Hybrid Evolutionary Techniques for Constrained Optimisation Design

Thesis submitted in accordance with the
requirements of the University of Liverpool
for the degree of Doctor in Philosophy by

Salam Nema

This research programme was carried out in collaboration
with the Knowledge Support Systems Retails Ltd.

Thesis Supervisors:
Dr. John Yannis Goulermas
Dr. Jason Ralph

Department of Electrical Engineering and Electronics
The University of Liverpool

November 2010

Abstract

This thesis a research program in which novel and generic optimisation methods were developed so that can be applied to a multitude of mathematically modelled business problems which the standard optimisation techniques often fail to deal with. The continuous and mixed discrete optimisation methods have been investigated by designing new approaches that allow users to more effectively tackle difficult optimisation problems with a mix of integer and real valued variables.

Over the last decade, the subject of optimisation has received serious attention from engineers, scientists, and modern enterprises and organisations. There has been a dramatic increase in the number of techniques developed for solving optimisation problems. Such techniques have been applied in various applications, ranging from the process industry and engineering, to the financial and management sciences, as well as operational research sectors. Global optimisation problems represent a main category of such problems. Global optimisation refers to finding the extreme value of a given nonconvex function in a certain feasible region. Solving global optimisation problems has made great gain from the interest in the industry, academia, and government.

In general, the standard optimisation methods have difficulties in dealing with global optimisation problems. Moreover, classical techniques may fail to solve many real-world problems with highly structured constraints, whereas achieving the exact global solution is neither possible nor desirable**.** One of the main reasons for their failure is that they can easily been trapped in local minima. To avoid this, the use of efficient evolutionary algorithms is proposed in order to solve difficult computational problems where acceptable solutions can be achieved. These techniques have many particular advantages over the

traditional optimisation methods, which allow them to be successfully applied in many difficult engineering problems.

The focus of this thesis presents practical suggestions towards the implementation of hybrid approaches for solving optimisation problems with highly structured constraints. This work also introduces a derivation of the different optimisation methods that have been reported in the literature. Major theoretical properties of the new methods have been presented and implemented. Here we present detailed description of the most essential steps of the implementation. The performance of the developed methods is evaluated against real-world benchmark problems, and the numerical results of the test problems are found to be competitive compared to existing methods.

Acknowledgments

I would like to thank my supervisor Dr. Yannis Goulermas for his crucial guidance, constant support and deep interest all the way through the work of this project. His encouragement and dedication was invaluable in developing those ideas presented here. I owe special thanks for him for carefully reading and invaluable suggestions and corrections of all the parts of this thesis.

Additionally, I am grateful to the help and support given by Dr. Phil Cook, for all research facilities that have been provided to me in KSS Ltd.

I also would also like to thank Mr. Graham Sparrow for his continual suggestions and support during this research. His suggestions have led to interesting and fertile discussion that made a vital contribution of the successful completion of this project.

I would like to thank all the staff of the KSS, for their help and encouragement in every little occasion. I also thank the Department of Electrical Engineering for their support and assistance since the start of my research work.

I wish to thank my parents whose love and support has been the foundation of all my achievements.

Contents

List of Figures

List of abbreviations

- AO Alternating Optimisation.
- BB Branch and Bound.
- COP Constrained Optimisation Programming.
- DE Differential Evolution.
- DF Derivative-Free.
- DP Deterministic Programming.
- EA Evolutionary Algorithms.
- ECP Extended Cutting Plane.
- EP Evolutionary Programming.
- GBD Generalized Bender's Decomposition.
- GO Global Optimisation.
- IP Integer Programming.
- GA Genetic Algorithms.
- LP Linear Programming.
- MDNLP Mixed Discrete Non-Linear Programming.
- MILP Mixed Integer Linear Programming.
- MINLP Mixed Integer Non-Linear Programming.
- MIP Mixed Integer Programming.
- MOO Multi-Objective Optimisation.
- MIQP Mixed Integer Quadratic Programming.
- NLP Non-linear Programming.
- OA Outer Approximation.
- PSO Particle Swarm Optimisation.
- QN Quasi-Newton.
- QP Quadratic Programming.
- SLP Successive Linear Programming.
- SP Stochastic Programming.
- SCP Sequential Cutting Plane.

Notation

$\boldsymbol{\mathcal{R}}$	The set of real numbers.
N	The set of non-negative integers.
Ø	The empty set (without any element).
$\{x, y, z\}$	The set consisting of the three elements x, y and z.
$x \in X$	x is an element of the set X.
$x \notin X$	x is not an element of the set X.
$\lvert X \rvert$	The number of elements in the set X, the cardinality of X.
$\{x/_{x\ such\ that\ \ldots}\}$	The set of elements x such that …
$\exists x$:	There exists an element x such that …
$\forall x \epsilon X$	For any element x of X.
$A \times B$	Cartesian product of A and B.
$[a, b]$	A closed interval: $\left\{\frac{x}{x} \in \mathcal{R}; \text{a} \leq \text{x} \leq \text{b}\right\}$, where a and b are real numbers (a ≤ b).
$\inf_{x \in X}(x)$ of X.	If X has a lower bound, then $\inf_{x \in X}(x)$ is by definition the largest of the lower bound If X has no lower bound, then by convention $\inf(x) = -\infty$.
$\{x^k\}_{x \in \text{N}}$	Sequence of elements x^k, for $k = 1,2, \ldots$
$\mathcal{R}^n$	Cartesian product of the set , multiplied n times by itself.
$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix}$	The vector of $\mathcal{R}^n$ with components $x_1, \ldots, x_n$.
x^T	Transpose of the vector x of $\mathcal{R}^n$.
$x^T \cdot y$	Scalar product of the vector x and y.
$\lVert x \rVert$	Euclidian norm of the vector x.
$A = [a_{ij}]_{\substack{i=1,\ldots,m \\ j=1,\ldots,n}}$	Matrix with m rows and n columns, a_{ij} is the element in row i and column j.
A^T	Transpose of matrix A.

Rank(*A*)	rank of matrix *A* (dimension of the largest regular square submatrix of *A*).
$\nabla f(x)$	If $f(\mathcal{R}^n \to \mathcal{R})$ is a function of the variables $x_1, \dots, x_n$, then$\nabla f(x)$ is the gradient of *f* at the point x, that is the n-vector with components $\frac{\partial f}{\partial x_1}(x), \dots, \frac{\partial f}{\partial x_n}(x)$.
$\partial f/\partial x(x)$	Equivalent notation to mean $\nabla f^T(x)$, the transposed vector of $\nabla f(x)$. $\partial f/\partial x(x)$ is thus the same as the row-matrix with components $\frac{\partial f}{\partial x_1(x)}, \dots, \frac{\partial f}{\partial x_n(x)}$.
$\nabla^2 f(x)$	If $f(\mathcal{R}^n \to \mathcal{R})$ is a function of the variables $x = (x_1, \dots, x_n)$, then $\nabla^2 f(x)$ is the Hessian of *f* at the point x, that is the real $n \times n$ (symmetrical) matrix whose (i, j) element is $\frac{\nabla^2 f}{\partial x_i \partial x_j}(x)$.
$\partial f(x)$	Subdifferential of f at x: set of the subgradients of f at x(for the convex or a concave function).
$\inf_{x \in X}\{f(x)\}$	By definition, $\inf_{x \in X}\{f(x)\} = \inf_{x \in X}(y)$. where $f: \mathcal{R}^n \to \mathcal{R}$ and $X \subset \mathcal{R}^n, X \neq \emptyset, Y = \{\frac{y}{y} \in \mathcal{R}; \exists x \in \text{X: } y=f(x)\}$.
$\sup_{x \in X}\{f(x)\}$	By definition $\sup_{x \in X}\{f(x)\} = -\inf_{x \in X}\{-f(x)\}$
$\underset{x \in X}{Min}\{f(x)\}$	Let $f: \mathcal{R}^n \to \mathcal{R}$ and $X \subset \mathcal{R}^n$. If $-\infty < \inf_{x \in X}\{f(x)\} < \infty$ and there exists $\bar{x} \in X$ such that $f(\bar{x}) = \inf_{x \in X}\{f(x)\}$,then $\underset{x \in X}{Min}\{f(x)\} = f(\bar{x})$. If $\inf_{x \in X}\{f(x)\} = -\infty$, then by convention $\underset{x \in X}{Min}\{f(x)\} = f(\bar{x}) = -\infty$. If $X = \emptyset$, then by convention $\underset{x \in X}{Min}\{f(x)\} = f(\bar{x}) = \infty$.
$\underset{x \in X}{Max}\{f(x)\}$	By definition: $\underset{x \in X}{Max}\{f(x)\} = -\underset{x \in X}{Min}\{-f(x)\}$.
$Min\{a, b, c\}$	Minimum of the three real numbers *a*, *b* and *c*.

Chapter 1

General Introduction

1.1 Overview

Optimisation problems are generally composed of three parts: an objective function that needs to be optimised, a set of variables that define the problem and on which the objective function depends, and a set of constraints that restrict feasible values of these variables. Constraints reduce the feasible space wherein solutions to the problem can be found. Formulation of an optimisation problem involves taking statements, defining general goals and requirements of a given activity, and transcribing them into a series of well-defined mathematical statements. More precisely, the formulation of an optimisation problem involves: selecting one or more optimisation variables, choosing an objective function, and identifying a set of constraints. Optimisation algorithms need to ensure that a feasible solution is found. That is, the optimisation algorithm should find a solution that both optimises the objective function and satisfies all constraints. If it is not possible to satisfy all constraints, the algorithm has to balance the trade-off between optimal objective function value and number of constrains violated.

Optimisation can be applied to all disciplines. Many recent problems in Engineering, Science, and Economics can be presented as computing globally optimal solutions. Using classical nonlinear programming techniques may fail to solve such problems because these problems usually contain multiple local optima [1]. Therefore, global search methods should be invoked in order to deal with such problems.

1.2 Objectives

In this research, constrained optimisation problem is considered in the continuous and discrete search space. The overall objectives of the present thesis were to develop novel hybrid versions of hybrid evolutionary approaches as promising solvers for the considered problems. The designed algorithms aimed to overcome the drawbacks of slow convergence and random constructions of stochastic methods. In these hybrid methods, local search strategies are inlaid inside the evolutionary approaches in order to guide them especially in the vicinity of local minima, and overcome their slow convergence especially in the final stage of the search.

The optimisation problems exist in many applications and achieving the exact global solution is neither possible nor desirable. Therefore, using efficient global search methods is highly needed in order to achieve optimal global solutions. It has been found that evolutionary algorithms produce good results when applied to these problems and they could obtain highly accurate solutions in many cases [2]. The power of evolutionary techniques come from the fact that they are robust and can deal successfully with a wide range of problem areas. However, these methods, especially when they are applied to complex problems, suffer from the slow convergence and the high computational cost. The main reason for this slow convergence is that these methods explore the global search space by creating random movements without using much local information about promising search direction. In contrast, local search methods have faster convergence due to their using local information to determine the most promising search direction by creating logical movements. However, local search methods can easily be entrapped in local minima.

Our work has the objective to combine evolutionary approaches with gradient-based search methods to design more efficient algorithms with relatively faster convergence than the pure evolutionary methods. Furthermore, these hybrid methods are not easily entrapped in local minima because they still maintain the merits of the stochastic search. In this study, generic hybrid algorithms that combine these methods

are developed in order to deal with the global optimisation problems that have the above characteristics. Specifically, local search guidance in the direct search methods is invoked to direct and control the global search features of evolutionary approach to design more efficient hybrid methods.

1.3 Organisation and Contributions

In this thesis, details of the implementation are rather technical, thus it makes sense to firstly investigate the details of the optimisation approaches for solving constrained optimisation problems, and demonstrate some examples of the implemented algorithms. Secondly, theoretical and technical details of the developed methods appear towards the end of the thesis detailing the lower level aspects of our approaches.

The thesis is organised into seven chapters as follows:

- The first chapter is introduction to this research program in which novel and generic optimisation methods were developed and implemented in order to tackle constrained optimisation problems.

- In chapter 2, some well-known deterministic and stochastic search methods are introduced to be used throughout this study. Stochastic search methods are a relatively recent development in the optimisation field, aimed to tackle difficult problems, such as ones afflicted by non-differentiability. Although a casual understanding of some deterministic methods will be useful here, especially where the methods are analogous to certain stochastic approaches.

- The third chapter presents the solution of a collection of test models for continuous and mixed-variables nonlinear programming. It also introduces some modern solvers that are used for solving such problems. Results are reported for testing a number of existing state-of-the-art solvers for global constrained optimisation and constraint satisfaction on different test problems, collected from the literature. This chapter also shows the implementation of different deterministic and evolutionary methods for solving nonlinear constrained optimisation problems.

- In chapter 4, a new hybrid coevolutionary algorithm is presented. This approach is capable of solving difficult real-world constrained optimisation problems formulated as min-max problems with the saddle point solution. A two-group model has been considered; in such models individuals from the first group interact with individuals from the second through a common fitness evaluation based on payoff matrix game. The new approach is fast and capable of global search because of combining particle swarm optimisation and gradient search to balance exploration and exploitation. When applying this algorithm to specific real-world problems, it is often found that the addition of gradient-search mechanism can aid in finding a good global optimal solution. The developed algorithm is particularly suited for difficult optimisation problems in the sense that the objective function and the constraints are nonsmooth functions and the problem has multiple local extrema.

- The fifth chapter introduces an original method for solving general Mixed Discrete Non-Linear Programming problems, based on the generic framework of Alternating Optimisation and Augmented Lagrangian Multipliers method. An iterative solution strategy is proposed by transforming the constrained problem into two unconstrained components or units; one solving for the discrete variables, and another for the continuous ones. During the search process, each unit focuses on minimising a different set of variables while the other type is frozen. While optimising each unit, the penalty parameters and multipliers are consecutively updated until the solution moves towards the feasible region. The performance of the algorithm is evaluated against real-world benchmark problems; the experiment results indicate that the algorithm achieves an exact global solution with better computational cost compared to the existing algorithms.

- The sixth chapter, we developed a hybrid particle swarm optimisation with branch and bound architecture, which is based on the fact that the evolutionary algorithm has the ability to escape from local minima, while the gradient-based method exhibits faster convergence rate. The designed algorithm retains and combines these attractive properties of both teqchneqies; while at

the same time mitigates significantly their aforementioned weaknesses. It is particularly suited for difficult nonlinear mixed discrete optimisation problems, in the sense that the objective function and the constraints are non-smooth functions and have multiple local extrema. The algorithm takes advantage of the rapid search of BB, when the evolutionary method has discovered a better solution in its globally processed search space. The hybridisation phase of the new algorithm depends primarily on a selective temporary switching from particle swarm optimisation and branch and bound methods, when it appears that the current optimum can be potentially improved. As will be described later, any such potential improvement is recorded and broadcasted to the entire swarm of the particles via its social component update. The validity, robustness and effectiveness of the proposed algorithm are exemplified through some well known benchmark mixed discrete optimisation problems

- The seventh chapter presents Conclusions and issues for further work, and indicate the potential usefulness of the developed approaches.

It has been found that applying a complete local search method in the final stage of the evolutionary search techniques helps them to obtain good accuracy quickly. The new hybrid methods are promising in practice and competitive with the other compared methods in terms of computational costs and the success of obtaining the global solutions. The algorithm developed in chapter 4 shows a superior performance in terms of the solution qualities against the compared methods. In Chapter 5 and 6, two new algorithms have been proposed as hybrid methods that combine specific strategies to fit the the development of the field of general mixed discrete nonlinear programming**.** Inheriting the advantages of the deterministic and stochastic approaches, the new methods are efficient and capable of global search. Simulation results based on well-known mixed continuous-discrete engineering design problems demonstrate the effectiveness and robustness of the proposed algorithm. The deterministic search method scheme applied in the final stage can overcome the slowness of the evolutionary algorithm in its final stage and helps in achieving higher quality solutions. Moreover, the proposed new methods are promising

in practice and competitive with some other population-based methods in terms of the solution qualities. The numerical results shown in chapters 3–6 show that creating gradient-based techniques while applying stochastic approach in the proposed methods give better performance of metaheuristics. In addition, accelerating the final stage of the evolutionary methods by applying a complete local search technique extricates evolutionary methods from wandering around the optimal solution.

Chapter 2

Methods for Constrained Optimisation

2.1 Introduction

This Chapter introduces the reader to elementary concepts of generic formulations for linear and nonlinear optimisation models, and provide some illustration for different optimisation methods. In general, there are two classes of optimisation methods for solving continuous and mixed discrete design problems: stochastic and deterministic ones. Stochastic search methods are a relatively recent development in the optimisation field, aimed to tackle difficult problems, such as ones afflicted by non-differentiability, multiple objectives and lack of smoothness. Although a casual understanding of some deterministic methods will be useful here, especially where the methods are analogous to certain stochastic approaches. In this chapter, some well-known deterministic methods and metaheuristics are introduced to be used throughout this study.

2.2 Deterministic Search Techniques

This section has its objective the discussion of techniques, most of which are derived from the gradient-based algorithms literature. It deals with techniques that are applicable to the solution of the continuous constrained optimisation problem: The Mathematical formulation can be stated as:

$$\begin{aligned} \underset{x}{Min} \quad & f(\vec{x}) \\ & g_i(\vec{x}) \le 0, \quad i = 1,\ldots,m \\ s.t \quad & h_j(\vec{x}) = 0, \quad j = 1,\ldots,l \\ & \vec{x} \in X \subseteq \Re^n \end{aligned} \tag{2.1}$$

where $\vec{x}$ represents a vector of n real variables subject to a set of m inequality constraints $g(\vec{x})$ and a set of l equality constraints $h(\vec{x})$.

There are many techniques available for the solution of a constrained nonlinear programming problem. All the methods can be classified into two broad categories: direct methods and indirect methods. In the direct methods, the constraints are handled in an explicit manner, whereas in most of the indirect methods, the constrained problem is solved as a sequence of unconstrained optimisation problems [3].

In this section, direct search methods are presented in order to deal with the constrained optimisation problems that have the above characteristics. In this research, local search guidance in the direct search methods is invoked to direct and control the global search features of metaheuristics to design more efficient hybrid methods. In the rest of this chapter, some well-known direct and indirect search methods are introduced briefly to be used throughout this study. These techniques can be classified as follows:

- **Direct Search Methods**
  - Random search methods.
  - Heuristic search methods.
  - Complex methods.
  - Objective and constraints approximation methods.
  - Sequential quadratic programming method.
  - Methods of feasible directions.
  - Zoutendijk's method.
  - Rosen's gradient projection method.
  - Generalized reduced gradient method.

- **Indirect Search Methods**
  - Transformation of variables techniques.
  - Sequential unconstrained optimisation techniques.
  - Interior penalty function method.
  - Exterior penalty function method.
  - Augmented Lagrange multiplier method.

These search methods have been designed for solving unconstrained optimisation problems. However, constrained handling techniques can be used to deal with constrained optimisation problems. More details about these methods can be found in [4].

2.2.1 Mixed Integer Continuous Programming

Mixed Integer Programming (MIP) refers to mathematical programming with continuous and discrete variables and linearity or non-linearity in the objective function and constraints. The use of MIP is a natural approach of formulating problems where it is necessary to simultaneously optimise the system structure (discrete) and parameters (continuous). These problems arise when some of the variables are required to take integer values. MIPs have been used in various applications, including the process industry and the financial, engineering, management science and operations research sectors.

A mixed-integer linear program (MILP) is a mathematical program with linear constraints in which specified subsets of the variables are required to take on integer values. Although MILPs are difficult to solve in general, the past ten years has seen a dramatic increase in the quantity and quality of software - both commercial and non-commercial - designed to solve MILPs. The MILP formulation with 0-1 variables is stated as

$$
\begin{aligned}
\min \quad & C^{T} x + d^{T} y \\
\text{subject to} \quad & Ax + By \le b \\
& x \ge 0, \quad x \in X \subseteq \Re^{n} \\
& y \in \{0,1\}^{Q}
\end{aligned}
\tag{2.2}
$$

where, x is a vector of n continuous variables,

y is a vector of q 0-1 variables,

c, d are $n \times 1$ and $q \times 1$ vectors of parameters,

A, B are matrices of appropriate dimension,

b is a vector of p inequalities.

Mixed Integer Linear programming methods and codes have been available and applied to many practical problems for more than twenty years. The major difficulty that arises in mixed- integer linear programming MILP problems for the form (2.2) is due to the combinatorial nature of the domain of y variables. Any choice of 0 or 1 for the elements of the vector y results in a LP problem on the x variables which can be solved for its best solution.

One may follow the brute-force approach of enumerating fully all possible combinations of 0 -1 variables for the elements of the y vector. Unfortunately, such an approach grows exponentially in time with respect to its computational effort. For instance, if we consider one hundred 0 -1 y variables then we would have 2^{100} possible combinations. As a result, we would have to solve 2^{100} LP problems. Hence, such an approach that involves complete enumeration becomes prohibitive [1]. MINLP refers to mathematical programming with continuous and discrete variables and nonlinearities in the objective function and constraints. MINLP problems are difficult to solve, because they combine all the difficulties of both, the nature of mixed integer programs (MIP) and the difficulty in solving convex or nonconvex nonlinear programs (NLP).

MINLPs have been used in various applications, including the process industry and the financial, engineering, management science and operational research sectors. As the number of binary variables y in form (2.2) increase, one is faced with a large combinatorial problem, and the complexity analysis results characterize the MINLP problems as NP- complete. At the same time, due to the nonlinearities the MINLP problems are in general nonconvex which implies the potential existence of multiple local solutions.

Considerable interest was shown for discrete variables engineering optimisation problems in the late 1960s and early 1970s. However, at that time optimisation methods for continuous nonlinear programming (NLP) problems were not developed, so the focus shifted to the development and evolution of numerical algorithms for such problems. In the 1970s and 80s, substantial effort was put into developing and evaluating algorithms for continuous NLP problems. Therefore, in recent years, the focus has shifted back to applications of optimisation techniques to practical engineering problems that naturally used mixed discrete and continuous variables in their formulation [4] .The component structure of MIP and NLP within MINLP provides a collection of natural algorithmic approaches. They can be classified as:

- **Classical solution methods**
  - Branch and Bound (BB)
  - Outer Approximation (OA)
  - Extended Cutting Plane methods (ECP)
  - Generalized Bender's Decomposition (GBD)
- **Hybrid methods**
  - LP/NLP based Branch and Bound
  - Integrating SQP with Branch-and-Bound
  - Sequential Cutting Plane (SCP)
  - Outer-Approximation based Branch-and-Cut algorithm

In the rest of this chapter, the important details of these methods are explained to be used throughout this study. Implementation and numerical examples will be presented in the following chapters.

2.2.2 Branch and Bound method

A general branch and bound method for MIP problems is based on the key ideas of separation, relaxation, and fathoming [2]. The branch and bound (BB) method starts by solving first the continuous NLP relaxation. If all discrete variables take integer values the search is stopped. Otherwise, a tree search is performed in the space of the integer variables. Then the algorithm selects one of those integer variables which take a non-integer value, and branch on it.

Branching generates two new sub problems by adding simple bounds to the NLP relaxation. One of the two new NLP problems is selected and solved next. If the integer variables take non-integer values then branching is repeated. If one of the fathoming rules is satisfied, then no branching is required, and the corresponding node has been fully explored. The fathoming rules are:

- Infeasible solution is detected.
- An integer feasible node is detected.
- A lower bound on the NLP solution of a node is greater or equal than the current upper bound.

Once a node has been fathomed the algorithm backtracks to another node which has not been fathomed until all nodes are fathomed.

Branching variable selection

Since branching is in the core of any BB algorithm, finding good strategies was important to practical MIP solving right from the beginning. Suppose we have chosen an active node i, associated with it is the non-linear programming solution x_i. Next we must choose a variable to define the division. The following describes the most commonly used variable selection strategies:

1. Lowest-Index-First:

It is possible to have some information on the importance of some of the integer variables in a given model. The integer variables are arranged in order of importance, the most important of these being processed first. This is accomplished by indexing the variables with the decreasing priorities of the integer variables and selecting the variable with the lowest index first.

2. Most Fractional Integer variables:

This strategy selects the variable which is farthest from the nearest integer value. This choice is aimed at getting the largest degradation of the objective when branching is carried out so that more nodes can be fathomed at an early stage.

3. Use of Pseudo-Costs:

The concept of Pseudo-cost was first developed for solving mixed integer linear programming problems. The pseudo-costs are used as a quantitative measure of the importance of the integer variables and this allows the assignment of some priority to the variables. For each integer variable x_j two quantities are defined, lower pseudo-cost (pcl_j) and upper pseudo-cost (pcu_j). The values of the lower and upper pseudo-costs are computed during the tree search.

Selection of Branching Node

The selection method for branching node may significantly affect the performance of branch and bound algorithm [1]. The most commonly used branching strategies are:

- *Depth-first:*

  Whenever a branching is carried out the nodes corresponding to the new problems are given preference over the rest of the unfathomed nodes. The child nodes are created as the next nodes to optimise.

- *Best-first:*

  In this strategy, the node which currently has the lowest bound in the objective is selected for branching. The first solution found is usually close the optimal solution of the problem.

- *Breadth-first:*

  We examine a certain level in the tree entirely before proceeding to the next level in the tree in order to get shorter search paths in the tree.

  Hybrid methods

- *Depth-first-then-breadth:*

  In the method a depth first search is performed until the first solution is found. The algorithm then switches to a breadth first search strategy.

- *Depth-first-then-best:*

  As the previous method, a depth first strategy is performed until the first solution is found, then switching to a best first method.

- *Depth-first-with-backtracking:*

  Backtracking is a systematic way to go through all the possible configurations of a space.

2.2.3 Outer Approximation method

This algorithm is based on the concept of defining an MILP master problem. Relaxations of such a master problem are then used in constructing algorithms for solving the MINLP problem. The method presented here is a generalization of Outer approximation proposed by Duran and Grossman [5]. We shall next present the reformulation of P as an MILP master problem. Based on this reformulation an algorithm is presented which solves a finite sequence of NLP sub problems and a MILP or MIQP master problem, respectively.

The MINLP model problem P is reformulated as an MINLP problem using Outer approximation; the reformulation employs projection onto the integer variables and linearization of the resulting NLP sub problems by means of supporting hyper planes.

It can be shown that it suffices to add the linearisation of strongly active constraints to the master program. This is very important since it reduces the size of the MILP master program relaxation that is

solved in the Outer Approximation Algorithms. In this subsection the simplifying assumption is made that all $y \in Y$ are feasible. The first step in reformulating P is to define the NLP sub problem.

$$NLP(y^j)\begin{cases}\min\limits_{x} & f(x, y^j) \\ s.t. & g(x, y^j) \le 0 \\ & x \in X\end{cases} \tag{2.3}$$

In which integer variables are fixed at the value $y = y^j$. By defining $v(y^j)$ as the optimal value of the sub problem $NLP(y^j)$ it is possible to express P in terms of a projection onto the y variables, that is $\min\limits_{y^j \in Y}(v(y^j))$

$$\text{Master problem}\begin{cases}\text{minimise}\limits_{y,\eta} & \eta \\ \text{subject to} & \eta \ge f_i + (\mu_i)^T (y - y_j) \forall y_j \in Y \\ & y \in Y \quad \text{integer}\end{cases} \tag{2.4}$$

The assumption that all are feasible implies that all sub problems are feasible. Let x^j denote an optimal solution of NLP(y^j) for $y^j \in Y$. In order to derive a correct representation it is necessary to consider how NLP solvers detect infeasibility. Infeasibility is detected when convergence to an optimal solution of a feasibility problem occurs. At such an optimum, some of the nonlinear constraints will be violated and other will be satisfied and the norm of the infeasible constraints can only be reduced by making some feasible constraints infeasible. The equivalent MILP problem is

$$MILP = \left\{\begin{array}{ll}\min\limits_{x,y,\eta} & \eta \\ s.t & \eta \ge f^{\,j} + [\nabla f^{\,j}]^{\mathrm{T}} \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} \\ & 0 \ge g^{\,j} + [\nabla g^{\,j}]^{\mathrm{T}} \begin{pmatrix} x - x^j \\ y - y^j \end{pmatrix} \\ \forall\ y^j \in Y,\ x \in X,\ y \in Y \text{ integer} & \end{array}\right\} \tag{2.5}$$

The relaxation of the master problem can be employed to solve the model problem P. The resulting algorithm is shown to iterate finitely between NLP sub problems and MILP master problem relaxations. This algorithm is shown to be efficient if curvature information is present in the problem.

Linear Outer Approximation Algorithm:

- Initialisation

Repeat $i = 0, T^{-1} = \phi, S^{-1} = \phi, UBD^{-1} = \infty$

Where $T = \{j = NLP(y^j) is\ feasible\ and\ x^j\ is\ an\ optimal\ solution\ to\ NLP(y^j)\}$

$S = \{k = NLP(y^k)\ is\ in feasible\ and\ x^k\ solves\ F(y^k)\}$

1. Solve NLP(y_j) of F(y_j),the solution is x_j
2. Linearize Objective and constraints function about (x_j , y_j).
3. If (NLP(y_j) feasible) Then

   update current best point by setting $x^* = x^j, y^* = y^j, UBD = f^i$

   else $UBD^j = UBD^{j-1}$
4. Solve the current relaxation Mj of the master program M, giving a new y_{j+1}.
5. Set j = j+1 Until (Mj is infeasible).

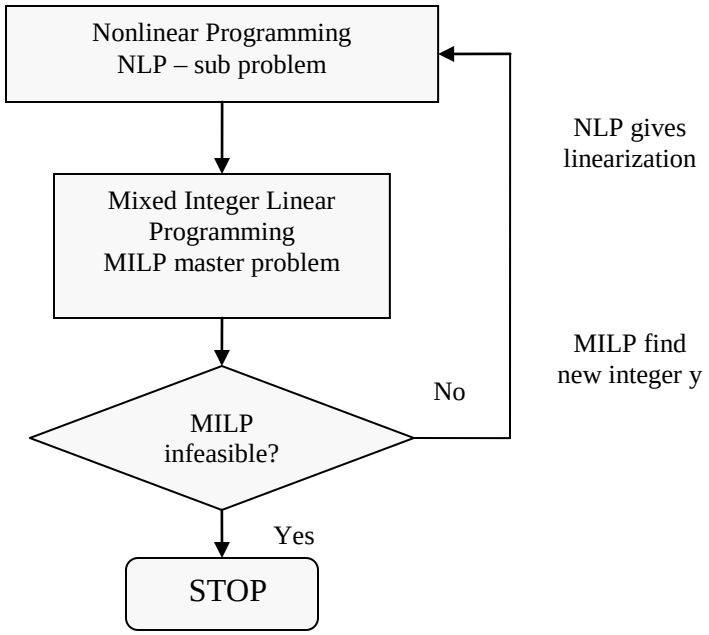


Fig. 1.1 Linear Outer Approximation algorithm.

2.2.4 Extended Cutting Plane method

The ECP method, which is an extension of Kelly's cutting plane algorithm for convex NLP, does not rely on the use of NLP sub problems and algorithms [2]. It relies only on the iterative solution of the problem (M-MIP) by successively adding a linearization of the most violated constraint at the predicted point

$$(x^k, y^k) \ : \ J^k = \{ \ \hat{J} \in \arg \{\max_{j \in J} \ g_j(x^k, y^k) \ \} \tag{2.6}$$

Convergence is achieved when the maximum constraint violation lies within the specified tolerance. The optimal objective value of (M-MIP) yields a non-decreasing sequence of lower bounds. It is possible to either add to (M-MIP) linearization of all the violated constraints in the set J^k, or linearization of all the nonlinear constraints $j \in J$. In the ECP method the objective must be defined as a linear function, which can easily be accomplished by introducing a new variable to transfer nonlinearities in the objective as an inequality.

The ECP method is able to solve MINLP problems, including general integer variables and not only binary variables, and no integer cuts are needed to ensure convergence. ECP methods are oftenly claimed to have slow convergence. The number of non-linear function evaluations used to obtain the optimal solution has in several cases been even magnitudes lower than when using MINLP methods based on solving NLP sub problems.

2.2.5 Generalized Bender's Decomposition (GBD) method

The GBD method is similar to the Outer- Approximation method [1]. The difference arises in the definition of the MILP master problem (M-MIP). The first step is to express P in terms of a projection onto the integer variables

$$\text{Proj(P)}\left\{\min_{y^j \in V}\left\{v'(y^j)\right\}\right\} \tag{2.7}$$

v is the set of all the integer assignments.

The optimal value of the NLP sub problems defined by:

$$\text{NLP}\begin{cases}\underset{x,y}{\text{minimise}} & f(x,y) \\ & g\ (x,y)\le 0 \\ \text{subject to} & y = y^{j} \\ & x\in X,\ y\in converged(Y)\end{cases} \tag{2.8}$$

Benders' Decomposition is able to treat certain nonconvex problems that are not readily solved by other methods such as BB or OA.

The reformulation does not contain the continuous variables x:

$$\underset{\text{GBD}}{\text{Master problem}}\begin{cases}\underset{y,\eta}{\text{minimise}} & \eta \\ \text{subject to} & \eta \ge f_i + (\mu_i)^T (y - y_j)\forall y_j \in Y \\ & y\in Y \quad \text{integer}\end{cases} \tag{2.9}$$

Decomposition models with integer variables usually decompose into a master problem that comprises al the integer variables and sub problems, which evaluate the remaining variables. Sub problems with integer variables introduce additional difficulties and require the use of nonlinear duality theory.

2.2.6 LP/NLP based Branch and Bound

This approach covers problems with nonlinearities in the integer variables. The motivation for the LP/NLP based branch and bound algorithm is that outer approximation usually spends an increasing amount of computing time in solving successive MILP master problem relaxation. This approach avoids the re-solution of MILP master problem relaxation by updating the branch and bound tree. Instead of solving successive relaxations of M, the algorithm solves only one MILP problem which is updated as new integer assignments are encountered during the tree search [2].

Initially an NLP sub problem is solved and the initial master program relaxation is set up from the supporting hyperplanes at the solution of the NLP - sub problem. The MILP problem is then solved by a branch and bound process with the exception that each time a node gives an integer feasible solution y^i.

Algorithm

1. Consider MILP branch and bound.
2. Interrupt MILP, when y_j found.
3. Solve NLP (y_j) to get x_j .
4. Linearise f , c about (x_j , y_j)
5. Add linearization to tree.
6. Continue MILP tree search.

   Until

   lower bound > upper bound

As in the two outer approximation algorithms the use of an upper bound implies that no integer assignment is generated twice during the tree search. Since both the tree and the set of integer variables are finite the algorithm eventually encounters only infeasible problems and the stack is thus emptied so that the procedure stops.

This method can also be applied to the GBD and ECP methods. The LP/NLP method commonly reduces quite significantly the number of nodes to be enumerated. The trade-off is that the number of MLP sub problems may increase. This method is better suited for problems in which the bottleneck corresponds to the solution of the MILP master problem.

2.2.7 Integrating SQP with Branch-and-Bound

An alternative to nonlinear branch-and-bound for convex MINLP problems is due Borchers and Mitchell [6]. They observed that it not necessary to solve the NLP at each node to optimality before branching and propose d an early branching rule, which branches on an integer variable before the NLP has converged.

The algorithm is based on branch and bound, but instead of solving an NLP problem at each node of the tree, the tree search and the iterative solution of the NLP are interlaced. Thus the nonlinear part of (P) is solved whilst searching the tree. The nonlinear solver that is considered in this method is a Sequential Quadratic Programming (SQP) solver. The basic idea underlying this approach is to branch early – possibly after a single QP iteration of the SQP solver.

This approach has a similar motivation as the Outer Approximation algorithm [5] but avoids the resolution of related MILP master problems by interrupting the MILP branch and bound solver each time an integer node is encountered. At this node an NLP problem is solved and new outer approximations are added to all problems on the MILP branch and bound tree. Thus the MILP is updated and the tree search resumes.

Algorithm

1. Initialisation: Obtain the continuous relaxation of P
2. Set the upper bound to infinity
3. While (there are pending nodes in the tree) do
   - Select an unexplored node
   - Repeat (SQP iteration)

     Solve QPs for a step dk.

     if (QPs infeasible) ***then*** fathom node and ***exit***

     Set $(x^{k+1}, y^{k+1}) = (x^k, y^k) + (d_x^k, d_y^k)$

     if(($y^{(k+1)}$) integral) ***then***

     Update current best point by setting

     $$(x^*, y^*) = (x^{k+1}, y^{k+1}),\ f^* = f^{k+1} \text{ and } U = f^*$$

     else choose a non integral $y^{(k+1)}$ and branch

     endif

     exit

     endif
4. Compute the integrity gap $\theta = \max_i | y_i^{k+1} - round(y_i^{k+1}) |$
5. ***if*** $(\theta > \tau)$ ***then***

   - Choose a non-integral $y^{(k+1)}$ and branch, ***exit***

   end if

   end while

The value of $\tau = 0.1$ is suggested for the early branching rule and this value has also been chosen here. The algorithm has to be modified if a line search or a trust region is used to enforce global convergence for SQP. The key idea in the convergence proof is that the union of the child problems that are generated

by branching is equivalent to the parent problem. This algorithm has two important advantages. First, the lower bounding can be implemented at no additional cost. Secondly, the lower bounding is available at all nodes of the branch and bound tree.

2.2.8 Sequential Cutting Plane (SCP)

This algorithm integrates cutting plane techniques with branching techniques. Rather than solving a linearized MILP problem to feasibility or optimality, it applies cutting planes in each node of the branch and bound tree. The technique differs from the α-ECP method, where the generation of cutting planes is separated from the branching process [7]. For the NLP sub problems, we solve a sequence of linear programming (LP) problems. Note that the SCP algorithm could also be considered to be a form of Successive Linear Programming (SLP). However, a more general version of the SCP algorithm could, if desired, also retain the cutting planes between the LP sub iterations. The algorithm also generates explicit lower bounds for each node in the branch and bound tree. The algorithm obtains explicit lower bounds on the nodes when performing NLP iterations in the nodes.

The first LP sub iteration within NLP iteration provides a lower bound on the node. When branching, the child nodes inherit the lower bound of the parent node. Whenever the current upper bound is improved, you may drop any node with a lower bound greater than or equal to the current upper bound. You may, therefore, in some cases drop nodes in the tree without solving any additional LP problems for those nodes. Explicit lower bounds have a significant impact on the convergence speed as it means less sub problems solved.

The algorithm builds a branch and bound tree where each node represents a relaxed NLP sub problem of the original problem (P). Each NLP sub problem is solved using a sequence of LP problems, but the NLP sub problem is not solved to optimality. We then choose an integer variable with a non-integral value in the current iteration and branch on this variable generating two new NLP sub problems. The first LP

problem in an NLP iteration provides a lower bound for the optimal value of the NLP sub problem. The lower bounds of the nodes can be used for removing nodes from the tree any time we improve the currently best known solution for the original problem (P).

The algorithm does not solve the NLP sub problems to optimality. It interrupts the NLP procedure before an optimal point has been found in order to make the branch and bound algorithm faster. If the current iteration is converging to a non-integral point, we may branch early on any variable in y^k that has a non-integral value, rather than waste effort on finding an optimal, non-integer, solution for the current sub problem.

The algorithm uses an NLP version of the Sequential Cutting plane algorithm to solve the NLP sub problems. The difference from the SQP approach is that it solves a sequence of LP problems rather than QP problems in order to find a solution to the NLP sub problems.

2.2.9 Outer-Approximation based Branch-and-Cut algorithm

The algorithm integrates Branch and Bound, Outer Approximation and Gomory Cutting Planes [5]. Only the initial Mixed Integer Linear Programming (MILP) master problem is considered. At integer solutions nonlinear Programming (NLP) problems are solved, using a primal-dual interior point algorithm. The objective function and constraints are linearised at the optimum solution of those NLP problems and the linearisations are added to all the unsolved nodes of the enumerations tree. Also, Gomory cutting planes, which are valid throughout the tree, are generated at selected nodes. These cuts help the algorithm to locate integer solutions quickly and consequently improve the linear approximation of the objective and constraints, held at the unsolved nodes of the tree.

