THE UNIVERSITY *of* LIVERPOOL

DEVELOPMENTS OF ANIMAL BEHAVIOUR INSPIRED OPTIMISATION ALGORITHMS AND THEIR APPLICATIONS

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

in

Electrical Engineering and Electronics

by

Shan HE, B.Sc.(Eng.), M.Sc.(Eng.)

November 2006

DEVELOPMENTS OF ANIMAL BEHAVIOUR INSPIRED OPTIMISATION ALGORITHMS AND THEIR APPLICATIONS

by

Shan HE

To my parents and family to whom I owe everything.

Acknowledgements

First of all, I would like to thank my supervisor, Professor Q.H. Wu, not only for his dedicated and thorough supervision, financial support and stimulating discussions, but also for his patient guidance both in an academic and personal level.

I would also like to thank Dr. Emmanuel Prempain, who severed as my second supervisor during my first 3 years PhD study before moving to the University of Leister. I shall always cherish his invaluable discussions and suggestions. My heartfelt thanks also goes out to Professor Kai-Sheng Huang, my undergraduate project supervisor, who always supports me with his kind assistance.

My gratitude goes to all of my colleagues, especially Dr. Wenhu Tang, Dr. JunQiu Feng, Mr. Zhen Lv and Dr. Sun Pu for their kind help throughout my studies.

Finally, I am greatly indebted to my dear parents, for their encouragement, patience, understanding and love. I dedicate this thesis to my parents for their unwavering love and trust.

Abstract

The thesis begins with a brief introduction to the study of animal behaviour, which served as the inspiration for this study. Then a general introduction is given to Natural Computation and Swarm Intelligence (SI), a natural computational paradigm inspired by animal behaviour. A question then is asked: is Particle Swarm Optimiser (PSO), an optimisation algorithm which has been well accepted as a SI algorithm, really a SI algorithm? In order to answer this question, the relationship between SI, self-organisation and animal behaviour is discussed. Then a new definition of SI is given; and a new concept, Animal Behaviour inspired Optimisation (ABO), is proposed. An existing ABO algorithms, PSO is described in detail and its relationship with animal behaviour is revealed. Subsequently, the background and motivations of this research are described.

A novel ABO algorithm, Group Search Optimiser (GSO), is proposed in this thesis. Optimisation is analogous to the resource searching process of animals in nature. The GSO algorithm employs the Producer-Scrounger (PS) model, which is a generic animal social foraging model, as a framework. In order to design optimum searching strategies under this framework, concepts and strategies of resource searching from animal searching behaviour are adopted. A large set of benchmark functions, including six 300-dimensional functions, are used to assess the performance of the GSO algorithm. The differences between the GSO and evolutionary algorithms and PSO are also discussed.

The study also covers the development of PSO based on the knowledge gained from observing animal aggregation. The PSO algorithm is inspired by the aggregation behaviour of animals such as the schooling of fish and the

flocking of birds. In this thesis, passive congregation, which is a type of biological mechanism that allows animals to aggregate into groups, is introduced to the standard PSO algorithm to improve its performance. Experimental results from ten 30-dimensional benchmark functions are also presented in comparison to three standard PSO variants.

The second part of the thesis is devoted to the applications of the ABO algorithms to real-world problems. The first application is a novel artificial neural network (ANN) training algorithm based on the GSO algorithm. In this thesis, a GSO-based ANN (GSOANN) training algorithm is proposed to overcome the difficulties faced by the traditional gradient-based ANN training algorithms. The performance of the GSOANN then is assessed using four real-world classification problems and one forecasting problem. The second application is mechanical design optimisation problems. Five problems including one nonlinear programming benchmark function and four mechanical design optimisation problems are successfully solved by the proposed algorithm. The third engineering application of ABO algorithms is to Optimal Power Flow (OPF) problems. The OPF problems are mixed-variable constrained optimisation problem. In order to solve the OPF problems, the PSOPC and GSO algorithms have been applied.

Finally, a systematic summary is presented, and future research work is suggested.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is concerned with the development of animal behaviour inspired optimisation algorithms and their applications to engineering optimisation problems. This chapter explains the historical background and concepts of animal behaviour; clarifies the relationship between animal behaviour, swarm intelligence, and self-organisation; gives a new concept of animal behaviour inspired optimisation; introduces the motivations behind this study; and summarises of the contributions from this research. The layout of the thesis and auto-bibliography are also given at the end of the chapter.

1.1 Introduction to Animal Behaviour

The scientific study of animal behaviour includes everything we can observe the animals doing, from all the static postures and active movements to all the noises and smells and the changes of colour and the shapes that characterise animal life. The animals studied include single-celled organisms, invertebrates, fish, amphibians, reptiles, birds, and mammals. The study of animal behaviour involves a variety of approaches. In [4], Niko Tinbergen, a pioneer ethologist, asked four main questions about animal behaviour.

1. Why do animals respond to environment stimuli in a particular way?
2. Why do animals respond to internal stimuli in a particular way?
3. Why do some animals respond in one way and others in another way to

the same situation?

4. Why do animals of a particular species, or group, characteristically behave in particular ways in particular situations?

These four questions actually reflect different facets of the research in animal behaviour. The first and the second questions are concerned with how do psychological and physiological mechanisms control behaviour. In other words, what are the psychological and physiological causes of behaviour? Researchers interested in this question are mainly concerned with both the external stimuli that affect behaviour, and the internal hormonal and neural mechanisms that control behaviour. The third question focuses on how these mechanisms develop within individuals and the adaptive value of a behavioural trait. Researchers try to answer the third question by investigating the functions of behaviour which include its immediate effects on animals and its adaptive value in helping animals survive or reproduce successfully in a particular environment. They are also interested in how the development of behaviour pertains to the ways in which behaviour changes over the lifetime of an animal, and how these changes are affected by both genes and experience. The fourth question can be paraphrased to ask, "How did behavioural traits originate and evolve in animals?" To answer this question, researchers investigate the evolution of behaviour as it relates to the origins of behaviour patterns and how these change over generations.

According to the Animal Behaviour Society, the research in animal behaviour can be roughly divided into four broad fields: ethology, comparative psychology, behavioural ecology, and anthropology, although these disciplines overlap greatly in their goals, interests, and methods. We will give a brief introduction to these fields and discuss the similarity and differences between them.

Ethology, according to the Merriam-Webster dictionary, is the scientific and objective study of animal behaviour especially under natural conditions. The research in ethology is concerned primarily with an animal's genetically-programmed behaviours often referred to as instincts. Actually, animal be-

haviour, at its earlier stage, was usually limited to ethology. The origins of animal behaviour can be traced back in the work of eighteenth century naturalists such as Gilbert White (1720-1793) and Charles Leroy (1723-1789). However, it was Charles Darwin (1809-1882) who laid the foundation of ethology. Because of Darwin's theory of natural selection, ethologists have been particularly concerned with the evolution of behaviour and the understanding of behaviour in terms of the theory of natural selection. The research of modern ethology revolves around two important discoveries made by Nobel prize winner Konrad Lornz (1903-1989). The first discovery was fixed action patterns (FAPs) which are instinctive responses that would occur reliably in the presence of identifiable stimuli. Much of the research focuses on problems in animal communication which can be mediated by a few simple FAPs. Another important discovery is imprinting, a specialized type of "programmed learning" observed in many higher animals such as young nidifugous birds and mammals. A central concept complementary to imprinting is the innate release mechanism, whereby organisms are genetically predisposed to be especially responsive to certain stimuli such that imprinting will become fixed on the parents.

Comparative psychology refers to the study of the behaviour and mental life of animals other than human beings. Comparative psychology is sometimes referred to by the less often used but more accurate name of "animal psychology". Comparative psychology was founded in the late nineteenth century by George Romanes (1849-1894), inspired by Charles Darwin, and was further developed as an important discipline within academic psychology by the experiments on instrumental learning of Edward L. Thorndike (1874-1949) and on classical conditioning by Ivan Pavlov (1849-1936). Unlike ethology, comparative psychology, which is also concerned with the regulation and functions of behaviour, can be seen as a branch of psychology. Moreover, early comparative psychologists concentrated on the study of learning and thus tended to look at behaviour in artificial situations. Comparative psychology usually involves the use of a comparative method in which similar studies are carried out on animals of different species, and the results interpreted in terms of their different

phylogenetic or ecological backgrounds [5]. The research in comparative psychology focuses on behaviour, cognition, perception, and social relationships of diverse species from a comparative perspective.

Compared to ethology and comparative psychology, behavioural ecology is a fairly new research area that evolved from ethology in the 1960s and early 1970s. Behavioural ecology is so named because "the way in which behaviour contributes to survival and reproduction depends on ecology" [6]. As a branch of evolutionary biology, behavioural ecology share the same interest in explaining how a behavioural characteristic observed today is likely to have been shaped by natural selection [7]. The evolutionary persistence of a trait depends upon its contribution to the survival and reproduction of the individual carrying the trait [7]. Therefore, the research in behavioural ecology focuses not only on animals' behaviours to survive by exploiting resource and avoiding predators, but also on how the roles of behaviour contribute to reproductive success from ecological and evolutionary perspectives. From this aspect, behavioural ecology is dealing with Tinbergen's fourth question as discussed above. Since an individual animal's behaviour is critically important to its ability to survive and reproduce, natural selection will tend to result in animals that are, for example, efficient foragers, efficient avoiders of predators, efficient copulators, and efficient parents [8]. The behavioural strategies of all living animals should be regarded as optimal to some degree [9].

To analyze behavioural strategies, behavioural ecology, as have other areas of evolutionary biology, has employed a number of techniques that are used in optimisation theory. The research focal points of behavioural ecology include: (1) foraging behaviour; (2) territoriality, that is, behaviour to defend a given territory against other animals, usually of the same species; and (3) group living [7].

Anthropology is the scientific study of the origin, the behaviours, and the physical, social, and cultural development of human beings. Anthropology traditionally can be divided into four research fields:

1). physical anthropology, sometimes called biological anthropology, which

studies primate behaviour, human evolution, and population genetics;

2). cultural anthropology, usually called social anthropology in the United Kingdom and now often known as socio-cultural anthropology. Cultural anthropologists study areas such as social networks, diffusion, social behaviour and kinship patterns;

3). linguistic anthropology study areas include variations in language across time and space, the social uses of language, and the relationship between language and culture; and

4). archaeology, which studies the material remains of human societies.