The complete algorithm is described in the following:

Algorithm

1. Initialisation: $\psi_o = \{0,1\}^p$ is given; set $i = 1, \hat{T}_{-1} = \hat{S}_{-1} = \phi$.

2. Set up initial master problem:

2.1 If NLP(ψ_o) is feasible, solve it and set $\hat{T}_0 = \{0\}$.

Otherwise solve F(ψ_0) and set $\hat{S}_0 = \{0\}$. Let x_0 be the optimum of NLP (ψ_0) or F(ψ_0) .

2.2 If x_0 is the optimum of NLP(ψ_0) then set $UBD_0 = f(x_0, \psi_o)$.

Otherwise set $UBD_0 = \infty$.

2.3 Linearise objective and constraints about (x_0, ψ_o) and form the initial 0-1 MILP master problem $\hat{M}_0$.

2.4 Define $\hat{M}_0$ as the root of the search tree. Let Λ be the list which contains the unsolved nodes and set.

3. Node selection: If $\Lambda = \phi$, then Stop. Otherwise select a node (R_0^i, R_1^i) and remove it from the list Λ .

4. Solve the LP relaxation of the 0-1 MILP problem $\hat{M}_i$ and let $(\hat{x}, \hat{\psi}, \eta)$ be its optimum solution.

5. If $\hat{\psi} \in \{0,1\}^p$ then

5.1 Set $\psi_i = \hat{\psi}$ and solve NLP(ψ_i) if it is feasible or F(ψ_i) otherwise. Let x_i be the optimum of NLP(ψ_i) or F(ψ_i).

5.2 Linearise objective and constraints around (x_i, ψ_i) and set $\hat{T}_i = \hat{T}_{i-1} \cup \{i\}$ or $\hat{S}_i = \hat{S}_{i-1} \cup \{i\}$ as appropriate.

5.3 Add the linearisations to $\hat{M}_i$ and to all the nodes in Λ . Place $\hat{M}_i$ back in Λ .

5.4 Update incumbent solution and upper bound:

If NLP(ψ_i) is feasible and $f(x_i, \psi_i) < UBD_i$ then

$(x_*, \psi_*) = (x_i, \psi_i)$ and $UBD_{i+1} = f(x_i, \psi_i)$

Otherwise $UBD_{i+1} = UBD_i$

5.5 Pruning: Delete all nodes from Λ with $\eta > UBD_{i+1}$.

Go to Step 3

6. If $\hat{\psi} \notin \{0,1\}^p$ then

6.1 Cutting versus Branching Decision: If cutting planes should be generated then go to Step 6.2. Otherwise go to Step 6.3

6.2 Cut generation: Generate a round of Gomory mixed integer cuts using every row corresponding to a fractional basic variable in the optimal tableau of an LP relaxation.

Add all those cuts to $\hat{M}_i$ and store them in the pool. Go to step 4.

6.3 Branching: Select a violated 0-1 variable in $\hat{\psi}$ say $\hat{\psi}^{(r)} \in (0,1)$. Create two new nodes $(R_0^{i+1}, R_1^{i+1}) = (R_0^i \cup \{r\}, R_1^i)$ and $(R_0^{i+1}, R_1^{i+1}) = (R_0^i, R_1^i \cup \{r\})$.
Add both nodes to the list Λ. Go to Step 4.

If $UBD_i < \infty$ 1 upon the termination of the algorithm, then (x_*, ψ_*) is the optimal solution of the original 0-1 MINLP problem. Otherwise the problem is infeasible.

Algorithm OA-BC requires an initial 0-1 vector to be given by the user. If such a vector is not available then the algorithm can start by solving the NLP relaxation of the initial 0-1 MINLP problem. If the solution of the NLP relaxation satisfies all the integrality constraints then that solution also solves the initial 0-1 MINLP problem and the algorithm can stop. If the NLP relaxation is infeasible then the initial 0-1 MINLP problem is also infeasible and the algorithm can stop. Finally if the NLP relaxation is feasible and has a non-integer optimum solution, then the initial 0-1 MILP master problem can be formulated by linearising the objective and constraints around that solution.

2.3 Comparison of MINLP optimisation methods

The MINLP optimisation methods represent quite different solution approaches. A comparison between all these methods can be described as the following:

Comparison of MINLP methods		
Method	*Advantages*	*Disadvantages*
BB	Finds an optimal solution if the problem if of limited size and enumeration can be done in reasonable time.	Extremely time consuming. The number of nodes in a branching tree can be too large.
OA	Avoids solving huge number of nonlinear programming problems.	- Potentially large number of iterations. - Adding Hessian term to the MILP becomes MIQP.
ECP	- Only solving MILP problems instead of NLP problems, in each iteration the nonlinear constraints need not be calculated at relaxed values of the integer variables. - solves MINLP problems including general integer variables and not only binary variables.	- It has slow convergence. - The most time consuming step in the ECP algorithm is the solution of the MILP sub problems.
GBD	- Solves a large scale of linear programs. - Problems have a special structure called Block Diagonal structure. Only active inequalities are considered.	The MILP master problem is given by a dual representation of a continuous space.

LP/NLP-BB	- This approach avoids the re-solution of MILP master program relaxations by updating the branch and bound tree. - This approach covers problems with non-linearities in the integer variables.	- Unlike ordinary branch and bound, a node cannot be assumed to have been fathomed, if it produces an integer feasible solution. - QP/NLP differs from LP/NLP because QP rather than LP problems are solved in the tree search.
SQP-BB	- Its not necessary to solve the NLP at each node to optimality before branching. - This algorithm gives a factor of about 3 improvement in terms of CPU time compared to nonlinear BB.	It needs a good NLP solver to interrupt the SQP method after each QP solve.
SCP	- It obtains the lower bound for the current NLP sub problem directly from the solution of the first LP in each NLP iteration. - It does not require any additional solution to Lagrangian duality problems. - It introduces a new method for selecting the next child node to solve in a depth-first search strategy.	- For larger MINLP problems, the performance of the algorithm is still open. More numerical tests on considerably larger problems must be performed in order to get a more detailed picture of algorithmic performance.
OA-BC	- Integrating the construction of the outer approximation of the master problem into a single tree search. - The sequential solution of several MILPs is avoided.	- It spends most of the running time solving the NLPs. - The number of NLPs solved by the algorithm is much larger than the OA.

Table 1.1 Comparison of MINLP methods.

2.4 Evolutionary Programming

A large number of Evolutionary Algorithms (EA) have been developed. These EAs can be grouped based on how individuals are represented, which evolutionary operators are used, and how these are implemented. This chapter discusses briefly the concepts of Evolutionary and Coevolutionary Programming. Evolutionary Algorithms can be classified into two classes; population-based methods and point-to-point methods. In the latter methods, the search invokes only one solution at the end of each iteration from which the search will start in the next iteration. The population-based methods invoke a set of many solutions at the end of each iteration. This chapter highlights the principles of genetic algorithms, particle swarm optimisation, and differential evolution as examples of population-based methods, and simulated annealing as an example of point-to-point methods.

2.4.1 Simulated Annealing

Simulated annealing is a simple technique that can be used to find a global optimiser for continuous, integer and discrete nonlinear programming problems. The approach does not require continuity or differentiability of the problem functions because it does not use any gradient or Hessian information [8]. The SA algorithm successively generates a trial point in a neighbourhood of the current solution and determines whether or not the current solution is replaced by the trial point based on a probability depending on the difference between their function values. Convergence to an optimal solution can theoretically be guaranteed only after an infinite number of iterations controlled by a procedure called cooling schedule. The main control parameter in the cooling schedule is the temperature parameter T. The main role of T is to let the probability of accepting a new move be close to 1 in the earlier stages of the search and to let it be almost zero in the final stage of the search. A proper cooling schedule is needed in the finite-time implementation of SA to simulate the asymptotic convergence behaviour of the SA.

2.4.2 Genetic algorithms

A genetic algorithm (GA) is a procedure that tries to mimic the genetic evolution of a species. Specifically, GA simulates the biological processes that allow the consecutive generations in a population to adapt to their environment. The adaptation process is mainly applied through genetic inheritance from parents to children and through survival of the fittest. Therefore, GA is a population-based search methodology. Nowadays, GAs are considered to be the most widely known and applicable type of metaheuristics [9].

GA starts with an initial population whose elements are called chromosomes. The chromosome consists of a fixed number of variables which are called genes. In order to evaluate and rank chromosomes in a population, a fitness function based on the objective function should be defined. Three operators must be specified to construct the complete structure of the GA procedure; selection, crossover and mutation operators. The selection operator cares with selecting an intermediate population from the current one in order to be used by the other operators; crossover and mutation. In this selection process, chromosomes with higher fitness function values have a greater chance to be chosen than those with lower fitness function values. Pairs of parents in the intermediate population of the current generation are probabilistically chosen to be mated in order to reproduce the offspring. In order to increase the variability structure, the mutation operator is applied to alter one or more genes of a probabilistically chosen chromosome. Finally, another type of selection mechanism is applied to copy the survival members from the current generation to the next one. The GA operators of selection, crossover and mutation have been extensively studied. Many effective settings of these operators have been proposed to fit a wide variety of problems. The GA algorithm can be described as follows:

Algorithm

- *Initialisation:* Generate an initial population P_0. Set the crossover and mutation probabilities $P_c \in (0,1)$ and $P_m \in (0,1)$, respectively. Set the generation counter t := 1.

- *Selection:* Evaluate the fitness function F at all chromosomes in P_t. Select an intermediate population $P_t^{'}$ from the current population P_t .
- *Crossover:* Associate a random number from (0, 1) with each chromosome in $P_t^{'}$ and add this chromosome to the parents pool set S_t^P if the associated number is less than P_c. Repeat the following Steps 1 and 2 until all parents in S_t^P are mated:
  1. Choose two parents P_1 and P_2 from S_t^P .Mate P_1 and P_2 to reproduce children c_1 and c_2 .
  2. Update the children pool set S_t^c though $S_t^c := S_t^c \cup \{c_1, c_2\}$ and update S_t^c through $S_t^c := S_t^c - \{p_1, p_2\}$.
- *Mutation:* Associate a random number from (0, 1) with each gene in each chromosome in $P_t^{'}$,mutate this gene if the associated number is less than P_m, and add the mutated chromosome only to the children pool set S_t^c .
- *Stopping Conditions:* If stopping conditions are satisfied, then terminate. Otherwise, select the next generation P_{t+1} from $P_t \cup S_t^c$ .Set S_t^c to be empty, set $t := t+1$, and go to the selection step.

2.4.3 Differential Evolution

Differential Evolution (DE) can be categorized into a class of floating-point encoded, evolutionary optimisation algorithms. Currently, there are several variants of DE. The particular variant used throughout this investigation is the DE/rand/1/bin scheme. Generally, the function to be optimised, f, is of the form:

$$f(X): \Re^n \to \Re$$

As with all evolutionary optimisation algorithms, DE works with a population of solutions, not with a single solution for the optimisation problem. Population *P* of generation *G* contains n_{pop} solution vectors called individuals of the population. Each vector represents a potential solution for the optimisation problem.

$$P^{(G)} = X_i^{(G)} \qquad i = 1,\ldots,n_{pop}, G = 1,\ldots,G_{\max}$$

Where G_{max} is maximum number of generations reached.

So, the population *P* of generation *G* contains n_{pop} individuals each containing n_{param} parameters (*chromosomes* of individuals):

$$P^{(G)} = X_i^{(G)} = x_{i,j}^{(G)} \qquad i = 1,\ldots,n_{pop},\ j = 1,\cdots,n_{param}$$

In order to establish a starting point for optimum seeking, the population must be initialized. Often there is no more knowledge available about the location of a global optimum than the boundaries of the problem variables. In this case, a natural way to initialise the population $P^{(0)}$ (initial population) is to seed it with random values within the given boundary constraints:

$$P^{(0)} = x_{i,j}^{(0)} = r_{i,j}(x_j^{(U)} - x_j^{(L)}) + x_j^L \qquad i = 1,\ldots,n_{pop},\ j = 1,\ldots,n_{param} \tag{2.10}$$

where *r* denotes a uniformly distributed random value range [0.0,1.0].

The population reproduction scheme of DE is different from the other evolutionary algorithms. From the 1st generation forward, the population of the following generation $P^{(G+1)}$ is created in the following way on the basis of the current population $P^{(G)}$. First a temporary (trial) population for the subsequent generation, $P^{(G+1)}$, is generated as follows:

$$x'^{(G+1)}_{i,j} = \begin{cases} x^{(G)}_{C_{i,j}} + F.(x^{(G)}_{A_{j,j}} - x^{(G)}_{B_{i,j}}) & \text{if} \quad r_{i,j} \le C_r \vee j = D_i \\ x^{(G)}_{i,j} & \text{otherwise} \end{cases}$$

where

$i = 1,\ldots,n_{pop},\ j = 1,\ldots,n_{param}$

$D = 1,\ldots,n_{param}$

$A = 1,\ldots,n_{pop}, B = 1,\ldots,n_{pop}, C = 1,\ldots,n_{pop}, A_i \neq B_i \neq C_i \neq i$

$C_r \in [0,1], F \in [0,2], r \in [0,1]$ (2.11)

A, *B* and *C* are three randomly chosen indices referring to three randomly chosen individuals of population. They are mutually different from each other and also different from the running index *i*. New,

random, values for *A*, *B* and *C* are assigned for each value of index *i* (for each individual). A new value for the random number *r* is assigned for each value of index *j* (for each chromosome).

The index *D* refers to a randomly chosen chromosome and it is used to ensure that at least one chromosome of each individual vector $X'^{(G+1)}$ differs from its counterpart in the previous generation $X^{(G)}$. A new random (integer) value is assigned to *D* for each value of index *i* (for each individual).

F and *Cr* are DE control parameters. Both values remains constant during the search process. As well the third control parameter, *npop* (population size), remain constant, too. *F* is a real-valued factor in range [0.0,2.0] that controls the amplification of differential variations and *Cr* is a real-valued crossover factor in range [0.0,1.0] controlling the probability to choose a mutated value for *x* instead of its current value. Generally, both *F* and *Cr* affect the convergence velocity and robustness of the search process. Their optimal values are dependent both on objective function, *f(X)*, characteristics and on the population size *npop*. Usually, suitable values for *F*, *Cr* and *npop* can be found by trial-and-error.

The selection scheme of DE also differs from the other evolutionary algorithms. On the basis of the current population $P^{(G)}$ and the temporary population $P'^{(G+1)}$, the population of the next generation $P^{(G+1)}$ is created as follows:

$$X_i^{(G+1)} = \begin{cases} X'^{(G+1)}_i & \text{if} \quad f_{cost}(x'^{(G+1)}_i) \le f_{cost}(x_i^{(G)}) \\ X_i^{(G)} & \text{otherwise} \end{cases} \tag{2.12}$$

Thus, each individual of the temporary (trial) population is compared with its counterpart in the current population. The one with the lower value of cost function *fcost(X)* (to be minimised) will survive in the population of the next generation. As a result, all the individuals of the next generation are as good or better than their counterparts in the current generation. An interesting point concerning the DE's selection

scheme is that a trial vector is not compared against all the individuals in the current population, but only against one individual, i.e., against its counterpart in the current population.

2.4.4 Particle Swarm Optimisation

The PSO is a stochastic optimisation method based on the simulation of the social behaviour of bird flocks or biological groups in general, that evolve by information exchange among particles in a group. The PSO algorithm was first introduced by Kennedy and Ebehart [10] followed by a more general work on swarm intelligence [11]. In the PSO, the population is called a swarm and the individuals are called particles. Each individual in PSO flies in the search space and returns in its memory the best position it ever experienced.

The trajectory of each individual in the search space is adjusted by dynamically altering the velocity of each particle, according to its own experience (cognitive component) and the progress of the other particles in the search space (social component). The different types of PSO algorithms will be described in the later chapters.

2.4.5 Constraint-handling methods

A key factor in solving optimisation problems is how the algorithm handles the constraints relating to the problem. In order to apply evolutionary algorithms to constrained optimisation problems, additional mechanisms need to be employed to ensure that the search process focuses on the feasible space. Over the last few decades, several methods have been proposed to handle constraints in evolutionary algorithms [12]. These methods can be grouped to four categories:

- Methods that preserve the feasibility of solution.
- Methods based on penalty functions.
- Methods which make a clear distinction between feasible and infeasible solutions.
- Other hybrid methods.

The most common approach to improve evolutionary algorithms performance to deal with constrained optimisation problems is based on penalty functions. However, the major disadvantage of this approach is that there are some parameters which must be adjusted to guarantee convergence of the proposed method.

2.4.6 Coevolution

Coevolution is the complementary evolution of closely related species. Based on the kind of interaction between individuals of the different spices, two kinds of coevolutionary algorithms are identified, namely competitive coevolution and cooperative coevolution [13]. In competitive coevolution, an inverse fitness interaction exists between the computing species. A win for one species means a failure for the other. To survive, the losing species adapts to counter the winning species. In cooperative coevolution, the success of one species improves the overall quality of all individuals in all species. Cooperative coevolution is achieved through a positive feedback among the species that take part in the cooperating process.

In standard EAs, evolution is usually viewed as if the population attempts to adapt in a fixed physical environment. In contrast, coevolutionary algorithms realize that in natural evolution the physical environment is influenced by other independently acting biological populations. Evolution is therefore not just local within each population, but also in response to environmental changes as they are caused by other populations. Another difference between standards EAs and CEAs is that EAs define the meaning of optimality through an absolute fitness function. This fitness function then drives the evolutionary process. On the other hand, CEAs do not define optimality using a fitness function but attempt to evolve an optimal species where optimality is defined as defeating opponents.

2.5 Summary

Many optimisation design problems can be formulated as constrained problems which often consist of many mixed equality and inequality constraints. In this chapter, we have presented deterministic and stochastic search methods. First, an overview of gradient-based search strategies is introduced to describe

their verity. Moreover, some approaches for solving mixed discrete continuous optimisation problems have been discussed in more depth. Finally, a comparison between all these methods has been provided.

The rest of the chapter has provided a short summary of evolutionary computation paradigm, with just enough information to support the discussions on algorithms that follow in the later parts of the research. The Evolutionary algorithms have received a lot of attention regarding their potential for solving the numerical constrained optimisation or mixed- variables optimisation problems. Evolutionary computation uses iterative progress, such as growth or development in a population. This population is then selected in a guided random search using parallel processing to achieve the desired end. Such processes are often inspired by biological mechanisms of evolution.

Chapter 3

Implementation and Numerical Experiments

3.1 Introduction

This chapter describes the solution of a collection of test models for continuous and mixed-variables nonlinear programming. It also presents the state-of-the-art solvers that are used for solving constrained optimisation problems. Results are reported for testing a number of existing state-of-the-art solvers for global constrained optimisation and constraint satisfaction on different test problems, collected from the literature. The test problems are available online in AMPL or GAMS and were translated into the input formats of the various solvers. This chapter also shows the implementation of the most powerful deterministic and evolutionary methods for solving constrained optimisation problems. These algorithms have been implemented in MATLAB 7 or C++, and some of the numerical results of the test problems have been compared to show their efficiency and robustness.

3.2 Optimisation problems models format

Since there is no standard format for nonlinear models, a translation server must be used to transform the models from basic format into a number of other formats. The Netlib collection of Linear programming (LP) models has been developed in the industry standard MPS format. The MIPLIB collection developed later has filled a similar need in the field of Mixed Integer Linear Programming (MIP). In the rapidly growing field of Mixed Integer Nonlinear Programming (MINLP) we do not have a similar computerized

and easily available collection of models. The MINLP models are represented in modelling system languages such as GAMS, AMPL, BARON, LINGO, and MINOPT. A GAMS model translation server able to translate the models into the format required by an algorithm. There are some books with test models for MINLP optimisation. Floudas and Leyffer [1, 14] have websites with models used to test their own codes.

The largest MINLP models collection is available through a web site at http://www.gams.com, where all the models are described in GAMS format but there is a translator that can transform a GAMS model into many other formats. The models in the MINLP library vary from small scale literature models to large scale real world models from different application areas.

3.3 The model in AMPL

AMPL is a language for large-scale optimisation and mathematical programming problems in production, distribution, blending, scheduling, and many other applications. The fundamental components for all the models should include: Sets, Parameters, variables, an Objective, and constraints. The AMPL language is intentionally as close to the mathematical form as it can get while still being easy to type on an ordinary keyboard and processed by a program. There are AMPL constructions for each of the basic components and ways to write arithmetic expressions, sum over sets, and so on.

3.3.1 COP Solvers

With the recent progress made in global optimisation, the importance of modelling systems has taken on a more significant role. In practice, most global solvers require more than black-box function evaluations. These solvers need structural information of algebraic expressions to build convex relaxations. These solvers can be described as follows:

- **SNOPT** Large scale SQP based NLP solver from Stanford University
- **PATHNLP** Large scale NLP solver for convex problems from the University of Wisconsin at Madison

- **MINOS** NLP solver from Stanford University
- **CONOPT** Large scale NLP solver from ARKI Consulting and Development
- **KNITRO** Large scale NLP solver from Ziena Optimisation, Inc.

Solvers Descriptions

SNOPT

SNOPT is a new large scale SQP for solving optimisation problems involving many variables and constraints. It minimises a linear or nonlinear function subject to bounds on the variables and sparse linear or nonlinear constraints. It is suitable for large-scale linear and quadratic programming and for linearly constrained optimisation, as well as for general nonlinear programs. SNOPT is most efficient if only some of the variables enter nonlinearly, or if the number of active constraints (including simple bounds) is nearly as large as the number of variables. SNOPT requires relatively few evaluations of the problem functions.

PATHNLP

The PATHNLP solver is suitable for NLP programs. PATHNLP solves an NLP by internally constructing the Karush-Kuhn-Tucker (KKT) system of first-order optimality conditions associated with the NLP and solving this system using the PATH solver for complementarity problems. The solution to the original NLP is extracted from the KKT solution and returned to GAMS. All of this takes place automatically - no special syntax or user reformulation is required.

MINOS

GAMS/MINOS is the oldest NLP solver available with GAMS and it is still the NLP solver that is used the most. MINOS has been developed at the Systems Optimisation Laboratory at Stanford University, and development is continuing today. Linearly constrained models are solved with a very efficient and reliable reduced gradient technique that utilizes the sparsity of the model. Models with nonlinear constraints are solved with a method that iteratively solves subproblems with linearized constraints and an

augmented Lagrangian objective function. This iterative scheme implies that only the final optimal solution is feasible for nonlinear models, in contrast to the feasible path method used by the other large scale NLP solver, CONOPT.

CONOPT

GAMS/CONOPT is an alternative solver to MINOS and other non linear problem (NLP) solvers available for use with GAMS. The availability of multiple nonlinear solvers in the GAMS system should be seen as an attempt to increase the overall usefulness of nonlinear modelling with GAMS. CONOPT, developed by ARKI Consulting and Development in Denmark, is a multi-method solver. CONOPT and the other GAMS NLP solvers often complement each other. If one solver fails, one of the others will often be able to solve the model. If all solvers fail it is a good indication that the model is very difficult or very poorly scaled, and manual intervention from an experienced modeller is necessary.

KNITRO

KNITRO is a software package for finding local solutions of continuous, smooth nonlinear optimisation problems, with or without constraints. Even though KNITRO has been designed for solving large-scale general nonlinear problems, it is efficient for solving all of the classes of smooth optimisation problems. KNITRO implements both state-of-the-art interior-point and active-set methods for solving nonlinear optimisation problems.

3.3.2 MINLP Solvers

There are several MINLP solvers available. The solvers differ in the methods they use, in whether they find globally optimal solution with proven optimality, in the size of models they can handle, and in the format of models they accept. The MINLP solvers can be classified as follows:

- **Alpha ECP** : Extended Cutting Plane Algorithm from T.Westerlund, Abo Akademi University, Finland.
- **BARON** : Branch-and-Reduce algorithm from N. Sahinidis, University of Illinois Urbana-Champaign.
- **GAMS/DICOPT** : Outer-Approximation algorithm from I.E. Grossmann, Carnegie Mellon University.
- **LOGMIP** : LogMIP (acronym of Logical Mixed Integer Programming) is a solver for generalized disjunctive programs (GDP).
- **MINLP** : Branch-and-Bound algorithm from R. Fletcher and S. Leyffer, The University of Dundee.
- **SBB** : Branch-and-Bound algorithm from ARKI Consulting and Development.
- **Visual Xpress** : MIP solver, free for models with at most 100 rows and 200 variables, and tables and variables having at most two dimensions.
- **LINDO,LINGO** : for linear and nonlinear mixed integer programs (small-scale versions available for free).
- **Setconst** : NLP solver with set constrained variables in Matlab , Mixed Integer Nonlinear Programming Solver.(seems to assume that for fixed discrete parameters, the problem is convex).
- **MINOPT** : A Modelling Language and Algorithmic Framework for Linear, Mixed-Integer, Nonlinear, Dynamic, and Mixed-Integer Nonlinear Optimisation.

Open Source MINLP solvers

- **BNB20**: solves mixed integer nonlinear optimisation problems based on branch and bound algorithm.
- **BONMIN (**Basic Open-source Nonlinear Mixed Integer programming): an experimental open source C++ code for solving general MINLP problems. It is distributed on COIN-OR(www.coin-or.org) under the Common Public License.

Solvers Descriptions

AlphaECP:

AlphaECP is a general purpose MINLP solver. The ECP (extended cutting plane) method is based on cutting plane techniques and the solution of a sequence of mixed integer linear programming problems only. Mixed integer problems, with a pseudo-convex objective function subject to pseudo convex inequality constraints, can be solved to global optimality with the method.

BARON:

BARON is a computational system for solving non convex optimisation problems to global optimality. Purely continuous, purely integer, and mixed-integer nonlinear problems can be solved with the software. The Branch And Reduce Optimisation Navigator derives its name from combining interval analysis and duality in its reduce arsenal with enhanced branch and bound concepts as it winds its way through the hills and valleys of complex optimisation problems in search of global solutions.

DICOPT:

DICOPT (DIscrete and Continuous Optimiser) is an extension of the outer-approximation algorithm with equality relaxation strategies. DICOPT solves a series of NLP and MIP sub-problems using any solver supported by GAMS. Although, the algorithm has provisions to handle non-convexities, it does not always find the global solution.

LOGMIP:

LogMIP has two main components:

- A language compiler for the definition of disjunctions.
- Disjunctive program solvers.

Those components are linked to GAMS (a computer system for the specification and solution of mathematical programs). Both parts are supersets of GAMS language and solvers, respectively. LogMIP is not independent of GAMS, it uses the declarations and definitions made into GAMS language format for the specifications and solution of a disjunctive problem.

MINLP:

MINLP implements a branch-and-bound algorithm searching a tree whose nodes correspond to continuous non linearly constrained optimisation problems. The continuous problems are solved using filterSQP, a Sequential Quadratic Programming solver which is suitable for solving large nonlinearly constrained problems.

SBB:

SBB is a GAMS solver for Mixed Integer Nonlinear Programming models. It is based on a combination of the standard Branch and Bound method known from Mixed Integer Linear Programming and some of the standard NLP solvers already supported by GAMS. During the solution process SBB solves a number relaxed MINLP models with tighter and tighter bounds on some of the integer variables. The solutions to these submodels are assumed to provide valid bounds on the objective function. SBB will find the global optimum if the underlying RMINLP model is convex. If the submodels are not convex then some submodels may be solved to a local optimum that is not global, and they may terminate in a locally infeasible point even if feasible solutions exist. If a submodel cannot be solved with the default NLP solver then SBB has the ability to try to solve it with a sequence of other solvers.

3.4 COP Numerical Experiments

This section describes solving a nonlinear constrained optimisation problem taken from [15]. First, the SQP method has been used for solving each problem. The problems can be described as follows:

$$\begin{aligned} \min_{x} \quad & f(x) = (x_1 - 10)^3 + (x_2 - 20)^3, \\ \text{s.t.} \quad & g_1(x) = (x_1 - 5)^2 + (x_2 - 5)^2 + 100 \le 0, \\ & g_2(x) = (x_1 - 5)^2 + (x_2 - 5)^2 - 82.81 \le 0, \\ & U_B = (100, 100), \\ & L_B = (13, 0). \end{aligned} \tag{3.1}$$

Iter	F-count	f(x)	Max constraint	Line search step length	Directional derivative	First-order optimality
0	3	-7973				
1	6	-7174.14	1.624	1	637	1.58e+003
2	9	-6966.43	0.03487	1	1.12e+003	42.2
3	12	-6961.82	1.756e-005	1	1.1e+003	0.0265
4	15	-6961.81	5.414e-012	1	1.1e+003	7.08e-005

Table 3.1 Experiment results.

Global minimum: $\vec{x}^* = (14.095, 0.84296)$, $f(\vec{x}^*) = -6961.81388$

The numerical results for solving this problem using the Augmented Lagrangian multipliers method are shown in Table 3.2

Optimal objective $f(\vec{x}^*)$	−6961.81388
Variables solution $\vec{x}^*$	[14.095, 0.84296]
Lagrangian multipliers $\vec{\lambda}^*$	[0 0 0 0 0 0]
Penalty parameters $\vec{r}^*$	[587.4151 717.9518 0.0575 0.1918 0.0007 0.0007]

Table 3.2 Optimal design of G6 function

We shall next, solve the same problem by using Genetic Algorithms, The obtained results are shown as follows:

Generation	f-count	Best f(x)	constraint	Stall Generations
1	1084	-6961.81	8.114e-009	0
2	2124	-6961.95	0.0007163	0
3	3176	-6961.81	9.78e-007	0
4	4232	-6961.81	1.947e-012	0

Table 3.3 Optimisation results.

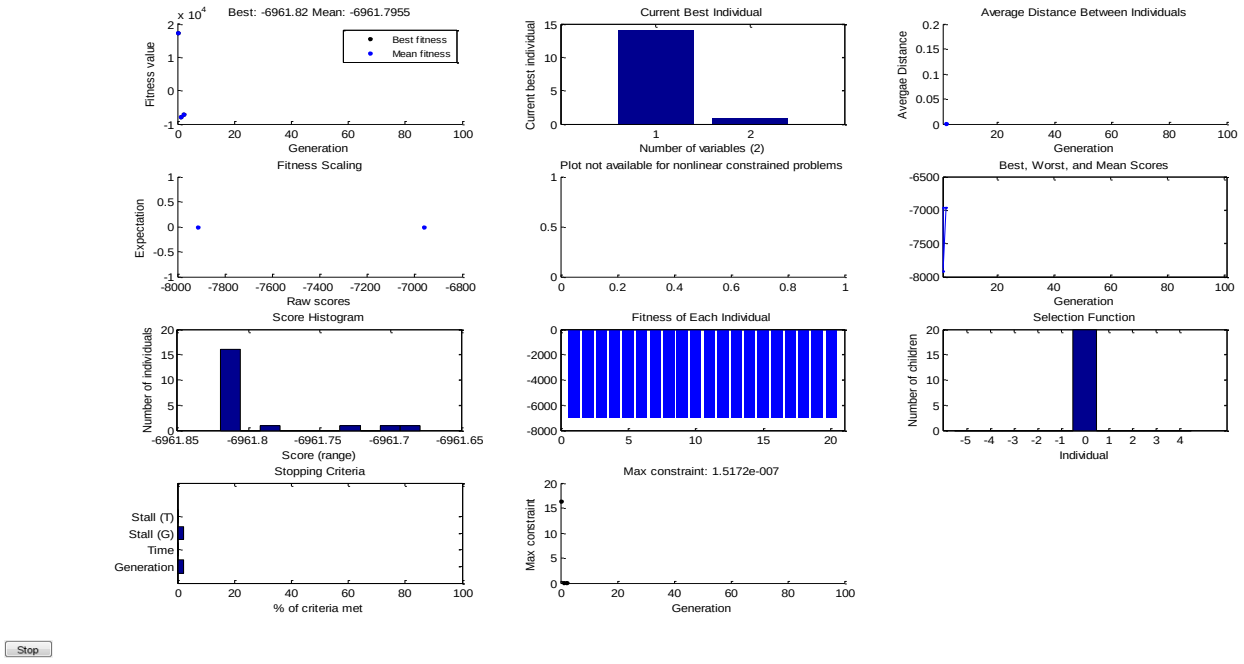


Fig 3.1 Convergence plots for constrained optimisation problem.

The numerical results using pattern search method can be seen as follows:

Iter	f-count	*f(x)*	constraint	MeshSize	Method
1	28	-6962	2.74e-007	0.001	Increase penalty
2	55	-6962	2.74e-007	9.333e-007	Update multipliers

Table 3.4 Iterations history.

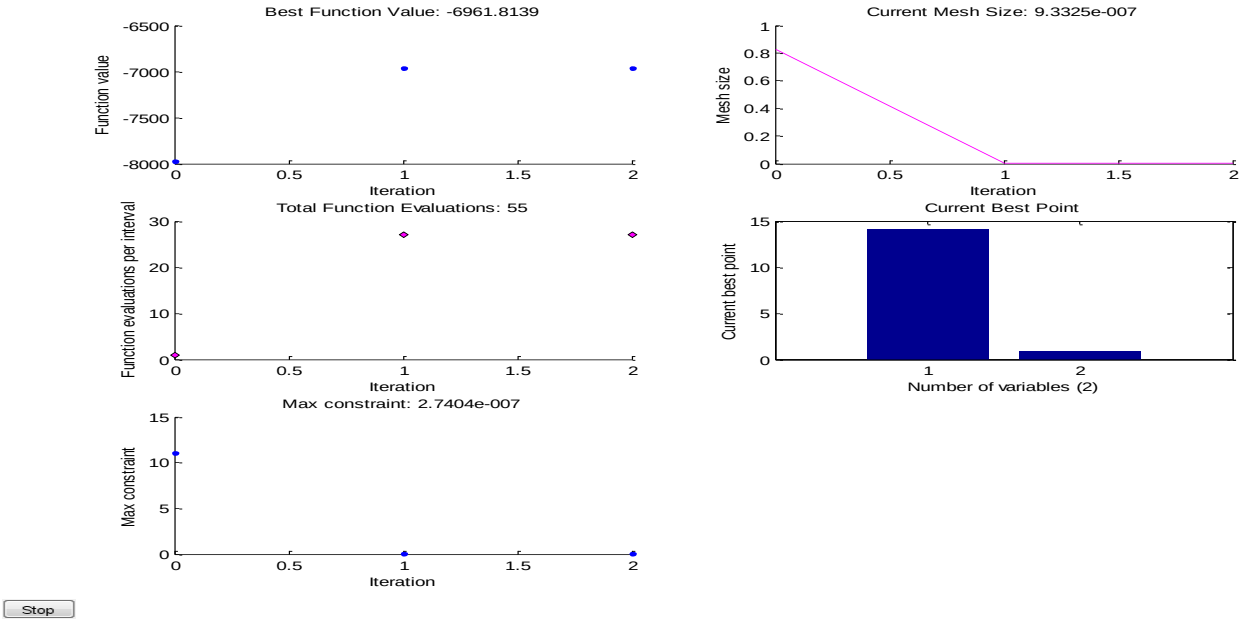


Fig 3.2 Convergence Characteristics plots.

3.5 MINLP Numerical Experiments

This section describes solving a collection of test models for mixed integer nonlinear programming. The SQP-BB algorithm has been implemented in MATLAB 7 using SQP as NLP solver. This algorithm is described in more details in [14]. The implemented algorithm is shown in Fig.3.3.

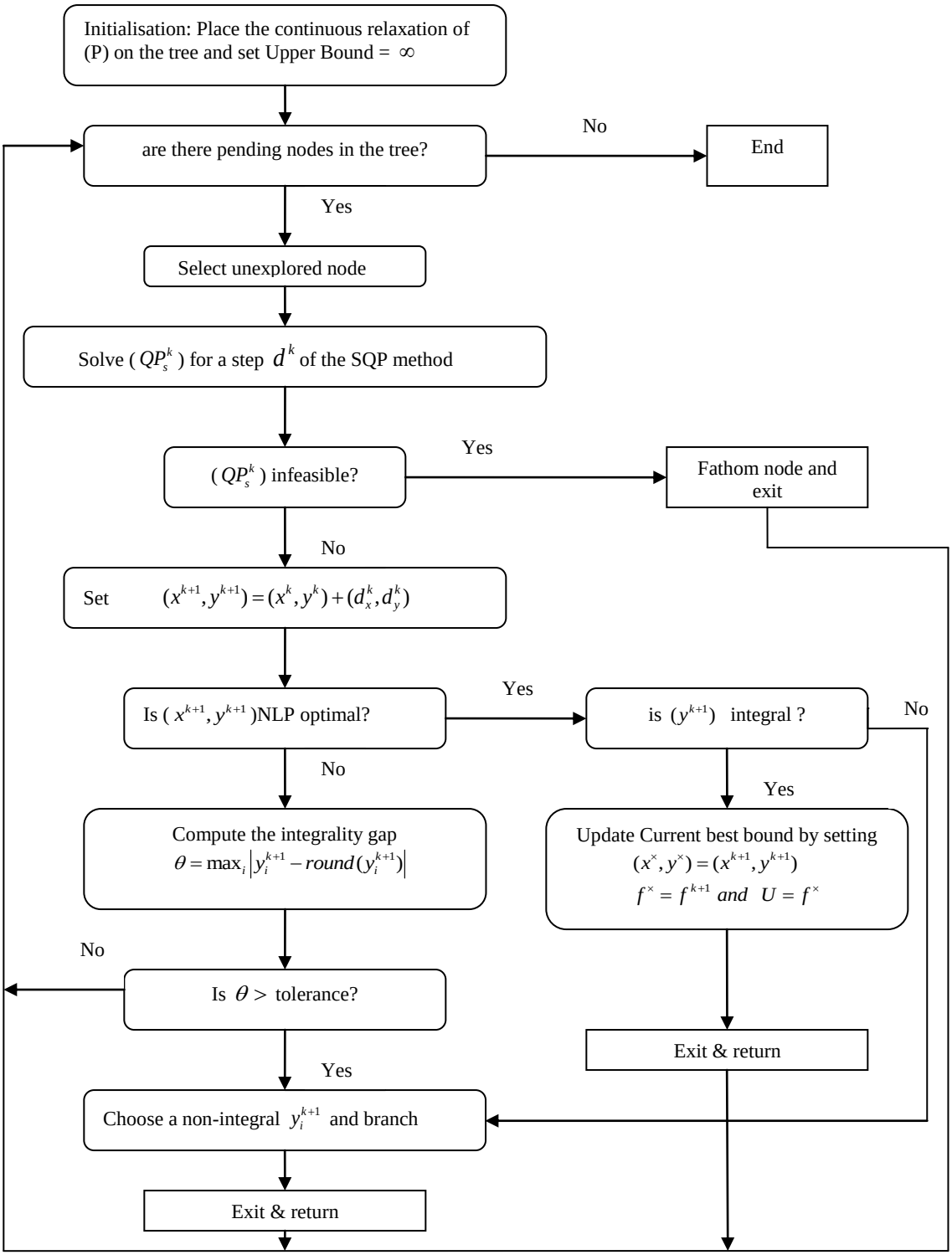


Fig. 3.3 Integrating SQP and BB

To illustrate the applicability and the efficiency of the implemented algorithm five test problems on process synthesis and design proposed by different authors have been chosen. These problems arise from

the area of Chemical Engineering, and represent difficult nonconvex optimisation problems with continuous and discrete variables. First the Process synthesis problem has a nonlinear constraint and has been proposed by Kocis and Grossman [16]. Process flow sheeting problem was first studied by Floudas [1] and is nonconvex because of the first constraint. Problems SYNTHES1, SYNTHES2, SYNTHES3 are process synthesis problems taken from [14].

Header	**Description**
nv	Total number of variables of the problem
nr	Total number of real variables of the problem
niv	Number of integer variables of the problem
nc	Total number of the constraints of the problem
nec	Number of linear equality constraints of the problem
nic	Number of linear inequality constraints of the problem
nlc	Total number of non-linear constraints

Table 3.5 Description of the headers used in table 3.6

Problem	**nv**	**nr**	**niv**	**nc**	**nec**	**nic**	**nlc**
Process Synthesis	2	1	1	2	0	1	1
flow sheeting	3	2	1	3	0	2	1
Synthes1	6	3	3	6	0	4	2
Synthes2	11	6	5	14	1	10	3
Synthes3	17	9	8	23	2	17	4

Table 3.6 Test problem Characteristics

Example: Process synthesis problem

This is a small problem with only one continuous and one discrete variable. It has linear and non-linear inequality constraints. This problem has also been solved by other authors [8]. The master problem formulation is given below:

$$
\begin{aligned}
&\text{minimize} && f(x,y) = 2x + y \\
&\text{subject to} && 1.25 - x^2 - y \le 0 \\
& && x + y \le 1.6 \\
& && 0 \le x \le 1.6 \\
& && y \in \{0,1\}
\end{aligned}
\tag{3.2}
$$

Since this problem has only one continuous and one binary variable, details can be illustrated graphically in order to provide a geometrical interpretation of the master problem.

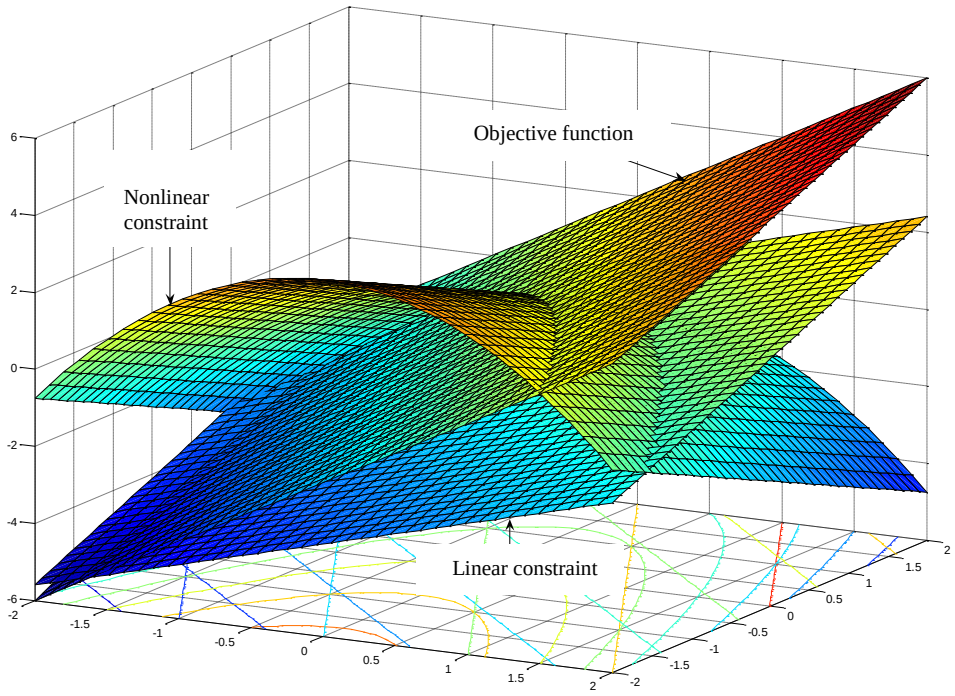


Fig. 3.4 Feasible region and objective function in process synthesis problem.

The results for this problem are:

- NLP relaxation value = 2.000
- The optimal solution y = 1.0000 , x = 0.5000
- Optimal objective function value = 2.000
- Number of nodes visited by algorithm = 1
- Number of QP problem solved = 6
- Seconds of the CPU time needed for the solve = 0.0160

Iter	F-count	f(x)	max constraint	Step-size	Directional derivative	First-order optimality	Procedure
0	3	3	0.4				Infeasible start point
1	7	2.25	-0.05	1	-0.75	0.81	
2	11	2.05714	0	1	-0.193	0.565	Hessian modified twice
3	15	2.00154	0	1	-0.0556	0.107	Hessian modified twice
4	19	2	0	1	-0.00154	0.00311	Hessian modified twice
5	23	2	0	1	-1.19e-006	3.03e-006	Hessian modified twice

Table 3.7 numerical results for process synthesis problem.

Fig. 3.4 shows the graphical representation of the constraints involved in the example. The graphical solution which can be read from the figure is y = 1, x= 0.5. It is intersection of the active linear and nonlinear constraints. The red area is the feasible region that is the design space in which all the constraints are satisfied, while the blue area shows the infeasible search design which some of the constraints are violated. The geometry evident in the graphical solution of the example is used to show some concepts associated with constrained optimisation problems and their characteristics.

It can be seen that after solving the master problem all integer variables take an integer value then this solution also solves the MINLP. So, we do not have to proceed with the searching tree because the integer solution has been found. The algorithm terminated with the global minimum.

Example: Process flow sheeting problem

This problem was fist studied by Floudas [1] and is nonconvex because of the first constraint. It has also been solved by Costa and Oliviera [17].The problem is given by

$$\begin{aligned} \text{minimize} \quad & f(x_1, x_2, y) = -0.7y + 5(x_1 - 0.5)^2 + 0.8 \\ \text{subject to} \quad & -\exp(x_1 - 0.2) - x_2 \le 0 \\ & x_2 + 1.1y \le -1.0 \\ & x_1 - y \le 0.2 \\ & 0.2 \le x_1 \le 1 \\ & -2.22554 \le x_2 \le -1 \\ & y \in \{0,1\} \end{aligned} \tag{3.3}$$

This problem has two continuous variables and one binary variable. It also has two linear inequality constraints. Since it is a pure 0-1 problem, the NLP sub problems for fixed Y^k require only an objective function evaluation rather than an optimisation. There are only two possible combinations of the binary variables.

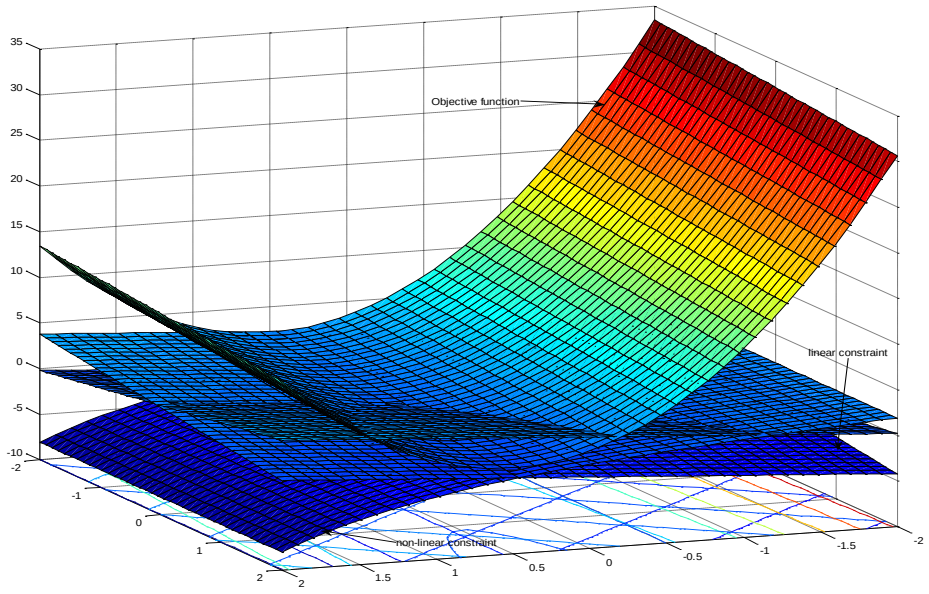


Fig. 3.5 Objective function contours and nonlinear feasible region

The results for this problem are:

- NLP relaxation value = 0.5369
- The optimal solution y = 1.0000 , x_1 = 0.9419 , x_2 = -2.1000
- Optimal objective function value = 1.0765
- Number of nodes visited by algorithm = 3
- Number of QP problem solved = 16
- Seconds of the CPU time needed for the solve = 0.0780

The feasible region and the objective function contours are show in Figure 3.5. The global optimum of the problem is located where the objective value = 1.0762. The figure also includes the solution to this example. The red area shows the feasible region that is the design space in which all the constraints are satisfied. In this example, the lower bound lies below the upper bound, so a new NLP sub problem is solved, but we have found an integer solution so the best current upper bound has been updated to provide a solution for the MINLP master problem.

Example: Synthes1

This problem will illustrate the use of nonlinear 0-1 variables; there are 8 possible combinations of the 3 binary variables, of which 4 are feasible as determined by the linear inequality constraint. The problem is formulated in AMPL and given by

- Source: Test problem 1 (Synthesis of processing system) in
M. Duran & I.E. Grossmann, "An outer approximation algorithm for a class of mixed integer nonlinear programs", Mathematical Programming 36, pp. 307-339, 1986.

```
Number of variables:   6 (3 binary variables)
Number of constraints: 6
Objective nonlinear: 1
Nonlinear constraints: 2

set I := 1..3;

param u {I} default 2;

var x {i in I} >= 0, <= u[i];
var y {I}      binary;

minimise Obj:
     5*y[1] + 6*y[2] + 8*y[3] + 10*x[1] - 7*x[3] - 18*log(x[2] + 1)
     - 19.2*log(x[1] - x[2] + 1) + 10;

s.t.c1: 0.8*log(x[2] + 1) + 0.96*log(x[1]-x[2]+1) - 0.8*x[3] >= 0;
    c2: log(x[2] + 1) + 1.2*log(x[1]-x[2]+1) - x[3] - 2*y[3] >= -2;
    c3: x[2] - x[1] <= 0;
    c4: x[2] - 2*y[1] <= 0;
    c5: x[1] - x[2] - 2*y[2] <= 0;
    c6: y[1] + y[2] <= 1;
```

Results found by the solver are:

- NLP relaxation value = 0.7593
- The optimal solution y1= 0, y2=1, y3=1 , x1= 1.3, x2 = 0, x3 =1
- Optimal objective function value = 6.9998
- Number of nodes visited by algorithm = 5
- Number of QP problem solved = 34
- Seconds of the CPU time needed for the solve = 0.2190

The example shows that global solution can be obtain by the algorithm. The current upper bound has been updated, and it used to derive linearization for the nonlinear functions that are to be included in the MINLP master problem.

Example: Synthes2

This problem has more continuous and integer variables; there are 32 possible combinations of the 5 binary variables, of which 11 are feasible as determined by the linear inequality constraint. there are 3 nonlinear inequality constraints and one linear equality constraint. The problem is formulated in AMPL and given by

- Source: Test problem 2(Synthesis of processing system) in
M. Duran & I.E. Grossmann, "An outer approximation algorithm for a class of mixed integer nonlinear programs", Mathematical Programming 36, pp. 307-339, 1986.

```
Number of variables:   11 (5 binary variables)
Number of constraints: 14

set I := 1..6;
set J := 1..5;
param u {I} default Infinity;
var x {i in I} >= 0, <= u[i];
var y {J}      binary;

minimise Obj:
     5*y[1] + 8*y[2] + 6*y[3] + 10*y[4] + 6*y[5]
   - 10*x[1] - 15*x[2] - 15*x[3] + 15*x[4] + 5*x[5] - 20*x[6]
   + exp(x[1]) + exp(0.833333*x[2]) - 60*log(x[4]+x[5]+1) + 140;

s.t. c1: - log(x[4]+x[5]+1) <= 0;
     c2: exp(x[1]) - 10*y[1] <= 1;
     c3: exp(0.833333*x[2]) - 10*y[2] <= 1;
     c4: 1.25*x[3] - 10*y[3] <= 0;
```

```
c5: x[4] + x[5] - 10*y[4] <= 0;
c6: -2*x[3] + 2*x[6] - 10*y[5] <= 0;
c7: -x[1] - x[2] - 2*x[3] + x[4] + 2*x[6] <= 0;
c8: -x[1] - x[2] - 0.75*x[3] + x[4] + 2*x[6] <= 0;
c9: x[3] - x[6] <= 0;
c10: 2*x[3] - x[4] - 2*x[6] <= 0;
c11: -0.5*x[4] + x[5] <= 0;
c12: -0.2*x[4] - x[5] <= 0;
c13: y[1] + y[2] = 1;
c14: y[4] + y[5] <= 1;
```

Results found by the solver are:

- NLP relaxation value = -8.3652
- The optimal solution

x1	x2	x3	x4	x5	x6	y1	y2	y3	y4	y5
0	2	1.07	0.65	0.32	1.07	0	1	1	1	0

- Optimal objective function value = 73.0353
- Number of nodes visited by algorithm = 13
- Number of QP problem solved = 80
- Seconds of the CPU time needed for the solve = 0.5290

Example: Synthes3

This problem has 9 continuous variables and; there are 32 possible combinations of the 5 binary variables, of which 11 are feasible as determined by the linear inequality constraint. There are 3 nonlinear inequality constraints and one linear equality constraint. The problem is formulated in AMPL and given by

➢ Source: Test problem 3 (Synthesis of processing system) in
M. Duran & I.E. Grossmann, "An outer approximation algorithm for a class of mixed integer nonlinear programs", Mathematical Programming 36, pp. 307-339, 1986.

```
Number of variables:   17 (8 binary variables)
Number of constraints: 23
Objective nonlinear: 1
Nonlinear constraints: 4

set I := 1..9;
set J := 1..8;
param u {I} default 2;
var x {i in I} >= 0, <= u[i];
var y {J}
```

```
minimise Obj:
     5*y[1] + 8*y[2] + 6*y[3] + 10*y[4]
   + 6*y[5] + 7*y[6] + 4*y[7] + 5*y[8]
   - 10*x[1] - 15*x[2] + 15*x[3] + 80*x[4] + 25*x[5]
   + 35*x[6] - 40*x[7] + 15*x[8] - 35*x[9]
   + exp(x[1]) + exp(0.833333*x[2]) - 65*log(x[3]+x[4]+1)
   - 90*log(x[5]+1) - 80*log(x[6]+1) + 120;

s.t. c1:  - 1.5*log(x[5]+1) - log(x[6]+1) - x[8] <= 0;
     c2:  - log(x[3]+x[4]+1) <= 0;
     c3:  - x[1] - x[2] + x[3] + 2*x[4] + 0.8*x[5]
          + 0.8*x[6] - 0.5*x[7] - x[8] - 2*x[9] <= 0;
     c4:  - x[1] - x[2] + 2*x[4] + 0.8*x[5] + 0.8*x[6]
          - 2*x[7] - x[8] - 2*x[9] <= 0;
     c5:  - 2*x[4] - 0.8*x[5] - 0.8*x[6] + 2*x[7]
          + x[8] + 2*x[9] <= 0;
     c6:  - 0.8*x[5] - 0.8*x[6] + x[8] <= 0;
     c7:  - x[4] + x[7] + x[9] <= 0;
     c8:  - 0.4*x[5] - 0.4*x[6] + 1.5*x[8] <= 0;
     c9:  0.16*x[5] + 0.16*x[6] - 1.2*x[8] <= 0;
     c10: x[3] - 0.8*x[4] <= 0;
     c11: - x[3] + 0.4*x[4] <= 0;
     c12: exp(x[1]) - 10*y[1] <= 1;
     c13: exp(0.833333*x[2]) - 10*y[2] <= 1;
     c14: x[7] - 10*y[3] <= 0;
     c15: 0.8*x[5] + 0.8*x[6] - 10*y[4] <= 0;
     c16: 2*x[4] - 2*x[7] - 2*x[9] - 10*y[5] <= 0;
     c17: x[5] - 10*y[6] <= 0;
     c18: x[6] - 10*y[7] <= 0;
     c19: x[3] + x[4] - 10*y[8] <= 0;
     c20: y[1] + y[2] = 1;
     c21: y[4] + y[5] <= 1;
     c22: - y[4] + y[6] + y[7] = 0;
     c23: y[3] - y[8] <= 0;
```

Results found by the solver are:

- NLP relaxation value = 15.0822
- The optimal solution

x1	x2	x3	x4	x5	X6	x7	x8	x9	y1	y2	y3	y4	y5	y6	y7	y8
0	2.0	.46	.58	2.0	0	0	.26	.58	0	1	0	1	0	1	0	1

- Optimal objective function value = 68.0097
- Number of nodes visited by algorithm = 29
- Number of QP problem solved = 298
- Seconds of the CPU time needed for the solve = 1.4060

This example shows that the algorithm requires a large number of QP solved per node in order to solve the MINLP problem. Eight nodes have been fathomed due to infeasibility or the current value of the objective function was higher than the current bound. If one of the fathoming rules is satisfied, then no branching is required. The algorithm uses backtracking search strategy to select which node to solve next. An optimal solution has been found after 29 NLP problems solved.

Results and discussion

The performance of the implemented algorithm was compared to the nonlinear branch and bound solver MINLP-BB [14], available at NEOS server and the Sequential Cutting Plane algorithm which uses CPLEX library to solve LP problems [18]. A numerical comparison with existing solvers shows that the performance of the algorithm is competitive with the existing algorithms. However, the overall number of QP problems that are being solved can be reduced using different NLP solvers.

Table 3.9 summarise the results obtained when the integration of SQP and BB algorithm were used. The number of nodes generated in the branch and bound tree, as well the number of mixed integer nonlinear programming problems solved gives an indication of the relative performance of the implemented algorithm.

Header	**Description**
Problem	Name of the optimisation problem
NLPs	Number of NLP problems solved in order to solve the problem
NLP-R	Non-linear Programming relaxation (initial solution)
Obj_f	The value of the function at the optimum point
Nodes	Number of nodes visited by the algorithm
QPs	Number of QP problem solved
CPU	Seconds of the CPU time needed for the solve

Table 3.8 Description of the headers used in table 3.9

Problem	NLP-R	Obj_f	Nodes	QPs	CPU
Process Synthesis	2.000	2.000	1	6	0.0160
Flow Sheeting	0.5369	1.0765	3	16	0.0780
Synthes1	0.7593	6.0098	5	34	0.2190
Synthes2	-8.3652	73.0353	13	80	0.5290
Synthes3	15.0822	68.0097	29	298	1.4060

Table 3.9 Test results for the test problems

The algorithm solves MINLP problems by solving a sequence of non-linear programming problems, in contrast to other solvers that typically solve a sequence of LP, QP, NLP, or MILP problems. It performs as well on problems with linear or non-linear constraints. The implemented algorithm uses slightly more CPU time than the MINLP-BB solver.

3.6 Penalty function approach for the discrete nonlinear problems

This approach treats the requirement of discreteness in the MDNLP problems by defining additional constraints and constructs a penalty function for them [4]. The added term only penalizes non-discrete design variables and it imposes penalty for deviations from the discrete values. The augmented function can be defined as follows:

$$F(x) = f(x) + s\phi(x) + r\sum_{k=1}^{ncon} \max[0, g_k(x)] \qquad \text{Augmented function} \qquad (3.5)$$

$$\phi(x) = \sum_{i=1}^{n} \frac{1}{2}\left[\sin \frac{2\pi\{x_{m+i}^{c} - 0.25(d_{i,j+1} + 3d_{i,j})\}}{d_{i,j+1} - d_{i,j}} + 1\right] \qquad \text{Penalty function} \qquad (3.6)$$

d_{ij} and d_{ij+1} are the two neighbouring discrete values for *x*.

s^k denote the penalty parameters of the penalty term for the discrete design variables.

- A convergence criterion guarantees that the optimisation process is ended if the design variables are sufficiently close to the prescribed discrete variables and can be expressed as follows:

$$q_i(1-q_i) \le \frac{\varepsilon}{d_{ij+1}-d_{ij}}\left(1-\frac{\varepsilon}{d_{ij+1}-d_{ij}}\right) \qquad \text{where,} \quad q_i = \frac{y_i - d_{ij}}{d_{ij+1}-d_{ij}} \tag{3.7}$$

Example:

Consider the following optimisation function.

$$f(x) = x^4 - \frac{8}{3}x^3 - 2x^2 + 8x$$

$$x \in \{-1, 0, 1, 2\}$$

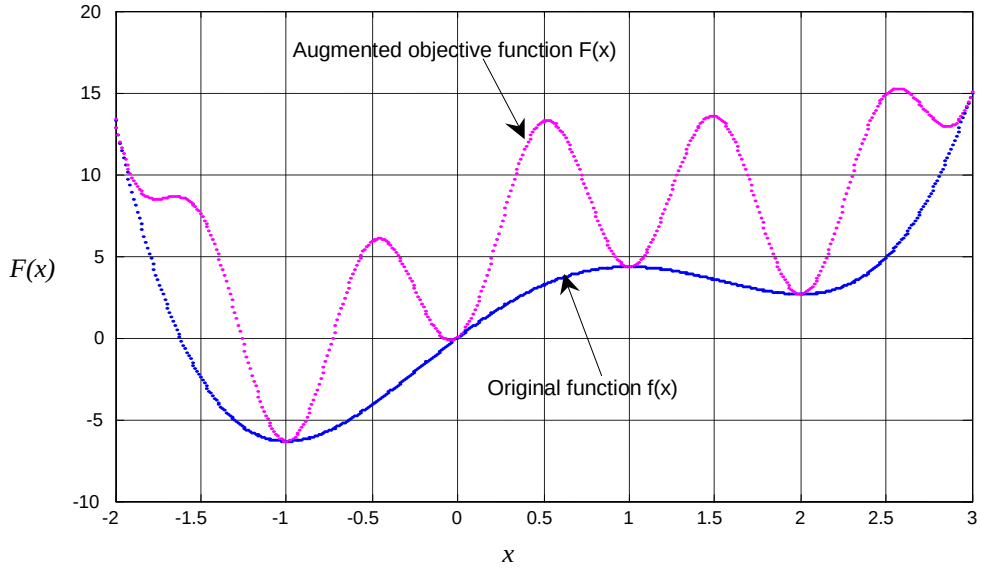


Fig 3.6 Objective function and augmented objective function.

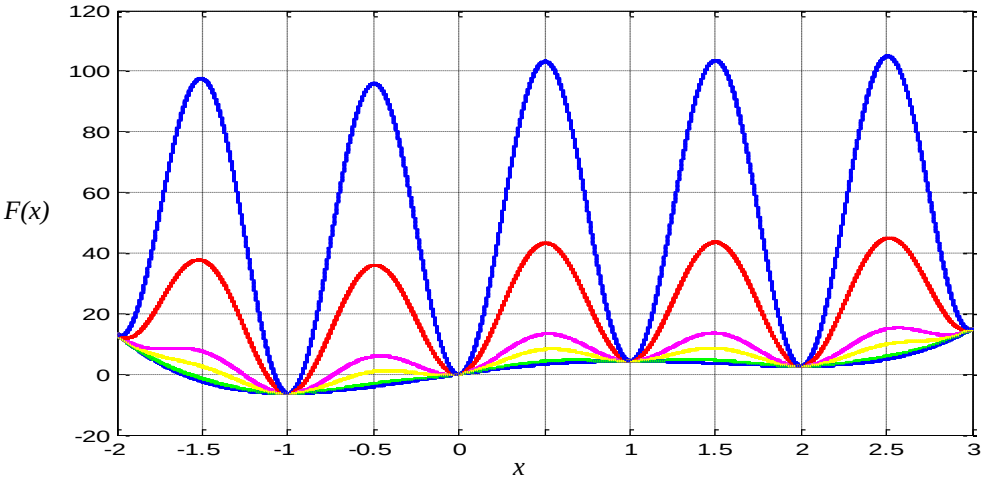
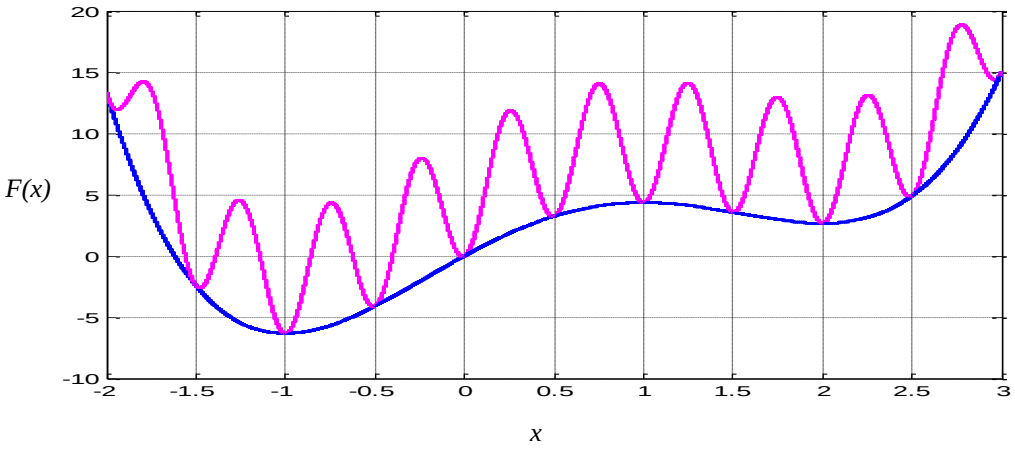


Fig 3.7 Behaviours of F(x) for the various penalty parameters S.

Case 2:

$x \in \{-2, -1.5, -1, -0.5, 0, 0.5, 1, 1.5, 2, 2.5, 3\}$



Case 3:

$x \in \{-1.7, -1.5, -1.4, -1.2, -1, -0.3, 0.1, 0.3, 0.5\}$

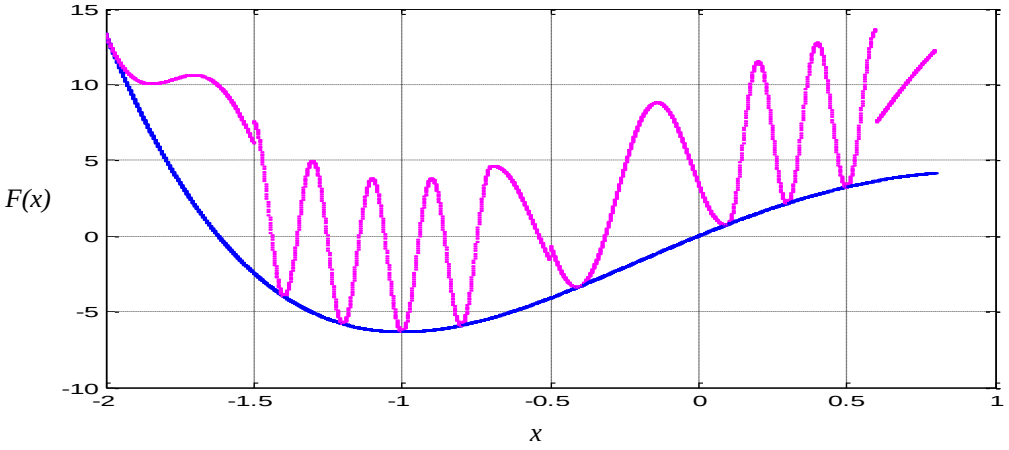


Fig 3.8 Behaviours of F(x) for different discrete requirements.

In this research, the original method has been developed to solve MINLP problems. The augmented Lagrangian function can be extended to solve MINLP problems. The augmented objective function can be formulated by combining the penalty function and the Lagrangian multiplier methods as follows:

$$L(x, y, \lambda, r^k, s^k) = f(x, y) + \sum_{i=1}^{m} \lambda_i \alpha_i + \sum_{i=1}^{n} \lambda_{m+i} h_i(x, y) + \sum_{i=1}^{m} r_i^k \alpha_i^2 + \sum_{i=1}^{n} r_i^k h_i^2(x, y) + s^k \phi^k(y) \quad (3.8)$$

where,

λ_i donate the values of Lagrangian multipliers for the inequality constraints

r_i^k are the penalty parameters for the ith constraints

α_i is necessary to convert the inequality constraints to equality constraints

$$\alpha_i = \max\left\{ g_i(x,y), -\frac{\lambda_i}{2r^k} \right\}$$

λ_{m+i} are the values of Lagrangian multipliers for the equality constraints

$h_i(x,y)$ are the equality constraints

s^k are the penalty parameters for the discrete penalty function $\phi(y)$

$$\phi^k(y) = \frac{1}{2}\left[\sin\left(\frac{2\pi\left(y^k - \frac{1}{4}(d_{ij+1} + 3d_{ij}) \right)}{d_{ij+1} - d_{ij}} \right) + 1 \right] \tag{3.9}$$

Where, d is a set of discrete values,

d_{ij} and d_{ij+1} are the two neighbouring discrete values for *y*.

The minimisation can be started by setting the values of $\lambda_i, \lambda_{m+i}, r^k$ to 0, 0, 1 respectively, and then these values have to be updated in each iteration as follows:

$$\begin{aligned}
&\lambda_i^{k+1} = \lambda_i^k + 2r^k \max\left\{ g_i(x,y), -\frac{\lambda_i^{(k)}}{2r^k} \right\} \\
&\lambda_{m+i}^{k+1} = \lambda_{m+i}^k + 2r^k h_i(x,y) \\
&r^{k+1} = cr^k \qquad c > 1 \\
&s^{k+1} = as^k \qquad a \geq 1
\end{aligned} \tag{3.10}$$

MDNLP example:

Consider the following optimisation problem:

$$\begin{aligned} &\text{minimise} && f(x,y) = 3x^2 + 8y \\ &\text{subject to} && g(x,y) = -x^2 - 1 \le 12.53 \\ & && h(x,y) = 1 + x + 2y^2 = 9.4 \\ & && 1 \le x \le 8 \\ & && 0 \le y \le 3 \end{aligned}$$

The solution to this problem using branch and bound method is $(x^*, y^*) = (6.4, 1)$ with $f = 130.88$, while the continuous solution using SQP is $(x^*, y^*) = (1, 1.923)$ with f = 18.388. The augmented Lagrangian multipliers function is as follows:

$$\begin{aligned} L(x,y,\lambda,r_k) &= f(x,y) + \sum_{i=1}^{m} \lambda_i \alpha_i + \sum_{i=1}^{n} \lambda_{m+i} h_i(x,y) + r_k \sum_{i=1}^{m} \alpha_i^2 + r_k \sum_{i=1}^{n} h_i^2(x,y) \\ &= f(x,y) + \lambda_1 \alpha_1 + \lambda_2 h_1(x,y) + r_k \alpha_1^2 + r_k h_1^2(x,y) \\ &= 3x^2 + 8y + \lambda_1 \times \max((-x^2 - 13.53), -\frac{\lambda_1}{2r_k}) + \lambda_2 \times \max((-x+1), -\frac{\lambda_2}{2r_k}) + \lambda_3 \times \max((x-8), -\frac{\lambda_3}{2r_k}) + \\ &\quad \lambda_4 \times (1 + x + 2y^2 - 9.4) + r_k \times \left[\max((-x^2 - 13.53), -\frac{\lambda_1}{2r_k}) \right]^2 + r_k \times \left[\max((-x+1), -\frac{\lambda_2}{2r_k}) \right]^2 \\ &\quad + r_k \times \left[\max((x-8), -\frac{\lambda_3}{2r_k}) \right]^2 + r_k \times \left(1 + x + 2y^2 - 9.4\right)^2 \end{aligned}$$

For the stationary point of A, the necessary conditions, $\frac{\partial A}{\partial x} = 0, \frac{\partial A}{\partial y} = 0$

$$\frac{\partial A}{\partial x} = (6x - 2x\lambda_1 - \lambda_2 + \lambda_3) - 4xr_k[\max((-x^2 - 13.53), -\frac{\lambda_1}{2r_k}) - 2r_k[\max((-x+1), -\frac{\lambda_2}{2r_k})] + r_k[\max((x-8), -\frac{\lambda_3}{2r_k})] = 0$$

$$\frac{\partial A}{\partial y} = 8 + 4\lambda_4 y + 2r_k(2y^2 + x - 8.4) = 0$$

The equations can be written as:

$$\min\left[(6x - 2x\lambda_1 - \lambda_2 + \lambda_3), (6x - 2x\lambda_1 - \lambda_2 + \lambda_3 - 4xr_k(-x^2 - 13.53) - 2r_k(-x+1) + r_k(x-8))\right] = 0$$
$$8 + 4\lambda_4 y + 2r_k(2y^2 + x - 8.4) = 0$$

In the first equation, if $6x - 2x\lambda_1 - \lambda_2 + \lambda_3 = 0 \Rightarrow x = 0$, which violates the constraints.

Then, the solution for the unconstrained problem is

$$(6x - 2x\lambda_1 - \lambda_2 + \lambda_3 - 4xr_k(-x^2 - 13.53) - 2r_k(-x+1) + r_k(x-8)) = 0$$

In the second equation, $4r_k y^2 + 4\lambda_4 y + 2r_k(x-8.4) + 8 = 0$

Let the value of $r_k = 5$ and $\lambda_1^{(1)} = 0, \lambda_2^{(1)} = 0$, this gives:

$$x^{*(1)} = 0.6891,\ y^{*(1)} = 1.9504$$

By updating the values of λ_1, λ_2 with fixed value of *r*=5, we get

$$x^{*(2)} = 0.8838,\ y^{*(2)} = 1.9385$$
$$x^{*(3)} = 0.9567,\ y^{*(3)} = 1.9291$$

The following table shows the parameters values in 10 iterations:

The iterations history								
$\lambda_1^{(i)}$	$\lambda_2^{(i)}$	$\lambda_3^{(i)}$	$\lambda_4^{(i)}$	$r_k^{(i)}$	$x^{*(i)}$	$y^{*(i)}$	$h(x,y)$	$f(x,y)$
0	0	0	0	5	0.6891	1.9504	-0.1025	17.5639
0	3.1091	0	-1.0254	5	0.8838	1.9385	-6.3125e-00	18.2808
0	4.2711	0	-1.0317	5	0.9567	1.9291	-5.0343e-00	18.3734
0	4.7037	0	-1.0368	5	0.9839	1.9256	-1.8743e-00	18.3862
0	4.8647	0	-1.0386	5	0.9940	1.9243	-7.0042e-00	18.3880
0	4.9247	0	-1.0393	5	0.9978	1.9238	-2.6114e-00	18.3883
0	4.9470	0	-1.0396	5	0.9997	1.9236	-3.6237e-00	18.3883
0	4.9553	0	-1.0397	5	0.9999	1.9236	-1.3448e-00	18.3883
0	4.9584	0	-1.0397	5	1	1.9235	-5.0234e-00	18.3883

Table 3.10 Optimal solutions.

3.7 Constrained Optimisation Experiments

This section presents solving a nonlinear constrained optimisation problem taken from [15]. Four different algorithms have been implemented and applied for tackling this problem. The first two are deterministic methods (SQP and ALM), while the other two are stochastic methods (PSO and GA). The

numerical results and the convergence behaviour for each method are shown below. The problems can be described as follows:

$$\min_{x} \quad f(x) = 5.35785x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141$$

$$\text{s.t.} \quad g_1(x) = u(x) - 92 \le 0$$
$$g_2(x) = -u(x) \le 0$$
$$g_3(x) = v(x) - 110 \le 0$$
$$g_4(x) = v(x) + 90 \le 0$$
$$g_5(x) = w(x) - 25 \le 0$$
$$g_6(x) = -w(x) + 20 \le 0$$

where

$$u(x) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - .0.0022053x_3x_5$$
$$v(x) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2$$
$$w(x) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4$$
$$U_B = (102,45,45,45,45),$$
$$L_B = (78,33,27,27,27).$$

(3.11)

SQP Method

Iter	F-count	f(x)	Max constraint	Line search step length	Directional derivative	First-order optimality
0	6	-32217.4	3.237			
1	12	-30374.6	0	1	84.6	305
2	18	-30665.9	0.000353	1	324	9.26
3	24	-30665.5	3.162e-009	1	265	0.00147

Table 3.11 Experimental results.

Global minimum: $\vec{x}^* = (78.0000 \quad 33.0000 \quad 29.9953 \quad 45.0000 \quad 36.775)$,

$f(\vec{x}^*)$ = -30665.539

ALM method

The numerical results for solving this problem using Augmented Lagrangian multipliers method are shown in Table 3.12

Optimal objective $f(\vec{x}^*)$	-30665.539
Variables solution $\vec{x}^*$	[78.0028, 32.9992, 29.9809, 45.0000,36.8109]
Lagrangian multipliers $\vec{\lambda}^*$	[398.7382, 0, 0,0, 0,807.249 , 49.0407, 83.6634, 0, 0, 0, 0,0, 0, 26.6518, 0]
Penalty parameters $\vec{r}^*$	[305,1,1,1,1,1234.8, 15,305,11,1,3,1,1,1,2296,1]

Table 3.12. Optimal design

Genetic Algorithm

Global minimum: $\vec{x}^*$ = (78.001, 33.012, 33.116, 45, 29.627)

$f(\vec{x}^*)$ = -30076.1152

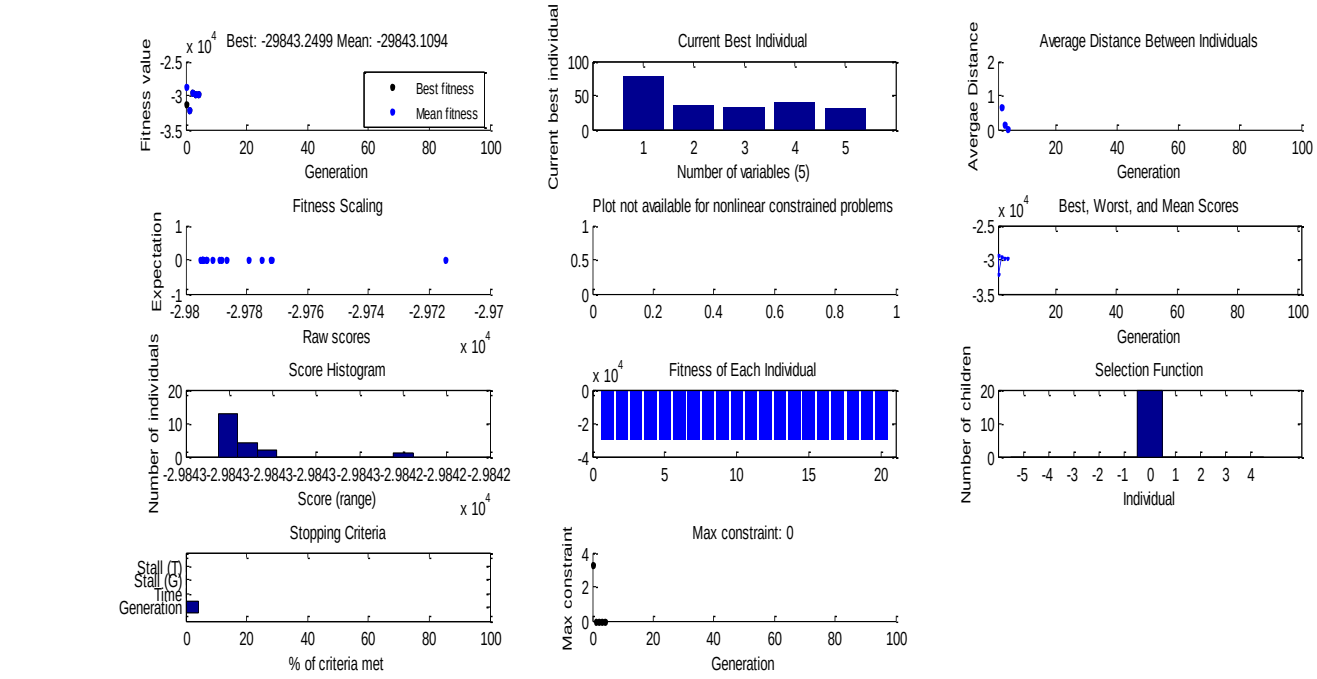


Fig.3.9 Convergence Characteristics plots

Particle Swarm Optimisation

Global minimum: $\vec{x}^*$ = (78.0027, 32.9992, 29.9807, 45.0000, 36.8114)

$f(\vec{x}^*)$ = -30667.9073

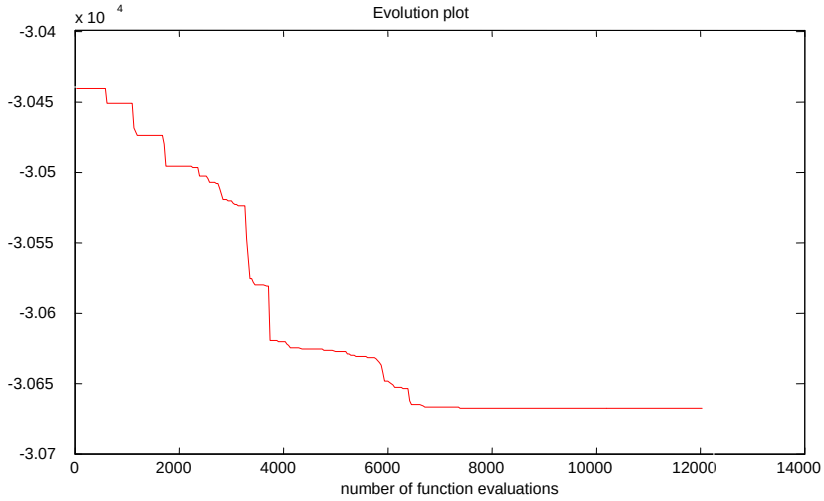


Fig. 3.10 Convergence plot for a nonlinear constrained optimisation problem using PSO.