In the past few decades, studies in animal behaviour have provided researchers in artificial intelligence with fertile inspirations. For example, the study of ethology, especially the discovery of fixed action patterns, inspired behaviour-based robotics [10] [11] which uses relatively internal variable states to model the environment. Recently, researchers have gleaned ideas from behavioural ecology to design optimisation algorithms. Ant foraging behaviour has served as an inspiration of Ant Colony Optimiser (ACO) algorithm. Group living behaviour, especially animal aggregation behaviour, inspired Particle Swarm Optimiser. In the next section, we will give a comprehensive introduction to the animal behaviour inspired optimisation algorithms.

1.2 Animal Behaviour Inspired Optimisation Algorithms

In this section, Natural Computation, which is an umbrella theme for many artificial intelligence techniques including Swarm Intelligence (SI), is introduced. Then a brief introduction to SI is given and its relationship to animal behaviour and self-organisation is discussed. A new definition of SI is given and a new concept, Animal Behaviour inspired Optimisation (ABO) Algorithms, is proposed. Finally, particle swarm optimiser, an ABO algorithm studied in this thesis, is described in detail.

1.2.1 Natural computation

Human beings have always drawn information and inspiration from nature to guide their assessments of how things work in their world. One notable example is the now ubiquitous Velcro, which was inspired by burrs. Swiss amateur inventor, Georges de Mestral, while pondering how to rid his dog's hair of the clinging burrs, concluded that the burrs might be a good model for fastening fabrics together. The result was that, in 1948, Velcro was invented.

For millions of years, nature has been doing a great job of solving complex problems. Due to evolutionary pressure, natural systems were forced to come up with highly optimised and efficient solutions to sustain life. Therefore, the transfer of problem-solving approaches from lifeforms in nature to synthetic constructs is helpful. There is a research field called biomimetics which aims at applying methods and systems found in nature to the study and design of engineering systems and modern technology. With the advance of computer science, researchers have taken the idea further by simulating natural process to solve computational problems in silico. This emerging field is called Natural Computation.

According to [12], Natural Computation can be divided into three main branches: 1) Computing inspired by nature, also known as natural computation which draws inspiration from nature to develop problem solving techniques for complex problems; 2) The simulation and emulation of nature by means of computing which aims at creating patterns, forms behaviours and organisms to mimic various natural phenomena by synthetic processes, thus increasing our understanding of nature and insights about computer models; and 3) Computing with natural materials which uses natural materials to perform computation for the purpose to substitute or supplement the current silicon-based computers.

This study falls into the first branch, *i.e.*, computing inspired by nature. There are several paradigms in this branch:

- Evolutionary computation (EC)

- Artificial neural networks (ANNs)
- Artificial immune systems (AISs)
- Swarm intelligence (SI)

EC generally involves techniques that are used to implement mechanisms inspired by evolution such as reproduction, mutation, recombination, natural selection and survival of the fittest. EC comprises Genetic Algorithms (GAs), evolutionary programming, evolution strategy, genetic programming, and learning classifier systems. In the past few decades, EC has been widely used to solve various scientific and engineering problems [13], due to their simplicity and flexibility [14]. ANNs were designed to simulate biological neural networks. They have been used to model complex relationships between inputs and outputs or to find patterns in data. AISs can be regarded as a type of optimisation algorithm inspired by immune systems, especially vertebrate immune systems. AISs are very similar to GAs but exploit the acquired immune system's characteristics of learning and memory to solve a problem. A detailed introduction of SI are given in the following section.

1.2.2 Swarm intelligence, self-organisation and animal behaviour

Current definitions of swarm intelligence

The expression "swarm intelligence" was coined by Beni and Wang in 1989 [15]. There is no commonly accepted definition of Swarm Intelligence (SI). As defined in [16]:

SI is *"an artificial intelligence technique based around the study of collective behavior in decentralized, self-organized systems."*

From the book *Swarm Intelligence: From Natural to Artificial Systems* [17]:

SI is *"the property of a system whereby the collective behaviors of (unsophisticated) agents interacting locally with their environment cause coherent functional global patterns to emerge."*

Later, one of the authors E. Bonabeau proposed a new definition [18]:

SI is the collective behaviour that emerges from a group social insects of which the team work is largely self-organised, and coordination arises from the different usually primitive interactions among individuals.

As summarised in [16], SI systems typically consist of a population of agents interacting with each other and with their environment using simple local rules. Normally there is no centralized control to dictate how individual agents should behave, rather, simple local interactions between such agents often lead to the emergence of complex global behavior. Examples of SI systems can be found in nature which include ant colonies, bird flocking, fish schooling, animal herding, and bacteria molding.

According to [16] and also generally accepted by most of the researchers in SI, the most prominent components of SI are Ant Colony Optimiser (ACO) and Particle Swarm Optimiser (PSO), both of which are based on observations of collective animal behaviour. ACO is inspired by real ants' foraging behaviour. In the ACO algorithm, artificial ants build solutions by moving on the problem graph and depositing artificial pheromone on the graph so that future artificial ants can build better solutions [16]. ACO has been successfully applied to a number of difficult optimisation problems, e.g., traveling salesman problems. PSO is another well-known SI algorithm which glean ideas from animal aggregation behaviour. Artificial life models, such as BOID, which can mimic animal aggregation vividly, serve as the direct inspiration of PSO. In the PSO algorithm, a set of individuals called particles fly in the N dimensional space in order to find the global minimum. Each particle has its own velocity determined by two factors, the best position it previously visited so far and the best position found by its neightbours (the local version of PSO) or the whole swarm (the global version of PSO). Then the individual updates its position according to the velocity. The PSO algorithm is particularly attractive to practitioners because it has only a few parameters to adjust. In the past few years, the PSO algorithm has been successfully applied in many areas. Although widely accepted as a SI algorithm by most of the researchers,

in this thesis, we ask:

Is a Particle Swarm Optimiser a real Swarm Intelligence algorithm?

Answer: **the global version of PSO is not a SI algorithm.**

Although there are different definitions of SI, there is one thing in common: the characteristic of SI which distinguishes it from other natural computational techniques is self-organisation. There are many different definitions of self-organisation across different disciplines from biology, cybernetics, thermodynamics and mathematics. The traditional definition from cybernetics is that the self-organising systems consist of four basic ingredients [19]:

1. Positive feedback
2. Negative feedback
3. Balance of exploitation and exploration
4. Multiple interactions

Since SI has its origin in biology, we are more interested in the definition from a biological point of view. In [20], self-organisation is defined as:

"a process in which a pattern at the global level of a system emerges solely from numerous interactions among the lower-level components of the system. Moreover, the rules specifying interactions among the system's components are executed using only local information, without reference to the global pattern."

In self-organising biological systems, there is no guidance from well-informed leaders, and no set of predetermined blueprints, recipes or templates to explicitly specify the pattern [20]. Instead, structure is as an emergent property of the dynamic, local interactions among components in the system. [1] The way in which the individuals interact in SI and self-organising systems provides them the advantages of robustness, flexibility and capability of scaling to enormous sizes.

[1] The environment is also a lower-level component in the systems as defined in [20].

From the description above, we can see the ACO algorithm satisfies the criterion of self-organising systems. Each ant interacts locally with environment by depositing artificial pheromone on the problem graph. Each ant chooses routes determined by the pheromone laid by other ants. There is no central control and external management of how an ant should move. The global complex behaviour of finding optimal solutions emerges from ants' simple local interaction with environment.

The PSO algorithm is inspired by animal swarm behaviour, *e.g.*, bird flocking and fish schooling. The swarm behaviour is self-organising and emerges from a few local interaction rules. However, for optimisation purpose, in the PSO algorithm, informed members who possess the best position are used to guide the whole swarm to the global minimum. It is unlikely for an individual to recognise the best particle at the swarm level only based on local information. Therefore, the best particle, especially in the global version, is "selected" from the swarm [21], not emerged from local interactions between individuals or between individuals and their environment. This "selection" is essentially an external source to guide the whole swarm. Therefore, we argue that the global version of PSO does not have the self-organising feature, and is not a SI algorithm. For the local version of PSO, the best particle is taken from some smaller number of adjacent particles of the population which is local and can be regarded as local interactions, *e.g.*, competitions, between individuals. Therefore the local version of PSO is essentially a SI algorithm.

Swarm Intelligence = Self-organisation + Inspiration from animal behaviour

Here we argue that the current definition of SI on Wikipedia is redundant and inaccurate. As defined on Wikipedia, SI is "an artificial intelligence technique based around the study of collective behavior in decentralised, self-organised systems." This definition has several problems. First of all, the term "decentralized" actually is an important property of self-organised systems. Therefore, "decentralised" is redundant when accompanied with "self-

organised". Secondly, the phrase "based around" only describes the SI's theoretical foundation which is self-organised systems, but does not depict explicitly the characteristic of SI. Moreover, besides the self-organising biological systems such as swarm of social insects or flock of birds, examples of self-organised systems also include magnetism, crystallization, lasers, cellular autocatalysis in chemical and physical systems [22]. However, from the American Heritage Dictionary, the word "swarm" often referred to an aggregation of animals, *e.g.*, ants and birds. Therefore, using only "self-organising" to define SI is too broad and may cause confusion. One example is Cellular Evolutionary Algorithms (CEAs) [23] which are also called diffusion or fine-grained models. CEAs are based on a spatially distributed population in which genetic interactions may only take place in a small neighborhood of each individual. The selection process also takes place in a small set of adjacent individuals which is similar to the local version of PSO. Therefore, CEAs are a self-organised system and also strictly fit the current definition of SI. There are some other optimsation algorithms that also possesses the self-organising feature, *e.g.*, Stochastic diffusion search (SDS) [24], Evolutionary Diffusion Optimisation (EDO) [25], etc. However, to simply catergorise these algorithms as SI deviate the meaning of swarm and may cause confusion.

So is the definition of SI in [17] better? If we compare this definition to the definition of self-organisation above [20], it is not difficult to notice that this definition of SI is indeed a simplified version of the definition of self-organisation. E. Bonabeau's new definition restricts the self-organisation to social insects. However, with the invention of local version of PSO, this definition is too narrow and cannot reflect current research in SI. Therefore, we here propose a new definition of SI:

Swarm Intelligence is a self-organising artificial intelligence technique inspired by animal behaviour.

1.2.3 Animal behaviour inspired optimisation algorithms

In this thesis, we also coin a new term: Animal Behaviour inspired Optimisation (ABO) which refers to a broad range of optimisation algorithms inspired by animal behaviour. ABO not only includes SI algorithms but also includes those algorithms who are inspired by animal behaviour but do not belong to SI because of lack of self-organising characteristic. From this point of view, we can add more algorithms recently developed to the category of ABO algorithms.

The first one is synthetic predator search (SPS) algorithm [26] which is inspired by area-restricted searching behavior. It is not a population-based algorithm like PSO. On the contrary, it is similar to simulated annealing which is suitable for solving combinatorial optimisation problems.

Intra and intersociety interactions of animal societies, *e.g.*, human and social insect societies, have been used to design a stochastic optimisation algorithm, society and civilization algorithm (SCA) [27]. This algorithm was proposed to solve single objective constrained optimisation problems based on a formal society and the civilization model.

Bacteria, which are simple single-celled organisms, have been studied for decades. Recently, bacterial foraging behavior, bacterial chemotaxis, has served as the inspiration of two different stochastic optimisation algorithms. The first one is bacterial chemotaxis (BC) algorithm, which was based on a bacterial chemotaxis model [28]. The way of bacterial react chemoattractants in concentration gradients are employed to tackle continuous optimisation problems.

Ideals from animal behavior have also been incorporated to multi-objective evolutionary algorithms. In [29], predator-prey model from animal behavior has been used to approximate the shape of the Pareto-optimal set of multi-objective optimisation problems.

In Chapter 2, we also proposed a new ABO algorithm, Group Search Optimiser (GSO), which is inspired by animal social foraging behaviour.

For those optimisation algorithms who are self-organising but are not inspired by animal behaviour, e.g., CEAs, SDS and EDO, we coined a new term, Self-Organising Optimisation (SOO) algorithms. The relationship be-

tween ABO, SI and SOO is illustrated in Fig. 1.1.

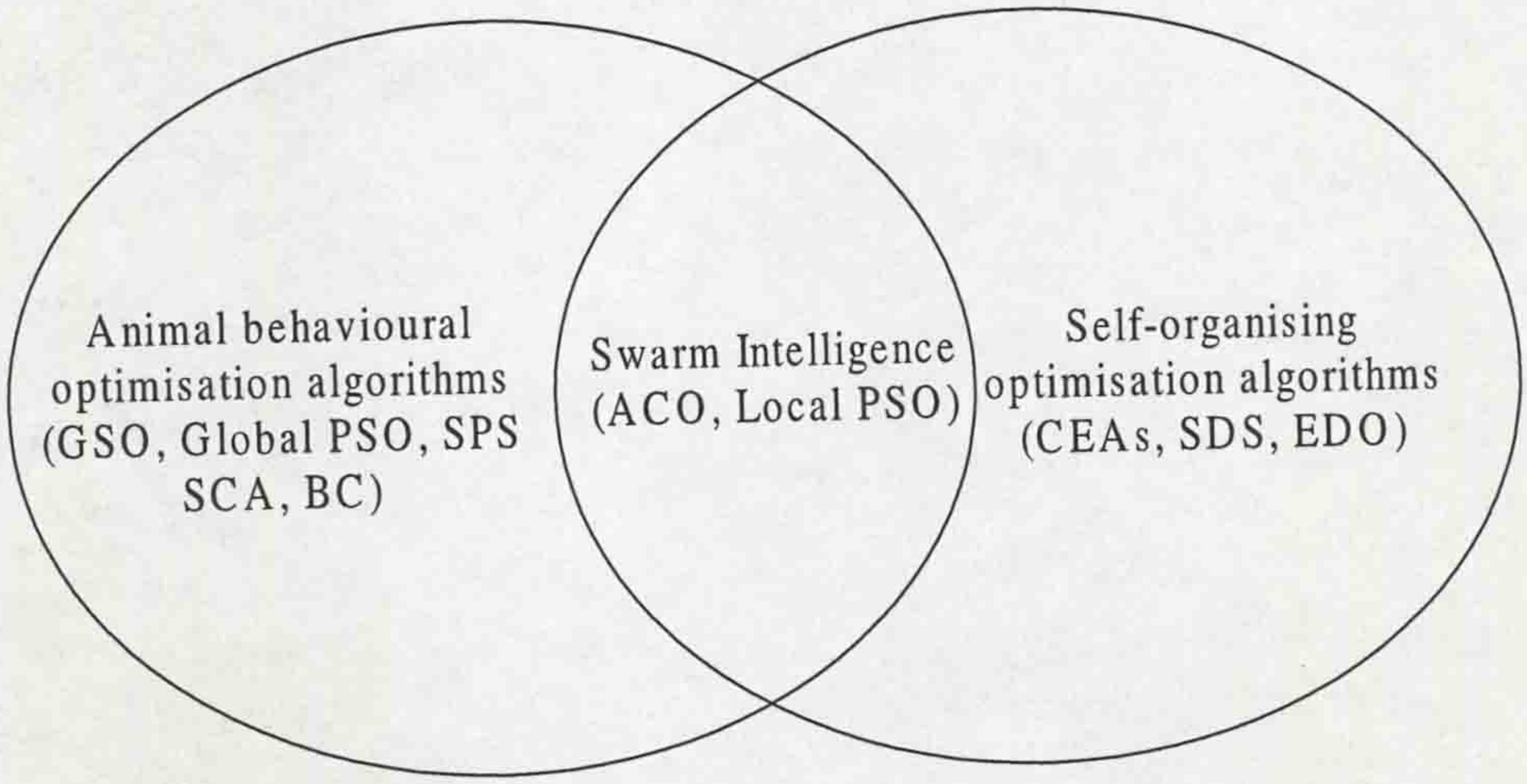


Figure 1.1: Relationship between SI, SOO and ABO algorithms.

1.2.4 Introduction to particle swarm optimiser

Before introducing Particle Swarm Optimiser (PSO), we first give a brief introduction to animal congregation behaviour and an Artificial Life (ALife) model BOID which served as a direct inspiration of the PSO algorithm.

The study of animal aggregation and BIOD

An aggregation of animals, *e.g.*, a swarm of bees, a flock of birds, or a school of fish always captures our attention. These congregations of animals are coordinated behaviourally in space and time. They move synchronously and wheel and twist in three-dimensional space, which inhibits our ability to visually focus on an individual animal and causes us instead to focus on the sum of the parts which comprises a cohesive whole [30].

In the research in animal aggregation, Parrish *et. al.* [30] proposed a set of questions which can be roughly divided into three themes. The first one deals with the basic conundrum of how a set of selfish individuals can act as a cohesive, coherent whole. The questions included in this theme are:

- What are the costs and benefits of group membership?
- What information can, and do, individuals use?
- Do individuals have a sense of the whole?
- Is there an optimal group size?

The second theme addresses the group as whole. The questions include:

- Why are there discrete boundaries?
- What is the appropriate scale for assessing pattern?
- Why should pattern exist in three-dimensional aggregations?

The third theme integrates elements of the individual with those of the group. The research attempts to define the whole as some function of the parts. The theme includes the following questions:

- What are the assembly rules?
- Which properties of the group are epiphenomena and which are functional properties that have selected for?
- Can models which predict epiphenomena be used to make predictions about individual behaviour?

These questions are depicted as "big picture" questions in [30] which define the research field of animal aggregation. These questions not only interest biologists but also computer scientists. For example, the first question in the third theme, "What are the assembly rules?" also interests researchers in artificial life. Here we will also present a brief introduction to an artificial life model which employs several simple rules to generate complex, coordinated animal motion such as bird flocks and fish schools. Actually, The Particle Swarm Optimiser algorithm was not directly inspired by animal aggregation; instead, it also originated from an artificial life model: BIOD. Therefore, we will introduce BIOD here in order to give background knowledge.

Artificial life, also known as alife or a-life, is the study of life through the use of human-made analogs of living systems. Computer scientist Christopher Langton coined the term when he held the first "International Conference on the Synthesis and Simulation of Living Systems" (otherwise known as Artificial Life I) at the Los Alamos National Laboratory in 1987. In 1986, Craig Reynolds developed a computer model of coordinated animal motion such as bird flocks and fish schools. Each individual of the model is called "boid" which is maneuvered by three simple predefined behaviours:

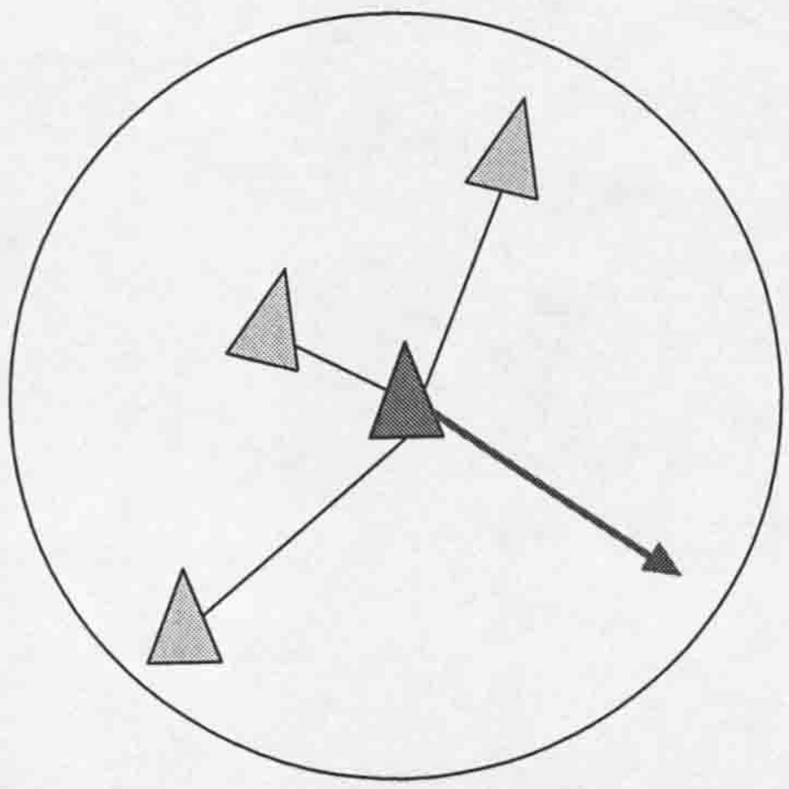


Figure 1.2: Separation.

1. Separation: steer to avoid crowding local flockmates
2. Alignment: steer towards the average heading of local flockmates
3. Cohesion: steer to move toward the average position of local flockmates

Each boid reacts only to flockmates within a small space around itself. The space is characterized by a distance, which is measured from the center of the boid, and an angle, which is measured from the boid's direction of flight. The boid will ignore flockmates outside this local neighborhood.