3.8 Summary

In this Chapter, a collection of test models for continuous and mixed-variables nonlinear programming has been presented. The results of the implemented algorithms have been checked with a number of existing state of the art solvers for global constrained optimisation. In the later chapters, the new algorithms will be described and the major theoretical properties of these methods will be illustrated. The validity, robustness and effectiveness of the new algorithms are compared through some well known benchmark optimisation problems.

Chapter 4

A Hybrid Cooperative Search Algorithm for Constrained Optimisation

4.1 Introduction

Many engineering design problems can be formulated as constrained optimisation problems which often consist of many mixed equality and inequality constraints. In this chapter, a hybrid coevolutionary method is developed to solve constrained optimisation problems formulated as min-max problems. The new method is fast and capable of global search because of combining particle swarm optimisation and gradient search to balance exploration and exploitation. It starts by transforming the problem into an unconstrained one using an augmented Lagrangian function, then using two groups to optimise different components of the solution vector in a cooperative procedure. In each group, the final stage of the search procedure is accelerated by a simple local search method on the best point reached by the preceding exploration based search. We validate the effectiveness and robustness of the proposed algorithm using several engineering problems taken from the specialist literature.

4.2 General Background

In the last decade, there has been a dramatic increase in the number of the techniques developed to solve Constrained Optimisation Problems (COP). Evolutionary Algorithms (EAs) have been widely used for solving such problems. These algorithms share the principle of being computer-based approximate

representations of natural evolution. They typically alter the population solutions over a sequence of generations according to statistical analogues of the process of evaluation. Such techniques have been applied in various works [19, 20, 21], where promising results have been obtained. However, classical EAs techniques may fail to solve many real-world optimisation problems with highly structured constraints, whereas achieving the exact global solution is neither possible nor desirable**.** Therefore, the use of Co-Evolutionary Algorithms (CEAs) is highly needed in order to solve difficult computational problems where acceptable solutions can be achieved. CEAs have many particular advantages over the traditional EA methods, which make them successfully applied in many difficult engineering problems [22, 23, 24]. These algorithms can perform better than the standard EA methods because of their parallelism and high efficiency in exchanging information between individuals.

In general, coevolutionary algorithms represent a natural approach to applying evolutionary computation to refine multi population behaviours. In order to apply a cooperative coevolutionary framework to solve a particular problem, the problem has to be decomposed into subcomponents and assign each subcomponent to a subpopulation [25]. By dividing a complicated problem into several relatively simple sub-problems, the algorithm can effectively solve complicated optimisation problems with highly structured constraints. These subpopulations should be evolved by a particular evolutionary method and co-evolved simultaneously. One of the most crucial steps in a CEA is the fitness of an individual, which should be based on its interaction with other individuals in the population.

In recent years, coevolutionary algorithms have received a lot of attention regarding their potential for solving the numerical constrained optimisation problems where several researchers proposed some new methods. Huang et al. [26] introduced an effective coevolutionary Differential Evolution (DE) for constrained optimisation where a special penalty function is designed to handle the constraints, then a coevolution model is presented and differential evolution is employed to perform evolutionary search in spaces of both solutions and penalty factors. Another similar work is that of Liu et al. [27] that presented a memetic coevolutionary DE algorithm for solving COP. Two cooperative populations are constructed

and evolved by independent DE algorithm. The purpose of the first population is to minimise the objective function regardless of constraints, and that of the second population is to minimise the violation of constraints regardless of the objective function. Furthermore, He and Wang [28] introduced coevolutionary Particle Swarm Optimisation (PSO) for constrained optimisation by employing the notion of coevolution to adapt penalty factors. PSO is applied with two kinds of swarms for evolutionary exploration and exploitation in spaces of both solutions and penalty factors. Krohling and Coelho [29] also presented an approach based on coevolutionary PSO to solve COP formulated as min-max problems. Two populations of PSO are evolved, and a Gaussian probability distribution is used to generate the acceleration coefficients of PSO. In the field of GAs, Tahk and Sun [30] proposed an efficient coevolutionary augmented Lagrangian method with an annealing scheme for COP, where the populations of the parameter vector and the multiplier vector approximate the zero-sum game by a static matrix game. Selection, recombination, and mutation are done by using the evolutionary mechanism of conventional evolutionary algorithms such as evolution strategies and evolutionary programming. The work of Park et al. [31] used a coevolutionary Genetic Algorithm (GA) specifically designed to optimise gas production system, whereas a fuzzy-formulation is combined with a coevolutionary GA for solving optimum gas production rates of each well to minimise investment cost with given constraints in order to enhance ultimate recovery.

At the mean time, hybrid metaheuristics have received considerable interest in recent years. A variety of hybrid coevolutionary approaches have been proposed in the literature. Son and Baldick [32] proposed a hybrid coevolutionary programming for Nash equilibrium search in games with local optima, whereas a parallel and global search coevolutionary algorithm is applied to locate the real Nash equilibrium. Another hybridisation strategy has been presented by Rivera et al. [33] which is a multiobjective hybrid methodology for cooperative-coevolutionary optimisation of radial basis function networks.

In this chapter, a new Hybrid Coevolutionary Particle swarm optimisation architecture, named HCP, is presented. This coevolutionary approach is capable of solving difficult real-world constrained

optimisation problems formulated as min-max problems with the saddle point solution. We consider a two-group model; in such models individuals from the first group interact with individuals from the second through a common fitness evaluation based on payoff matrix game. In each group, a population of independent PSO is evolved, the first population evolves the variable vector while the other the Lagrangian multiplier vector. In order to enhance the poor convergence of simple coevolutionary programming, a hybrid approach is suggested, where the optimisation of each group takes advantage from combining the global but slow search of PSO with the rapid but local search of Quasi-Newton method (QN). When applying HCP to specific real-world problems, it is often found that the addition of gradient-search mechanism can aid in finding good solutions quickly and reliably. The hybridization phase of HCP based on fine-tuning the PSO performance in an attempt to achieve improved results in exploring the search space. At the end of the optimisation, the first group provides the variable vector, and the second group provides the Lagrange multiplier vector.

4.3. Cooperative Coevolutionary framework

4.3.1 Augmented Lagrangian method

A constrained optimisation problem can be converted to an equivalent unconstrained problem by using the penalty function approach, where there is no need to adjust the optimisation algorithm to work on constraints [2]. Then, an algorithm for unconstrained problems can be used to find solutions that do not violate the constraints. The Augmented Lagrangian Method (ALM) is the most robust of the penalty function methods. This method can also be used to convert the constrained problem into an unconstrained one by defining the Lagrangian for the constrained problem, and then by maximising the Lagrangian [3]. More importantly it also provides information on the Lagrangian multipliers at the solution. The equality and inequality constraints can be introduced into the objective function by augmenting it with a weighted sum of the constraint functions.

We shall next describe the concept of Lagrangian duality where the optimisation problem has two different presentations, the primal and the dual problem; the relation between the two problems is provided by an appropriate duality theory [34]. The general constrained problem formulation which is called the primal problem (P) can be stated as:

$$\begin{aligned} \underset{x}{Min} \quad & f(\vec{x}) \\ & g_i(\vec{x}) \le 0, \quad i = 1,\ldots,m \\ s.t \quad & h_j(\vec{x}) = 0, \quad j = 1,\ldots,l \\ & \vec{x} \in X \subseteq \Re^n \end{aligned} \tag{4.1}$$

where $\vec{x}$ represent a vector of n real variables subject to a set of m inequality constraints $g(\vec{x})$ and a set of l equality constraints $h(\vec{x})$. The Lagrangian function $L(\vec{x},\vec{\lambda},\vec{\mu})$ of (P) can be defined as:

$$L(\vec{x},\vec{\lambda},\vec{\mu}) = f(\vec{x}) + \sum_{i=1}^{m} \lambda_i g_i(\vec{x}) + \sum_{i=1}^{l} \mu_i h_i(\vec{x}) \tag{4.2}$$

where $\vec{\lambda}$ is a vector of Lagrangian multipliers for the inequalities $g(\vec{x})$, and $\vec{\mu}$ is a vector of the Lagrangian multipliers for the equalities $h(\vec{x})$. The problem (P) can be solved if one can find a saddle-point $(\vec{x}^*,\vec{\lambda}^*,\vec{\mu}^*)$ of the Lagrangian function. The dual function $\psi(\vec{\lambda},\vec{\mu})$ can be defined by

$$\psi(\vec{\lambda},\vec{\mu}) = \underset{x\in\Re^n}{Inf} \; L(\vec{x},\vec{\lambda},\vec{\mu}) \tag{4.3}$$

If the problem (P) has a saddle point $(\vec{x}^*,\vec{\lambda}^*,\vec{\mu}^*)$, then we have

$$Max(D) = \psi(\vec{\lambda}^*,\vec{\mu}^*) = f(\vec{x}^*) = Min(P) \tag{4.4}$$

Where the optimal value of the primal problem (P) equals the optimal value of the dual problem (D). Conversely, if there is a solution $\vec{x}^*$ of (P) and $\vec{\lambda}^* \ge 0$, such that $\psi(\vec{\lambda}^*,\vec{\mu}^*) = f(\vec{x}^*)$, then (P) has a saddle point and $(\vec{x}^*,\vec{\lambda}^*,\vec{\mu}^*)$ is such a saddle point. This property applies, in particular, to convex programming and corresponds to the saddle point of the Lagrangian function defined by Eq. (4.2), and then we have:

$$L(\vec{x}^*,\vec{\lambda},\vec{\mu}) \le L(\vec{x}^*,\vec{\lambda}^*,\vec{\mu}^*) \le L(\vec{x},\vec{\lambda}^*,\vec{\mu}^*) \tag{4.5}$$

The dual function ψ can be described as a concave function of $\vec{\lambda}$ and $\vec{\mu}$. The concavity of ψ makes it possible to state that every local optimum of ψ is a global optimum. The dual problem (D) will therefore in general be more easily solved than the primal problem (P), where:

$$\underset{\lambda,\mu}{Max}\phi(\vec{\lambda},\vec{\mu}) = \phi(\vec{\lambda}^*,\vec{\mu}^*) \tag{4.6}$$

In order to design an efficient duality framework for solving nonconvex constrained optimisation problems; a combination of the Lagrangian approach with penalty function method has to be used by adding a quadratic penalty term to the Lagrangian function in Eq. (4.2), which can now be defined as:

$$\phi\left(\vec{x},\vec{\lambda},\vec{\mu},\vec{r}\right) = f(\vec{x}) + \sum_{i=1}^{m}\theta_i(\vec{x})\cdot\left[\lambda_i + r_i\theta_i(\vec{x})\right] + \sum_{j=1}^{l}h_j(\vec{x})\cdot\left[\mu_j + r_j h_j(\vec{x})\right] \tag{4.7}$$

The *r* represents the positive penalty terms for the corresponding two types of constraints. $\theta_i(\vec{x})$ is used to convert the inequalities to equalities via setting

$$\theta_i(\vec{x}) = \max\left\{g_i(\vec{x}), -\frac{\lambda_i}{2r}\right\} \tag{4.8}$$

It can be shown that the solution of the primal problem is identical to that of the augmented function $\phi\left(\vec{x},\vec{\lambda},\vec{\mu},\vec{r}\right)$. The optimal solution $\vec{x}^*$ can be obtained by an unconstrained search if $(\vec{\lambda}^*,\vec{\mu}^*)$ is known and the search starts from point close to $\vec{x}^*$. The most critical point of the deterministic augmented Lagrangian methods is how to select an appropriate updating strategy so that it converges to $(\vec{\lambda}^*,\vec{\mu}^*)$. The proposed HCP algorithm overcomes this difficulty by the use of a cooperative coevolutionary approach to applying evolutionary computation to achieve the saddle point $(\vec{x}^*,\vec{\lambda}^*,\vec{\mu}^*)$.

Example:

The example presented here is a constrained problem to show that the property in Eq.(4.7). Consider a convex problem given by

$$\begin{cases} Min\ \ 2x_1^2 + 3x_2^2 \\ s.t \\ 4x_1 + x_2 \le -5 \end{cases}$$

Here we have,

$$f(x_1, x_2) = 2x_1^2 + 3x_2^2 \quad , \; g(x_1, x_2) = 4x_1 + x_2 + 5$$

The objective function *f* and the constraints *g*, and *h* are convex. Therefore there is a saddle point where

$$f(x^*) = \phi(\lambda^*, \mu^*)$$

$$L(x, \lambda) = 2x_1^2 + 3x_2^2 + \lambda(4x_1 + x_2 + 5)$$

Hence, the dual function:

$$\phi(\lambda) = -\frac{25}{12}\lambda^2 + 5\lambda$$

The maximum of ϕ is obtained when $\frac{\partial \phi}{\partial \lambda} = 0$, hence $\lambda^* = 1.2$, then we have:

$$f(x^*) = \phi(\lambda^*) = -\frac{25}{12}(1.2)^2 + 5(1.2) = 3.$$

The solution for this problem is found as: $x_1^* = -\lambda = -1.2$, and $x_2^* = -\lambda / 6 = -0.2$, which is the unconstrained minimum of the primal and dual problems.

4.3.2 PSO Module

The PSO is a stochastic method for global optimisation that exploits a population of potential solutions to probe the search space. This derivative free approach is particularly suited to continuous variable problems and has recently been successfully applied to many optimisation problems [11,19]. In this evolutionary method, a group of particles optimises a certain objective function $f(.)$, where the trajectory of each individual in the search space is adjusted by dynamically alternating the velocity of each particle, according to its own experience (cognitive knowledge) and the progress of other particles (social knowledge) in the search space.

In the PSO, we have a set of possible solutions $x \in \Re^n$ (the particle positions), with x_{js} denoting the s^{th} vector component of the j^{th} particle. At each k^{th} iteration of the swarm operation, each position x_j^k is updated according to the velocity v_j^k of the j^{th} particle, according to

$$\begin{aligned} \vec{v}_{js}^{\,k} &= \vec{v}_{js}^{\,k-1} + c_1 \cdot r_{1\,js}^k \cdot \left(P_{js}^k - \vec{x}_{js}^{\,k-1}\right) + c_2 \cdot r_{2\,js}^k \cdot \left(G_s^k - \vec{x}_{js}^{\,k-1}\right) \\ \vec{x}_{js}^{\,k} &= \vec{x}_{js}^{\,k-1} + \vec{v}_{js}^{\,k} \end{aligned} \tag{4.9}$$

where c_1 and c_2 are acceleration constants that control how far each particle moves in a single iteration, $r_{1\,jk}, r_{2\,jk} \in [0,1]$ are uniform randomly generated numbers that attain the stochastic swarm behaviour. It can be seen from the three components of Eq.(4.9), that the trajectory of each j^{th} particle is adjusted to take into account its own best known solution P_j (individual experience), and the best known solution G in the entire swarm (collaboration between the N members). The personal and global best positions are correspondingly given by

$$\begin{aligned} P_j^k &= \underset{x_j^i : 0 \le i \le k-1}{\arg\min} \left\{ f\left(\vec{x}_j^{\,i}\right) \right\}, \ \forall\, j = 1,\ldots,N \\ G^k &= \underset{P_j^k : 1 \le j \le N}{\arg\min} \left\{ f\left(P_j^k\right) \right\} \end{aligned} \tag{4.10}$$

In the proposed HCP algorithm, an efficient automation strategy for the PSO with linear time-varying acceleration coefficients was adopted [35]. Under this development, the cognitive parameter c_1 starts with a high value $c_{1\max}$ and linearly decreases to $c_{1\min}$, whereas the social parameter c_2 starts with a low value $c_{2\min}$ and linearly increases to $c_{2\max}$. Both parameters can be updated according to

$$\begin{aligned} c_1(k) &= c_{1\min} + \frac{k_{\max} - k}{k_{\max}} \cdot \left(c_{1\max} - c_{1\min}\right) \\ c_2(k) &= c_{2\max} + \frac{k_{\max} - k}{k_{\max}} \cdot \left(c_{2\min} - c_{2\max}\right) \end{aligned} \tag{4.11}$$

where, $k_{\max}$ is the maximal number of iterations and k is the current number of iterations.

4.3.3 Gradient search Module

In HCP algorithm, the search toward the optimal solution is guided by the Quasi-Newton method, which based on the idea of building up curvature information as the iterations of a descent method proceed [3]. After a continuous constrained problem is transformed to a continuous unconstrained one through the augmented Lagrangian function, a standard QN algorithm can be employed to efficiently minimise it. Second-order gradient-based algorithms proceed towards the minimum point of a minimising function

$f(\vec{x})$ in a sequential manner by updating the current solution in each *(k+1)*th iteration as

$$\vec{x}^{k+1} = \vec{x}^{k} - H\left(\vec{x}^{k}\right)^{-1} \cdot \nabla f\left(\vec{x}^{k}\right) \tag{4.12}$$

To reduce the computational load of estimating the Hessian *H* at point $\vec{x}^{k}$ in each iteration, QN builds up curvature information using first-order derivatives by applying the Sherman-Morrison formula

$$\begin{aligned} H^{k+1} &= H^{k} + \frac{\left(y^{k} - H^{k}s^{k}\right)\cdot\left(y^{k} - H^{k}s^{k}\right)^{T}}{\left(Q^{k} - H^{k}s^{k}\right)^{T}s^{k}} \\ \textit{where} \quad Q^{k} &= \nabla f\left(\vec{x}^{k+1}\right) - \nabla f\left(\vec{x}^{k}\right) \\ s^{k} &= -H^{k}\left(\vec{x}^{k}\right)^{-1} \cdot \nabla f\left(\vec{x}^{k}\right) \end{aligned} \tag{4.13}$$

where H^{0} is usually taken to be the identity matrix.

4.3.4 Coevolutionary Game Theory

In this section, we shall confine our intention to zero-sum games with two players [36]. In such games, the competitive aspect is extreme, since whatever is won by one player is lost by the other. Assume that player 1 (P1) has strategies $A_1,....,A_{N_A}$ and that player 2 (P2) has strategies $B_1,....,B_{N_B}$. Then, for every pair of strategies (A_i, B_j), there will be a pay-off $P(A_i, B_j)$ for Player P1. Since the game is a zero-sum game, the corresponding pay-off for player P2 must be $-P(A_i, B_j)$. Thus, the game may be described by the $N_A \times N_B$ matrix of real number in which the entry of the in the *i*th row and *j*th column is $P(A_i, B_j)$. In a more general framework, these outcomes represent utility transfers from one player to the other. Thus, we can view each element of the matrix as the net change in the utility of P2 for a particular play of the game, which is equal to the negative of the net change in the utility of P1. Then, regarded as a rational decision maker, P1 will seek to minimise the outcome of the game, while the P2 will seek to maximise it, by independent decisions.

Let $\vec{x}$ and $\vec{y}$ denote individuals representing the options A_i and B_j, respectively, where $\vec{x}$ varies over all mixed strategies for *A*, and $\vec{y}$ varies over all mixed strategies for *B*. Assume that the game is to be

played only once, then it would seem to be plausible for P1 and P2 to get values that are at least as big as the values that they can be sure of obtaining the so-called security levels. Define these values S_1 and S_2 as follows:

$$S_1 = \max_x \min_y P(\vec{x}, \vec{y}) \tag{4.14}$$

$$S_2 = \max_y \min_x P(\vec{x}, \vec{y}) \tag{4.15}$$

The score of the match (A_i, B_j) is defined as the value of $S(x_i, y_j)$. It should be noted that, the security level of P1 (the minimiser) never falls below the security level of P2 (the maximiser), i.e.

$$\min_x P(x_i, \vec{y}^*) = \underline{S}(P) \leq \overline{S}(P) = \max_y P(\vec{x}^*, y_i) \tag{4.16}$$

Where $\vec{x}^*, \vec{y}^*$ denote security strategies for P1 and P2, respectively.

To illustrate these facets of matrix game, let us now consider the following (3×5) matrix:

		P2				
P1	–4	5	–1	2	0
	0	–2	5	1	1
	2	1	7	–3	4

Here P2 has unique security strategy, "column 3" (i.e. j^*=3), securing him a gain-floor of $\underline{S} = \min_i P(i,3) = -1$. P1, on the other hand, has two security strategies, "row 1" and "row 2" (i.e. i_1^*=1, i_2^*=2), yielding him a loss-ceiling of $\overline{S} = \max_j P(1, j) = \max_j P(2, j) = 5$ which is above the security level of P2. Now, if P2 plays first, then he chooses his security strategy " column 3", with P1's unique response being "row 1" (i°=1), resulting in an outcome of $\underline{S}$ = –1. If P1 plays first, he is actually indifferent between his two security strategies. In case he chooses "row 1", then P2's unique response is "column 2" (j° = 2), whereas if he chooses "row 2", his opponent's response is "column 3" (j° = 3), both pairs of strategies yielding an outcome of $\overline{S}$ = 5.

4.3.5 The proposed HCP Algorithm

The basic idea of the proposed method is to decompose the search space into two subpopulations, and then use the best individuals from each subpopulation collaborating together for fitness evaluation. The algorithm descends in the original continuous variable space $\vec{x}$ and ascends in the Lagrangian-multiplier space of $(\vec{\lambda}, \vec{\mu})$. For each subpopulation being evolved, the search towards the optimal solution is processed by integrating the PSO and QN methods to facilitate exploration and exploitation. The two groups are involved to solve saddle point problems, whereas the potential solutions of each group form their own sub-population, and evolve only in their own group. Both populations create a matrix game which is defined by the individuals of the two opposing groups. The structure of the HCP method is shown in Fig. 4.1, and the algorithm can be described by the following steps:

(1) Lagrangian formulation

At the beginning of the HCP algorithm, it requires a complete unconstrained optimisation before performing any decomposition procedures. Hence, the concept of augmented Lagrangian is used to transform the problem to unconstrained one. The ALM function can be incorporated into the HCP algorithm according to Eq.(4.7), whereas the approach preserves separability without violating or using explicit constraints. The choice of the penalty weight plays an important role in the convergence behavior of the algorithm. It should be kept as low as possible, just above the limit below which infeasible solutions are optimal. Therefore, In each iteration k, the penalty parameters can be updated as the following:

$$r_i^t = t \times r_i^0, \quad \forall i = 1, \ldots, m \tag{4.17}$$

Where t and r_i^0 are the user-defined values according to $t > 1$, and $r_i^0 > 0$. The same rule applies to updating the equality penalty r_j^k, $\forall j = 1, \ldots, l$. The user can also control the growth of the penalties by setting an upper value for them.

(2) Initialisation of populations

Two populations of PSO are created randomly within the design space. The size of each population is predefined and fixed throughout. The first population (P1) presents the parameter solution vector $\vec{x}$, while the second population (P2) presents the Lagrangian multipliers vector $(\vec{\lambda}, \vec{\mu})$. The algorithm takes each function variable as a separate subpopulation of individuals, so each population group has its own evolutionary process.

(3) Matrix game

Based on the description in the previous sections, the coevolutionary matrix game can be presented by an $N_A \times N_B$ payoff matrix, where each group plays some pure strategy. Each population is defined as the collection of all particles representing the potential solutions. The security level of the first population group (P1) is the maximum value it can expect in the worst possible case, that is, the case when the second population group (P2) tries to minimise P1's pay-off. Similarly, the security level of P2 can be obtained by applying the same procedures to P2's payoff matrix.

(4) Evaluation of population

The security strategy of the matrix games has been applied to determine the fitness of each particle in the swarm. The first group is focusing on evolving the solution vector $\vec{x}^*$, with the Lagrangian vector fixed. The second group has its objective maximising the Lagrangian by finding the appropriate vectors $(\vec{\lambda}^*, \vec{\mu}^*)$ while fixing $\vec{x}$. For the first population, the fitness function for each particle is defined as:

$$\overline{f}(x_i) = \max_{\lambda,\mu \in P2} \phi(x_i, \vec{\lambda}, \vec{\mu}) \qquad i = 1,2,\ldots,U \tag{4.18}$$

For the second population, the fitness function is defined as:

$$\underline{f}(\lambda_j, \mu_j) = \min_{x \in P1} \phi(\vec{x}, \lambda_j, \mu_j) \qquad j = 1,2,\ldots,V \tag{4.19}$$

Where U and V are the size of each population group.

(5) Invoke PSO and QN modules for P1

In general, a stochastic optimiser algorithm will need a very large number of iterations to make it more probable that it will reach a global optimum. Therefore, the HCP algorithm combines the advantages of random search and deterministic search methods. At the beginning of the search procedure, the PSO is used mainly for minimising the continuous solution vector $\vec{x}$ in order to determine the optimal feasible region surrounding the optimum point. The inherent search mechanism of PSO should be modified to locate the feasible region near the optimum solution. Therefore, we use an efficient method, called fly-back mechanism proposed by He et al. [19], which utilizes the information about the feasible region that the particle learned from its fly experience during the search process, where the optimal feasible region is determined according to the sorted fitness values. The process terminates when the population fitness value becomes stable.

At the end of the search of PSO, the best known solution of all the swarm particles G is chosen as a starting search point for the iterative QN method that is subsequently used to replace PSO to find the final optimum solution, since the gradient-based method usually has faster convergence rate and higher convergence efficiency compared to stochastic random search method. The hybridisation phase in each group is designed to provide greater emphasis on optimality during the search process, so as to avoid the algorithm to converge to the local optima.

(6) Invoke PSO and QN modules for P2

As discussed earlier, the second population group has its own objective and separate evolutionary process. It only focuses on optimising the Lagrangian multiplier vector $(\vec{\lambda}, \vec{\mu})$, whereas a hybrid mechanism combining PSO and QN methods has been used to obtain the solution vector $(\vec{\lambda}^*, \vec{\mu}^*)$. However, the fitness of an element in the second population depends on the outcome of the first population, which is the leader group.

(7) Termination criteria

As a result, the global best in the first population group is the solution for the continuous variables vector $\vec{x}$, while the global best in the second population group is the solution for the Lagrangian multiplier vector $(\vec{\lambda}, \vec{\mu})$. The process of the HCP algorithm is terminated when the relative error between the augmented function in two successive iterations becomes very small, or the maximum number of iterations is reached.

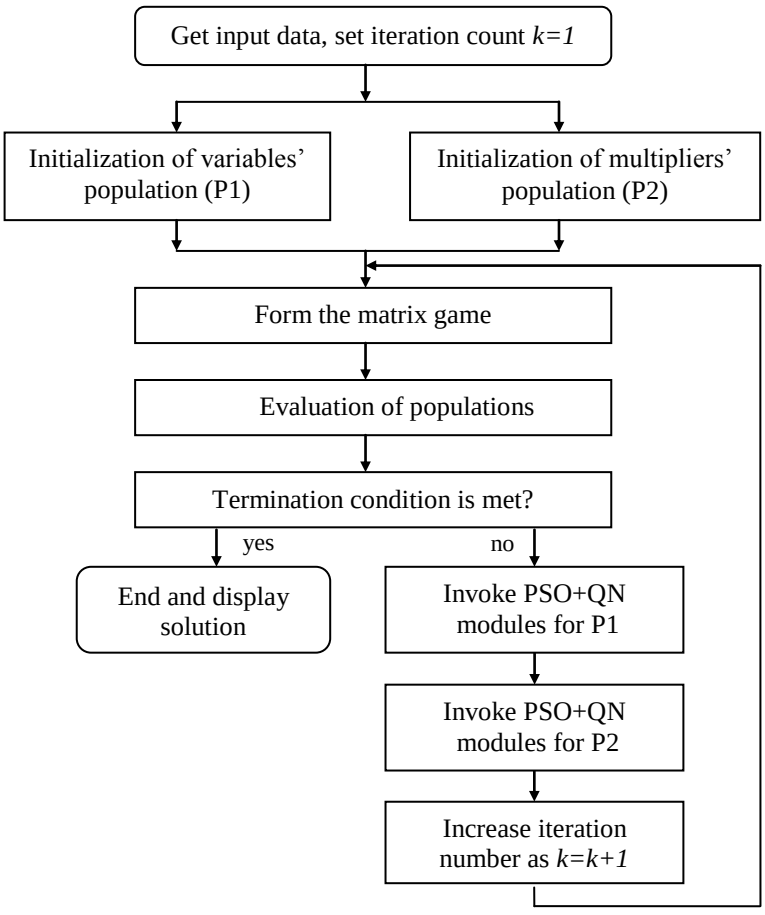


Fig. 4.1 Flowchart of the proposed HCP algorithm.

4.4 Benchmark problems

The performance of the proposed coevolutionary algorithm was investigated on well known and widely used problems frequently employed in the literature, where several coevolutionary algorithms developed for constrained optimisation are compared. The algorithm has been implemented in MATLAB 7.7, using some minor components available as part of the optimisation toolbox. All the programs were run on a 3.2-GHz AMD Athlon processor with 2 GB of RAM. Five test problems proposed by different authors have been chosen to illustrate the applicability and the efficiency of the implemented algorithm. This has been done by performing 100 independent runs for each test case.

In order to investigate the best performance of the HCP algorithm, different population sizes are used for each problem with the maximum number of search iterations $k_{\max}$ set to 2000. The experimental results suggested that for all benchmark tests, a small population could produce quickly good results. However, in some cases, the choice of a smaller population size produces solutions that are a little worse than the cases of larger population. Hence, the population size's choice should depends on the complexity of such problem, because the different optimisation problems converge differently. It also observed that the population ratio is not a critical parameter for the proposed coevolutionary approach. Hence, we recommend using the same population size for each optimisation group.

A. Himmelblau's Function

The first example is a common benchmark problem in nonlinear constrained optimisation, which was originally proposed by Himmelblau [37]. This problem has five design variables, six nonlinear constraints, and ten boundary conditions as follows:

$$
\begin{aligned}
&\textit{Min} \quad f(\vec{x}) = 5.3578547 x_3^2 + 0.8356891 x_1 x_5 + 37.293239 x_1 - 40792.141 \\
&\textit{s.t} \qquad 0 \le g_1(\vec{x}) \le 92 \\
&\qquad\qquad 90 \le g_2(\vec{x}) \le 110 \\
&\qquad\qquad 20 \le g_3(\vec{x}) \le 25 \\
&\textit{where} \\
&\qquad g_1(\vec{x}) = 85.334407 + 0.0056858 x_2 x_5 + 0.0006262 x_1 x_4 - 0.0022053 x_3 x_5 \\
&\qquad g_2(\vec{x}) = 80.51249 + 0.0071317 x_2 x_5 + 0.0029955 x_1 x_2 + 0.0021813 x_3^2 \\
&\qquad g_3(\vec{x}) = 9.300961 + 0.0047026 x_3 x_5 + 0.0012547 x_1 x_3 + 0.0019085 x_3 x_4 \\
&\textit{and} \\
&\quad 78 \le x_1 \le 102, \quad 33 \le x_2 \le 45, \quad 27 \le x_3 \le 45, \quad 27 \le x_4 \le 45, \quad 27 \le x_5 \le 45
\end{aligned}
$$

Several attempts have been made to solve this problem. For instant, Himmelblau used a generalised reduced gradient method to search for the optimal solution and the best result obtained was *–30,373.94*. This problem was also investigated by Runarsson and Yao [38], where an evolutionary strategy with stochastic ranking has been used to solve the problem. They reported a best result of *–30,665.53*. Gen and Cheng [39] have tackled this problem using an improved GA and the best solution they obtained was *–30,183.57*. In our approach, the HCP had the lowest objective function value with lower computational cost than all of the other algorithms. The algorithm found an optimal objective function value of *–30,665.57* when using a population size of *30* particles. Furthermore, we have investigated the convergence behaviour of the HCP algorithm when varying the population size in each PSO+QN model. For instance, when using a population size of *15* particles, an optimal objective function value of *–30,661.19* was produced after performing *100* runs. The number of fitness function evaluations for the best run was *7,400*. In the mean time, a value of *–30,665.57* with *16,520* function evaluations was obtained when using a population size equal to *40*. Obviously, a larger number of particles produces the best optimal solution but involves a higher computational cost, while a smaller number of particles has better computational efficiency but with a considerably poorer performance. In conclusion, for this problem, we recommend using *30* particles for each population group.

The optimal results of Himmelblau's problem are shown in Table 4.1, where the final outcome of each group has been presented ($\vec{x}^*$ and $\vec{\lambda}^*$). The mean fitness value for *100* independent runs was *–30629.36*

with a standard deviation of *83.67*. The run that resulted in the best objective function value performed *12,000* function evaluations. Fig.4.2 shows a plot of the performance of the HCP algorithm.

Optimal objective $f(\vec{x}^*)$	–30665.57
Variables solution $\vec{x}^*$	[78.0027, 32.9992, 29.9809, 45.00, 36.8109]
Lagrangian multipliers $\vec{\lambda}^*$	[398.7, 0, 0, 0, 0, 807.2, 49.0, 83.6, 0, 0, 0, 0, 0, 0, 26.6, 0]
Penalty parameters $\vec{r}^*$	[304, 1, 1, 1, 1, 12347, 14, 304, 10, 1, 2, 1, 1, 1, 2295, 1]

Table 4.1. Optimal design of Himmelblau's Function.

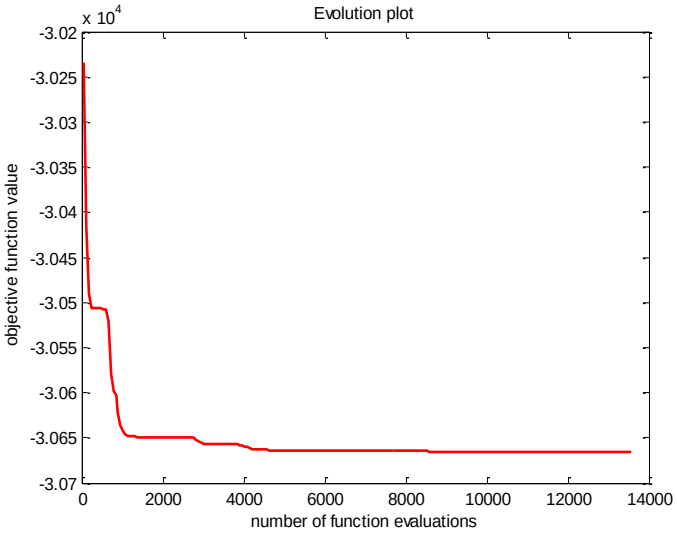


Fig.4.2 Evolution plot for Himmelblau's Function.

B. ***Minimisation of the weight of a tension/compression string***

This is a more complicated example taken from [15], which tackles the minimisation of the weight of a tension/compression spring subject to constraints on minimum deflection, shear stress, surge frequency, and limits on outside diameter. As shown in Fig.4.3, the design variables are the wire diameter $d(x_1)$, the mean coil diameter $D(x_2)$, and the number of active coils $P(x_3)$. The mathematical formulation of this problem is stated as:

$$Min \quad f(\vec{x}) = (x_3 + 2)x_2 x_1^2$$

$$s.t \quad g_1(\vec{x}) = 1 - \frac{x_2^3 x_3}{71785 x_1^4} \le 0,$$

$$g_2(\vec{x}) = \frac{4x_2^2 - x_1 x_2}{12566(x_2 x_1^3 - x_1^4)} + \frac{1}{5108 x_1^2} - 1 \le 0,$$

$$g_3(\vec{x}) = 1 - \frac{140.45 x_1}{x_2^2 x_3} \le 0,$$

$$g_4(\vec{x}) = \frac{x_1 + x_2}{1.5} - 1 \le 0.$$

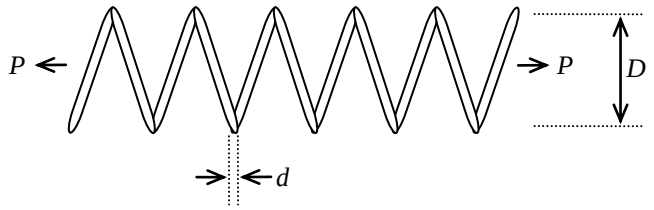


Fig.4.3 Tension/compression string design problem.

This problem has been investigated by previous researchers to minimise the weight of a tension spring. Belegundu [15] used a method based on Lagrange multipliers technique to solve this problem, while Arora [4] has used a numerical optimisation method called constraint correction at constant cost. The best known results was obtained by Coello and Montes [40] using a dominance-based selection scheme to incorporate constraints into the fitness function of a genetic algorithm. In HCP algorithm, the optimum value of the objective function is found to be slightly better than the result reported by Coello and Montes [41], but with a significant improvement in the number of function evaluations compared. MDNLPs typically include continuous, discrete and integer design variables. In this case, a rounding off technique based on the work of Ringertz [42] is used for treating the discreteness requirements on the number of active coils $P(x_3)$ .As has been observed in our numerical experiments, this simple truncation of their real values does not effect the search performance and keeps the handling of continuous and discrete variables uniform. The mean value from this problem after *100* runs was *0.0127* with a standard deviation of *0.00019*. The number of fitness function evaluations for each run was *650*, when using a population size of *10* particles. The convergence plot of the best solution produced by all runs are shown in Fig. 4.4,

while a comparison of results of the proposed algorithm, as well as results published in the literature are shown in Table 4.2.

Quantity	HCP	[Coe02]	[Aro89]	[Bel82]
x_1	0.051987	0.051989	0.053396	0.050000
x_2	0.363964	0.363965	0.399180	0.315900
x_3	10.890521	10.890522	9.185400	14.25000
g_1	-0.0014	-0.0000	0.0000	-0.0000
g_2	0.0000	-0.0000	-0.0000	-0.0037
g_3	-4.0611	-4.0613	-4.1238	-3.9383
g_4	-0.7226	-0.7226	-0.6982	-0.7560
$f(\vec{x}^*)$	0.012679	0.012681	0.012730	0.012833

Table 4.2 Comparison of the results for the minimisation of the weight of a tension spring.

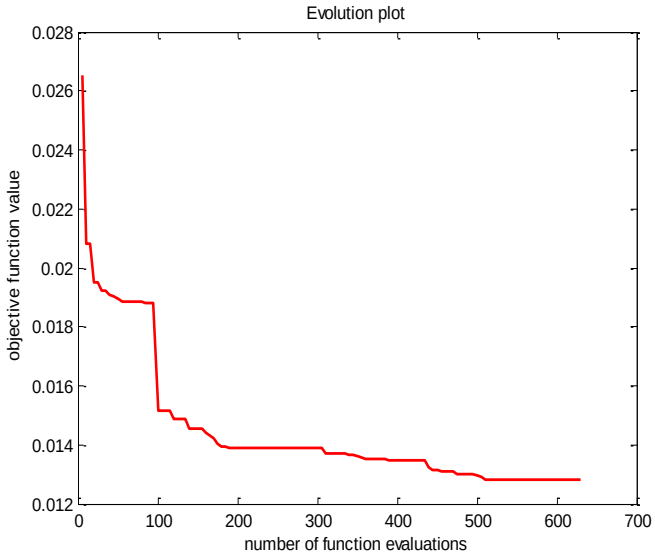


Fig.4.4 Performance of the minimisation of the weight of a tension spring problem.