With these simple behaviours and control mechanisms mentioned above, complex yet organized group behaviour emerges. The group behaviour has a

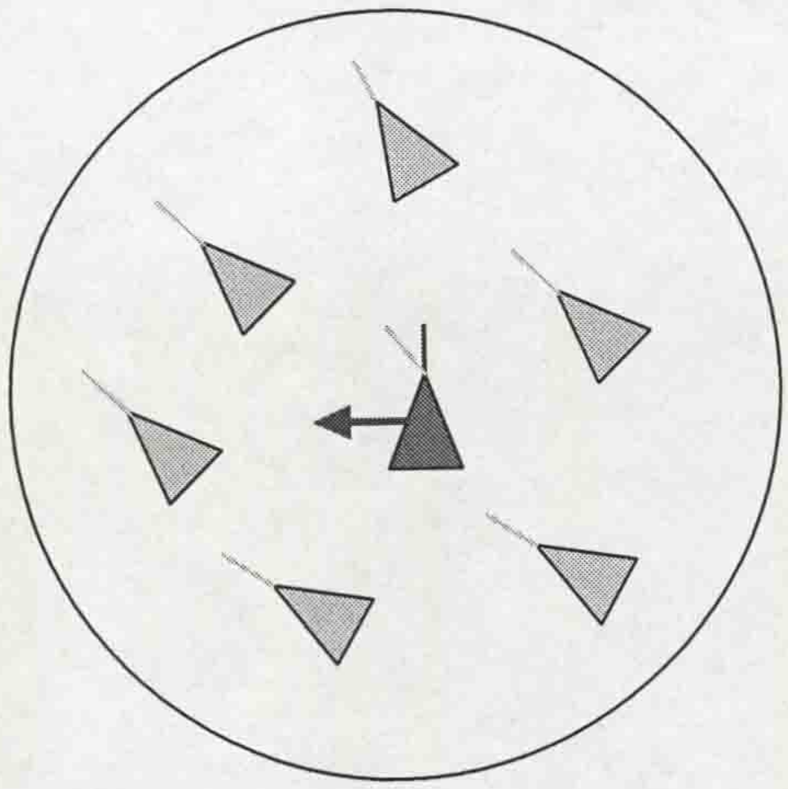


Figure 1.3: Alignment.

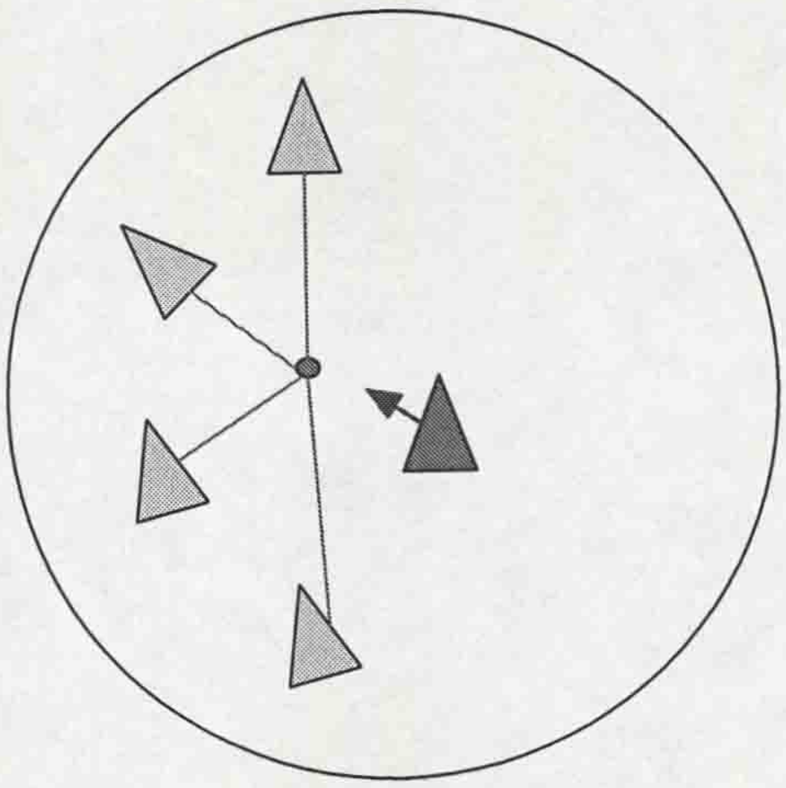


Figure 1.4: Cohesion.

chaotic aspect since each individual's simple behaviour is inherently nonlinear. However, with the negative feedback provided by the behavioral controllers, the group dynamics tend to be ordered. As a result, the group displays life-like behaviours which are characterised by unpredictability over moderate time scales.

The first and natural application of BOID is computer animation. In 1987, Craig Reynolds with his coworkers at the Symbolics Graphics Division and Whitney Demos Productions made the first computer animation film based on BIOD: Stanley and Stella in: Breaking the Ice. This film was first shown at the Electronic Theater at SIGGRAPH '87. Since then, mainstream film makers have adopted BOID to create computer animation. The first Hollywood film that employed BIOD was Batman Returns directed by Tim Burton. Other famous films that used BIOD include Disney's "The Lion King" and "The Hunchback of Notre Dame".

Apart from computer animation, BIOD has many other applications and also has spawned some novel research fields. One example is robotics. Directly or indirectly inspired by BIOD, researchers in robotics adopted the concept of group behaviour to control a group of robots. Other examples include the design of coordinated groups of aircraft or spacecraft and data visualization [31]. The most distinguished example is Particle Swarm Optimiser (PSO) which is a continuous optimisation algorithm inspired by BOID. We will describe PSO in detail in the following section.

Particle Swarm Optimiser (PSO)

PSO is a nonlinear stochastic optimisation technique developed by Dr. Eberhart and Dr. Kennedy. As they mentioned in their seminal paper published in 1995, "the method was discovered through simulation of a simplified social model." It was inspired by "computer simulations of various interpretations of the movement of organisms in a bird flock or fish school", especially BIOD. The population of PSO is called a *swarm* and each individual in the population of PSO is called a *particle*. The i_{th} particle at iteration k has the

following two attributes:

1) A current position in an N-dimensional search space $X_i^k = (x_1^k, \ldots, x_n^k, \ldots, x_N^k)$, where $x_n^k \in [l_n, u_n], 1 \leq n \leq N, l_n$ and u_n are lower and upper bounds for the n_{th} dimension, respectively.

2) A current velocity V_i^k, $V_i^k = (v_1^k, \ldots, v_n^k, \ldots, v_N^k)$, which is bounded by a maximum velocity $V_{\max}^k = (v_{\max,1}^k, \ldots, v_{\max,n}^k, \ldots, v_{\max,N}^k)$ and a minimum velocity $V_{\min}^k = (v_{\min,1}^k, \ldots, v_{\min,n}^k, \ldots, v_{\min,N}^k)$.

In each iteration of PSO, the swarm is updated by the following equations [32]:

$$V_i^{k+1} = \omega V_i^k + c_1 r_1 (P_i^k - X_i^k) + c_2 r_2 (P_g^k - X_i^k) \tag{1.2.1}$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \tag{1.2.2}$$

where P_i is the best previous position of the i_{th} particle (also known as *pbest*). According to the different definitions of P_g, there are two different versions of PSO. If P_g is the best position among all the particles in the swarm (also known as *gbest*) such a version is called the global version. If P_g is taken from some smaller number of adjacent particles of the population (also known as *lbest*) such a version is called the local version. P_i and P_g are given by the following equations respectively:

$$P_i = \begin{cases} P_i & : \quad f(X_i) \geq P_i \\ X_i & : \quad f(X_i) < P_i \end{cases} \tag{1.2.3}$$

$$P_g \in \{P_0, P_1, \ldots, P_m\} | f(P_g) = \min(f(P_0), f(P_1), \ldots, f(P_m)) \tag{1.2.4}$$

where f is the objective function, $m \leq M$ and M is the total number of particles, r_1 and r_2 are elements from two uniform random sequences in the range $(0,1)$: $r_1 \sim U(0,1)$; $r_2 \sim U(0,1)$, and ω is an inertia weight [33], which is initialized typically in the range of [0,1]. A larger inertia weight facilitates global exploration and a smaller inertia weight tends to facilitate

local exploration to fine-tune the current search area [34]. The variables c_1 and c_2 are acceleration constants [35], which control how far a particle will move in a single iteration.

Recent Advances in PSO

Since its introduction in 1995, PSO has been intensively studied by researchers around the world. The current research trends can be categorized into five parts: algorithms, topology, parameters, merging/combination with other algorithms, and applications.

One of the important developments of the standard PSO algorithm is the constriction factor approach PSO (CPSO), which was proposed by [36]. The velocity of CPSO is updated by the following equation:

$$V_i^{k+1} = \chi(V_i^k + c_1 r_1(P_i^k - X_i^k) + c_2 r_2(P_g^k - X_i^k)) \tag{1.2.5}$$

where χ is called a constriction factor, given by:

$$\chi = \frac{2}{|2 - \varphi + \sqrt{\varphi^2 - 4\varphi}|} \quad \text{where} \, \varphi = c_1 + c_2, \, \varphi > 4 \tag{1.2.6}$$

The CPSO ensures the convergence of the search procedures and can generate higher-quality solutions than the standard PSO with inertia weight on some studied problems [37]. However, mathematically speaking, CPSO is equivalent to standard PSO with an inertia weight.

In [38], van den Bergh and Engelbrecht proposed a cooperative PSO (COPSO). There are two cooperative models in their paper. The first one is called CPSO-S_k which is a direct extension of Potter's cooperative coevolutionary genetic algorithm (CCGA). The n-dimensional search space is partitioned into n one-dimensional search space. There are n swarms to optimise each partitioned one-dimensional search space. The main difference between the COPSO and the CCGA, as claimed by the authors, is that the optimisation process of a PSO is driven by the social interaction of the individuals within that swarm instead of exchange of genetic information. In contrast, "the CCGA is driven

by changes in genetic or behavioral traits within individuals of the populations". The second model is CPSO-H_k, which is a hybrid algorithm combines the CPSO-S_k and standard PSO. The results on five benchmark functions obtained by these two COPSOs are excellent.