C. ***Pressure Vessel Design***

This problem was introduced by Sandgren [43] and aims to minimise the total cost of materials for forming and welding of a pressure vessel. A cylindrical vessel is capped at both ends by hemispherical heads as shown in Fig. 4.5. There are four design variables: T_s (Thickness of the shell), T_h (Thickness of the head), R (inner radius), and L (length of the cylindrical section of the vessel, not including the head). T_s and T_h are integer multiples of *0.0625* in., which are the available thicknesses of rolled steel plates, and R and L are continuous. The problem can be described as follows:

$$Min \quad f(\vec{x}) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3$$

$$s.t \quad g_1(\vec{x}) = 0.0193x_3 - x_1 \le 0$$

$$g_2(\vec{x}) = 0.00954x_3 - x_2 \le 0$$

$$g_3(\vec{x}) = 1{,}296{,}000 - \pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 \le 0$$

$$g_4(\vec{x}) = x_4 - 240 \le 0$$

and

$$0.0625 \le x_1 \le 6.1875, \quad 0.0625 \le x_2 \le 6.1875$$

$$10 \le x_3 \le 200, \quad 10 \le x_4 \le 200$$

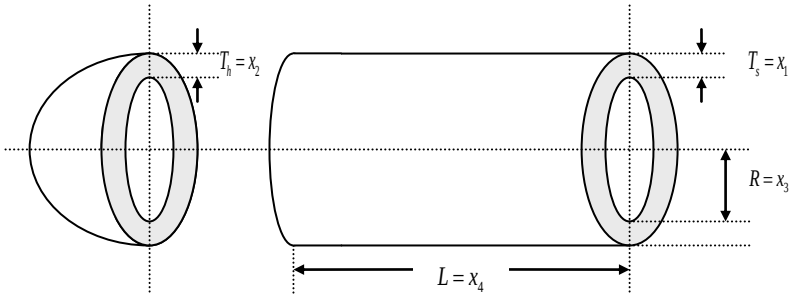


Fig.4.5 Pressure vessel design problem.

This problem has been solved by previous researchers as a mixed integer problem. It was tackled by Deb [44] using evolutionary algorithm called GeneAS and the best solution he found was *6410.38*. It was also dealt with by Coello [40] using GA with dominance-based tournament selection to handle the constraints. He reported an optimal objective value of *6059.94*. The best result was obtained by He et al. [19] by applying improved PSO, and it was *6059.71*. The present algorithm has found a slightly better solution of *6059.69* when the number of particles was set to *30*. The total number of function evaluations performed was *30,000*, while the mean value for all the runs was *6174.13* with a standard deviation of *112.81*. The convergence behaviour of the HCP algorithm has been shown in Fig. 4.6, where the search process performed in the space of the integer and continuous variables.

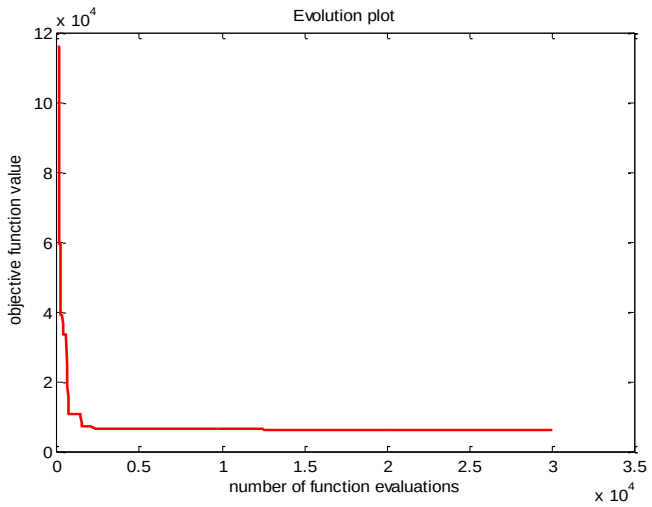


Fig. 4.6 Evolution plot of pressure vessel design.

D. *Welded Beam Design*

This problem was first investigated by Ragsdell and Phillips [45]. The objective is to find the minimum cost design of a structural welded beam design, with seven linear and nonlinear constraints on (g_1) shear stress (τ), (g_2) bending stress in the beam (σ), (g_3, g_4, g_5) side constraints, (g_6) end deflection of the beam (δ), and (g_7) buckling load on the bar (P_c). As shown in Fig.4.7, there are four design variables $h(x_1), l(x_2), t(x_3), \text{and } b(x_4)$. The problem can be formulated as follows:

$$
\begin{aligned}
&\textit{Min} \quad f(\vec{x}) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14.0 + x_2) \\
&\textit{s.t} \quad g_1(\vec{x}) = \tau(x) - \tau_{\max} \le 0 \\
&\qquad g_2(\vec{x}) = \sigma(x) - \sigma_{\max} \le 0 \\
&\qquad g_3(\vec{x}) = x_1 - x_4 \le 0 \\
&\qquad g_4(\vec{x}) = 0.10471 x_1^2 + 0.04811 x_3 x_4 (14.0 + x_2) - 5.0 \le 0 \\
&\qquad g_5(\vec{x}) = 0.125 - x_1 \le 0 \\
&\qquad g_6(\vec{x}) = \delta(x) - \delta_{\max} \le 0 \\
&\qquad g_7(\vec{x}) = P - P_c(x) \le 0 \\
&\textit{and} \\
&\quad 0.1 \le x_1 \le 2.0, \quad 0.1 \le x_2 \le 10.0, \quad 0.1 \le x_3 \le 10.0, \quad 0.1 \le x_4 \le 2.0
\end{aligned}
$$

where

$$\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}$$

$$\tau' = \frac{P}{\sqrt{2}x_1x_2} \quad , \quad \tau'' = \frac{MR}{J} \quad , \quad M = P\left(L + \frac{x_2}{2}\right)$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}$$

$$J = 2\left\{\sqrt{2}x_1x_2\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$$

$$\sigma(\vec{x}) = \frac{6PL}{x_4x_3^2} \quad , \quad \delta(\vec{x}) = \frac{4PL^3}{Ex_3^3x_4}$$

$$P_c(\vec{x}) = \frac{4.013\sqrt{EG(x_3^2x_4^6/36)}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$$

The numerical parameters for the model are chosen as: $P = 6000lb,\ L = 14in.,\ E = 30\times10^6\ psi,$

$G = 12\times10^6\ psi,\ \tau_{\max} = 13600psi,\ \sigma_{\max} = 30000psi,\ \delta_{\max} = 0.25in.$.

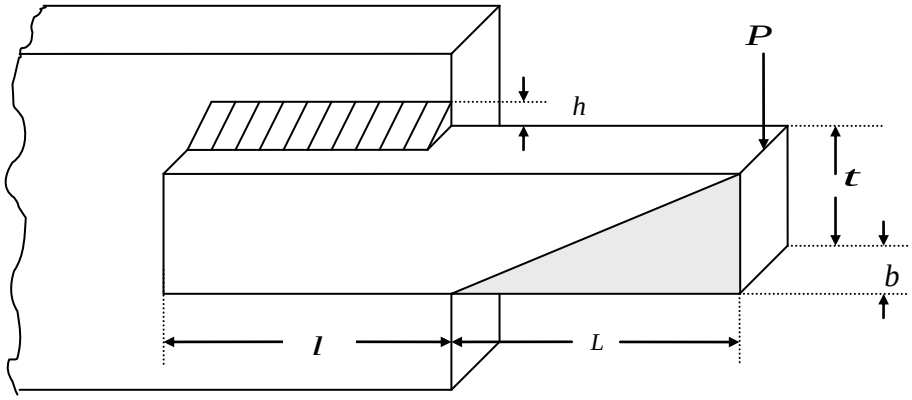


Fig.4.7 Welded beam design problem.

Many researchers have tried to solve this problem using different techniques. Ragsdell and Phillips [45] have tackled the problem using geometric programming, and the best solution they obtained was *2.3859*. Ray and Liew [46] used a society and civilization algorithm to deal with this problem. They reported a best result of *2.3854*. The best known results were obtained by He et al. [19] using an improved PSO, where the optimal solution was *2.3809*. It has been found that the HCP algorithm obtain an optimal objective function value of *2.3809* when using a population size of *25*. It can be noted that the HCP found

the same global optimum as He et al. However, it is worth mentioning that the number of fitness function evaluations of He et al. was *30,000*, while only *6,200* function calls are needed to achieve the optimum solution by using the present algorithm. The mean value for the *100* runs performed was *2.382*, with a standard deviation *0.00628*. Fig. 4.8 shows a plot of the performance of the HCP algorithm.

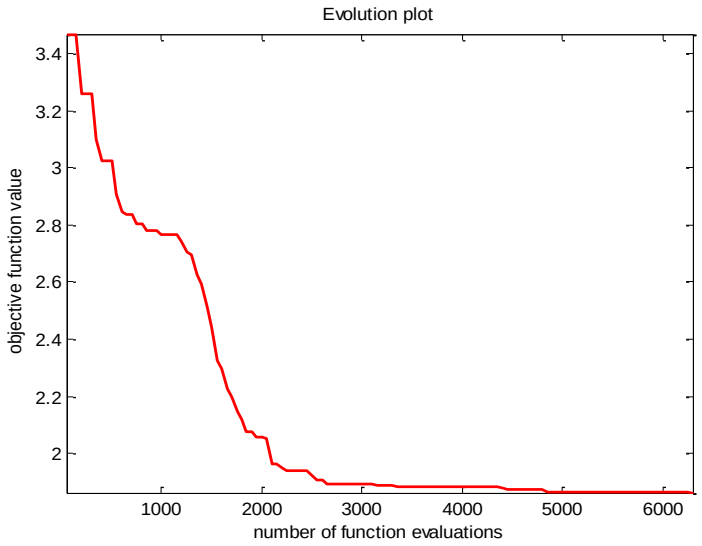


Fig.4.8 Evolution plot of welded beam design problem.

E. *10-Bar Planar Truss Structure*

The geometry of 10-bar truss structure is show in Fig. 4.9. This optimisation problem has been studied by many researchers such as Haftka [47] and Achtziger [48]. The objective of this problem is to minimise the weight of the structure. The variables in this case were the cross sectional areas of each member. There are *10* design variables and the minimum cross-sectional area of each member is *0.1* in^2. The problem contains two bays, each of *360* inches in length as well as height. There are two loads of P *=100* kip located at nodes 2 and 4, respectively. The members are subject to stress limitations of *±25* ksi, but for the ninth member, those limits are modified to *±75* ksi. The material mass density is set as *0.1*lbm/in^3. The entire cross sectional areas are design variables and can range from *0.1* to *10.0* in^2. All nodes in both directions are subject to a displacement limitation of *±2.0* in.

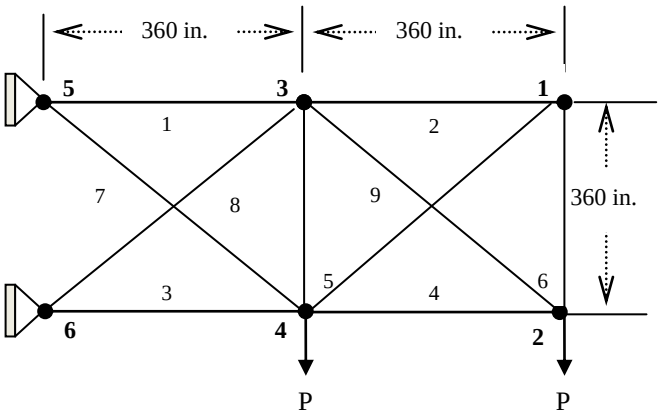


Fig.4.9 The 10-Bar Planar Truss Structure.

Due to the nature of the problem, it was determined to use a finite element analysis of the truss for optimisation. The proposed coevolutionary algorithm procedure was developed for the optimisation of planar truss. This problem has been investigated by previous researchers to minimise the weight of the structure. It was solved by Simulated Annealing Technique by Haftka et al. [47], and the best optimal solution he obtained was *1.5283* lb. The same problem was also solved by using Sequential Quadratic Programming, and the minimum weight was *1.497* lb. The present algorithm has found an optimal solution of *1.514* lb when the number of particles was set to *30*. Table 4.3 presents the results obtained for a continuous variable problem using different optimisation methods.

Quantity	HCP	SA	SQP
x_1	7.51	7.37	7.09
x_2	0.46	0.62	0.10
x_3	8.43	8.61	8.10
x_4	3.54	3.38	3.90
x_5	0.10	0.10	0.10
x_6	0.46	0.10	0.10
x_7	6.28	6.52	5.80
x_8	5.00	4.77	5.51
x_9	3.35	3.18	3.68
x_{10}	0.64	0.87	0.14
$f(\vec{x}^*)$	1,514	1,528	1,497

Table 4.3. Comparison of the results for the minimisation of 10-bar truss.

In this problem of truss structure, the HCP algorithm achieved the good solution after *800* iterations. It has been found that the proposed approach is quite efficient and capable of finding lighter and reasonable structural designs in most cases with different parameter settings. As seen in table 4.3, The results from SQP showed better accuracy than the present algorithm for this problem, while HCP was better than the SA method.

4.5 Analysis of HCP

This article presented a method of swarm evolution under a cooperative framework. Simulation results based on well-known constrained engineering design problems suggest that the HCP algorithm works well in different cases and is capable of locating the global optimum for all the problems in a reliable manner. Moreover, the HCP obtains some better solutions than those previously reported in the literature. The main reason is that, each group is of much smaller scale than the original problem and can be solved in less effort with more accuracy than the original problem. Additionally, in each sub-problem, the gradient search becomes effective when the solution region is found and local search with fast convergence is needed. This is an important advantage of the proposed algorithm over previous coevolutionary algorithms since the the hybridisation phase provides a good opportunity to escape from the local solution so that the algorithm has more chances to get a better solution with less computational cost.

The HCP algorithm does not need additional fitness evaluation for each sub-problem. Instead, the individuals from the first group interact with individuals from the second through a common fitness evaluation based on coevolutionary matrix game. It should also be noted that, for each variables vector $\vec{x}^*$, the cost function $f(\vec{x}^*)$ and the constraints $g(\vec{x}^*)$, $h(\vec{x}^*)$ have been calculated *i* times in total, where $i = 1,2,\ldots,U$, then the augmented function $\phi(\vec{x},\vec{\lambda},\vec{\mu},\vec{r})$ can be calculated for all Lagrangian multipliers by simple calculation according to Eq.(4.7). So, there is no need to evaluate $f(\vec{x}^*)$, $g(\vec{x}^*)$, and $h(\vec{x}^*)$ again

as long as the variables solution vector $\vec{x}^*$ is the same. Therefore, the computational cost of the proposed approach can be compared to other methods based on evolution of a single group.

With some numerical experiments, it has been found that, the sizes of the population could be chosen based on the complexity of the problem. For instant, the welded beam design problem contains a highly nonlinear constrains, which required a large number of individuals in each population group, while it was much easier to tackle other constrained problem (G1-G13) presented in the literature which can be found in [49]. For these problems, we observe that with average population size of *10*, the HCP algorithm was able to achieve the global optimum solution. However, this is not the case when dealing with difficult optimisation problems considered in this paper, whereas the evolutionary process is unlikely to come up with exact best response with such low population size. Hence, increasing the population size increases the convergence ability of the proposed algorithm.

In order to further assess the benefits from the proposed algorithm, the first problem is considered here for the purpose of comparing with other evolutionary algorithms published in the literature such as improved PSO [19], and coevolutionary PSO using Gaussian distribution [29]. When the iteration number is *3000* and swarm size is *30* for *100* executions, the proportion of PSO runs converged to global optimum is *78%,* while PSO-GD has *54%,* when the maximum iteration number is *2000* with swarm size *30*. When applying the proposed method to this problem, the convergence rate is found to be better than both algorithms. The proportion of HCP runs converged to global optimum is *81%.* Moreover, the total number of function evaluations was reduced by approximately *86%* compared to the standard PSO method. The performance behaviour of these three algorithms is shown in Fig.4.10. It can be seen that, the HCP converges more quickly with better solution than other algorithms.

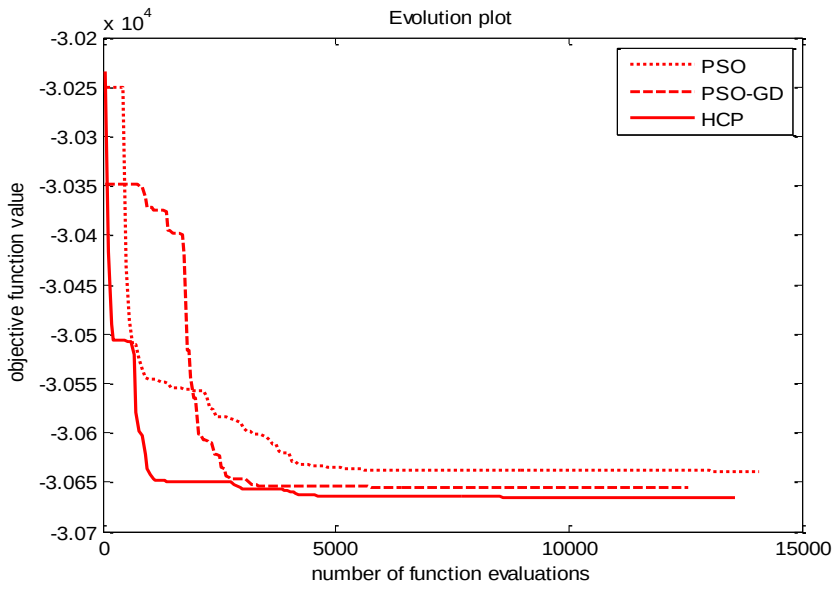


Fig.4.10 Search process comparison for Himmelblau's function.

4.6 Summary

In this chapter, we have developed and investigated a novel coevolutionary method whereas several test cases have been chosen to reflect the efficiency of our framework in dealing with variety of real-world optimisation problems. The HCP algorithm is developed to solve constrained optimisation problems formulated as min-max problems. Through a coevolutionary game approach, we exploit the success of HCP in processing non-linear and non-convex problems.

We have found that the hybridisation phase used during the evolutionary process of each sub-population is very efficient in increasing the convergence rate of the algorithm. The HCP is also very suitable for parallel computation that decreases the run time required for achieving the optimum solution. It also has been proved that, this method allows the use of different evolutionary methods for optimising any set of variables. Moreover, experiments of the developed algorithm show that it outperforms other methods presented in the literature in terms of both accuracy and computational efficiency.

Chapter 5

An Alternating Optimisation Approach for Mixed Discrete Non Linear Programming

5.1 Introduction

This chapter contributes to the development of the field of Alternating Optimisation (AO) and general Mixed Discrete Non-Linear Programming (MDNLP) by introducing a new decomposition algorithm (AO-MDNLP) based on the Augmented Lagrangian Multipliers method. In the proposed algorithm, an iterative solution strategy is proposed by transforming the constrained MDNLP problem into two unconstrained components or units; one solving for the discrete variables, and another for the continuous ones. Each unit focuses on minimizing a different set of variables while the other type is frozen. During optimising each unit, the penalty parameters and multipliers are consecutively updated until the solution moves towards the feasible region. The two units take turns in evolving independently for a small number of cycles. The validity, robustness and effectiveness of the proposed algorithm are exemplified through some well known benchmark mixed discrete optimisation problems.

5.2 General Background

This work addresses the mixed discrete programming problem, which seeks a global optimum to an optimisation formulation with an objective function subject to a set of linear and nonlinear constraints where the decision variables are both continuous and discrete. In the last decade, there has been a

dramatic increase in the techniques developed to solve MDNLP problems [14, 19, 34]; such techniques have been applied in various domains, ranging from the process industry and engineering, to the financial and management sciences as well as operational research sectors. The challenging difficulty of MDNLP problems is their high nonlinearity and non-differentiability due to the combinatorial nature of the associated discrete-valued variables.

The categories of algorithms for solving MDNLPs can be mainly divided into stochastic and deterministic ones. The stochastic methods are employing randomised searches and aim to tackle the problem of local optimality. Examples include Simulated Annealing [8], Genetic Algorithms [34, 43], Differential Evolution [50], Particle Swarm Optimisation [19, 51], and other hybrid methods [20, 52, 53]. The deterministic ones take a different approach and adopt a systematic way of approaching the optimum; popular examples include the Non-Linear Branch-and-Bound [6, 14], Sequential Linearization [54, 55], the Penalty Function approach [43,56], and the Lagrangian Relaxation methods [57, 58].

This article proposes an original method for solving MDNLP problems, based on the generic framework of Alternating Optimisation (AO) introduced by Bezdek and Hathaway [59]. AO is a very efficient iterative procedure for solving large problems by alternating between restricted subsets of variables. It has good convergence properties, reduced development times and the ability to reduce the risk of getting trapped in a local minima. Its main drawback, however, is that AO cannot be adapted easily for use with constrained optimisation problems. In this article, the AO procedure was applied to the constrained formulation of MDNLP by partitioning and processing each discrete and continuous subset of the mixed decision variables with different, and more suitable to each subset, solvers. The solvers combine a standard Quasi-Newton gradient-based method [2, 3], with a Lagrangian formulation of the MDNLP, together with a Branch-and-Bound search [6, 60] for the continuous and discrete variables, respectively.

5.3 Employed Optimisation Models and Algorithms

5.3.1 The MDNLP formulation

An MDNLP optimisation problem contains continuous, integer and discrete variables, with linear and nonlinear constraints, and also constraints on the value sets of the discrete variables. It can be stated as

$$\min_{x,y} \quad f(\vec{x},\vec{y})$$
$$\text{s.t.} \quad \begin{cases} g_i(\vec{x},\vec{y}) \le 0, & i=1,\ldots,m \\ h_j(\vec{x},\vec{y}) = 0, & j=1,\ldots,l \\ \vec{x} \in X \subseteq \Re^{n_c} \\ \vec{y} \in Y \equiv Y_1 \times \ldots \times Y_{n_d} \end{cases} \tag{5.1}$$

where $\vec{x}$ is the vector of n_c continuous variables and $\vec{y}$ is the vector of n_d discrete variables within the value set *X*. The problem also accounts for m inequalities $g(\vec{x},\vec{y})$ and l equalities $h(\vec{x},\vec{y})$. Y_k is the discrete value set of each k^{th} discrete variable y_k. The major difficulty that arises in the MDNLP problems is due to the combinatorial nature of the *Y* variables, as the number of possible solutions rises exponentially with the discrete variables domain. Therefore, complexity analysis characterises MDNLP problems as Non-Polynomial Complete [61].

5.3.2 The augmented Lagrangian multipliers method

An effective way for solving a continuous optimisation problem with constraints using solvers for unconstrained problems, is to convert it to an equivalent unconstrained one by using the penalty approach, where all constraints $g(\vec{x})$ and $h(\vec{x})$ are converted to extra penalty terms added to the objective function $f(\vec{x})$. Such an advanced penalty method is the Augmented Lagrangian Penalty Function (ALPF) [3, 62] which combines the properties of the quadratic penalty function and the Lagrangian formulation of the problem. The ALPF can be expressed as

$$\phi(\vec{x},\vec{\lambda},\vec{r}) = f(\vec{x}^k) + \sum_{i=1}^{m} \beta_i(\vec{x}^k) \cdot \left[\lambda_i^k + r_i^k \beta_i(\vec{x}^k)\right] + \sum_{j=1}^{l} h_j(\vec{x}^k) \cdot \left[\lambda_{m+j}^k + r_{m+j}^k h_j(\vec{x}^k)\right] \tag{5.2}$$

where $\{\lambda_1,\ldots\lambda_m\}$ and $\{\lambda_{m+1},\ldots\lambda_{m+l}\}$ are the Lagrangian multipliers for the m inequalities and the l equalities, respectively. The r_i represent the positive penalty terms for the corresponding two types of

constraints. $\beta_i(\vec{x})$ is a used to convert the inequalities to equalities via setting

$$\beta_i(\vec{x}) = \max\left\{ g_i(\vec{x}^k), -\frac{\lambda_i^k}{2r_i^k} \right\} \tag{5.3}$$

The optimum solution $\vec{x}^*$ is computed as a sequence of iterative unconstrained subproblems with regular updates of the penalties r_i^k and the multipliers λ_i^k at each iteration *k*. The optimisation is initialized with the values of $\lambda_i^0 = \lambda_{i+m}^0 = 0$ and $r_i^0 = r_{i+m}^0 = 1$ as suggested by Rao [2]. Because the correct penalty factors and the Lagrangian multipliers are problem dependent and, thus, unknown, they are continually updated as

$$\begin{aligned} \lambda_i^{k+1} &= \lambda_i^k + 2 \cdot r_i^k \cdot \beta_i(\vec{x}^k), && \forall\ i = 1, \ldots, m \\ \lambda_{m+j}^{k+1} &= \lambda_{m+j}^k + 2 \cdot r_{m+j}^k \cdot h_j(\vec{x}^k), && \forall\ j = 1, \ldots, l \end{aligned} \tag{5.4}$$

To make the procedure more efficient, instead of fixing the penalties r_i, an adaptation strategy [63] has been used to regulate the penalty decrease/increase. For instance, if a current point $\vec{x}^k$ violates the i^{th} inequality constraint $g_i(\vec{x}^k)$, r_i^k must be increased to eventually move the final solution to the feasible region. The following heuristic is used to update the penalty parameters

$$r_i^{k+1} = \begin{cases} \frac{6}{5} r_i^k & \text{if } g_i(\vec{x}) > \varepsilon \\ \frac{4}{5} r_i^k & \text{if } g_i(\vec{x}) < \varepsilon \\ r_i^k & \text{otherwise} \end{cases} \qquad \forall i = 1, \ldots, m \tag{5.5}$$

where ε is the user-defined tolerance for acceptable constraint violations. The same rule applies to updating the equality penalty r_{m+j}^{k+1}, based on the violation condition $|h_j(\vec{x})| > \varepsilon$.

The following termination criterion has been used to examine how close the search approaches to the optimum solution. Firstly, the solution is obtained when the relative error between the augmented function in two successive iterations becomes small, according to

$$\left| \frac{\phi(\vec{x}^{k+1}, \vec{\lambda}^{k+1}, \vec{r}^{k+1}) - \phi(\vec{x}^k, \vec{\lambda}^k, \vec{r}^k)}{\phi(\vec{x}^{k+1}, \vec{\lambda}^{k+1}, \vec{r}^{k+1})} \right| \leq \varepsilon \tag{5.6}$$

Since ϕ becomes a nonconvex function, it is important to check for optimality of the obtained solution $\vec{x}^*$, as this is the case when the corresponding multiplier vector $\vec{\lambda}^k$ approaches the optimal one $\vec{\lambda}^*$ [35]. The algorithm was terminated if the current feasible point $\vec{x}^*$ satisfies the Karush-Kuhn-Trucker (KKT) conditions which are necessary for $\vec{x}^*$ to be a global optimum of ϕ

$$\begin{aligned} &\nabla f\left(\vec{x}^*\right)+\sum_{i=1}^{m}\lambda_i\nabla g_i\left(\vec{x}^*\right)+\sum_{j=1}^{l}\lambda_{m+j}\nabla h_j\left(\vec{x}^*\right)\approx\varepsilon \\ &\lambda_i\geq 0,\quad \lambda_i\cdot g_i\left(\vec{x}^*\right)=0,\quad \forall i=1,\ldots,m \end{aligned} \tag{5.7}$$

5.3.3 Unconstrained optimisation

After a continuous constrained problem is transformed to a continuous unconstrained one through the ALPF, a standard Quasi-Newton (QN) algorithm [62] can be employed to efficiently minimise it. Second-order gradient-based algorithms proceed towards the minimum point of a minimizing function $f\left(\vec{x}\right)$ in a sequential manner by updating the current solution in each *(k+1)*th iteration as

$$\vec{x}^{k+1}=\vec{x}^k-H\left(\vec{x}^k\right)^{-1}\cdot\nabla f\left(\vec{x}^k\right) \tag{5.8}$$

To reduce the computational load of estimating the Hessian *H* at point $\vec{x}^k$ in each iteration, QN builds up curvature information using first-order derivatives by applying the Sherman-Morrison formula

$$\begin{aligned} &H^{k+1}=H^k+\frac{\left(y^k-H^k s^k\right)\cdot\left(y^k-H^k s^k\right)^T}{\left(y^k-H^k s^k\right)^T s^k} \\ &\text{where}\quad y^k=\nabla f\left(\vec{x}^{k+1}\right)-\nabla f\left(\vec{x}^k\right) \\ &\qquad\quad s^k=-H^k\left(\vec{x}^k\right)^{-1}\cdot\nabla f\left(\vec{x}^k\right) \end{aligned} \tag{5.9}$$

where H^0 is usually taken to be the identity matrix.

5.3.4 The Branch-and-Bound algorithm

This is an established generic algorithm for efficiently enumerating and searching parts of optimisation problems. The BB method for discrete problems [1,64] is based on the mechanisms of separation,

relaxation and fathoming in a search tree. Its principle lies in successive decompositions of the original problem to smaller disjoint subproblems until an optimal solution is found.

The algorithm starts by solving first the continuous relaxation problem using a Non-Linear Programming (NLP) solver. If all discrete variables take discrete values the search is stopped. Otherwise, a tree search is performed in the space of the discrete variables. Then the algorithm selects one of those discrete variables which take a non-discrete value and branch on it. Branching generates two new subproblems by adding simple bounds to the NLP relaxation. Then, one of the two new NLP problems is selected and solved. If the discrete variables take non-discrete values then branching is repeated, while if one of the fathoming rules is satisfied, then no branching is required, and the corresponding node is flagged as fully explored. When during the search discrete solutions are found, they can provide upper bounds on the optimal value of the original problem. Once a node has been fathomed the algorithm backtracks to another node which has not been explored until all nodes are fathomed. The general operations of the algorithm are shown in Table 5.1.

Place the continuous relaxation and set upper bound to infinity.
while there are unexamined subproblems/nodes in the tree
    Select an unexplored node.
    Solve the NLP problem on the discrete variable *y*.
    Obtain lower bound.
    if the solution is optimal and y value is fractional:
        Branch on *y*.
    endif
    Solve NLP problem until:
        - The subproblem is infeasible, or
        - A discrete feasible solution is found (record the value of this solution as upper bound), or
        - The lower bound is greater than the objective value of a previous discrete solution.
    Continue branching and solving NLP subproblems.
Endwhile

Table 5.1 Main Branch-and-Bound operations.

5.4 The Proposed AO-MDNLP Framework

5.4.1 Alternating Optimisation (AO)

AO is a generic methodology for locating the solution of an optimisation problem by partitioning and treating independently the design variables. It has been shown that the AO method very efficiently converges to at least a local minimum regardless of the initialisation [65]. The principle advantage of AO is that it replaces the optimisation of the objective function with a sequence of easier optimisations involving the different partitions of the design variables.

If we assume that we have to minimise a function $f(\vec{x})$ of n variables, the original problem can be partitioned into N autonomous subsets of variables (with n_s variables in each s^{th} subset, with $\sum_{s=1}^{N} n_s = n$) and the process of optimisation alternates between these subsets until the global problem is completed. The flowchart in Figure 1 illustrates the operation sequencing of AO, where the strikethrough notation $\not{x}_i$ indicates variables that are fixed with respect to the current subproblem at index i. In later sections, the parameter t is used to define the number of cycles to be used during the AO optimisation process.

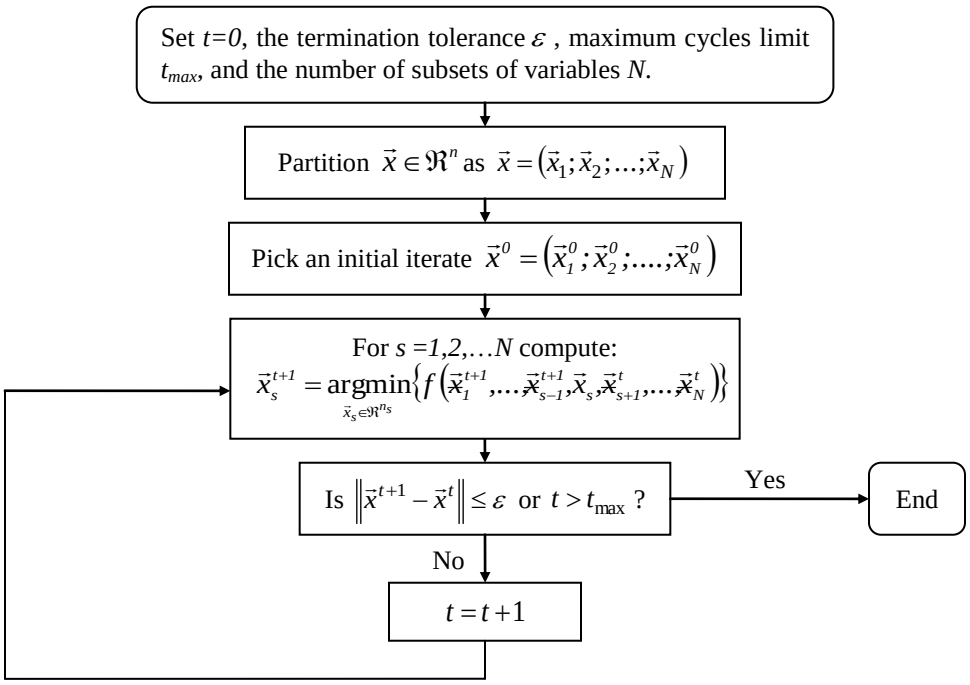


Figure 5.1 Iteration procedure of Alternating Optimisation.

Example: *Hartmann function*

A classic benchmark problem in nonlinear optimisation introduced by Dixon and Szego [66] has been used to demonstrate the application of AO, It minimises the following

$$f(\vec{x}) = -\sum_{i=1}^{4} \alpha_i \, exp\left[-\sum_{j=1}^{6} B_{ij}\left(x_j - Q_{ij}\right)^2 \right], \qquad \alpha = [1, 1.2, 3, 3.2]'$$

$$B = \begin{bmatrix} 10 & 3 & 17 & 3.05 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}, \qquad Q = 10^{-4} \times \begin{bmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{bmatrix} \tag{5.10}$$

This problem has six continuous variables $\vec{x} = (x_1, \ldots, x_6) \in \Re^6$. Suppose we choose $N=2$ arbitrary partitions $\vec{x}_1 = (x_1, x_2, x_3) \in \Re^3$, and $\vec{x}_2 = (x_4, x_5, x_6) \in \Re^3$ of $n_1=n_2=3$ variables each. The minimisation can start by setting the initialisation point to $\vec{x}^0 = (1,1,1,1,1,1)$, and then minimise $f(\vec{x})$ by alternatively minimizing each subset of the partitioned variables independently.

Cycle t	$\vec{x}_1^t$	$\vec{x}_2^t$	$\left\|\vec{x}^t - \vec{x}^{t-1}\right\|$
0	*(1, 1, 1)*	*(1, 1, 1)*	----
1	*(0.1312, 0.2005, 0.5683)*	*(0.2718, 0.3128, 0.6595)*	*1.6428*
2	*(0.2015, 0.1501, 0.4774)*	*(0.2753, 0.3117, 0.6573)*	*0.1256*
3	*(0.2017, 0.1500, 0.4769)*	*(0.2753, 0.3117, 0.6573)*	*0.0005*
4	*(0.2017, 0.1500, 0.4769)*	*(0.2753, 0.3117, 0.6573)*	*0.0000*

Table 5.2 Applying AO on the Hartmann function.

Table 5.2 shows a possible outcome of applying AO on this example. The algorithm converges quickly to the optimum solution requiring only four cycles to satisfy the stopping condition $\left\|\vec{x}^t - \vec{x}^{t-1}\right\| \le 10^{-4}$. This simple example indicates that the AO framework can provide the means for solving many large scale problems that are difficult to process by existing methods, and it leads to easier subproblems with solution spaces much more reduced than the original *n*-dimensional one.

5.4.2 The AO-MDNLP algorithm

Based on our observation that MDNLPs have highly structured constraints with mixed variables, This method proposes to partition these MDNLPs by their variables into two subproblems and solve each subproblem as in the example before but using the Lagrangian transformation and also with different and appropriately efficient subsolvers for each subset. This new architecture combines the previously discussed robust components, namely the AO framework, the ALPF model and the QN and BB algorithms. The rationale behind this variable partitioning is to allow many computationally expensive MDNLP problems to be solved by existing solvers more efficiently. This is possible because the proposed AO-MDNLP method leads to smaller and simpler structured subproblems that are easier to minimise, while the Lagrangian framework supports resolution of the violated constraints across the subproblems using an effective updating strategy.

Because the original problem in Eq. (5.1) consists of the objective function $f(\vec{x},\vec{y})$ and the constraints $g_i(\vec{x},\vec{y})$ and $h_j(\vec{x},\vec{y})$ the constraints (without assuming a specific problem structure) are always associated with both continuous and discrete variables. In order to apply AO, the problem was decomposed into two subproblems; one optimising the set of $\vec{x}$ and the other the set of $\vec{y}$ variables. To make the handling of the constraints more uniform and also efficient, the ALPF has been used to allow the continuous subproblem to be converted to an unconstrained one. Overall, the proposed method decomposes the MDNLP problem of Eq. (5.1) to two units, where an unconstrained problem is solved at each unit. Unit-*A* fixes all variables $\vec{y}$ and minimises the ALPF using QN, defined as

$$\phi(\vec{x},\vec{y},\vec{\lambda},\vec{r}) = f(\vec{x}^k,\vec{y}^k) + \sum_{i=1}^{m}\beta_i(\vec{x}^k,\vec{y}^k)\cdot\left[\lambda_i^k + r_i^k\beta_i(\vec{x}^k,\vec{y}^k)\right] + \sum_{j=1}^{l}h_j(\vec{x}^k,\vec{y}^k)\cdot\left[\lambda_{m+j}^k + r_{m+j}^k h_j(\vec{x}^k,\vec{y}^k)\right]$$

$$\beta_i(\vec{x},\vec{y}) = \max\left\{g_i(\vec{x}^k,\vec{y}^k), -\frac{\lambda_i^k}{2r_i^k}\right\} \tag{5.11}$$

After Unit-*A* performing the full minimisation of $\phi(\vec{x},\vec{y},\vec{\lambda},\vec{r})$ with respect to $\vec{x}$, some of the penalty parameters r_i^k and the Lagrangian multipliers λ_i^k are consecutively updated. The unconstrained optimisation of $\phi(\vec{x},\vec{y},\vec{\lambda},\vec{r})$ has to be carried out for a sequence of values $\vec{r}$ and $\vec{\lambda}$ until the solution moves towards the feasible region, where the Lagrangian multipliers can be estimated more accurately.

The iterative process stops when the augmented function is not changing much between two successive iterations. In the mean time, a test for the satisfaction of the KKT conditions is performed before taking the current solution as an optimum solution.

Subsequently, Unit-*B* takes turn in the optimisation process and the continuous variables $\vec{x}$ become fixed. The unit invokes a Branch-and-Bound algorithm to minimise the discrete variables $\vec{y}$ only, but instead of solving a constrained problem at each node of the BB tree, the augmented function $\phi(\vec{x},\vec{y},\vec{r},\vec{\lambda})$ is minimised for the relaxed component of $\vec{y}$ using QN. This setup is efficient, because at each node the subproblem has n_c less dimensions that the standard unpartitioned BB. The penalty parameters r_i^k and the Lagrangian multipliers λ_i^k have to be consecutively updated at each node in order to find the feasible continuous solution. When solving each subproblem in the BB tree, the following condition must be satisfied before taking the obtained point as a discrete solution

$$\max\left\| y_i^k - d_i^k \right\| \le \varepsilon \tag{5.12}$$

where y_i^k is the discrete value of the i^{th} discrete variable at the iteration k, and d_i^k is the nearest discrete value for the discrete design variable y_i^k. Once a node has been fully explored, the global search procedures of BB have to be carried out until a discrete solution has been found. The selection method for branching node may significantly affect the performance of BB. Our approach uses the depth-first with backtracking strategy (Ringertz 1988) until all the nodes have been explored.

After convergence of both units, the algorithm composes the final solution by combining the partial final

solution generated by each unit. The algorithm terminates with the current solution, if the maximum number of cycles is reached or if the following necessary stopping criteria is met

$$\left\| \left(\vec{x}^{t+1}, \vec{y}^{t+1} \right) - \left(\vec{x}^{t}, \vec{y}^{t} \right) \right\| \leq \varepsilon \tag{5.13}$$

The overall implementation of the proposed AO-MDNLP is presented in Table 5.3.

Stage 1: ***Initialisation***

Set cycle count $t = 0$, termination tolerance ε, and maximum cycles limit $t_{\max}$.

Pick an initial iterate $\left(\vec{x}^{0}, \vec{y}^{0} \right)$, and set $\left(\vec{\lambda}_i^0, \vec{\lambda}_{i+m}^0, \vec{r}_i^{\,0}, \vec{r}_{i+m}^{\,0} \right)$.

Stage2: ***Optimisation***

while ($t \leq t_{max}$)

Form $\phi(\vec{x}, \vec{y}, \vec{\lambda}, \vec{r})$ according to Eq. (5.11).