Ratnaweera *et al.* introduced a Self-Organizing hierarchical PSO with time-varying acceleration coefficients (HPSO-TVAC) [39]. In addition to the time-varying inertia, the authors also introduced time-varying acceleration coefficients. In order to preserve the diversity of the swarm, a mutation operator was incorporated into PSO. Furthermore, in order to escape being trapped by local minima, a mechanism called "self-organising hierarchy" was proposed. Under this method, only the "social" part (P_g) and the "cognitive" part (P_i) of the particle swarm strategy are considered to estimate the new velocity of each particle and particles are reinitialized whenever they are stagnated in the search space.

Researchers also investigated the topology of PSO in order to improve its performance. In [40], Mendes et al. propose a fully informed PSO algorithm based on coefficient analysis. The authors argue that there is no assumption that the best neighbor actually found a better region than the second-best or third-best neighbors. They use the following topologies in their paper: All, where all vertexes are connected to every other; Ring, where every vertex is connected to two others; Four clusters, with four cliques connected among themselves by gateways; Pyramid, a triangular wire-frame pyramid, and Square, which is a mesh where every vertex has four neighbors that wrap around on the edges as a torus. They found that the algorithm with all the neighbors of a particle are involved in calculating the next movement has better performance than original PSO using the previous best positions.

PSO has also been extended to handle multi-objective optimisation [41]. By incorporating Pareto dominance into PSO, the algorithm stores the non-dominated vectors found so far in a second population of particles. Then these vectors will be used by the primary population of particles to update their velocities. In order to generate well-distributed Pareto fronts, an adaptive grid

is also introduced. In order to enhance the exploratory capabilities, mutation operators are employed to mutate both the particles and their dynamic ranges.

In recent years, there are more and more applications of PSO to engineering and scientific optimisation problems. For example, PSO has been employed to tackle optimisation problems in power systems, *e.g.*, reactive power and voltage control [42], optimal power flow [43] [44], economic dispatch [45], dynamic security border identification [46] and distribution state estimation [47].

1.3 Motivations and Objectives

This study primarily focuses on the ABO algorithms. In the past few years, PSO has attracted more and more attention because of its fast convergence rate and simplicity. However it also suffers some disadvantages, *e.g.*, poor global search performance. Researchers have proposed several approaches for improving the global PSO algorithms. For example, mutation operation has been introduced to PSO to help the swarm escape from local minima when search process stagnates [48] [49] [39]. Other attempts to improve the PSO algorithms include the combination of other evolutionary operators, *e.g.*, selection and crossover [50].

All the research mentioned above focused only on the algorithmic side of PSO and ignored the the important biological background of PSO. Although the algorithmic improvement of standard PSO worked efficiently for solving some problems, these improved PSO algorithms cannot be seen as a natural extension of the original PSO. On the other hand, after studying animal behaviour, the inspiration source of ABO, we found that current research in ABO only employs a small portion of research in animal behaviour, *e.g.*, social insects' swarm behaviour and ants' foraging behaviour. The research of animal behaviour remains largely unexplored by the researchers in computational intelligence. Therefore, we want to take a different approach from that of other researchers who only see the ABO algorithms, *e.g.*, PSO, from an algorithmic perspective. Firstly, we aim at extending the existing ABO algorithms,

especially PSO, by drawing inspiration from animal aggregation which is the biological root of PSO. On the other hand, optimisation, which is a process of seeking optima in a search space, can be analogous to the resource searching process of animals in nature. Shaped by natural selection, the searching strategies of all living animals are sufficient enough to survive in nature [51]. Therefore, it is natural to turn to animal behaviour, especially animal social searching behaviour to seek information for developing novel optimisation algorithms. Since the development of such algorithms would be useless unless there are reasonable applications for the algorithms, this thesis also focuses on the applications of the ABO algorithms, *e.g.*, PSO and GSO to real-world problems.

The objectives of this study therefore become twofold: 1) to develop novel ABO algorithms based on animal behaviour and 2) to apply the ABO algorithms developed in this research to the solution of real-world problems. The first objective can be divided into two stages, *i.e.*, 1) the further improvement of the existing ABO algorithms, particularly the global PSO algorithm, by transferring knowledge from animal behaviour and 2) the development of novel SI algorithms directly inspired by the research in animal behaviour untouched by other researchers.

1.4 Thesis Overview

This thesis is structured as follows:

Chapter 2 introduces a novel ABO algorithm, Group Search Optimiser (GSO), which is inspired animal group searching behavior. Based on a generic social foraging model, Producer-Scrounger model, it provides an open framework to utilize research in animal behavioral ecology to solve difficult optimisation problems. A large set of 29 benchmark functions, including six 300-dimensional large-scale benchmark functions are employed to evaluate the GSO algorithm. In this chapter, results on the 29 benchmark functions from Genetic Algorithm, Evolutionary Program-

ming, Evolution Strategies and Particle Swarm Optimiser are also given for comparison. From the comparison, it can be concluded that the GSO algorithm has competitive performance to other EAs in terms of accuracy and convergence speed, especially on high-dimensional multi-modal problems. The differences between GSO, EAs and PSO are also discussed in this chapter.

Chapter 3 describes an improved PSO algorithm with with passive congregation. In nature, passive congregation is an important biological force preserving swarm integrity. It is an attraction of an individual to other group members in which there is no display of social behavior. In this study, passive congregation is introduced to transfer information among individuals that will help individuals to avoid misjudging information and becoming trapped by poor local minima. Following the introduction and details of this algorithm, the PSO with passive congregation (PSOPC) is tested with a set of 10 benchmark functions with 30 dimensions and compared to their standard PSO variants respectively. The experimental results show that the PSO with passive congregation improves the search performance on the benchmark functions significantly.

Chapter 4 presents the application of the GSO algorithm to Artificial Neural Networks (ANNs) training. The ANN training process can be regarded as a difficult optimisation problem. In this chapter, parameters of a 3-layer feed-forward ANN, including connection weights and bias term are tuned by the GSO algorithm. Following the introduction of ANN and the GSO based ANN (GSOANN) training algorithm, four real-world classification problems and one forecasting problem are used to access the GSOANN algorithm. Four EAs-based and one gradient-based training algorithms are also implemented to solve these problems. Results from the literature on these benchmark problems are also presented in this chapter for comparison. From the comparison, GSOANN achieved better generalization performance than those of sophisticated machine learning

techniques proposed in recent year on several benchmark problems.

Chapter 5 extends standard PSO to handle problem-specific constraints and mixed variables such as integer, discrete and continuous variables. A constraint handling method called the "fly-back-mechanism" is introduced to maintain a feasible population. In order to handle mixed variables, a simple but effective scheme is employed. Then the improved PSO algorithm is applied to solve five benchmark problems commonly used in the literature of engineering optimisation and nonlinear programming. The experimental results on these five benchmark functions indicate that the proposed algorithm is better than or equal to other existing methods while requiring less computational time.

Chapter 6 begins by giving a brief literature review of Optimal Power Flow (OPF) problems in power systems. Then PSOPC and GSO are introduced to solve OPF problems. The proposed two algorithms are evaluated using an IEEE 30-bus test system. Three different OPF problems are solved by the two algorithms which include minimizing the fuel cost, improving the voltage profile and enhancing the voltage stability. Due to its superior searching performance in high-dimensional benchmark problems, the GSO algorithm then is applied to solve an OPF problem on a large-scale, practical IEEE 118-bus system.

Chapter 6 concludes the thesis based on the results obtained in this study. Ideas for future work are also listed in this chapter.

1.5 Contributions of Research

There are several major contributions arising from this research:

- The successful development of a novel ABO optimisation algorithm, Group Search Optimiser (GSO), inspired by animal social searching (foraging) behaviour;

- An improved Particle Swarm Optimiser with Passive Congregation (PSOPC). This study is inspired by animal social aggregation models;
- An extended Particle Swarm Optimiser (PSO) which is capable of handling mixed variables and constrains;
- Applications of GSO to neural networks training for data mining problems;
- Applications of extended PSO to mechanical optimal design problems;
- Applications of PSOPC and GSO to Optimal Power Flow problems.

The contributions listed above can be grouped as two categories: 1). algorithm developments of ABO algorithms and 2). real-world applications of ABO algorithms.

For the algorithm developments, this thesis describes, for the first time, a novel ABO algorithm, GSO. The algorithm is based on a generic social foraging model, Producer-Scrounger (PS) model, which is different from the metaphors used by the ACO and PSO. In order to evaluate its performance, extensive experimental study has been carried out. From the experimental results, it was found that the GSO algorithm has better search performance on large-scale multi-modal benchmark functions. Probably the most significant merits of GSO is that it provides an open framework to utilize research in animal behavioral ecology to tackle hard optimisation problems. This framework is more flexible than PSO and other Evolutionary Algorithms (EAs). For example, different local search techniques can be naturally incorporated into the framework of GSO as the searching strategies of producers. In the past, the combination of local search techniques and EAs was called "memetic" algorithms [52] or also known as Lamarckian Genetic Algorithms [53]. However, after the publication of Charles Darwin's theory of natural selection and the development of Mendelian genetics, the modern biologists have abandoned the Lamarckian theory of evolution [54]. Therefore, although the performance of memetic algorithms on many problem domains is superior to the traditional

GAs, their biological background is questionable. By employing local search techniques as producing strategies, GSO provides a more biologically sound framework than the memetic algorithms.

In addition to the development of GSO, this research has also contributed to the development of PSO. In order to improve the search performance of the standard PSO algorithm, passive congregation, which is a biological mechanism that allows animals to aggregate into groups, has been incorporated into PSO. The test results on 10 benchmark functions show that the proposed PSO algorithm has better performance than standard PSO in terms of accuracy and convergence speed. This algorithm has been employed by other researchers to tackle optimisation problems in power systems [55]. This study has also extended standard PSO which can only handle continuous unconstrained optimisation problems. The extension provided by this research effort allows the handling of mixed-variable constrained optimisation problems. The techniques used to handle mixed variable and constraints are comparatively simple but generate better results than many sophisticated methods.

The SI techniques developed in this study, e.g., GSO and PSOPC, have been applied to solve real-world problems. The first application of GSO is training Artificial Neural Networks (ANNs). Several real-world classification and forecasting problems have been solved by the GSO trained ANN. For some problems, the results we obtained in this research are the best in the literature. The GSO and PSOPC algorithms have also been used to solve optimal power flow problems. Results from IEEE 30-bus and IEEE 118-bus systems confirm that the two algorithms have better performance than other EAs and standard PSO.

1.6 Auto-bibliography

List of the publications produced from this work:

1. **S. He**, E. Prempain, and Q. H. Wu. An improved particle swarm optimiser for mechanical design optimisation problems, *Engineering Optimi-*

sation, 36 (5): 585-605, Oct. 2004.

2. **S. He**, Q. H. Wu, J. Y. Wen, J. R. Saunders and R. C. Paton, A particle swarm optimiser with passive congregation, *BioSystems*, 78 (1-3): 135-147, Dec. 2004.

3. **S. He**, E. Prempain, Q. H. Wu, J. Fitch, S. Mann. An improved particle swarm optimisation for optimal power flow, *2004 International Conference on Power Systems Technology*. Singapore. Nov. 2004.

4. W. H. Tang, **S. He**, E. Prempain, Q. H. Wu and J. Fitch. A particle swarm optimiser with passive congregation approach to thermal modeling for power transformers, *2005 IEEE Congress on Evolutionary Computation (CEC 2005)*. Edinburgh. Sep. 2005.

5. **S. He**, Q. H. Wu, and J. R. Saunders. A group search optimiser for neural network training, *2006 International Conference on Computational Science and its Applications (ICCSA 2006)*, Glasgow. May 2006. Lecture Notes in Computer Science, 3982: 934-943, Springer.

6. **S. He**, Q. H. Wu and J. R. Saunders. A Novel Group Search Optimiser Inspired by Animal Behavioural Ecology, *in Proceeding of 2006 IEEE Congress on Evolutionary Computation (CEC 2006)*. Sheraton Vancouver Wall Centre, Vancouver, BC, Canada. July 16-21, 2006. Tue PM-10-6

7. **S. He**, Q. H. Wu, and J. R. Saunders. Group search optimiser - an optimisation algorithm inspired by animal searching behaviour, 16 pages. Submitted to *IEEE Transaction on Evolutionary Computation*.

8. Q. H. Wu, **S. He**, and J. R. Saunders. Group Search Optimiser For Optimal Power Flow. 8 pages. Submitted to *IEEE Transactions on Power Systems*.

9. **S. He**, Q. H. Wu, and J. R. Saunders. Breast cancer diagnosis using a artificial neural network trained by group search optimiser, Submitted

to *Transactions of the Institute of Measurement and Control.* (Invited paper)

Part 1

Developments of Animal Behaviour Inspired Optimisation Algorithms

Chapter 2

From Animal Social Searching Behaviour to Group Search Optimiser

Nature-inspired optimisation algorithms [?] [56], notably Evolutionary Algorithms (EAs), have been widely used to solve various scientific and engineering problems [13], due to their simplicity and flexibility [14]. In this chapter we report a novel optimisation algorithm, Group Search Optimiser (GSO), inspired by animal behavior, especially animal social searching behavior. This algorithm belongs to the so-called Animal Behaviour inspired Optimisation algorithm. The framework is mainly based on the Producer-Scrounger model [57], which assumes group members search either for 'finding' (producer) or for 'joining' (scrounger) opportunities. Based on this framework, concepts from animal searching behavior, *e.g.*, animal scanning mechanisms, are employed metaphorically to design optimum searching strategies for solving continuous optimisation problems. We also disperse some group members from their current positions to perform random walks to avoid entrapment in local minima. When tested against benchmark functions, in low and high dimensions respectively, the GSO algorithm has competitive performance to other EAs in terms of accuracy and convergence speed, especially on high-dimensional multi-modal

problems.

2.1 Introduction

In the past few decades, natural computation has attracted more and more attentions. Nature serves as a fertile source of concepts, principles and mechanisms for designing artificial computation systems to tackle complex computational problems. In recent years, a new kind of computational intelligence: Swarm Intelligence (SI), which was inspired by animal collective behavior, has been developed. In Chapter 1, we have revealed the relationship between SI, animal behaviour and self-organisation. A new concept, Animal Behavioiur inspired Optimisation (ABO) has been proposed.

In this chapter, inspired by animal searching (foraging) behavior, we propose an novel ABO algorithm, Group Search Optimiser (GSO), primarily for continuous optimisation problems. GSO is mainly based on a social foraging model, Producer-Scrounger (PS) model proposed by C.J. Barnard and R.M. Sibly [57]. Under this framework, concepts and strategies of resource searching from animal searching behavior are adopted metaphorically for designing optimum searching strategies. General animal scanning mechanisms (*e.g.*, vision) are employed for producers. Scrounging strategies [57] of house sparrows (*Passer domesticus*) are used in the GSO algorithm. Besides the producer and scroungers, some group members are dispersed from a group to perform random walks to avoid entrapments in local minima.

In order to evaluate the performance of the implemented GSO algorithm, extensive studies based on a set of 23 benchmark functions have been carried out. For comparison purposes, we also implemented one evolutionary algorithms, GA, and one Swarm Intelligence algorithm, PSO, on these functions respectively. We also adopted published results of EP, ES and their improved variants, namely, Fast EP (FEP) [58], Fast ES (FES) [59], for comparison. Experimental results show that, compared to the other algorithms, GSO has markedly superior search performance for multi-modal functions, whilst main-

taining modest performance for high-dimensional uni-modal functions. The 23 benchmark functions used in our experiments have been widely employed by other researchers to evaluate evolutionary algorithms. However their dimensions are relatively small (up to 30) compared with real-world optimisation problems which usually involve hundreds even thousands of variables. In order to further investigate whether GSO can be scaled up to handle large-scale optimisation problems, we tested our GSO algorithm on 6 multi-modal benchmark functions (*e.g.*, f_8 to f_{13} studied in this chapter) in 300 dimensions in comparison to GA and PSO. The results are encouraging, the GSO algorithm generated results as good as those in 30-dimensional cases while GA and PSO yielded poor results or even failed to converge.

The rest of the chapter is organized as follows. In Section 2.3, GSO will be introduced and the details of implementation will be given. In Section 4.3, the experiment studies of the proposed GSO are presented with descriptions of the benchmark functions, experimental settings including the parameter setting of the GSO algorithm and the experimental results. The differences between GSO and other SI algorithms and EAs will be discussed in Section 3.5. The chapter is concluded in Section 4.4.

2.2 Animal Social Searching Theory

Searching (Foraging) behavior may be described as an active movement by which an animal finds or attempts to find resources such as food, mates, oviposition or nesting sites, and it is perhaps the most important kind of behavior in which an animal engages [1]. Searching behavior represents the confluence of three aspects of an animal: (1) the characteristics and abilities of an animal, including its perceptual and locomotory skills; (2) external environment factors determining what resources are available and the risks generated in obtaining them; and (3) internal factors, such as the level of physiological need relative to a certain kind of resource. The ultimate success of an animal's searching depends on [1]: (1) the strategies it uses in relationship to the availability of

resources and their spatial and temporal distributions in the environment; (2) its efficiency in locating resources; and (3) the ability of a species to adapt to long-term or even short-term environmental changes and the ability of an individual to respond. Shaped by natural selection, the searching strategies of all living animals are sufficient enough to survive in nature. For example, an animal can move in a way that optimises its chances of locating sparse, randomly located resources [60].

In animal behavioral ecology, group-living, which is a widespread phenomenon in the animal kingdom, has been studied intensively. One consequence of living together is that group searching allows group members to increase patch finding rates as well as to reduce the variance of search success [61]. This has usually led to the adoption of two foraging strategies within groups: (1) producing, *e.g.*, searching for food; and (2) joining (scrounging), *e.g.*, joining resources uncovered by others. The latter has also been referred to as conspecific attraction, kleptoparasitism, etc. [62]. Joining is an ubiquitous trait found in most social animals such as birds, fish, spiders and lions. Individuals in a group that are successful at searching resource provide resources at their expense to less successful individuals [63].

In order to analyze the optimal policy for joining, two models have been proposed: Information-Sharing (IS) [64] and Producer-Scrounger (PS) [57]. The IS model assumes foragers search concurrently for their own resource, whilst searching for opportunities to join. On the other hand, foragers in the PS model are assumed to use producing or joining strategies exclusively. Recent studies suggest that, at least for the joining policy of ground-feeding birds, the PS model is more plausible than IS mode [63].

Recently, Couzin *et. al.* [65] suggested that the larger the group, the smaller the proportion of informed individuals need to guide the group with better accuracy. Therefore, for accuracy and convenience of computation, we simplify the PS model by assuming that there is only one producer at each searching bout. The simplest joining policy, which assumes all scroungers will join the resource found by the producer, is used. In optimisation problems,

unknown optima can be regarded as open patches randomly distributed in a search space. Group members therefore search for the patches by moving over the search space [66]. It is also assumed that the producer and the scroungers do not differ in their relevant phenotypic characteristics. Therefore, they can switch between the two roles [57] [66].

Producer strategy consists of searching for one's food. An important component of search orientation is scanning; it is a set of mechanisms by which animals move sensory receptors and some times their bodies or appendages so as to capture information from the environment [1]. Scanning can be accomplished through physical contact or by visual, chemical, or auditory mechanisms. In nature, vision is the main scanning mechanism used by most of the animal species. To perform visual searches, many animals encode a large field of view with retinas having variable spatial resolution, and then use high-speed eye movements to direct the highest resolution region towards potential target locations [67] [68]. Good scanning performance is essential for survival. Najemnik and Geisler [69] showed humans use almost optimal scanning strategies for selecting fixation locations in visual search.

In nature, group members often have different searching and competitive abilities; subordinates, who are less efficient foragers than the dominant will be dispersed from the group [70] [71]. Various forms of dispersions are observed in range from simple insects to human being [72]. Dispersed animal may adopt ranging behavior to explore and colonize new habitats. Ranging is an initial phase of a search that starts without cues leading to a specific resource [73].

2.3 Group Search Optimiser

In this chapter, optimisation, which is a process of seeking optima in a search space, is analogous to the resource searching process of animals in nature. Based on the theoretical frame work presented in the previous section, we develop a GSO algorithm for continuous optimisation problems by incorporating concepts and strategies of animal searching behavior. The population of

GSO is called a *group* and each individual in the population is called a *member*.