%Unit-A:

while (the termination criterion in (5.6) and (5.7) is not met)

Minimise $\phi(\vec{x}, \vec{y}, \vec{\lambda}, \vec{r})$ using QN method.

Update the parameters $\vec{\lambda}, \vec{r}$ according to Eq. (5.4) and (5.5).

end while

Record the obtained solution $(\vec{x}^{t}, \vec{y}^{t})$.

%Unit-B:

while (there are unexamined nodes in the BB tree)

Minimise $\phi(\vec{x}, \vec{y}, \vec{\lambda}, \vec{r})$ at each node on the discrete variable y.

end while

Record the obtained solution $(\vec{x}^{t}, \vec{y}^{t})$.

Stage 3: ***Convergence***

if (the necessary stopping condition in (5.13) is met)

Terminate the algorithm with $(\vec{x}^{*}, \vec{y}^{*})$ as an optimum solution.

else

Increase cycle number as *t=t+1*.

end if

end while

Table 5.3 Pseudo-Code for the AO-MDNLP algorithm.

5.5 Numerical Experimentation

In this section, the performance of the proposed AO-MDNLP was investigated with a number of difficult real-world bench problems from mechanical engineering and chemical process synthesis, frequently employed in the literature. In all experiments, the constraint tolerances $\varepsilon_{g,h} = 10^{-4}$ are used for both equality and inequality constraints. A complete implementation of the AO-MDNLP algorithm has been developed in Matlab 7.4, running on a 2.0GHz Pentium 4 CPU with 1GB of RAM.

5.5.1 Results

Experiment 1:

This is a nonconvex problem from [1], which involves a process flow sheeting problem. It has two continuous variables and one discrete variable with three linear and nonlinear inequality constraints, and is given by

$$\begin{aligned} min \quad & f(\vec{x},\vec{y}) = -0.7y_1 + 5(x_1 - 0.5)^2 + 0.8 \\ subject\,to & \begin{cases} -\exp(x_1 - 0.2) - x_2 \le 0 \\ x_2 + 1.1y_1 \le -1.0 \\ x_1 - y_1 \le 0.2 \\ 0.2 \le x_1 \le 1 \\ -2.22554 \le x_2 \le -1 \\ y_1 \in \{0,1\} \end{cases} \end{aligned} \tag{5.14}$$

The optimum results obtained by the present approach are listed in Table 5.4.

Cycle t	$(\vec{x}_A^t; \vec{y}_A^t) = (x_1, x_2, y_1)$	$(\vec{x}_B^t; \vec{y}_B^t) = (x_1, x_2, y_1)$	$f(\vec{x},\vec{y})$	$\left\|(\vec{x}^t, \vec{y}^t) - (\vec{x}^{t-1}, \vec{y}^{t-1})\right\|$
0	*(1, 1, 1)*	*(1, 1, 1)*	*1.35*	*----*
1	*(0.5944, -1.4835, 0.4395)*	*(0.5944, -1.4835, 1.00)*	*0.1446*	*2.5164*
2	*(0.9418, -2.0998, 1.00)*	*(0.9418, -2.0998, 1.00)*	*1.0758*	*0.7075*
3	*(0.9418, -2.0998, 1.00)*	*(0.9418, -2.0998, 1.00)*	*1.0758*	*0.0000*

Table 5.4 Alternating Optimisation results of experiment 1.

Experiment 2:

This problem arises in the synthesis of chemical process, and it was investigated by Duran and Grossmann [5]. The goal is to determine the optimal solution of a chemical process system. The problem

has three continuous variables and three discrete variables with six linear and nonlinear inequality constraints. The master problem can be formulated as follows

$$
\begin{aligned}
min \quad & f(\vec{x},\vec{y}) = 5y_1 + 6y_2 + 8y_3 + 10x_1 - 7x_3 - 18\times\log(x_2+1) - 19.2\times\log(x_1 - x_2 + 1) + 10 \\
subject\,to \quad & \begin{cases}
0.8\times\log(x_2+1) + 0.96\times\log(x_1 - x_2 + 1) - 0.8x_3 \geq 0 \\
\log(x_2+1) + 1.2\times\log(x_1 - x_2 + 1) - x_3 - 2y_3 \geq -2 \\
x_2 - x_1 \leq 0 \\
x_2 - 2y_1 \leq 0 \\
x_1 - x_2 - 2y_2 \leq 0 \\
y_1 + y_2 \leq 1 \\
0 \leq x_1 \leq 2\ , 0 \leq x_2 \leq 2\ ,\ 0 \leq x_3 \leq 1 \\
y_1, y_2, y_3 \in \{0,1\}
\end{cases}
\end{aligned}
\tag{5.15}
$$

The optimal solutions obtained using the AO-MDNLP algorithm, are presented in Table 5.5.

Cycle t	$(\vec{x}_A^t; \vec{y}_A^t) = (x_1, x_2, x_2, y_1, y_2, y_3)$	$(\vec{x}_B^t; \vec{y}_B^t) = (x_1, x_2, x_3, y_1, y_2, y_3)$	$f(\vec{x},\vec{y})$	$\left\|(\vec{x}^t,\vec{y}^t) - (\vec{x}^{t-1},\vec{y}^{t-1})\right\|$
0	*(1, 1, 1, 1, 1, 1)*	*(1, 1, 1, 1, 1, 1)*	*19.5234*	----
1	*(1.1542, 0.5502, 1, 0.2751, 0.3020, 0)*	*(1.1542, 0.5502, 1, 0, 1, 1)*	*11.5790*	*1.1073*
2	*(1.3, 0, 0.9995, 0, 1, 1)*	*(1.3, 0, 0.9995, 0, 1, 0)*	*6.0098*	*0.5692*
3	*(1.3, 0, 0.9995, 0, 1, 1)*	*(1.3, 0, 0.9995, 0, 1, 0)*	*6.0098*	*0.0000*

Table 5.5 Optimal design of process synthesis problem.

Experiment 3:

Consider the optimal design problem of a pressure vessel given in [43]. The objective of this problem is to minimise the total cost of materials for forming and welding of a pressure vessel. The design variables of the problem are specified as: $(\vec{x},\vec{y}) = (x_1, x_2, y_1, y_2)^T$, which correspond respectively to the sell thickness, spherical head's thickness, shell radius, and shell length, where y_1 and y_2 represent discrete values, integer multiples of 0.0625, while x_1 and x_2 are continuous variables. The mathematical formulation of the problem is

$$
\begin{aligned}
min \quad & f(\vec{x},\vec{y}) = 0.6224 y_1 x_1 x_2 + 1.7781 y_2 x_1^2 + 3.1661 y_1^2 x_2 + 19.84 x_1^2 x_1 \\
subject\,to \quad & \begin{cases}
0.0193 x_1 - y_1 \le 0 \\
0.00954 x_1 - y_2 \le 0 \\
1{,}296{,}000 - \pi x_1^2 x_2 - \frac{4}{3}\pi x_1^3 \le 0 \\
x_2 - 240 \le 0 \\
0.0625 \le y_1 \le 6.1875 \\
0.0625 \le y_2 \le 6.1875 \\
10 \le x_1 \le 200 \\
10 \le x_2 \le 200
\end{cases}
\end{aligned}
\tag{5.16}
$$

The AO cycles results of the pressure vessel design problem are shown in Table 5.6.

Cycle t	$(\vec{x}_A^t; \vec{y}_A^t) = (x_1, x_2, y_1, y_2)$	$(\vec{x}_B^t; \vec{y}_B^t) = (x_1, x_2, y_1, y_2)$	$f(\vec{x},\vec{y})$	$\left\|(\vec{x}^t, \vec{y}^t) - (\vec{x}^{t-1}, \vec{y}^{t-1})\right\|$
0	*(1, 1, 1, 1)*	*(1, 1, 1, 1)*	*25.4066*	*----*
1	*(159.84, 209.1, 2.733, 1.625)*	*(159.84, 209.1, 2.9375, 1.6250)*	*168,004.3*	*261.8*
2	*(46.19, 180.4, 2.9375, 1.6250)*	*(46.19, 180.4, 2.8750, 1.4375)*	*32,659.5*	*117.2*
3	*(42.0989, 176.6305, 2.8750, 1.4375)*	*(42.0989, 176.6305, 0.8125, 0.4375)*	*6,059.65*	*6.01*
4	*(42.0989, 176.6305 , 0.8125, 0.4375)*	*(42.0989, 176.6305, 0.8125, 0.4375)*	*6,059.65*	*0.000*

Table 5.6 Optimal design of the pressure vessel.

Experiment 4:

This problem was studied by Duran and Grossmann [5]. It has more continuous and discrete variables; there are 32 possible combinations of the 5 binary variables, of which 11 are feasible as determined by the linear inequality constraint. There are 3 nonlinear inequality constraints and one linear equality constraints. The problem formulation is given below

$$
\begin{aligned}
\min \quad & f(\vec{x},\vec{y}) = 5y_1 + 8y_2 + 6y_3 + 10y_4 + 6y_5 - 10x_1 - 15x_2 - 15x_3 + 15x_4 + 5x_5 - 20x_6 \\
& + \exp(x_1) + \exp(0.83333x_2) - 60 \times \log(x_4 + x_5 + 1) + 140 \\
\text{subject to} \quad & \begin{cases}
-\log(x_4 + x_5 + 1) \le 0 \\
\exp(x_1) - 10y_1 \le 1 \\
\exp(0.83333x_2) - 10y_2 \le 1 \\
1.25x_3 - 10y_3 \le 0 \\
x_4 + x_5 - 10y_4 \le 0 \\
-2x_3 + 2x_6 - 10y_5 \le 0 \\
-x_1 - x_2 - 2x_3 + x_4 + 2x_6 \le 0 \\
-x_1 - x_2 - 0.75x_3 + x_4 + 2x_6 \le 0 \\
x_3 - x_6 \le 0 \\
2x_3 - x_4 - 2x_6 \le 0 \\
-0.5x_4 + x_5 \le 0 \\
-0.2x_4 - x_5 \le 0 \\
y_1 + y_2 = 1 \\
y_4 + x_5 \le 1 \\
0 \le x_1 \le 2 \ , 0 \le x_2 \le 2 \ , \ 0 \le x_3 \\
0 \le x_4 \ , \ 0 \le x_5 \le 2 \ , \ 0 \le x_6 \le 3 \\
y_1, y_2, y_3, y_4, y_5 \in \{0,1\}
\end{cases}
\end{aligned}
\tag{5.17}
$$

This example shows that the global solution can be obtained by the algorithm as shown in Table 5.7.

Cycle t	$(\vec{x}_A^t; \vec{y}_A^t) = (x_1, x_2, x_3, x_4, x_5, x_6, y_1, y_2, y_3, y_4, y_5)$	$(\vec{x}_B^t; \vec{y}_B^t) = (x_1, x_2, x_3, x_4, x_5, x_6, y_1, y_2, y_3, y_4, y_5)$	$f(\vec{x},\vec{y})$	$\left\|(\vec{x}^t, \vec{y}^t) - (\vec{x}^{t-1}, \vec{y}^{t-1})\right\|$
0	*(1,1,1,1,1,1,1,1,1,1,1)*	*(1,1,1,1,1,1,1,1,1,1,1)*	*74.1025*	----
1	*(1.904, 1.9995, 2.6218, 0.6264, 0.3132, 2.6217, 0.5713, 0.4292, 0.3277, 0.094,0)*	*(1.904, 1.9995, 2.6218, 0.6264, 0.3132, 2.6217, 1,1,0,1,0)*	*6.4246*	*3.1125*
2	*(1.999, 2.121, 0, 2.761, 1.381, 0, 1, 1, 0, 1, 0)*	*(1.999, 2.121, 0, 2.761, 1.381, 0, 0, 1, 1, 1, 0)*	*73.5040*	*4.4122*
3	*(0, 2, 1.0784, 0.652, 0.326, 1.0784, 0, 1, 1, 1, 0)*	*(0, 2, 1.0784, 0.652, 0.326, 1.0784, 0, 1, 1, 1, 0)*	*73.0353*	*3.4498*
4	*(0, 2, 1.0784, 0.652, 0.326, 1.0784, 0, 1, 1, 1, 0)*	*(0, 2, 1.0784, 0.652, 0.326, 1.0784, 0, 1, 1, 1, 0)*	*73.0353*	*0.0000*

Table 5.7 Alternating optimisation of process synthesis problem.

Experiment 5:

This problem was investigated by Kocis and Grossmann [16], and Costa [17]. It tackles the optimal design of multi-product batch plant with M serial processing stages, where fixed amounts Q_i from N

products must be produced. This problem contains a large number of nonlinear inequality constraints; it also has 22 continuous variables and 24 discrete variables. The master problem formulation can be stated as:

$$
\begin{aligned}
min \quad & f = \sum_{j=1}^{M} \alpha N_j V_j^B \\
subject\ to & \begin{cases}
\sum_{i=1}^{N} \frac{Q_i T_{Li}}{B_i} \le H \\
V_j \ge S_{ij} B_i \\
N_J T_{Li} \ge t_{ij} \\
1 \le N_j \le N_j^u \\
V_j^l \le V_j \le V_j^u \\
T_{Li}^l \le T_{Li} \le T_{Li}^u \\
B_j^l \le B_j \le B_j^u
\end{cases}
\end{aligned}
\tag{5.18}
$$

where, the numerical parameters for the model are chosen as: $M = 6$, $N = 5$, $H = 6000$, $\alpha_j = 250$, $\beta_j = 0.6$, $N_j^u = 3$, $V_j^l = 300$, and $V_j^u = 3000$. The values of $T_{Li}^l, T_{Li}^u, B_j^l$, and B_j^u are given by

$$
T_{Li}^l = \max \frac{t_{ij}}{N_j^u}, \quad T_{Li}^u = \max t_{ij}, \quad B_j^l = \frac{Q_i^* T_{Li}}{H}, \quad B_j^u = \min\left(Q_i \min_j V_j^u / S_{ij} \right)
$$

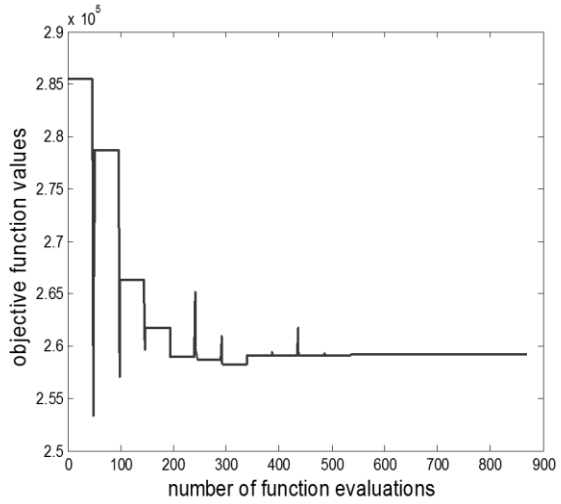
$$
where,\ S_{ij} = \begin{bmatrix}
7.9 & 2.0 & 5.2 & 4.9 & 6.1 & 4.2 \\
0.7 & 0.8 & 0.9 & 3.4 & 2.1 & 2.5 \\
0.7 & 2.6 & 1.6 & 3.6 & 3.2 & 2.9 \\
4.7 & 2.3 & 1.6 & 2.7 & 1.2 & 2.5 \\
1.2 & 3.6 & 2.4 & 4.5 & 1.6 & 2.1
\end{bmatrix}_{N \times M}
\quad ,and\ t_{ij} = \begin{bmatrix}
6.4 & 4.7 & 8.3 & 3.9 & 2.1 & 1.2 \\
6.8 & 6.4 & 6.5 & 4.4 & 2.3 & 3.2 \\
1.0 & 6.3 & 5.4 & 11.9 & 5.7 & 6.2 \\
3.2 & 3.0 & 3.5 & 3.3 & 2.8 & 3.4 \\
2.1 & 2.5 & 4.2 & 3.6 & 3.7 & 2.2
\end{bmatrix}_{N \times M}
\tag{5.19}
$$

Table 5.8 summarizes the optimal results of the batch plant problem, where the optimal solution has been found in four AO cycles with an optimal objective function value of 2.8551×10^5.

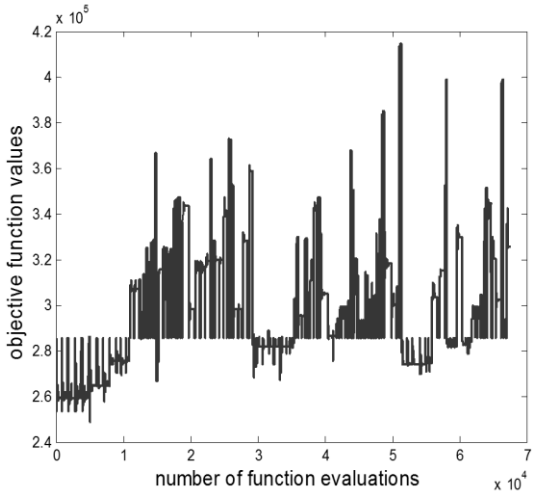
The optimal solution (x^*, y^*)
$(\vec{x}, \vec{y}) = [5.9395, 6.6468, 6.5896, 6.4588, 6.2642, 1.1632, 1.2238, 1.8245, 1.2238, 1.3083,$ $0.6931, 0.6931, 1.0986, 0.6931, 0, 0, 8.0064, 7.5452, 7.5882, 7.8706, 7.7528, 7.6544,$ $0, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]_{1 \times 46}$

Table 5.8 Optimal design of batch plant problem.

Note that, not all the AO iterations results for the batch plan problem were included because of its large size. However, the convergence behaviour of the proposed algorithm has been shown in Figure 5.2, where the AO search process performed in the space of the discrete and continuous variables. The convergence graph Figure 2(a) shows the decrease of the objective function in Unit-*A* while performing the full minimisation of $\phi\left(\vec{x},\vec{y},\vec{\lambda},\vec{r}\right)$ with respect to $\vec{x}$. Figure 5.2(b) shows the development of the objective function in Unit-B when applying the BB method to minimise $\phi\left(\vec{x},\vec{y},\vec{\lambda},\vec{r}\right)$ with respect to $\vec{y}$.



(a)Unit 1 evaluation plot (QN method)



(b)Unit 2 evaluation plot (BB method)

Figure 5.2 Objective function evaluations during the AO process.

5.5.2 Discussion of results

The performance of the AO-MDNLP algorithm is investigated using five MDNLP problems. Problems 1 to 4 are considered here for the purpose of comparing with the improved Particle Swarm Optimisation (PSO) introduced by He *et al.* [19] and a Hybrid Genetic Algorithm (MDHGA) proposed by Rao and Xiong [67] which are the state of the art methods for solving MDNLP problems. Problem 3 has been chosen to evaluate the efficiency of the proposed algorithm against other stochastic methods presented in the literature. Problems 5 can be considered as more complex global optimisation problems, where AO-MDNLP algorithm is needed to find the optimal values of the discrete and continuous variables. Table 5.9 summarizes all the obtained results using the proposed approach.

Experiment index	No. of continuous variables	No. of discrete variables	No. of Constraints	No. of AO cycles	Optimal objective function value
1	2	1	3	3	1.0758
2	3	3	6	3	6.0098
3	2	2	4	4	6,059.65
4	6	5	14	4	73.0353
5	22	24	73	4	285,506.5

Table 5.9 Experimental results of the AO-MDNLP algorithm.

It is important to note that, when the problem is partitioned by its variables, each subproblem is of much smaller scale than the original problem and can be solved in less time with more accuracy than the original problem. This can be exploited in a multi-processor architecture with relaxed synchronisation between the units to enable faster execution. The proposed AO-MDNLP algorithm for handling mixed-variables is found to work efficiently because of using an appropriate solver for optimising each different type of variables.

For Experiment 1, the optimal solution is *1.0758* which agrees with Floudas [1]. In Experiment 2, the AO-MDNLP converges to the optimum solution after only three cycles; the optimal objective function value of *6.0098* is similar to the best known results reported by Duran and Grossmann [3].

Experiment	*MDHGA algorithm*		*PSO algorithm*		*AO-MDNLP algorithm*	
	$f^*(\vec{x},\vec{y})$	Function evaluations	$f^*(\vec{x},\vec{y})$	Function evaluations	$f^*(\vec{x},\vec{y})$	Function evaluations
1	1.077	1,221	1.076	1,802	1.0758	690
2	6.15	10,352	6.01	11,589	6.0098	5880
3	7284.02	26,459	6,059.71	28,187	6,059.65	9,765
4	73.124	25,616	73.0468	26,432	73.0353	17,226

Table 5.10 Comparison of the proposed algorithm performance with PSO and MDHGA.

As shown in Table 5.10, a comparison is made to evaluate the performance of our approach with the popular PSO and a hybrid GA in terms of both solution accuracy and computational cost. The AO-MDNLP algorithm slightly outperformed both algorithms in terms of solution accuracy. However, the

proposed approach provides much better performance in terms of computational cost, as it requires significantly fewer function evaluations to solve each problem.

In order to further assess the efficiency of the proposed algorithm, its results have been compared with those published in the literature such as Evolutionary Algorithm (EA) [68], Evolutionary Programming method (EP) [69], and Genetic Algorithm (GA) [70]. As shown in Table 5.11 for Experiment 3, the optimum value of the objective function is only found to be slightly better than that of the best known solution found by He *et al.* [19], but with a significant improvement in the number of function evaluations. The number of function evaluations needed is *28,187* in He *et al.*, while in our algorithm the total number of function evaluations required to converge is *9,765.*

Quantity	***MDHGA***	***EP***	***EA***	***GA***	***PSO***	**AO-MDNLP**
x_1	1.1875	1.000	0.9345	0.8125	0.8125	0.8125
x_2	0.625	0.625	0.5000	0.4375	0.4375	0.4375
x_3	61.4483	51.1958	48.3290	40.097398	42.0984456	42.0989
x_4	27.4037	90.7821	112.6790	176.65404	176.636595	176.6305
g_1	-0.0015	-0.0119	-0.00475	-0.00002	0.00000	0.00000
g_2	-0.0388	-0.1366	-0.038941	-0.035891	-0.0358808	-0.0358
g_3	-963.9357	-13584.5631	-3652.876	-27.886075	0.00000	0.00000
g_4	-212.5963	-149.2179	-127.321	-63.345953	-63.363404	-63.6948
$f(\vec{x},\vec{y})$	**7,284.02**	**7,108.6160**	**6,410.381**	**6,059.946**	**6,059.714**	**6,059.654**

Table 5.11 Optimal solution of pressure vessel design problem (Experiment 3).

In Experiment 5, the batch plant problem with larger size is used to illustrate the efficiency of the algorithm. It was able to find the optimum solution in only four AO cycles. The optimal objective function obtained was *285,506.5*, with *104,319* function calls. The computational time increases for this problem, but optimal solutions are provided at the end, where other algorithms such as outer approximation method [5] fails to give any results as soon as the problem size begins to be larger. Overall, our experiments show that one advantage of the proposed approach is that it is more likely to

find the global optimum solution, where it is difficult to achieve in practice. Furthermore, it can cope with problems that involve different search spaces, and makes it possible to solve large-scale optimisation problems that may otherwise be computationally difficult and cause the algorithm to fail.

5.5. Summary

In order to improve upon existing optimisation methods, this Chapter examines the idea of modifying traditional alternating optimisation by introducing an algorithm for solving MDNLP problems. A decomposition technique has been discussed and some computationally efficient procedures have been presented. The key to this technique is an augmented Lagrangian function which preserves separability without violating or using explicit constraints. The proposed approach shows robustness in a diverse range of problems and that it can be beneficial for cases where the problem has many strongly interacting variables. It should be noted that, this technique allows the use of any method for optimising each set of variables. The idea should be also extendable to other decomposition strategies; future work could attempt to address further decomposing or portioning subproblems in order to exploit their special structure, so that instead of having two units more units are used to hierarchically decompose the problem.

Chapter 6

A Hybrid Particle Swarm Branch-and-Bound (HPB) Optimiser for Mixed Discrete Nonlinear Programming

6.1 Introduction

This work proposes a new algorithm for solving Mixed Discrete Non-Linear Problems (MDNLP), designed to efficiently combine the Particle Swarm Optimiser (PSO), a well-known global optimisation technique, and the Branch-and-Bound (BB), a widely used systematic deterministic algorithm for solving discrete problems. The proposed algorithm combines the global but slow search, and the rapid but local search capabilities of the PSO and the BB, respectively, to simultaneously achieve improved optimisation accuracy and low computational resources. It is capable of handling arbitrary continuous and discrete constraints without the use of the frequently cumbersome to parameterise penalty function. At the same time, it maintains a simple, generic and easy to implement architecture, and it is based on the Sequential Quadratic Programming (SQP) for solving the NLP subproblems in the BB tree. The performance of HPB is evaluated against real-world MDNLP benchmark problems, and it is found to be highly competitive compared to existing algorithms.

6.2. General Background

MDNLP refers to mathematical optimisation problems with multiple variables and nonlinearities in the objective function and/or the likely constraints, in which specified subsets of the variables are required to

take on discrete values, while the remaining are continuous. In the last decade, there has been a dramatic increase in the techniques developed to solve MDNLPs [14, 19, 34]. Such techniques have been applied in various applications, ranging from the process industry and engineering, to the financial and management sciences, as well as operational research sectors. MDNLP problems are notoriously difficult to solve, because they combine two difficult types of subproblems, namely the mixed discrete problem and the convex or non-convex nonlinear one.

The general MDNLP formulation can be stated as

$$\text{MDNLP problem} \begin{cases} \min\limits_{\vec{x},\vec{z}} & f(\vec{x},\vec{z}) \\ & g_i(\vec{x},\vec{z}) \le 0, \quad i = 1,\ldots,n_{ineq} \\ \text{s.t.} & h_j(\vec{x},\vec{z}) = 0, \quad j = 1,\ldots,n_{eq} \\ & \vec{x} \in X \subseteq \Re^{n_c} \\ & \vec{z} \in Z \equiv Z_1 \times \ldots \times Z_{n_d} \end{cases} \tag{6.1}$$

where $\vec{x}$ is a vector of n_c continuous variables and $\vec{z}$ is a vector of n_d discrete variables. The problem also accounts for n_{ineq} inequalities $g(\vec{x},\vec{z})$ and n_{eq} equalities $h(\vec{x},\vec{z})$. Z_k is the discrete set of values where each k^{th} discrete variable is allowed to take values from. The primary difficulty that arises in Eq. (6.1) is caused by the discrete nature of domain Z, since for large n_d and discrete domain sizes $|Z|$, the number of possible combinations increase exponentially. Therefore, complexity analysis characterises MDNLP problems as Non-Polynomial Complete (NPC) [1].

In general, there are two classes of optimisation methods for solving mixed discrete design problems: stochastic and deterministic ones. Stochastic search methods are a relatively recent development in the optimisation field, aimed to tackle difficult problems, such as ones afflicted by non-differentiability, multi-modality, multiple objectives and lack of smoothness. Examples of such methods include Simulated Annealing [71], Genetic Algorithms [8, 72], Genetic Programming [17, 73], Evolution Strategies [13, 74], and hybrid methods [75, 76]. Various population-based stochastic methods have been recently proposed

for MDNLP. Successful examples include the PSO [19], Genetic Algorithms [39], and Differential Evolution [50]. The principal advantage of these methods is that the search can be performed through the entire design space without specific knowledge of the problem. The disadvantage however, is that they cannot guarantee to find the global optimum within finite time, and as such, they are all computationally intensive and frequently exhibit slow convergence.

The deterministic search algorithms, on the other hand, take a different approach and adopt a systematic way of approaching the optimum. Diverse variations of these methods exist [77], with recent representative examples the Nonlinear Branch-and-Bound [6, 14], the Sequential Linearisation [54, 55], the Penalty Function approach [78, 79], and the Lagrangian Relaxation methods [80, 81]. Although, their main advantage is that they can use the structure of the problem to speed up the search for the discrete variables, their shortcoming is that they cannot cope with very non-smooth functions and multiple local minima.

In this article we propose the Hybrid PSO-BB architecture (HPB), which is based on the fact that the PSO has the ability to escape from local minima, while the BB exhibits faster convergence rate. HPB retains and combines these attractive properties of PSO and BB, while at the same time mitigates significantly their aforementioned weaknesses. It is particularly suited for difficult MDNLP problems, in the sense that the objective function and the constraints are non-smooth functions and have multiple local extrema. The HPB takes advantage of the rapid search of BB, when the PSO has discovered a better solution in its globally processed search space. The hybridisation phase of HPB depends primarily on a selective temporary switching from PSO to BB, when it appears that the current optimum can be potentially improved. As will be described later, any such potential improvement is recorded and broadcasted to the entire swarm of PSO particles via its social component update.

6.3 The Proposed Architecture

6.3.1 The Particle Swarm Optimisation (PSO) module

The PSO is a stochastic optimisation method based on the simulation of the social behaviour of bird flocks or biological groups in general, that evolve by information exchange among particles in a group. The PSO algorithm was first introduced by Kennedy and Ebehart [10] followed by a more general work on swarm intelligence [11]. In the PSO, the population is called the swarm and the individuals are called particles. Each particle flies in the search space and remembers the best position it ever experienced. The trajectory of each individual in the search space is adjusted by dynamically altering the velocity of each particle, according to its own experience (cognitive component) and the progress of the other particles in the search space (social component).

In the PSO (see Fig.6.1), we have a set of possible solutions $x \in \Re^n$ (the particle positions), with x_{jk} denoting the k^{th} vector component of the j^{th} particle. The initial velocity $v_j^{(t)}$ is set at random for every particle, velocity should be maintained within the range $[-V_{\max}, +V_{\max}]$ to reduce the likelihood of particles leaving the space. if velocity of particle is greater than $+V_{\max}$ or less than $-V\max$, its set to $+V_{\max}$. At each t^{th} iteration of the swarm operation, each position $x_j^{(t)}$ is updated according to the velocity $v_j^{(t)}$ of the j^{th} particle, according to

$$\begin{aligned} v_{jk}^{(t)} &= \underbrace{\omega \cdot v_{jk}^{(t-1)}}_{\substack{\text{momentum} \\ \text{component}}} + \underbrace{c_c \cdot r_{c\,jk}^{(t)} \cdot \left(P_{jk}^{(t)} - w_{jk}^{(t-1)}\right)}_{\substack{\text{cognitive} \\ \text{component}}} + \underbrace{c_s \cdot r_{s\,jk}^{(t)} \cdot \left(G_k^{(t)} - w_{jk}^{(t-1)}\right)}_{\substack{\text{social} \\ \text{component}}} \\ x_{jk}^{(t)} &= x_{jk}^{(t-1)} + v_{jk}^{(t)} \end{aligned} \tag{6.2}$$

where c_c and c_s are acceleration constants typically fixed to *~2.0* that control how far each particle moves in a single iteration, $r_{c\,jt}, r_{s\,jt} \in [0,1]$ are uniform randomly generated numbers that attain the stochastic swarm behaviour, and $\omega \in [0,1]$ is an inertia term regulating each particle's momentum. It can be seen from the three components of Eq.(6.2), that the trajectory of each j^{th} particle is adjusted to take into account its own best known solution P_j (individual experience), and the best known solution G in the

entire swarm (collaboration between the N members). The personal and global best positions are correspondingly given by

$$\begin{aligned} P_j^{(t)} &= \underset{x_j^{(i)}:0\le i\le t-1}{arg\,min}\left\{ f\left(x_j^{(i)}\right)\right\},\ \forall\ j=1,\ldots,N \\ G^{(t)} &= \underset{P_j^{(t)}:1\le j\le N}{arg\,min}\left\{ f\left(P_j^{(t)}\right)\right\} \end{aligned} \tag{6.3}$$

The role of the inertia weighting function ω is critical to the PSO's convergence, since it controls the influence of the previous history of the velocities on the current one. Accordingly, the inertia weighting function regulates the trade-off between the global and local exploration abilities of the swarms [82]. For the current work, we employ a very effective PSO variant, the Time Varying Inertia Weight (TVIW) PSO [83] that employs an adaptive acceleration term defined as

$$\omega^{(t)} = \omega_{max} - \frac{\omega_{max} - \omega_{min}}{t_{max}} \times t \tag{6.4}$$

$\omega_{\max}$ and $\omega_{\min}$ are the maximum and minimum values of the inertia term (typically set to *0.9* and *0.4*, respectively) and t_{max} the maximum number of iterations. Using this mechanism, the TVIW-PSO manages a more wandering early search, while towards the end of the run, when the space area containing the global optimum is found; it gradually assumes a more locally fine-tuning mode.

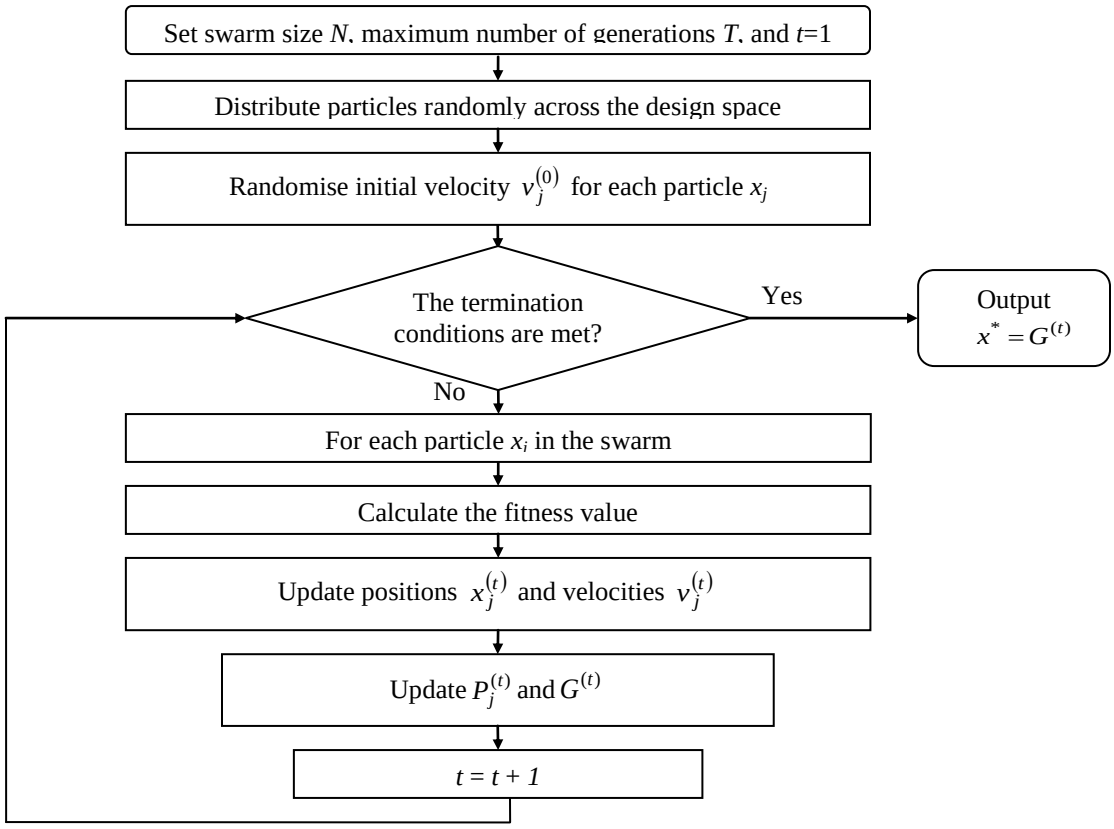


Fig.6.1 The TVIW-PSO algorithm.

6.3.2 Handling of Constraints

A critical part of MDNLP is how the algorithm handles the constraints, such as the $g_i \le 0$, $h_j = 0$, or the membership of the discrete variables $\vec{z} \in Z$ in Eq.(6.1). In the context of evolutionary optimisation, several mechanisms have been proposed to enforce inequality and equality constraints [12]. Overall, these can be grouped to four categories: ones that preserve the feasibility of solutions [84], penalty functions [85], ones which differentiate between feasible and infeasible solutions [9], and hybrid methods [86, 87]. Penalty functions are one of the most popular and direct approaches to handle constraints [6, 7, 82, 88]. However, their major disadvantage is the need for careful and often problem-specific calibration of coefficients and parameters.

In this work, we employ a very efficient constraint handling method referred to as the Fly-Back

mechanism [19], which is capable of maintaining a feasible population throughout the entire swarm lifetime. The idea is each particle has to fly back to its previous position when Eq.(6.2) takes it outside the feasible region. Experimental evaluations in [19] and our current comparisons with other methods here, indicate that this technique can locate better minima with fewer iterations, and more importantly, with no penalty parameters and coefficients needed tuning. The Fly-Back is managed by a simple modification of the particle update, such as

$$x_j^{(t)} = \begin{cases} x_j^{(t-1)} + v_j^{(t)} & \text{if} \quad g_i\left(x_j^{(t-1)} + v_j^{(t)}\right) \le 0 \quad \forall i \in \left\{1,\ldots,n_{ineq}\right\} \\ x_j^{(t-1)} & \text{if} \quad \exists i : g_i\left(x_j^{(t-1)} + v_j^{(t)}\right) > 0 \quad \text{(fly back)} \end{cases} \tag{6.5}$$

where only inequalities and continuous variables are shown for simplicity.

In order to apply fly-back to constrained optimisation problems, all randomly initialised solutions need to be located within the feasible search space. In general, but more critically for MDNLP cases with many complex nonlinear constraints g_i and h_j, it is time consuming to randomly find a swarm composed of N fully feasible solutions $x_j^{(0)}$. In this work, we have implemented an efficient constraint attainment formulation based on the SQP algorithm to generate random particles inside the feasible space with the minimum number of iterations, while maintaining at the same time an adequate initial sampling of the search space.

6.3.3 Treatment of Discrete Variables

MDNLPs typically include continuous, discrete and integer design variables. There are several methods that allow evolutionary techniques to handle discrete variables. Examples of such methods include rounding off [41], cutting plane technique [18], and pure discrete ones [89]. In this work, a truncation operator is used for enforcing the discreteness requirements on $\vec{z} \in Z_1 \times \ldots \times Z_{n_d}$. Specifically, we maintain all n_d components of $\vec{z}$ as real variables. Within each j^{th} particle x_j, these are concatenated with the n_c continuous variables $\vec{x}$ of the generic formulation of Eq.(6.1), so that the PSO velocity and position updates in Eqs.(6.2, 6.3) are left unrestricted. As has been observed in [42], this simple

truncation of their real values does not affect the search performance and keeps the handling of continuous and discrete variables uniform.