In an n-dimensional search space, the i_{th} member at the k_{th} searching bout (iteration), has a current position $X_i^k \in \mathbb{R}^n$, a head angle $\varphi_i^k = (\varphi_{i1}^k, \ldots, \varphi_{i(n-1)}^k) \in \mathbb{R}^{n-1}$ and a head direction $D_i^k(\varphi_i^k) = (d_{i1}^k, \ldots, d_{in}^k) \in \mathbb{R}^n$ which can be calculated from φ_i^k via a Polar to Cartesian coordinates transformation:

$$\begin{aligned} d_{i1}^k &= \prod_{p=1}^{n-1} \cos(\varphi_{ip}^k) \\ d_{ij}^k &= \sin(\varphi_{i(j-1)}^k) \cdot \prod_{p=i}^{n-1} \cos(\varphi_{ip}^k) \\ d_{in}^k &= \sin(\varphi_{i(n-1)}^k) \end{aligned} \tag{2.3.1}$$

At each iteration, a group member, located in the most promising area, conferring the best fitness value, is chosen as the producer. It then stops and performs visual scanning of the environment to seek resources (optima). In our GSO algorithm, basic scanning strategies introduced by white crappie (*Pomoxis annularis*) [74] is employed. The scanning field of vision is generalized to a n-dimensional space, which is characterized by maximum pursuit angle $\theta_{\max} \in \mathbb{R}^{n-1}$ and maximum pursuit distance $l_{\max} \in \mathbb{R}^1$ as illustrated in a 3D space in Fig. 2.1. In the GSO algorithm, at the k_{th} iteration the producer X_p behaves as follows:

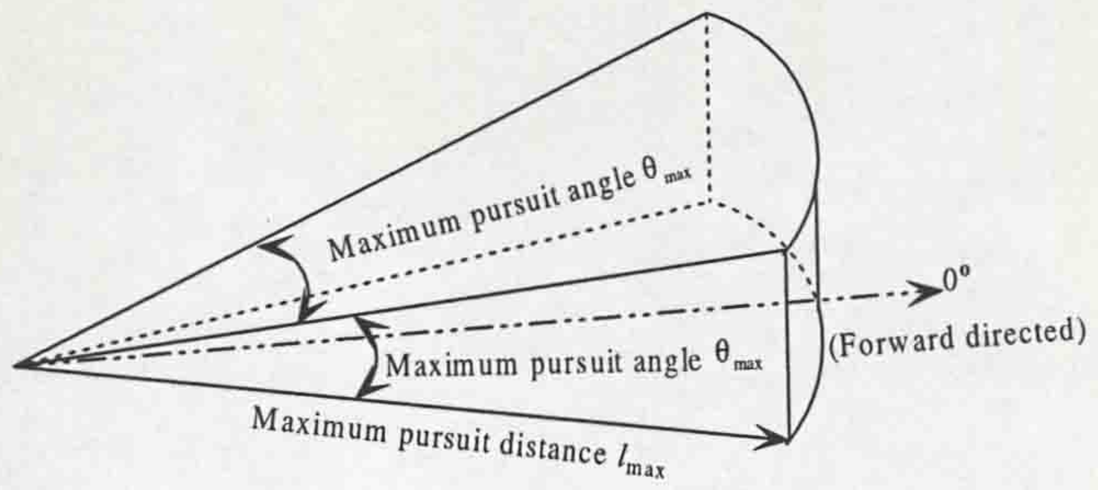


Figure 2.1: Scanning field in 3D space [1]

1) The producer will scan at zero degree and then scan laterally [74]. This is simulated by randomly sampling three points in the scanning field: one point

at zero degree:

$$X_z = X_p^k + r_1 l_{\max} D_p^k(\varphi^k) \tag{2.3.2}$$

one point in the right hand side hypercube:

$$X_r = X_p^k + r_1 l_{\max} D_p^k(\varphi^k + r_2 \theta_{\max}/2) \tag{2.3.3}$$

and one point in the left hand side hypercube:

$$X_l = X_p^k + r_1 l_{\max} D_p^k(\varphi^k - r_2 \theta_{\max}/2) \tag{2.3.4}$$

where $r_1 \in \mathbb{R}^1$ is a normally distributed random number with mean 0 and standard deviation 1 and $r_2 \in \mathbb{R}^{n-1}$ is a uniformly distributed random sequence in the range $(0, 1)$.

2) The producer will then find the best point with the best resource (fitness value) among the three points it scanned. If the best point has a better resource than its current position, then it will fly to this point. Or it will stay in its current position and turn its head to a new angle:

$$\varphi^{k+1} = \varphi^k + r_2 \alpha_{\max} \tag{2.3.5}$$

where $\alpha_{\max}$ is the maximum turning angle.

3) If the producer cannot find a better area after a iterations, it will turn its head back to zero degree:

$$\varphi^{k+a} = \varphi^k \tag{2.3.6}$$

where a is a constant.

During each searching bout, a number of group members are selected as scroungers. The scroungers will keep searching for opportunities to join the resources found by the producer. The basic scrounging strategies [57] in house sparrows (*Passer domesticus*) include: (a) Area copying: moving across to search in the immediate area around the producer; (b) Following: following another animal around without exhibiting any searching behavior; and (c) Snatching: taking a resource directly from the producer. In the GSO algorithm, only area copying, which is the commonest scrounging behavior in sparrows,

is adopted. At the k_{th} iteration, the area copying behavior of the i_{th} scrounger can be modeled as a random walk towards the producer:

$$X_i^{k+1} = X_i^k + r_3(X_p^k - X_i^k) \tag{2.3.7}$$

where $r_3 \in \mathbb{R}^n$ is a uniform random sequence in the range $(0, 1)$. The typical paths of scroungers in 5 iterations are illustrated in Fig. 2.2.

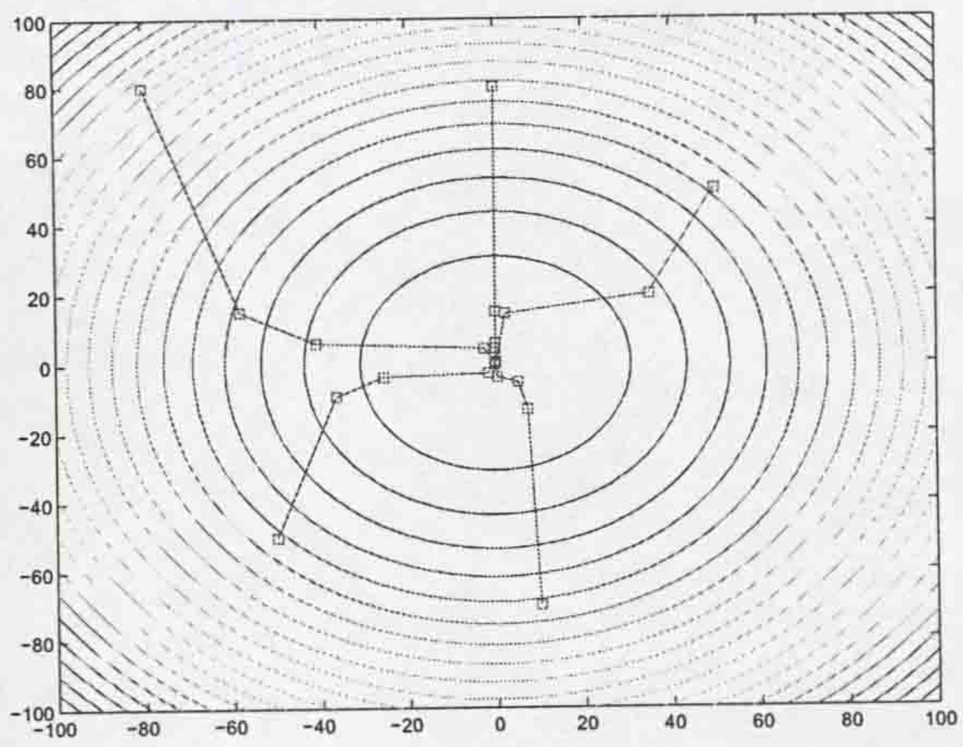


Figure 2.2: The paths of five scroungers moving towards the producer (in the center) in 5 iterations.

The rest of the group members will be dispersed. If the i_{th} group member is dispersed, it will perform ranging. In nature, ranging animals perform searching strategies, which include random walks and systematic search strategies to locate resources efficiently [75]. Random walks, which are thought to be the most efficient searching method for randomly distributed resources [60], are employed by the dispersed members. At the k_{th} iteration, (1) it generates a random head angle φ_i:

$$\varphi_i^{k+1} = \varphi_i^k + r_2\alpha_{\max} \tag{2.3.8}$$

where $\alpha_{\max}$ is the maximum turning angle; and (2) it chooses a random distance:

$$l_i = a \cdot r_1 l_{\max} \tag{2.3.9}$$

and move to the new point:

$$X_i^{k+1} = X_i^k + l_i D_i^k(\varphi_i^{k+1}) \tag{2.3.10}$$

To maximize their chances of finding resources, animals use several strategies to restrict their search to a profitable patch. One important strategy is turning back into a patch when its edge is detected [76]. This strategy is employed by the GSO algorithm to handle the bounded search space: when a member is outside the search space, it will turn back to its previous position inside the search space. The flowchart of the GSO algorithm is presented in Fig. 2.3. The pseudocode for the GSO algorithm is listed in Table 2.1.

2.4 Experimental Studies

2.4.1 Test functions

According to the No Free Lunch theorem, "for any algorithm, any elevated performance over one class of problems is exactly paid for in performance over another class" [77]. To fully evaluate the performance of the GSO algorithm without a biased conclusion towards some chosen problems, we employed a large set of standard benchmark functions which are given in Table 2.2. The set of 23 benchmark functions can be grouped into uni-modal functions (f_1 to f_7), multi-modal functions (f_8 to f_{13}), and low-dimensional multi-modal functions (f_{14} to f_{23}). Although this set of benchmark functions have been widely adopted by other researchers [58], their dimensions are chosen relatively small (up to 30) compared to that of real-world optimisation problems. It is our interest to investigate whether our GSO algorithm can be scaled up to handle large-scale optimisation problems. Therefore, multi-modal functions f_8 to f_{13}, for which the number of their local minima increases exponentially with respect to the increase of dimension, are selected and extended to 300 dimensions as listed in Table 2.3.