6.3.4 The Sequential Quadratic Programming (SQP) module

The SQP [2, 35] is a well-known method to solve nonconvex, nonlinear optimisation problems with linear or nonlinear constraints, and its overall operation is as follows. Assuming, for simplicity, that we minimise a function *f(x)* with only inequality constraints $g(\vec{x}) \le 0$, the SQP is executed for a number of iterations starting from an initial, not necessarily feasible solution point $\vec{x}_0$. At each k^{th} iteration, each current solution $\vec{x}_k$ is updated to $\vec{x}_{k+1}$ by finding a direction $\vec{d}_k$ and a step α_k towards that direction as

$$\vec{x}_{k+1} = \vec{x}_k + \alpha_k \cdot \vec{d}_k \tag{6.6}$$

At the beginning of the k^{th} iteration, we update a Hessian matrix H_k which is maintained as a positive definite matrix using the following scheme

$$H_k = H_{k-1} - \frac{H_{k-1}\vec{\delta}\vec{\delta}^T H_{k-1}}{\vec{\delta}^T H_{k-1}\vec{\delta}} + \frac{\vec{s}\vec{s}^T}{\vec{s}^T\vec{\delta}} \tag{6.7}$$

where,

$$\begin{aligned}
&\vec{\delta} = \vec{x}_k - \vec{x}_{k-1} \\
&\vec{s} = \beta\vec{q} + (1-\beta)H_{k-1}\vec{\delta} \\
&\vec{q} = \nabla_{\vec{x}} L(\vec{x}_k, \vec{\lambda}_{k-1}) - \nabla_{\vec{x}} L(\vec{x}_{k-1}, \vec{\lambda}_{k-1}) \\
&\beta = \begin{cases} 1.0 & \text{if } \vec{\delta}^T\vec{q} \ge 0.2\vec{\delta}^T H_{k-1}\vec{\delta} \\ \dfrac{0.8\vec{\delta}^T H_{k-1}\vec{\delta}}{\vec{\delta}^T H_{k-1}\vec{\delta} - \vec{\delta}^T\vec{q}} & \text{otherwise} \end{cases}
\end{aligned} \tag{6.8}$$

In the above, $\vec{\lambda}_k$ is the Lagrange multiplier vector at the k^{th} iteration while $L(\vec{x}, \vec{\lambda}) = f(\vec{x}) + \vec{\lambda}^T g(\vec{x})$ is the underlying Lagrangian. The moving direction $\vec{d}_k$ in Eq.(6.6) is found by solving the quadratic problem

$$\begin{aligned}
&\min_{\vec{d}_k} \left\{ \nabla f(\vec{x}_k)^T \vec{d}_k + \frac{1}{2}\vec{d}_k^T H_k \vec{d}_k \right\} \\
&s.t. \quad g(\vec{x}_k) + \nabla g_i(\vec{x}_k)^T \vec{d}_k \le 0, \quad \forall i = 1,\ldots,m
\end{aligned} \tag{6.9}$$

Finally, α_k in Eq.(6) is calculated by minimising the following exterior penalty function

$$\tilde{f}(\alpha_k) = f(\vec{x}_k + a_k\vec{d}_k) + \sum_{i=1}^{m} \tilde{\lambda}_{i,k} \max\left(0, g_i(\vec{x}_k + a_k\vec{d}_k)\right) \tag{6.10}$$

where $\tilde{\lambda}_{i,k}$ is the weight of the i_{th} constraint at iteration k and depends on the multiplier $\lambda_{i,k}$ from solving Eq.(8) and the previous weight $\tilde{\lambda}_{i,k-1}$ as

$$\tilde{\lambda}_{i,k} = \begin{cases} \lambda_{i,k} & \text{if } k = 1 \\ max\left(\lambda_{i,k}, \frac{1}{2}(\tilde{\lambda}_{i,k-1} + \lambda_{i,k})\right) & \text{otherwise} \end{cases} \tag{6.11}$$

The termination of Eq.(6) is achieved when the relative objective value decrease cannot exceed a user-defined tolerance ε as

$$\frac{\left|f(\vec{x}_{k+1}) - f(\vec{x}_k)\right|}{\left|f(\vec{x}_{k+1})\right|} \le \varepsilon \tag{6.12}$$

6.3.5 The Branch-and-Bound (BB) module

Branch-and-Bound is an established and generic algorithm for efficiently enumerating and searching parts of discrete problems. The algorithm generally alternates between two main steps: branching, which is a recursive subdivision of the search space, and bounding, which is the computation of lower and upper bounds for the global minimum of the objective function in a sub-region of the search space. The nonlinear BB for mixed-integer problems [14] is based on the mechanisms of separation, relaxation, and fathoming. The algorithm performs a tree-search, and starts by solving first the continuous problem relaxation using a Non-Linear Programming (NLP) solver. If all discrete variables take discrete values the search is stopped. Otherwise, a tree search is performed in the space of the discrete variables. Then the algorithm selects one of those discrete variables which take a non-discrete value, and branch on it. One can eliminate more branches if a smaller upper bound is generated as early as possible. This can be accomplished by choosing the right variable for branching at each step. In our work, for efficiency we used the Min-Clearance Difference method [90] to choose the k^{th} variable by determining the minimum difference between the non-discrete optimum values and their nearest allowable values for the next

branching step. The criterion can be defined as

$$\begin{aligned} \Delta z_i &= \left| \, | z_i - z_i^* | - | z_i - z_i^* + 1 | \, \right| \\ k &= \underset{i \in \{1,\ldots,n_d\}}{arg\,min} \{ \Delta z_i \} \end{aligned} \tag{6.13}$$

where Δz_i is the clearance for the i^{th} design variable.

Branching generates two new sub problems by adding simple bounds to the NLP relaxation. One of the two new NLP problems is selected and solved next by using the SQP method. If the discrete variables take non-discrete values then branching is repeated. Otherwise, if one of the fathoming rules is satisfied, then no branching is required and the corresponding node has been fully explored. The following three fathoming rules may be used to fathom a given candidate problem: 1. Infeasible subproblem, 2. Discrete feasible solution (record the value of this solution as upper bound), 3. Lower bound greater than the objective values of a known discrete solution. If discrete solutions are found, they provide upper bounds on the optimal value of the original problem. Once a node has been fathomed, the algorithm backtracks to another node which has not been fathomed until all nodes are fathomed. An overview of the nonlinear BB algorithm is shown in Fig.2.

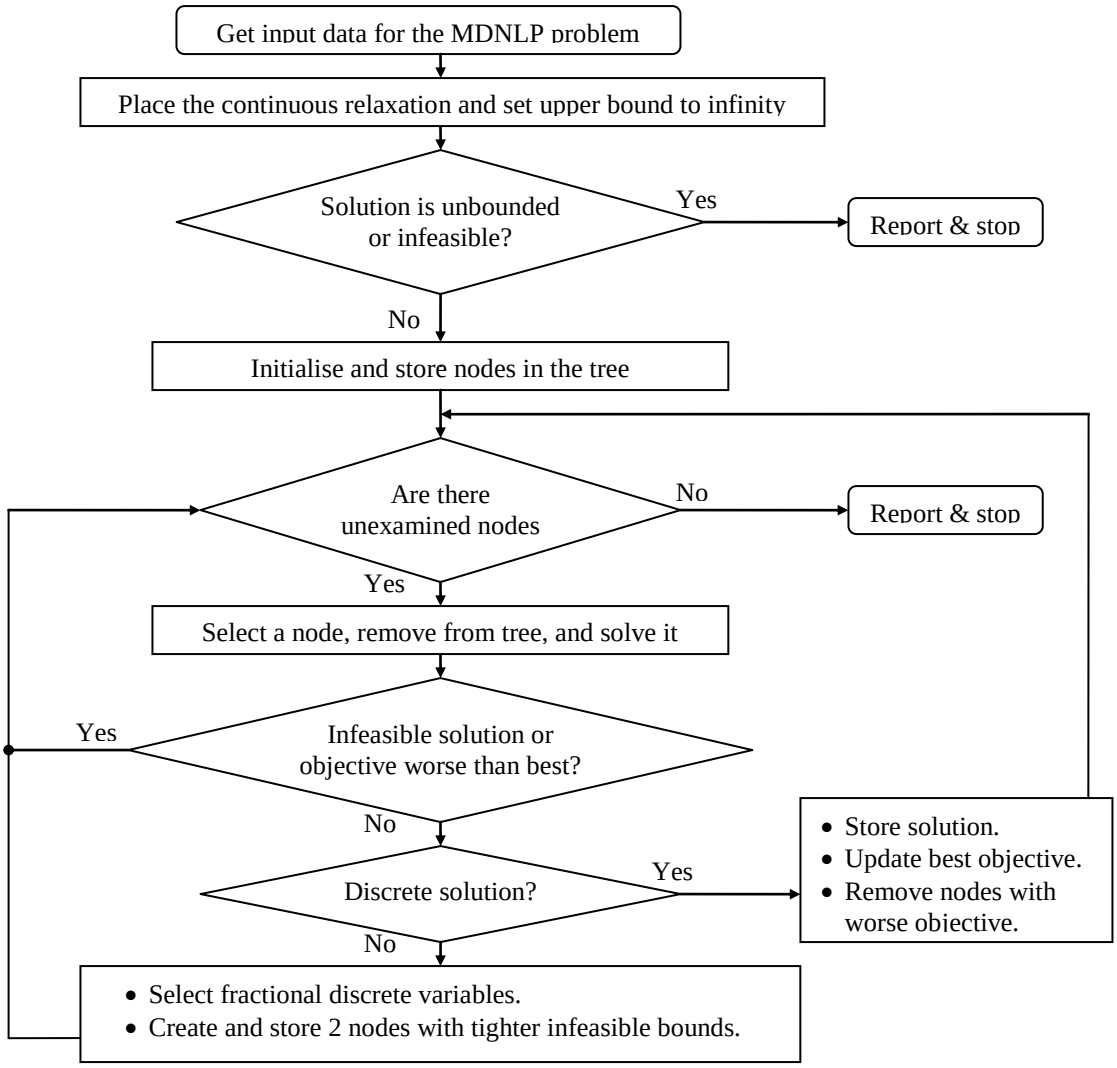


Fig.6.2 The nonlinear Branch-and-Bound algorithm.

Example:

This problem arises in the synthesis of chemical process, and it was investigated by Duran & Grossmann [5]. The goal is to determine the optimal solution of a chemical process system. The problem has three continuous variables $[x_1, x_2, x_3]^T$ and three integer variables $[x_4, x_5, x_6]^T$ with six linear and nonlinear inequality constraints. The mathematical formulation of the problem is given by:

$$
\begin{aligned}
&min \qquad f(X)=5x_4+6x_5+8x_6+10x_1-7x_3-18\times\log(x_2+1)-19.2\times\log(x_1-x_2+1)+10\\
&s.t\begin{cases}
0.8\times\log(x_2+1)+0.96\times\log(x_1-x_2+1)-0.8x_3\geq 0\\
\log(x_2+1)+1.2\times\log(x_1-x_2+1)-x_3-2x_6\geq -2\\
x_2-x_1\leq 0\\
x_2-2x_4\leq 0\\
x_1-x_2-2x_5\leq 0\\
x_4+x_5\leq 1\\
0\leq x_1\leq 2\ ,0\leq x_2\leq 2\ ,\ 0\leq x_3\leq 1\\
x_4,x_5,x_6\in\{0,1\}
\end{cases}
\end{aligned}
\tag{14}
$$

The optimal solutions obtained during the tree-search process are presented in Fig.6.3.

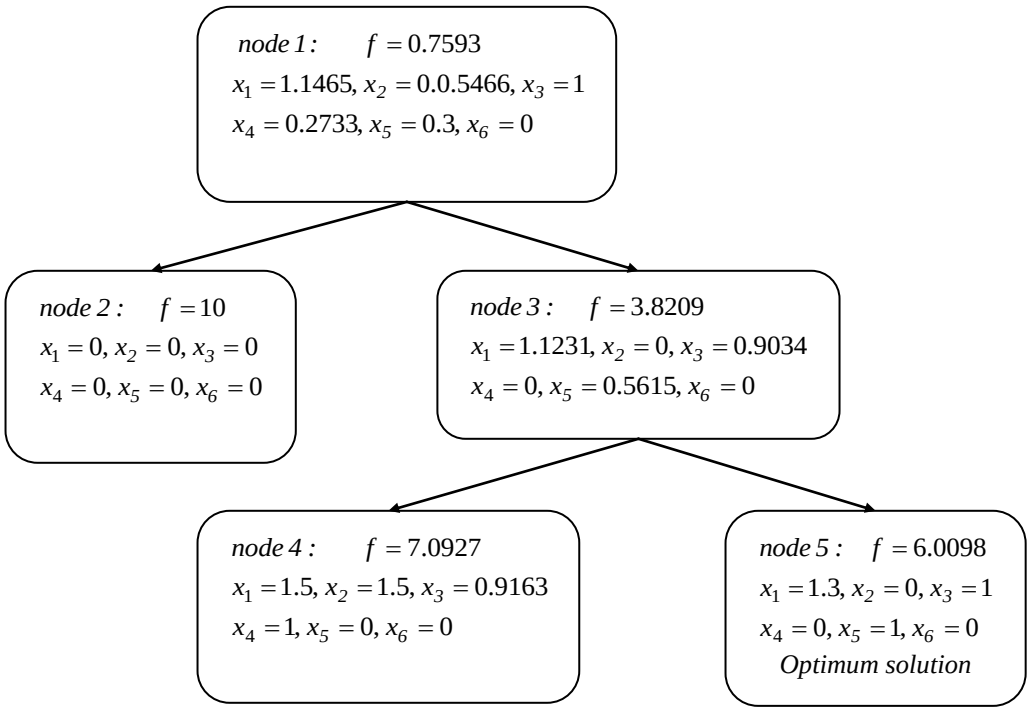


Fig.6.3 Branch-and-Bound tree.

The example shows that, the branching process starts at node 1, where new constraints are added into the subproblems, and NLP problem is solved inside each node. When the backtracking begins for the depth first search, we find the upper bound of *7.0927* in node 4, subsequently we check nodes 2 and 5, and terminate with the least upper bound as the optimal solution. It can be noted that, the algorithm stops searching in the direction of node 2, 4, and 5 because each node contains an integer solution and should be discarded. The optimal solution can be obtained in node 5, where all the discrete variables take integer

solutions, with the minimum objective function value. The search process terminates when there are no unexplored parts of the solution space.

6.3.6 Integrating and sequencing the PSO and BB

As discussed earlier, the integration framework introduced in this work aims to improve the solution accuracy compared to existing methods. Multi-method integration and hybridisation approaches have been shown effective in various works. Recently, [91] proposed a hybrid Genetic Algorithm (GA) and PSO for solving mixed-variables optimisation problems by incorporating evolutionary optimisation elements into the socially inspired PSO. [52] also presented a hybrid PSO-GA for recurrent network design, based on the concept of the maturation phenomenon in nature. The work of [60] presented a model for the hybridisation of memetic GAs with a truncated BB algorithm trying to boost performance through mutual collaboration. An efficient method designed for a specific application, the solution of the economic dispatch problem was proposed in [92]. This combined the PSO with SQP, by calling SQP selectively in each iteration to fine-tune the PSO solution. Our method is a direct extension of [92], aimed for generic optimisation in the MDNLP domain and not only NLP type problems. Another similar work is that of [93] that presented a hybrid BB-PSO algorithm, but specifically designed to solve integer separable concave programming problems, where the lower bound of the optimal value was determined with linear relaxation and the upper bound with PSO. Another hybridisation strategy has been proposed by [94] which is a PSO-based memetic algorithm for the flow shop scheduling problem that applies the evolutionary mechanism of PSO to perform exploration and several adaptive local searches.

Concerning MDLNP optimisation, previously proposed related approaches include the work of [79] that incorporated a dynamic penalty approach and PSO. A penalty function for the discrete design variables was introduced to handle them similar to the continuous design variables. [67] proposed a hybrid GA for solving MDNLP. In their approach, the GA was used to determine the optimal feasible region surrounding the global optimum, and a gradient method was subsequently used to find the final solution.

Furthermore, [68] combined adaptive genetic search strategies for MDNLP, while [50] introduced a Differential Evolution algorithm capable of optimising integer, discrete, and continuous variables and handling multiple constraints using penalty functions. Recently, [95] proposed an improved PSO algorithm for solving nonconvex MDNLP problems with equality constraints; the original problem is transformed into one with no constraints after mixed variables are partitioned and reduced. [96] presented an improved GA that uses information theory to refresh the population as prematurity occurs. A modified local search is performed to determine the more-feasible solutions in a period of generations.

In this work, the proposed HPB algorithm integrates the PSO and the BB methods to facilitate an accurate and at the same time rapid search for generic MDNLP problems. At the beginning of the proposed algorithm, the BB is used to determine a feasible initial solution for the PSO. This solution is taken as the best known solution $G^{(t)}$ of all the swarm agents. In each iteration, whenever an improvement in the currently stored global solution is achieved by the PSO, this improved $G^{(t)}$ is passed over to the BB module as a starting point. This is the principal link between PSO and the BB, as by exploiting the rapid convergence properties of BB, the PSO global search is influenced by the improved $G^{(t)}$, which propagates to all particles through the social interactions of Eq.(6.2). As seen in Fig.6.4, it is advantageous that the BB is not needed to be called in every single iteration but, similar to the NLP work of [92], only when the PSO has found a better solution. When BB completes, $G^{(t)}$ is updated and the PSO resumes. During PSO search, the discrete elements $\vec{z}$ of position vectors $x^{(t)}$ are truncated to the nearest valid discrete points. In any case, all solutions are guaranteed not to exceed the bounds of the search space because the infeasible particles are pulled back to the feasible region using the fly-Back mechanism. The search continues until a termination criterion, such as the maximum number of iterations *T*, is satisfied.

Numerical experimentations show that PSO and BB have their individual advantages and characteristics when solving different optimisation problems. The principal objective of HPB is to combine and preserve these characteristics for a wide range of difficult problems. As the experimental results show that the

proposed HPB framework combines the PSO and BB algorithms in a highly efficient manner in terms of both accuracy and computational cost. Experiments show that HPB outperforms other methods because of using an efficient strategy which allows the utilisation of the fast search mechanism of the BB method while maintaining the global optimisation properties of PSO. Furthermore, stochastic search methods require many function evaluations as compared to derivative-based optimisation methods, which is the cost of not using derivatives. The proposed algorithm can quickly find the optimum point by using deterministic search method. Hence, the HPB algorithm requires significantly fewer function evaluations to converge, and at the same time reduces the number of iterations for most of the studied problems.

As described in the previous subsections, an important user-oriented advantage of HPB is that there are no penalty functions, and it thus has much fewer parameters needed adjusting compared to other techniques, such as [6, 22, 97]. To avoid expensive global optimisation, HPB uses an efficient fly-back method to handle the nonlinear constraints, where particles are allowed to be attracted only to feasible solutions; this ensures that the personal best positions are always feasible. Moreover, the HPB algorithm uses SQP to determine the upper bounds needed in BB, which reduces the computational expenses when compared to other works, such as [93] which renewed the upper bound using PSO. The overall operation sequencing of the HPB is provided in Fig.6.4 and Table 6.1.

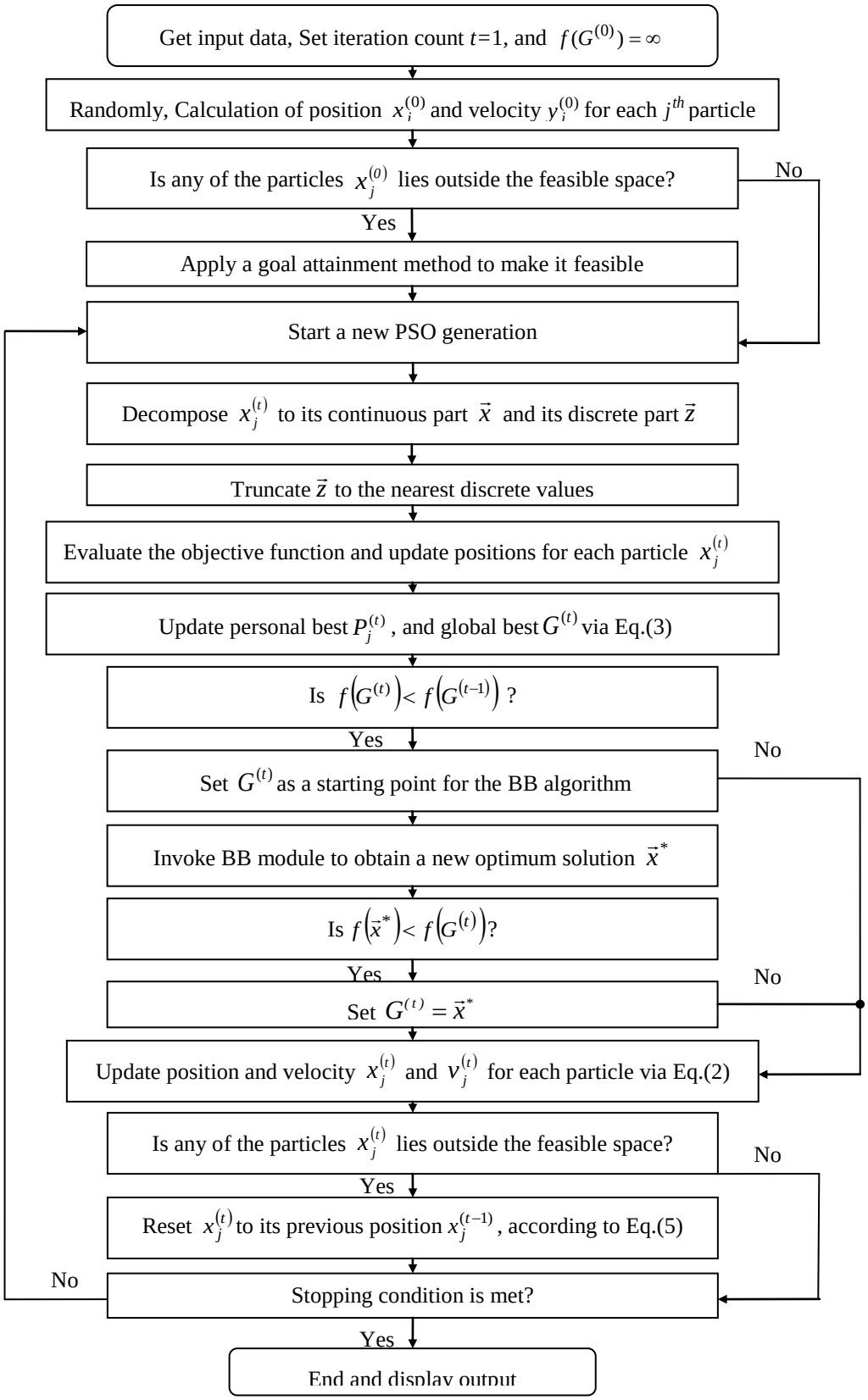


Fig.6.4 Flowchart of the proposed HPB algorithm

Stage 1: Initialisation Stage

set iteration count $t = 1$, and other user-defined algorithmic parameters
for each j^{th} particle, where $j = 1,\ldots,N$
set a random initial position $x_j^{(0)}$
if $x_j^{(0)}$ lies outside the feasible space
Apply a goal attainment method to make it feasible
endif
set a random initial velocity $v_j^{(0)}$
endfor

Stage 2: Optimisation Stage

while termination condition is not satisfied

%Evaluation phase
for each particle $x_j^{(t)}$
Decompose $x_j^{(t)}$ to its continuous part $\vec{x}$ and its discrete part $\vec{z}$
Truncate $\vec{z}$ to the nearest discrete values
Evaluate the objective function $f(\vec{x},\vec{z})$, and record it
Update personal best $P_j^{(t)}$ & global best $G^{(t)}$ via Eq.(3)
endfor

%Hybridisation phase
if $t == 1$ OR $f(G^{(t)}) < f(G^{(t-1)})$ (i.e., first entry or PSO achieved improvement)
Invoke BB module, starting the optimisation from $G^{(t)}$,
and record its final solution $\vec{x}^*$
if $f(\vec{x}^*) < f(G^{(t)})$ (i.e., BB achieved improvement)
set $G^{(t)} = \vec{x}^*$
endif
endif

%Creation of next generation swarm
set $t = t + 1$
for each particle $x_j^{(t)}$
Update position and velocity $x_j^{(t)}$ and $v_j^{(t)}$ via Eq.(2)
if the position of particle $x_j^{(t)}$ lies outside the feasible space
Fly-back the current particle to its previous position $x_j^{(t-1)}$, according to Eq.(5)
endif
endfor

endwhile

Table 6.1 Detailed sequencing of operations for the proposed HPB algorithm.

6.4 Numerical Experiments

In this section, we evaluate the proposed HPB with a number of difficult real-world MDNLP bench problems frequently employed in the literature. These problems arise in mechanical engineering, and represent highly nonconvex optimisation problems. In order to investigate the best performance of the HPB algorithm, different population sizes are used for each problem with different number of generations. The experimental results suggested that for all benchmark tests, a very small population of *20* could produce quickly good results. This value was fixed for all the experiments, and we only varied the total number of search iterations for experimental efficiency as the different test cases converge differently. In addition, *50* independent runs were carried out for each case. A linear decrease in the inertia term ω, with a maximum and minimum of *0.9* and *0.4*, respectively was adequate to improve the convergence rate of the HPB algorithm for all cases. The default values of acceleration constants c_c and c_s were both set to *2.0* for the same reason. The constraint tolerances $\varepsilon_{g,h} = 10^{-4}$ are used for both equality and inequality constraints in all runs.

A. *Pressure Vessel Design*

In practical design optimisation problems, continuous, and discrete variables occur quite frequently. Here we take a pressure vessel design optimisation problem from Sandgren [43]. The objective of this problem is to minimise the total cost of materials for forming and welding of a pressure vessel. The design variables of the problem are as shown in Table 6.2.

Design Variables	Definition	Unit	Remarks	Discrete Length
x_1	thickness (T_s)	*inch*	discrete	0.0625
x_2	thickness (T_h)	*inch*	discrete	0.0625
x_3	radius (R)	*inch*	continuous	---
x_4	length (L)	*inch*	continuous	---

Table 6.2 Design variables of a Pressure Vessel.

The mathematical formulation of the problem is

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

$$\text{subject to} \quad \begin{cases} g_1(X) = 0.0193x_3 - x_1 \le 0 \\ g_2(X) = 0.00954x_3 - x_2 \le 0 \\ g_3(X) = 1{,}296{,}000 - \pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 \le 0 \\ g_4(X) = x_4 - 240 \le 0 \\ 0.0625 \le x_1 \le 6.1875 \\ 0.0625 \le x_2 \le 6.1875 \\ 10 \le x_3 \le 200 \\ 10 \le x_4 \le 200 \end{cases} \tag{6.15}$$

We now compare the performance of our approach with other methods presented in the published literature. Pressure vessel design is a common benchmarking problem for MDNLP, and many researches have tried to solve it using different techniques [19, 40, 44, 69]. In this example, HPB had the lowest objective function value over the *50* test runs, with a significantly lower computational cost than all of the other algorithms. The optimal results of this problem are shown in Table 6.3.

Quantity	*EP* [69]	*EA* [44]	*GA* [40]	*PSO* [19]	HPB
x_1	1.000	0.9345	0.8125	0.8125	0.8125
x_2	0.625	0.5000	0.4375	0.4375	0.4375
x_3	51.1958	48.3290	40.097398	42.0984456	42.09893
x_4	90.7821	112.6790	176.654047	176.636595	176.6305
g_1	-0.0119	-0.00475	-0.00002	0.00000	0.00000
g_2	-0.1366	-0.038941	-0.035891	-0.0358808	-0.03587
g_3	-13584.5631	-3652.87683	-27.886075	0.00000	0.00000
g_4	-149.2179	-127.321	-63.345953	-63.363404	-63.69484
f(X)	**7,108.6160**	**6,410.3811**	**6,059.94634**	**6,059.7143**	**6,059.65457**

Table 6.3 Optimal solution of pressure vessel design problem.

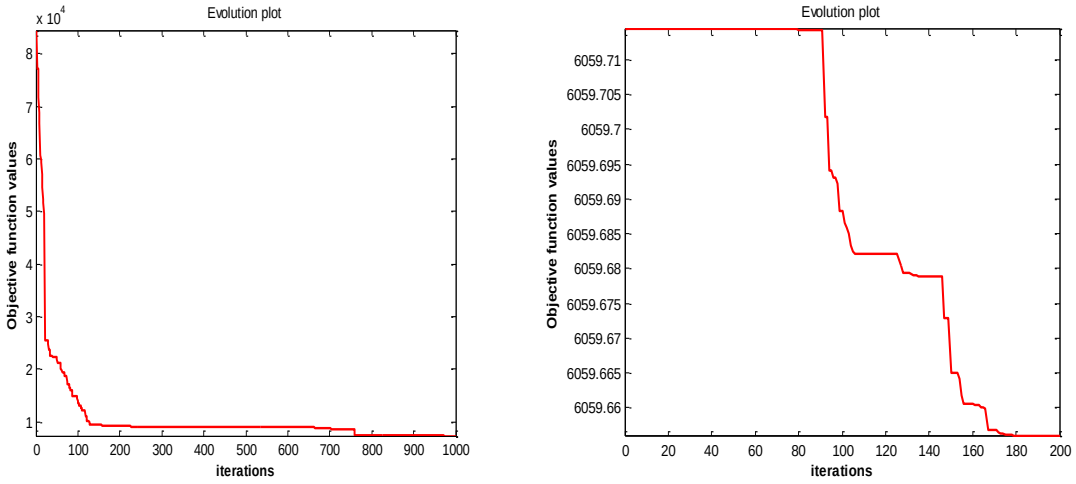
In our algorithm, we have used a population size of *N=20*, and the maximum number of search iterations was set to *T=200*. From Table 6.3, it can be seen that, the present algorithm reported the best performance to this problem. The run that resulted in the best objective function value performed *4,013* function evaluations and required *3.9* seconds of CPU time. The mean value of the objective function over the *50*

test runs was *6,059.84*, with standard deviation of *0.02194*. The best-known result was obtained by [19] using an improved PSO. The HPB algorithm slightly outperformed that algorithm in terms of solution accuracy. However, the proposed approach provides much better performance in terms of computational cost.

The performance of HPB and the algorithm described in [19] is compared in Table 6.4. The run that resulted in the best convergence properties are presented in Fig.6.5. The faster HPB convergence can be observed from Fig.6.5(b).

Method		Iterations	Particles	Runs	Function evaluations	Mean value	Standard deviation	CPU time
PSO	$f(X)$=6,059.71	1000	30	100	30,000	6,289.92	305.78	8.2s
HPB	$f(X)$=6,059.65	200	20	50	4,013	6,060.08	0.02194	3.9s

Table 6.4 Comparison of the HPB algorithm performance on the pressure vessel design problem.



(a) Improved PSO (1000 generations) (b) HPB (200 generations)

Fig.6.5 Performance comparison for the pressure vessel design problem.

B. *Spring Design*

This example tackles the design of a compression coil spring under constant load for minimum volume of materials as shown in Fig.6.6. This problem has been solved by many authors [19, 50, 43, 44], as it illustrates the use of continuous, discrete, and integer variables. There are two linear and six nonlinear

constraints, while the design variables are summarised in Table 6.5.

Design Variables	Definition	Unit	Remarks	Discrete Length
x_1	the wire diameter (d)	*inch*	discrete	vary
x_2	the mean coil diameter (D)	*inch*	continuous	---
x_3	the number of active coils (N)	---	integer	1

Table 6.5 Design variables of Compression coil spring.

The master problem formulation is given below

$$\min \quad f(X) = \frac{\pi^2 x_2 x_1^2 (x_3 + 2)}{4}$$

$$\text{subject to} \quad \begin{cases} g_1(X) = \dfrac{8C_f F_{max} x_2}{\pi x_1^3} - S \le 0 \\ g_2(X) = l_f - l_{max} \le 0 \\ g_3(X) = d_{min} - x_1 \le 0 \\ g_4(X) = x_2 - D_{max} \le 0 \\ g_5(X) = 3.0 - \dfrac{x_2}{x_1} \le 0 \\ g_6(X) = \sigma_p - \sigma_{pm} \le 0 \\ g_7(X) = \sigma_p + \dfrac{(F_{max} - F_p)}{K} + 1.05(x_3 + 2)x_1 - l_f \le 0 \\ g_8(X) = \sigma_w - \dfrac{(F_{max} - F_p)}{K} \le 0 \\ 0.2 \le x_1 \le 1.0 \\ 0.6 \le x_2 \le 3 \\ 1.0 \le x_3 \le 70 \end{cases} \tag{6.16}$$

where $C_f = \dfrac{4(x_2/x_1)-1}{4(x_2/x_1)-4} + \dfrac{0.615x_1}{x_2}$, $K = \dfrac{Gx_1^4}{8x_3x_2^3}$, $\sigma_p = \dfrac{F_p}{K}$, $l_f = \dfrac{F_{max}}{K} + 1.05(x_3 + 2)x_1$

The discrete variable d is having 42 possible discrete values

$$x_1 \in \left\{ \begin{matrix} 0.009,0.0095,0.0104,0.0118,0.0128,0.0132,0.014,0.015,0.0162,0.0173,0.018,0.020,0.023, \\ 0.025,0.028,0.032,0.035,0.041,0.047,0.054,0.063,0.072,0.080,0.092,0.105,0.120,0.135,0.148, \\ 0.162,0.177,0.192,0.207,0.225,0.244,0.263,0.283,0.307,0.331,0.362,0.394,0.4375,0.500 \end{matrix} \right\}$$

The values of pre-assigned parameters are chosen as

$F_{max} = 1{,}000.0lb,\ l_{max} = 14.0in.,\ d_{min} = 0.2in.,\ S = 189{,}000.0\,psi,\ D_{max} = 3.0in.,\ F_p = 300.0lp,$
$\sigma_{pm} = 6.0in.,\ \sigma_w = 1.25in.,\ G = 11.5 \times 10^6\ psi.$

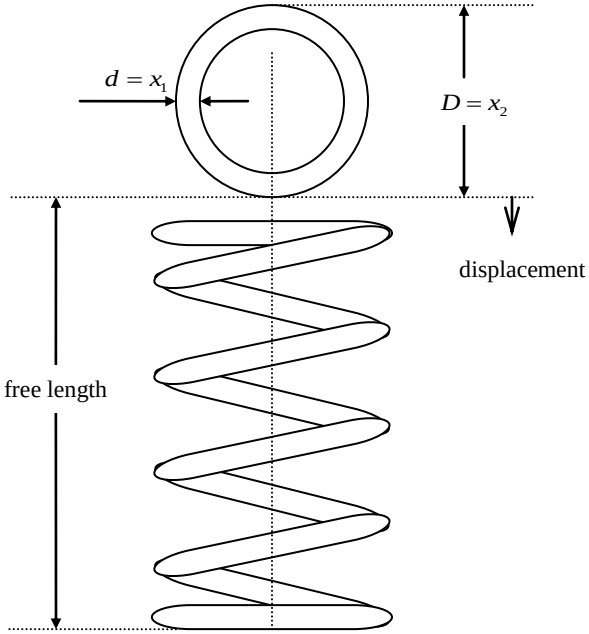


Fig.6.6 Compression coil spring.

Comparisons of results of the proposed algorithm, as well as results published in the literature are shown in Table 6.6.

Quantity	*Sandgren* [43]	*GeneAS* [44]	*DE* [50]	*PSO* [19]	HPB
x_1	0.283	0.283	0.283	0.283	0.283
x_2	1.180701	1.226	1.22304101	1.22304101	1.22301421
x_3	10	9	9	9	9
g_1	-54309	-713.510	-1008.8114	-1008.8114	-1011.6168
g_2	-8.8187	-8.933	-8.9456	-8.9456	-8.9457
g_3	-0.08298	-0.083	-0.083	-0.083	-0.0830
g_4	-1.8193	-1.491	-1.777	-1.777	-1.7769
g_5	-1.723	-1.337	-1.3217	-1.3217	-1.3216
g_6	-5.4643	-5.461	-5.4643	-5.4643	-5.4643
g_7	0.0000	0.0000	0.0000	0.0000	0.0000
g_8	0.0000	-0.009	0.0000	0.0000	0.0000
f(X)	**2.7995**	**2.665**	**2.65856**	**2.65856**	**2.65850**

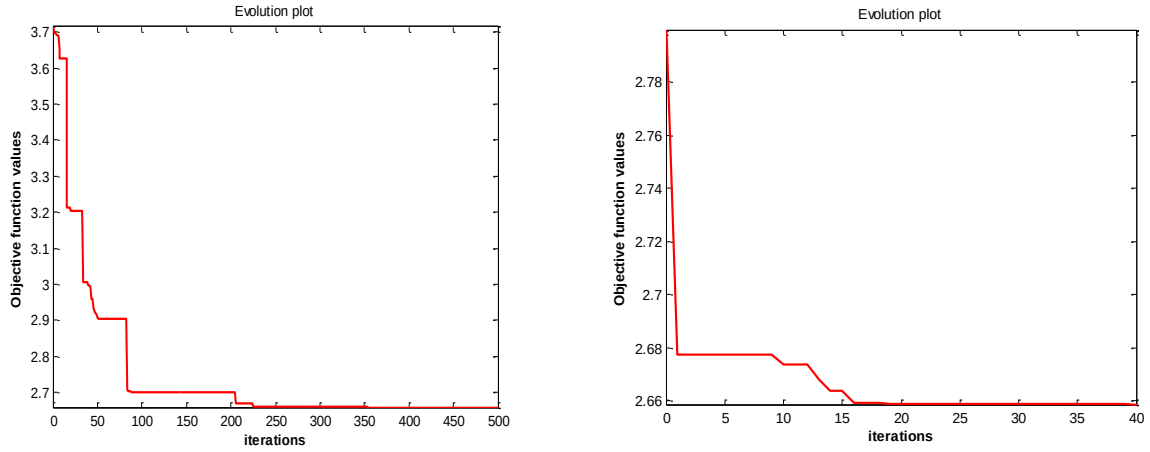
Table 6.6 Optimal solution of spring design problem.

In this case, the optimum value of the objective function is found to be slightly better than that of [19, 50] but with a significant improvement in the number of function evaluations compared. The mean value for the objective function obtained from *50* runs was *2.6621*, with a standard deviation *0.0239*. From

Table 6.7, HPB demonstrates substantial gain in effectiveness and performance compared to [19] algorithm. The convergence plots of the best solutions produced by all runs are shown in Fig.6.7.

Method	Iterations	Particles	Runs	Function evaluations	Mean value	Standard deviation	CPU time
PSO $f(X)$= 2.65856	500	30	100	15,000	2.73802	0.10706	5.8s
HPB $f(X)$= 2.65850	40	20	50	835	2.6985	0.0239	2.0s

Table 6.7 Comparison of the proposed algorithm performance on the spring design problem.



(a) Improved PSO (500 generations) (b)HPB (40 generations)

Fig.6.7 Performance comparison for the spring design problem.

C. *Welded Beam Design*

This problem is chosen from [45] and involves finding the minimum cost design of the structural welded beam design, with seven linear and nonlinear constraints. It has four design variables expressed in Table 6.8.

Design Variables	Definition	Unit	Remarks	Discrete Length
x_1	thickness of the weld (h)	*inch*	integer	1
x_2	length of the welded joint (l)	*inch*	integer	1
x_3	bar thickness (t)	*inch*	discrete	0.5
x_4	bar breadth (b)	*inch*	discrete	0.5

Table 6.8 Design variables of a welded beam.

The problem can be mathematically formulated as follows:

$$\begin{aligned}
\min \quad & f(X) = 1.10471 x_1^2 x_2 + 0.04811 x_3 x_4 (14.0 + x_2) \\
\text{subject to} \quad & \begin{cases}
g_1(X) = \tau(X) - \tau_{max} \le 0 \\
g_2(X) = \sigma(X) - \sigma_{max} \le 0 \\
g_3(X) = x_1 - x_4 \le 0 \\
g_4(X) = 0.10471 x_1^2 + 0.04811 x_3 x_4 (14.0 + x_2) - 5.0 \le 0 \\
g_5(X) = 0.125 - x_1 \le 0 \\
g_6(X) = \delta(X) - \delta_{max} \le 0 \\
g_7(X) = P - P_c(X) \le 0 \\
0.1 \le x_1 \le 2.0 \\
0.1 \le x_2 \le 10.0 \\
0.1 \le x_3 \le 10.0 \\
0.1 \le x_4 \le 2.0
\end{cases}
\end{aligned} \tag{6.17}$$

where,

$$\tau(X) = \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2}$$

$$\tau' = \frac{P}{\sqrt{2}x_1 x_2} \quad , \quad \tau'' = \frac{MR}{J} \quad , \quad M = P\left(L + \frac{x_2}{2}\right)$$

$$R = \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2}$$

$$J = 2\left\{\sqrt{2}x_1 x_2 \left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$$

$$\sigma(X) = \frac{6PL}{x_4 x_3^2} \quad , \quad \delta(X) = \frac{4PL^3}{E x_3^3 x_4}$$

$$P_c(X) = \frac{4.013\sqrt{EG(x_3^2 x_4^6 / 36)}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$$

The numerical parameters for the model are chosen as:

$$P = 6000lb, L = 14in., E = 30 \times 10^6\ psi,$$
$$G = 12 \times 10^6\ psi, \tau_{\max} = 13600psi,$$
$$\sigma_{max} = 30000psi, \delta_{max} = 0.25in.$$

This problem has been solved by previous researchers [12, 45] as a continuous optimisation problem, while other author [67] solved it as a mixed discrete problem. This problem is highly nonlinear and

non-convex. The number of particles was set to *N=20*, and the number of search iterations was *T=40*. From Table 6.9, HPB reported the best result to this problem over *50* runs. The total number of function evaluations performed was *702*. The mean value for all the runs performed was *4.3923* with a standard deviation *0.9267*. The performance of HPB can be summarised in Table 6.10, while Fig.6.8 shows convergence characteristic of HPB algorithm in welded beam design.

Quantity	*Ragsdell* [45]	*GA* [12]	*MDHGA* [67]	HPB
x_1	0.2455	0.205986	1	1
x_2	6.1960	3.471328	2	1
x_3	8.2730	9.020224	4.5	4.5
x_4	0.2455	0.206480	1	1
g_1	-5,743.826517	-0.074092	-6,685.2615	-891.365
g_2	-4.715097	-0.266227	-5,111.111	-5,111.111
g_3	0.000000	-0.000495	0.00000	0.00000
g_4	-3.020289	-3.430043	-1.4313	-1.6478
g_5	-0.120500	-0.080986	-0.8750	-0.8750
g_6	-0.234208	-0.235514	-0.2259	-0.2259
g_7	-74.2768560	-58.666440	-248,338.48	-248,338.48
f(X)	**2.38593732** continuous solution	**1.728226** continuous solution	**5.67334** discrete solution	**4.352135** discrete solution

Table 6.9 Optimal solution of welded beam design.

Method	Iterations	Particles	Runs	Function evaluations	Mean value	Standard deviation
HPB *f(X)*= 4.352135	40	20	50	702	4.3923	0.9267

Table 6.10 Computational performance of HPB algorithm for welded beam design problem.

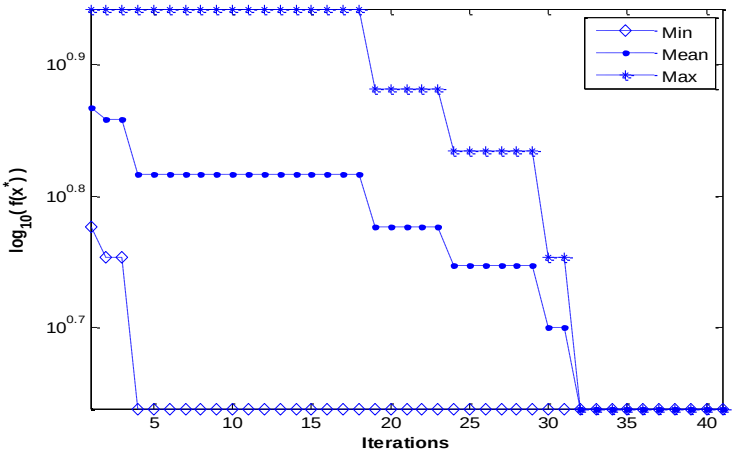


Fig.6.8 Evolution plots of welded beam design.

D. ***Speed Reducer Design***

This is a more complicated example taken from [2]. The objective of this problem, shown in Fig.6.9, is to minimise the weight of the speed reducer subject to constraints on bending stress of the gear teeth, surface stress, transverse deflections of the shafts and shaft stresses. The design variables with their types are shown in Table 6.11. The mathematical formulation of the problem is given by:

$$
\begin{aligned}
\min \quad & f(X) = 0.7854 x_1 x_2^2 (3.3333 x_3^2 + 14.9334 x_3 - 43.0934) - 1.508 x_1 \\
& (x_6^2 + x_7^2) + 7.477 (x_6^3 + x_7^3) + 0.7854 (x_4 x_6^2 + x_5 x_7^2) \\
\text{subject to} \quad & \begin{cases}
g_1(X) = 27 x_1^{-1} x_2^{-2} x_3^{-1} \le 1 \\
g_2(X) = 397.5 x_1^{-1} x_2^{-2} x_3^{-2} \le 1 \\
g_3(X) = 1.93 x_2^{-1} x_3^{-1} x_4^3 x_6^{-4} \le 1 \\
g_4(X) = 1.93 x_2^{-1} x_3^{-1} x_5^3 x_7^{-4} \le 1 \\
g_5(X) = \left[\left(\dfrac{745 x_4}{x_2 x_3} \right)^2 + (16.9) 10^6 \right]^{0.5} / (0.1 x_6^3) \le 1{,}100 \\
g_6(X) = \left[\left(\dfrac{745 x_5}{x_2 x_3} \right)^2 + (157.5) 10^6 \right]^{0.5} / (0.1 x_7^3) \le 850 \\
g_7(X) = x_2 x_3 \le 40 \\
g_8(X) = x_1 / x_2 \ge 5 \\
g_9(X) = x_1 / x_2 \le 12 \\
g_{10}(X) = (1.5 x_6 + 1.9) x_4^{-1} \le 1 \\
g_{11}(X) = (1.1 x_7 + 1.9) x_5^{-1} \le 1 \\
2.6 \le x_1 \le 3.6 \\
0.7 \le x_2 \le 0.8 \\
17 \le x_3 \le 28 \\
7.3 \le x_4 \le 8.3 \\
7.3 \le x_5 \le 8.3 \\
2.9 \le x_6 \le 3.9 \\
5.0 \le x_7 \le 5.5
\end{cases}
\end{aligned}
\tag{6.18}
$$

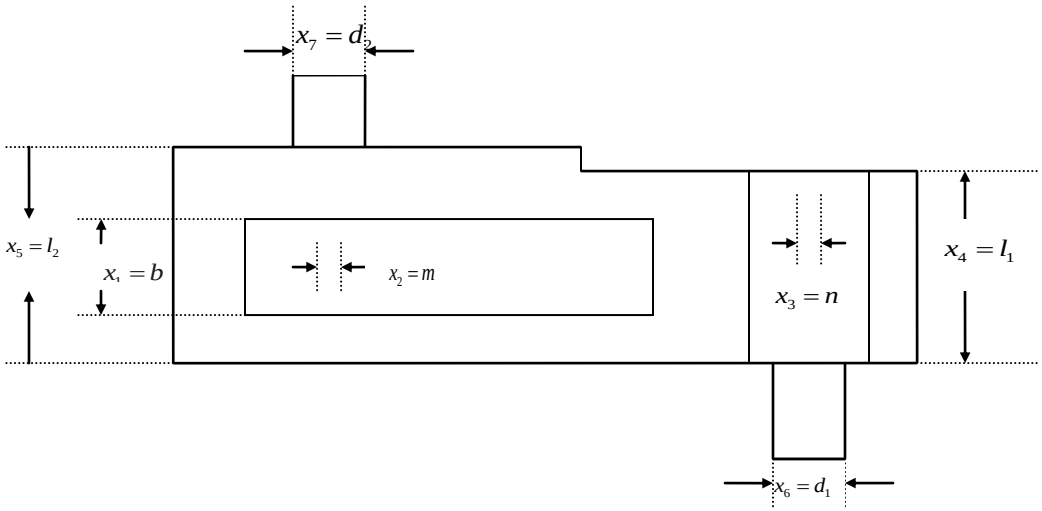


Fig.6.9 Speed reducer design.

Design Variables	Definition	Unit	Remarks	Discrete Length
x_1	face width (b)	*inch*	discrete	0.1
x_2	module of teeth (m)	*inch*	discrete	0.1
x_3	number of teeth on pinion (n)	*inch*	integer	1
x_4	length of shaft 1 between bearings (l_1)	*inch*	discrete	0.1
x_5	length of shaft 2 between bearings (l_2)	*inch*	discrete	0.1
x_6	diameter of shaft 1 (d_1)	*inch*	discrete	0.01
x_7	diameter of shaft 2 (d_2)	*inch*	discrete	0.01

Table 6.11 Design variables for a speed reducer.

This problem was investigated by Li and Papalambros [98], Azarm and Li [99], and Rao and Xiong [67]. The results in [98, 99] violate the fifth and the eleventh constraints which lead to infeasible solution. The best-known result was obtained by [67] using a hybrid genetic algorithm. In our algorithm, we have used a population size of *N=20*, and the maximum number of search iterations was set to *T=100*. As shown in Table 6.12, HPB found an optimal objective function value of *2,998.6*. The mean fitness value for *50* independent runs was *3,044.16* with a standard deviation of *57.9806*. The run that resulted in the best objective function value performed *3,029* function evaluations and required *7.6* seconds of CPU time. The over all computational results have been shown in Table. 6.13, while Fig.6.10 shows a plot of the performance of the HPB algorithm.

Quantity	*Li* [98]	*Azarm* [99]	*MDHGA* [67]	HPB
x_1	3.5	3.5	3.5	3.50
x_2	0.7	0.7	0.7	0.70
x_3	17	17	17	17.0
x_4	7.3	7.3	7.3	7.30
x_5	7.3	7.71	7.8	7.70
x_6	3.35	3.35	3.36	3.36
x_7	5.29	5.29	5.29	5.29
f(X)	**2,985.22**	**2,996.3**	**3,000.83**	**2,998.6**

Table 6.12 Optimal solution of speed reducer design.

Method	Iterations	Particles	Runs	Function evaluations	Mean value	Standard deviation
HPB *f(X)*= 2,998.6	100	20	50	3,029	3,044.16	57.9806

Table 6.13 Computational performance of HPB algorithm for speed reducer design problem.

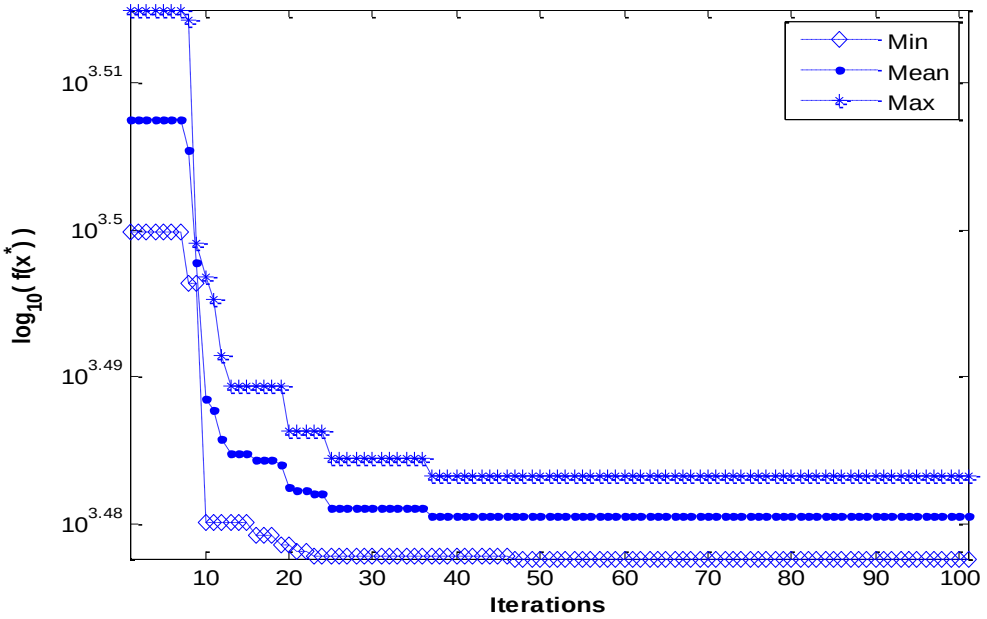


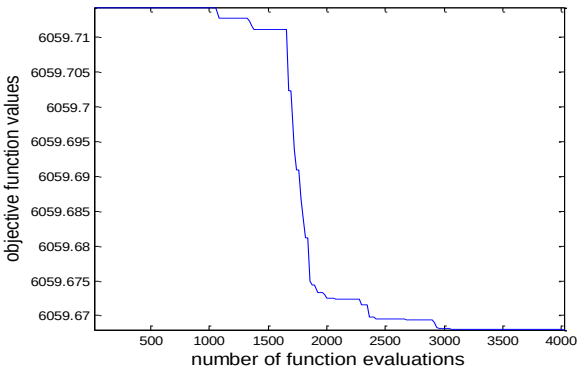
Fig.6.10 Evolution plots of speed reducer design.

6.5 Discussion

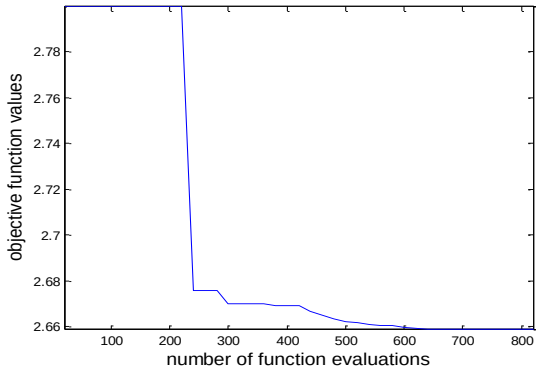
The experimental results suggest that the algorithm works well in different cases and is capable of locating the global optimum for all the problems in the present study in a reliable manner. For cases A and B, Tables 6.4 and 6.7 present the computational performance of the HPB and the improved PSO [19].

The difference in the convergence capability of HPB and PSO is apparent, and shows that the integration of the population-based evolutionary search with the global methodical search is justified. For comparison purposes, we have selected the problem of pressure vessel in case A to evaluate the performance of HPB against the PSO.

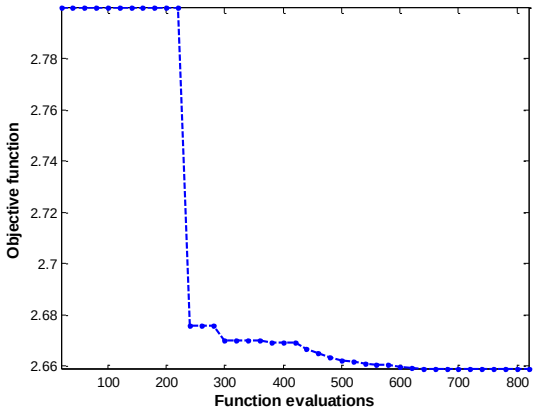
With the iteration number and the swarm size set to *1000* and *30* respectively, the proportion of PSO runs converging to global optimum in *100* executions is about *63%*, while HPB has *84%*, when the maximum iteration number is *200* with swarm size *20* and for *50* executions. Furthermore, the total number of function evaluations was reduced by *86.8%*. For case B, the optimal objective function value has been found to be nearly the same as [19], which corresponds to a much lower percentage improvement than before, but with a significant improvement in the number of function evaluations, as HPB has reduced the number of function calls by *94.4%*. Similarly, for the test cases C and D, the approach was able to achieve good results with relatively small populations and by using a relatively low number of generations. The optimum value of the objective function is found to be better than the one presented in [67].



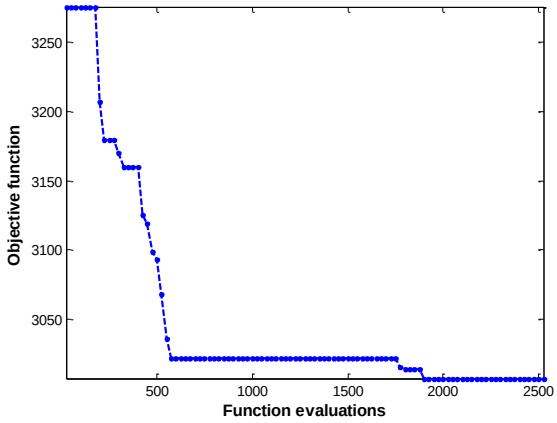
Case A: Pressure vessel design



Case B: Spring design



Case B: Spring design



Case D: Speed reducer design

Fig.6.11 The decrease of discovered objective value during HPB's optimisation process.

The main advantage of HPB is its computational efficiency. Specifically, regardless of how many iterations the algorithm is run, the number of evaluations of the fitness function, which is the most time consuming part, is reduced since the BB module is only invoked when the PSO locates a new prosperous solution. As shown in Fig.6.11, we have presented case A, B, C, and D to illustrate the convergence characteristic of HPB during the search process where it can be seen that, HPB is very efficient in terms of the number of function evaluations, because the information generated inside the hybrid algorithm can be shared by every algorithm involved. Additionally, the cooperation of PSO and BB is synchronised in such a way that it balances the frequency of invoking the expensive components and searching globally the solution space. However, the convergence speed of HPB is not the same for all the benchmark tests,

and it was much dependent on the nature of the MDNLP problem. Nevertheless, it was found that a steady solution could always be obtained for all test cases because the BB algorithm guarantees a good starting point for the PSO. Another advantage of HPB is its user-friendliness, as apart from the total number of iterations, it does not require the user to set any parameters critical to the convergence and global optimality of a particular problem.

6.6 Summary

We have introduced a hybrid algorithm which combines the characteristics of PSO and BB to facilitate an accurate and rapid search for generic MDNLP problems. The designed HPB algorithm combines the global but slow search, and the rapid but local search capabilities of the PSO and the BB, respectively, to simultaneously achieve improved optimisation accuracy and low computational resources. Additionally, it uses the fly-back method to deal with constraints, thus eliminating the penalty factors required for constraint handling, and providing initial feasible particles which lead to faster convergence. The PSO is used mainly to determine the optimal feasible region surrounding the optimum point. Then, the best known solution of all the swarm particles is chosen as a starting search point for the iterative gradient method that is subsequently used to replace PSO to find the final optimum solution. Experiments of HPB show that it outperforms other methods presented in the literature in terms of both accuracy and computational efficiency. The HPB algorithm also produces better solutions than the ones found by the BB and PSO methods when used separately, with lower number of function evaluations in each run.

Chapter 7

Conclusions and Issues for Further Work

This project has developed novel and generic optimisation methods that can be applied to a multitude of mathematically modelled business problems. It presents a derivation of the different Continuous and Mixed Discrete Nonlinear Programming algorithms that have been reported in the literature. Major theoretical properties of these methods have been presented. All the algorithms have been implemented and the numerical results of the test problems have been compared with the existing algorithms.

For the constrained global optimisation problems, different search methods have been presented in Chapters 2–3 based on stochastic and deterministic methods. Moreover, a new hybrid coevolutionary method has been invoked in chapter 4 in order to overcome the drawbacks of metaheuristics. The numerical results shown in Chapters 3–6 show that creating gradient-based techniques while applying stochastic approach in the proposed methods give better performance of metaheuristics. In addition, accelerating the final stage of the evolutionary methods by applying a complete local search technique extricates evolutionary methods from wandering around the optimal solution.

In the forth Chapter, we have developed and investigated a novel coevolutionary method for solving constrained optimisation problems through a coevolutionary game approach, we exploit the success of HCP in processing non-linear and non-convex problems. The hybridisation phase used during the evolutionary process of each sub-population is very efficient in increasing the convergence rate of the

algorithm. Furthermore, experiments of HCP show that it outperforms other methods presented in the literature in terms of both accuracy and computational efficiency. The proposed algorithm is also very suitable for parallel computation that decreases the run time required for achieving the optimum solution. It should be noted that, this method allows the use of different methods for optimising any set of variables. Our future work will focus on the techniques that combine the advantage of different evolutionary methods that may further increase the capability of the designed algorithm to tackle problems with mixed discrete-continuous variables.

In order to improve upon existing optimisation methods, the fifth Chapter examines the idea of modifying traditional AO method by developing an algorithm for solving MDNLP problems. A decomposition approach has been discussed and some computationally efficient procedures have been presented. The AO-MDNLP algorithm shows robustness in a diverse range of problems and that it can be beneficial for cases where the problem has many strongly interacting variables. The idea should be also extendable to other decomposition strategies; future work could attempt to address further decomposing or portioning subproblems in order to exploit their special structure, so that instead of having two units more units are used to hierarchically decompose the problem. For larger MDNLP problems, the performance of the AO-MDNLP algorithm is still open, where more numerical tests on considerably larger problems can be performed in order to get a more detailed picture of algorithm performance.

In the sixth Chapter, we have introduced a hybrid method which combines the characteristics of PSO and BB to facilitate an accurate and rapid search for generic MDNLP problems. The combined algorithm produces better solutions than the ones found by the BB and PSO methods when used separately, with lower number of function evaluations in each run. Future work could attempt to develop techniques that combine the advantages of different evolutionary algorithms with other deterministic methods, that may further increase efficiency and automate the termination conditions and the total number of iterations for arbitrary problems. For instance, a new hybrid optimisation algorithm that combines PSO method with a negative subgradient search technique can be developed for solving MDNLP problems. Furthermore,

because of the limitation that the current HPB cannot efficiently exploit parallel architectures since BB is invoked under exact conditions, further work is needed to adapt such hybrid algorithms for parallel architectures, so that different parts of the problem can be solved with suitable decompositions of the search space.

All in all, the author believes that the developed approaches have introduced efficient algorithms for optimisation theory. These algorithms are successfully demonstrated against real-world benchmark problems, and it is found to be highly competitive compared to existing algorithms. The effectiveness and robustness of the designed methods are validated using several engineering optimisation problems.

Bibliography

[1] C.A. Floudas, *Nonlinear and Mixed-Integer Optimisation*. London, U.K.: Oxford Univ. Press, 1995.

[2] S.S. Rao, Engineering Optimisation, *3rd ed. New York: Wiley*, 1996.

[3] J. Nocedal, and S.J. Wright, *Numerical optimisation*, NewYork: Springer-Verlag,1999.

[4] J.S. Arora, Introduction to optimum design. New York: *McGraw-Hill*, 1989.

[5] M. Duran and I.E. Grossmann, An outer-approximation algorithm for a class of mixed-integer nonlinear programs, *Math. Program.*, vol. 36, no. 3, pp. 307–339, Dec. 1986.

[6] B. Borchers and J.E. Mitchell, An improved branch and bound algorithm for mixed integer nonlinear programs, *Comput. Oper. Res.*, vol. 21, no. 4, pp. 359–367, Apr. 1994.

[7] R.E. Gomory, An algorithm for integer solutions to linear programs, in *Recent Advances in Mathematical Programming*, G. W. Graves and P. Wolfe, Eds. New York: McGraw-Hill, 1963.

[8] M.F. Cardoso, R.L Salcedo, S. Feyo de Azevedo, and D. Barbosa, A simulated annealing approach to the solution of minlp problems, *Computer Chemical. Engineering*, vol. 21, no. 12, pp. 1349–1364, 1997.

[9] M. Schoenauer and S. Xanthakis, Constrained GA Optimisation, in *Proc. 5th Int. Conf. Genetic Algorithms*, S. Forrest, Ed.,Morgan Kaufmann, pp. 573–580, 1993.

[10] J. Kennedy, and R. Eberhart, Particle swarm optimisation, in *Proc. IEEE Int. Conf. Neural Netw*, vol. 4, pp. 1942–1948, 1995.

[11] J. Kennedy and R.C. Eberhart, *Swarm Intelligence*. San Mateo, CA:Morgan Kaufmann, 2001.

[12] C.A.C. Coello and E.M. Montes, Constraint-handling in genetic algorithms through the use of dominance-based tournament selection, *Adv. Eng. Inf.*, vol. 16, no. 3, pp. 193–203, Jul. 2002.

[13] T. Back, Evolutionary Algorithms in Theory and Practice, *NewYork: Oxford Univ. Press*, 1996.

[14] S. Leyffer, Integrating SQP and branch-and-bound for mixed integer nonlinear programming, *Comput. Optim. Appl.*, vol. 18, no. 3,pp. 295–309, Mar. 2001.

[15] A.D. Belegundu, A Study of Mathematical Programming Methods for Structural Optimisation, *PhD Thesis. Department of Civil and Environmental Engineering, University of Iowa, Iowa*, 1982.

[16] G.R. Kocis, and I.E. Grossmann, Global optimisation of nonconvex mixed-integer non-linear programming (MINLP) problems in process synthesis, *Industrial & Engineering Chemistry Research*, vol. 27, pp. 1407–1421, 1988.

[17] L.Costa, and P. Oliviera, Evolutionary algorithms approach to the solution of mixed integer non-linear programming problems, C*omputers & Chemical Engineering*, vol. 25, pp. 257–266, 2001.

[18] J. Czyzyk, J. and M.P Mesnier, The Network-Enabled Optimization System (NEOS), *Computational Science & Engineering, IEEE*, vol. 5,no. 3, pp. 68–75, 1998.

[19] S. He, E. Prempain, and Q. H. Wu, An improved particle swarm optimiser for mechanical design optimisation problems, *Eng. Optim.*,vol. 36, no. 5, pp. 585–605, Oct. 2004.

[20] S. Nema, J.Y.Goulermas, G. Sparrow, and P.Cook, A hybrid particle swarm branch-and-bound (HPB) optimiser for mixed discrete non-linear programming, *IEEE Transaction Systems, Man ,Cybernetics A.*, vol. 38, no. 6, pp. 1411–1424, 2008.

[21] R. Storn, System Design by Constraint Adaptation and Differential Evolution, *IEEE Transactions on Evolutionary Computation*, pp. 1:22–34, 1999.

[22] R. Subbu, A. Sanderson, Network-Based Distributed Planning Using Coevolutionary Agents: Architecture and Evaluation, *IEEE Transaction Systems, Man, Cybernetics Part A*, vol. 34, no. 2, pp. 257–269, 2004.

[23] K.C. Tan, Y.J. Yang, C.K. Goh, A distributed cooperative coevolutionary algorithm for multiobjective optimisation, *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 527–549, 2006.

[24] X. Wang, S. Wang, J.J Ma, An improved co-evolutionary particle swarm optimisation for wireless sensor networks with dynamic deployment, *Sensors*, vol. 7, no. 3, pp. 354–370, 2007.

[25] M.A. Potter, A cooperative coevolutionary approach to function optimisation. *In: The Third Parallel Problem Solving From Nature*, pp. 249–257, 1994.

[26] F.Z. Huang, L. Wang, Q. He, An effective co-evolutionary differential evolution for constrained optimisation, *Applied Mathematics and Computation*, vol. 186, no. 1, pp. 340–356, 2007.

[27] B. Liu, H. Ma et al, A memetic co-evolutionary differential evolution algorithm for constrained optimisation, *IEEE Congress on Evolutionary Computation, Singapore, Singapore*, 2007.

[28] Q. He, L. Wang, An effective co-evolutionary particle swarm optimisation for constrained engineering design problems, *Engineering Applications of Artificial Intelligence*, vol. 20, pp. 89–99, 2007.

[29] R. Krohling, L. Coelho, Coevolutionary Particle Swarm Optimisation using Gaussian distribution for solving constrained optimisation problems, *IEEE Transaction Systems, Man, Cybernetics B*, vol. 36, no. 6, pp. 1407–1416, 2006.

[30] M.J. Tahk, B.C. Sun, Coevolutionary augmented Lagrangian methods for constrained Optimisation, *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 2, pp. 114–124, 2000.

[31] H.J. Park, J.S. Lim, and J.M. Kang, Optimisation of gas production systems using fuzzy nonlinear programming and co-evolutionary genetic algorithm, Energy Sources part A-Recovery Utilization and Environmental Effects, vol. 30, no. 9, pp. 818–825, 2008.

[32] Y.S. Son, R. Baldick, Hybrid coevolutionary programming for Nash equilibrium search in games with local optima, *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 4, pp. 305–315, 2004.

[33] A.J. Rivera, I. Rojas, J. Ortega, M.J. del Jesus, A new hybrid methodology for cooperative-coevolutionary optimisation of radial basis function networks, *Soft Computing* , vol.11, no. 7, pp. 655–668, 2007.

[34] P.E. Gill, W. Murray, and M. H Wright, Practical Optimisation, *London, U.K.: Academic*, 1981.

[35] A. Ratnaweera, S.K Halgamuge, H.C. Watson, Self-organizing hierarchical particle swarm optimiser with time-varying acceleration Coefficients, *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 240–255, 2004.

[36] M.J. Osborne, An introduction to game theory, *Oxford University Press, Oxford*, 2004.

[37] D.M. Himmelblau, Applied Nonlinear Programming, *McGraw-Hill, New York*, 1972.

[38] T.P. Runarsson, X. Yao, Stochastic ranking for constrained evolutionary Optimisation, *IEEE Transactions on Evolutionary Computation*, vol. 4, no. 3, pp. 284–294, 2000.

[39] M. Gen, R. Cheng, Genetic Algorithms and Engineering Design, *John Wiley and Sons*, 1997.

[40] C.A.C. Coello, E.M. Montes,Use of dominance-based tournament selection to handle constraints in genetic algorithms, in *Proc. ANNIE*. vol. 11, pp. 177–182, 2002.

[41] C. Coello, Use of a self-adaptive penalty approach for engineering optimisation problems, *Comput. Ind.*, vol. 41, no. 2, pp. 113–127, Mar. 2000.

[42] U.T. Ringertz, On methods for discrete structural optimisation, *Eng. Optim.*, vol. 13, pp. 47–64, 1988.

[43] E. Sandgren, Nonlinear integer and discrete programming in mechanical design optimisation, *J. Mech. Des.*, vol. 112, pp. 223–229, 1990.

[44] K. Deb, GeneAS: A robust optimal design technique for mechanical component design, in *Evolutionary Algorithms in Engineering Applications*. Berlin, Germany: Springer-Verlag, pp. 497–514, 1997.

[45] K.M. Ragsdell and D. T. Phillips, Optimal design of a class of welded structures using geometric programming, *Trans. ASME, J. Eng. Ind.*, vol. 98, no. 3, pp. 1021–1025, 1976.

[46] T. Ray, K.M. Liew, An optimisation algorithm based on the simulation of social behavior, *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 4, pp. 386–396, 2003.

[47] R.A. Haftka, Z. Gürdal, Elements of structural optimisation. *Solid mechanics and its applications*, Springer, 1991.

[48] W. Achtziger, M. Stolpe, Global optimisation of truss topology with discrete bar areas-Part II: Implementation and numerical results, *Computational Optimisation and Applications*, vol.44, no. 2, pp. 315–341, 2009.

[49] Z. Michalewics, Evolutionary algorithms for constrained parameter optimisation problems, *Evolutionary Computation*, vol. 4, no. 1, pp. 1–32, 1996.

[50] J. Lampinen, I. Zelinka, Mixed integer-discrete-continuous optimisation by differential evolution, Part 1: The optimisation method, in *Proc. 5th MENDEL*, P. Osmera, Ed., Brno, Czech Republic, pp. 71–76, 1999.

[51] L.Yiqing, Y. Xigang, and L. Yongkian, An improved PSO algorithm for solving non-convex NLP/MINLP problems with equality constraints, *Computers and Chemical Engineering*, vol. 31, pp. 153–162, 2007.

[52] C.-F. Juang, A hybrid of genetic algorithm and particle swarm optimisation for recurrent network design, *IEEE Trans. Syst., Man, Cybern. B,Cybern.*, vol. 34, no. 2, pp. 997–1006, Apr. 2004.

[53] W. Zhong, J. Liu, M. Xue, and L. Jiao, A multiagent genetic algorithm for global numerical optimisation, *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 2, pp. 1128–1141, Apr. 2004.

[54] H.T. Loh and P.Y. Papalambros, A sequential linearization approach for solving mixed-discrete nonlinear design optimisation problems, *Trans. ASME, J. Mech. Des.*, vol. 113, pp. 325–334, 1991.

[55] L. Lamberti and C. Pappalettere, An efficient sequential linear programming algorithm for engineering optimisation, *J. Eng. Des.*, vol. 16, no. 3, pp. 353–371, Jun. 2005.

[56] D. K. Shin, Z. Gurdal, and O. H. Griffin, Jr., A penalty approach for nonlinear optimisation with discrete design variables, *Eng. Optim.*, vol. 16,no. 1, pp. 29–42, 1990.

[57] K.M. Anstreicher, and H. Wolkowicz, Lagrangian relaxation of quadratic matrix constraints. *SIAM Journal on Matrix Analysis and Applications*, vol. 22,pp. 41–55, 2000.

[58] J.D. Dillon, and M.J. O'Malley, A Lagrangian augmented Hopfield network for mixed integer non-linear programming problems, *Neuro Computing*, vol. 42, pp. 323–330, 2002.

[59] J.C. Bezdek, and R.J. Hathaway, Convergence of alternating optimisation. *Neural, Parallel & Scientific Computations*, vol. 11, pp. 351–368, 2003.

[60] J.E. Gallardo, C. Cotta, and A.J. Fernandez, On the hybridization of memetic algorithms with branch-and-bound techniques, *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 1, pp. 77–83, Feb. 2007.

[61] S. Vavasis, Non-linear optimisation*: Complexity issues*. *NewYork: Oxford University Press*, 1991.

[62] M.S. Bazaraa, H.D. Sherali, and C.M. Shetty, *Non-linear programming: Theory and algorithms*. 2nd edn. NewYork:Wiley, 1993.

[63] J.C. Bean, and A.B. Hadj-Alouane,.A dual genetic algorithm for bounded integer programs. University of Michigan. Technical report.

[64] G. Nemhauser, and L.Wolsey,*Integer and combinatorial optimisation*, NewYork: JohnWiley and Sons Interscience, 1988.

[65] R.J. Hathaway, and J.C. Bezdek, Local convergence analysis of tri-level alternating optimisation, *Neural, Parallel, and Scientific Computation*, vol. 9, pp. 19–28, 2001.

[66] L.C.W. Dixon, and G.P. Szego,*The optimisation problem: An introduction*, NewYork: North Holland, 1978.

[67] S.S. Rao and Y. Xiong, A hybrid genetic algorithm for mixed-discrete design optimisation, *Trans. ASME, J. Mech. Des.*, vol. 127, no. 6,pp. 1100–1112, Nov. 2005.

[68] K. Deb and M. Goyal, Optimizing engineering designs using a combined genetic search, in *Proc. 7th Int. Conf. Genetic Algorithms*, T. Back, Ed., pp. 512–528, 1997.

[69] J.K. Koza, *Genetic Programming: On the Programming of Computers byMeans of Natural Selection*. Cambridge, MA: MIT Press, 1992.

[70] S. Impedovo, M.G. Lucchese, and G. Pirlo, Optimal zoning design by genetic algorithms, *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*,vol. 36, no. 5, pp. 833–846, Sep. 2006.

[71] S. Kirkpatrick, C.D Gelatt, Jr., and M.P. Vecchi, Optimisation by simulated annealing, *Science*, vol. 220, no. 4598, pp. 671–680,May 1983.

[72] Z. Michalewicz, *Genetic algorithms + Data Structure = Evolution Programs*, 3rd ed. New York: Springer-Verlag, 1996.

[73] K. Rodriguez-Vazquez, C. M. Fonseca, and P. J. Fleming, Identifying the structure of nonlinear dynamic systems using multiobjective genetic programming, *IEEE Trans. Syst., Man, Cybern. A, Syst., Humans*, vol. 34, no. 4, pp. 531–545, Jul. 2004.

[74] S.L. Cheng and C. Hwang, Optimal approximation of linear systems by a differential evolution algorithm, *IEEE Trans. Syst., Man, Cybern. A,Syst., Humans*, vol. 31, no. 6, pp. 698–707, Nov. 2001.

[75] H. Pierreval and J.L. Paris, Distributed evolutionary algorithms for simulation optimisation, *IEEE Trans. Syst., Man, Cybern. A, Syst.,Humans*, vol. 30, no. 1, pp. 15–24, Jan. 2000.

[76] R. Xu, G.K. Venayagamoorthy, and D.C. Wunsch, Modeling of gene regulatory networks with hybrid differential evolution and particle swarm optimisation, *Neural Netw.*, vol. 20, no. 8, pp. 917–927, Oct. 2007.

[77] J.S. Arora and M.W. Huang, Methods for optimisation of nonlinear problems with discrete variables, *Struct. Optim.*, vol. 8, pp. 69–85, 1994.

[78] J.F. Fu, R.G. Fenton, and W.L. Cleghorn, A mixed integer-discrete continuous programming method and its application to engineering design optimisation, *Eng. Optim.*, vol. 17, no. 4, pp. 263–280, 1991.

[79] S. Kitayama, M. Arakawa, and K. Yamazaki, Penalty function approach for the mixed discrete nonlinear problems by particle swarm optimisation, *Struct. Multidiscpl. Optim.*, vol. 32, no. 3, pp. 191–202, Sep. 2006.

[80] A.M. Geoffrion, Lagrangian relaxation for integer programming, *Math. Program. Stud.*, vol. 10, pp. 237–260, 1974.

[81] V. Jeet and E. Kutanoglu, Lagrangian relaxation guided problem space search heuristics for generalized assignment problems, *Eur. J. Oper. Res.*, vol. 182, no. 3, pp. 1039–1056, Nov. 2007.

[82] Y. Shi and R.C. Eberhart, Empirical study of particle swarm optimisation, in *Proc. IEEE Int. Congr. Evol. Comput.*, vol. 3, pp. 101–106, 1999.

[83] R.C. Eberhart and Y. Shi, Tracking and optimizing dynamic systems with particle swarms, in *Proc. IEEE Congr. Evol. Comput.*, Seoul, Korea, pp. 94–97, 2001.

[84] X. Hu and R. C. Eberhart, Solving constrained nonlinear optimisation problems with particle swarm optimisation, in *Proc. 6th World Multiconference SCI*, Orlando, FL, 2002.

[85] K. Parsopoulos and M. N. Vrahatis, Particle swarm optimisation method for constrained optimisation problems, in *Intelligent Technologies-Theory and Applications: New Trends in Intelligent Technologies*,ser. Frontiers in Artificial Intelligence and Applications, vol. 76, V. Sincak and J. Vascak, Eds. Amsterdam, The Netherlands: IOS Press, pp. 214–220, 2002.

[86] D. Waagen, P. Diercks, and J. McDonnell, The stochastic direction set algorithm: A hybrid technique for finding function extrema, in *Proc.1st Annu. Conf. Evol. Program.*, D. B. Fogel and W. Atmar, Eds., pp. 35–42, 1992.

[87] H. Myung, J.-H. Kim, and D. Fogel, Preliminary investigation into a two-stage method of evolutionary optimisation on constrained problems,in *Proc. 4th Annu. Conf. Evol. Program.*, MIT press, pp. 449–463, 1995.

[88] J. Joines and C. Houck, On the use of non-stationary penalty functions to solve nonlinear constrained optimisation problems with GA's, in *Proc.1st IEEE Conf. Evol. Comput.*, 1994, pp. 579–584.

[89] J. Bauer, W. Gutkowski, and Z. Iwanow, A discrete method for lattice structures optimisation, *Eng. Optim.*, vol. 5, no. 2, pp. 121–128, 1981.

[90] C.H. Tseng, L.W. Wang, and S.F. Ling, Enhancing branch-and-bound method for structural optimisation, *J. Struct. Eng.—ASCE*, vol. 121, no. 5, pp. 831–837, 1995.

[91] G.G. Dimopoulos, Mixed-variable engineering optimisation based on evolutionary and social metaphors, *Comput. Methods Appl. Mech. Eng.*,vol. 196, no. 4–6, pp. 803–817, Jan. 2007.

[92] T.A.A. Victoire, and A.E. Jeyakumar, Hybrid PSO–SQP for economic dispatch with valve-point effect, *Elect. Power Syst. Res.*, vol. 71, no. 1,pp. 51–59, 2004.

[93] Y. Gao, Z. Ren, and C. Xu, A branch and bound-PSO hybrid algorithm for solving integer separable concave programming problems, *Appl Math. Sci.*, vol. 1, no. 11, pp. 517–525, 2007.

[94] B. Liu, L. Wang, and Y-H. Jin, An effective PSO-based memetic algorithm for flow shop scheduling, *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 37, no. 1, pp. 18–27, Feb. 2007.

[95] Y.Q. Luo, X.G. Yuan, and Y.J. Liu, An improved PSO algorithm for solving non-convex NLP/MINLP problems with equality constraints, *Comput. Chem. Eng.*, vol. 31, no. 3, pp. 153–162, Jan. 2007.

[96] C.T. Young, Y. Zheng, and C.W. Yeh, Information-guided genetic algorithm approach to the solution ofMINLP problems, *Ind. Eng. Chem.Res.*, vol. 46, no. 5, pp. 1527–1537, 2007.

[97] C.J. Shih, Fuzzy and improved penalty approaches for multiobjective mixed-discrete optimisation in structural systems, *Comput. Struct.*, vol. 63, no. 3, pp. 559–565, May 1997.

[98] H.L.Li and P. Papalambros, A production system for use of global optimisation knowledge, *Trans. ASME, J. Mech. Transm. Autom. Des.*, vol. 107, no. 2, pp. 277–284, 1985.

[99] S. Azarm and W. C. Li, Multi-level design optimisation using global monotonicity analysis, *Trans. ASME, J. Mech. Transm. Autom. Des.*, vol. 111, no. 2, pp. 259–263, 1989.