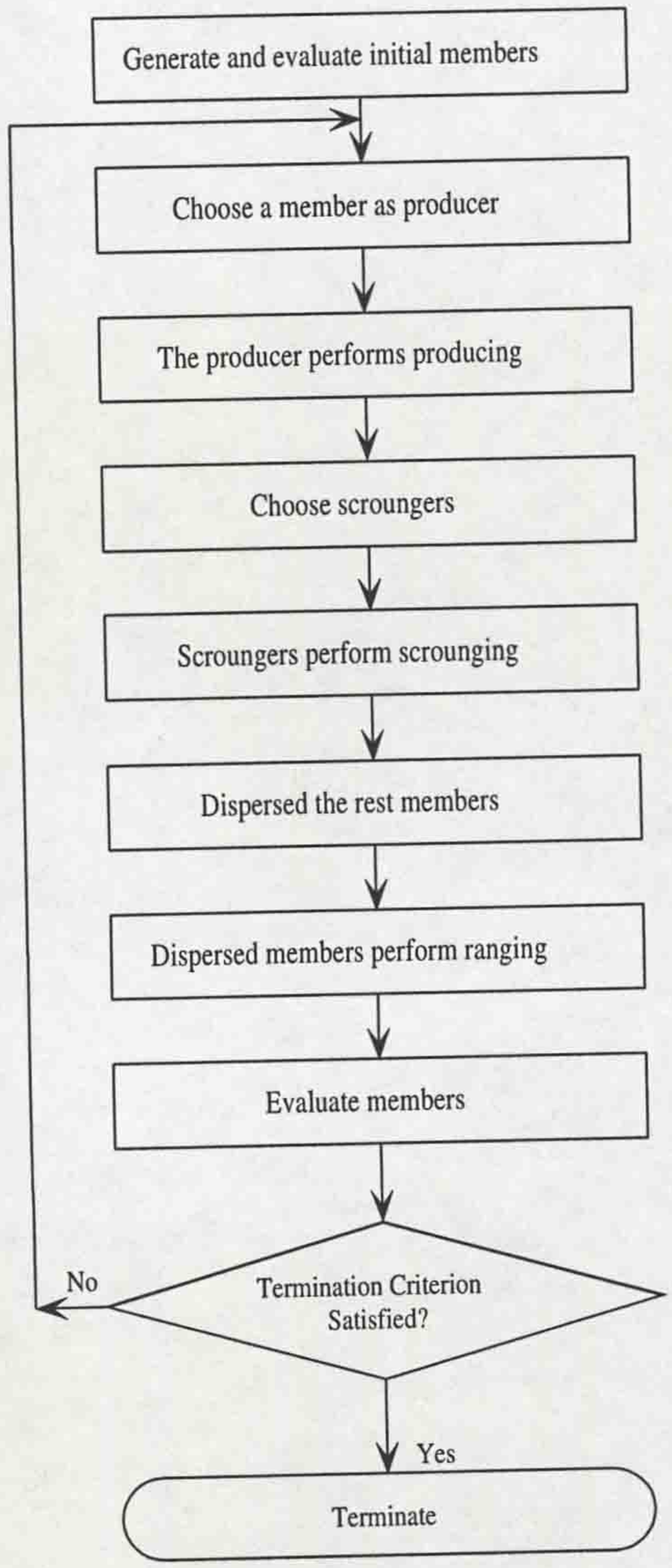


Figure 2.3: Flowchart of the GSO algorithm.

Table 2.1: Pseudo code for the GSO algorithm.

Set $k := 0$;

Randomly initialize positions X_i and head angles φ_i of all members;

Calculate the fitness values of initial members: $f(X_i)$

WHILE (the termination conditions are not met)

FOR (each members i in the group)

Choose producer: Find the producer X_p of the group;

Perform producing: 1) The producer will scan at zero degree and then scan laterally by randomly sampling three points in the scanning field using equations (2.3.2) to (2.3.4).

2) Find the best point with the best resource (fitness value). If the best point has a better resource than its current position, then it will fly to this point. Otherwise it will stay in its current position and turn its head to a new angle using equation (2.3.5).

3) If the producer can not find a better area after a iterations, it will turn its head back to zero degree using equation (2.3.6);

Perform scrounging: Randomly select 80% from the rest members to perform scrounging;

Perform dispersion: For the rest members, they will be dispersed from their current positions to perform ranging: 1). Generate a random head angle using equation (2.3.8); and 2). Choose a random distance l_i from the Gauss distribution using equation (2.3.9) and move to the new point using equation (2.3.10);

Calculate fitness: Calculate the fitness value of current member: $f(X_i)$

END FOR

Set $k := k + 1$;

END WHILE

Table 2.2: The 23 benchmark functions, where n is the dimension of the function, $f_{\min}$ is the global minimum value of the function.

Test function		n	S	$f_{\min}$
$f_1(x)$	$= \sum_{i=1}^{n} x_i^2$	30	$[-100, 100]^n$	0
$f_2(x)$	$= \sum_{i=1}^{n} \lvert x_i \rvert + \prod_{i=1}^{n} \lvert x_i \rvert$	30	$[-10, 10]^n$	0
$f_3(x)$	$= \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2$	30	$[-100, 100]^n$	0
$f_4(x)$	$= \max_i \{ \lvert x_i \rvert, 1 \leq i \leq n \}$	30	$[-100, 100]^n$	0
$f_5(x)$	$= \sum_{i=1}^{n-1} (100(x_{i+1} - x_i^2)^2 + (x_i - 1))^2$	30	$[-30, 30]^n$	0
$f_6(x)$	$= \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2$	30	$[-100, 100]^n$	0
$f_7(x)$	$= \sum_{i=1}^{n} i x_i^4 + \text{random}[0, 1)$	30	$[-1.28, 1.28]^n$	0
$f_8(x)$	$= -\sum_{i=1}^{n} \left(x_i \sin \left(\sqrt{\lvert x_i \rvert} \right) \right)$	30	$[-500, 500]^n$	-12569.5
$f_9(x)$	$= \sum_{i=1}^{n} (x_i^2 - 10 \cos(2\pi x_i) + 10)^2$	30	$[-5.12, 5.12]^n$	0
$f_{10}(x)$	$= -20 \exp \left(-0.2 \sqrt{\frac{1}{n} \sum_{i=1}^{n} x_i^2} \right) - \exp \left(\frac{1}{n} \sum_{i=1}^{n} \cos 2\pi x_i \right)$ $+20+\text{e}$	30	$[-32, 32]^n$	0
$f_{11}(x)$	$= \frac{1}{4000} \sum_{i=1}^{30} (x_i - 100)^2 - \prod_{i=1}^{n} \cos(\frac{x_i - 100}{\sqrt{i}}) + 1$	30	$[-600, 600]^n$	0
$f_{12}(x)$	$= \frac{\pi}{n} \left\{ 10 \sin^2(\pi y_1) + \sum_{i=1}^{29} (y_i - 1)^2 [1 + 10 \sin^2(\pi y_{i+1})] \right.$ $\left. +(y_n - 1)^2 \right\} + \sum_{i=1}^{30} u(x_i, 10, 100, 4)$ $y_i = 1 + \frac{1}{4}(x_i + 1)$ $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \leq x_i \leq a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$	30	$[-50, 50]^n$	0
$f_{13}(x)$	$= 0.1 \left\{ \sin^2(\pi 3x_1) + \sum_{i=1}^{29} (x_i - 1)^2 [1 + \sin^2(3\pi x_{i+1})] \right.$ $\left. +(x_n - 1)^2 [1 + \sin^2(2\pi x_{30})] \right\} + \sum_{i=1}^{30} u(x_i, 5, 100, 4)$	30	$[-50, 50]^n$	0
$f_{14}(x)$	$= \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2} (x_i - a_{ij})^6} \right]^{-1}$	2	$[-65.536, 65.536]^n$	1
$f_{15}(x)$	$= \sum_{i=1}^{11} \left[a_i - \frac{x_1 (b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4} \right]^2$	4	$[-5, 5]^n$	0.0003075
$f_{16}(x)$	$= 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$	2	$[-5, 5]^n$	-1.0316285
$f_{17}(x)$	$= \left(x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left(1 - \frac{1}{8\pi} \right) \cos x_1 + 10$	2	$[-5, 10] \times [0, 15]$	0.398
$f_{18}(x)$	$= [1 + (x_1 + x_2 + 1)^2 (19 - 14x_1 + 3x_1^2 - 14x_2$ $+6x_1 x_2 + 3x_2^2)] \times [30 + (2x + 1 - 3x_2)^2 (18 - 32x_1$ $+12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2)]$	2	$[-2, 2]^n$	3
$f_{19}(x)$	$= -\sum_{i=1}^{4} c_i \exp \left[-\sum_{j=1}^{4} a_{ij} (x_j - p_{ij})^2 \right]$	3	$[0, 1]^n$	-3.86
$f_{20}(x)$	$= -\sum_{i=1}^{4} c_i \exp \left[-\sum_{j=1}^{6} a_{ij} (x_j - p_{ij})^2 \right]$	6	$[0, 1]^n$	-3.32
$f_{21}(x)$	$= -\sum_{i=1}^{5} [(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^n$	-10
$f_{22}(x)$	$= -\sum_{i=1}^{7} [(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^n$	-10
$f_{23}(x)$	$= -\sum_{i=1}^{10} [(x - a_i)(x - a_i)^T + c_i]^{-1}$	4	$[0, 10]^n$	-10

Table 2.3: The 6 300-dimensional multi-modal benchmark functions, where n is the dimension of the function, S is the feasible search space, and $f_{\min}$ is the global minimum value of the function.

Test function	n	S	$f_{\min}$
$f_8(x)^{300}$	300	$[-500, 500]^n$	-125694.7
$f_9(x)^{300}$	300	$[-5.12, 5.12]^n$	0
$f_{10}(x)^{300}$	300	$[-32, 32]^n$	0
$f_{11}(x)^{300}$	300	$[-600, 600]^n$	0
$f_{12}(x)^{300}$	300	$[-50, 50]^n$	0
$f_{13}(x)^{300}$	300	$[-50, 50]^n$	0

2.4.2 Experimental setting

The parameter setting of the GSO algorithm is summarized as follows. The initial population of GSO is generated uniformly at random in the search space. The initial head angle φ^0 of each individual is set to be $\frac{\pi}{4}$. The constant a is given by $round(\sqrt{n+1})$ where n is the dimension of the search space. The maximum pursuit angle $\theta_{\max}$ is $\frac{\pi}{a^2}$. The maximum turning angle α is set to be $\frac{\pi}{2a^2}$. The maximum pursuit distance $l_{\max}$ is calculated from the following equation:

$$l_{\max} = \parallel U_i - L_i \parallel = \sqrt{\sum_{i=1}^{n} (U_i - L_i)^2}$$

where L_i and U_i are the lower and upper bounds for the i_{th} dimension. The parameter need to be tuned is the percentage of dispersed members; our recommended percentage is 20%, which was used throughout all our experiments. The population size of the GSO algorithm was set to at 48 in order to keep the number of function evaluations as same as other algorithms in a generation.

We compared the performance of GSO with that of 4 different EAs:

1) Genetic Algorithm (GA) [78]

2) Evolutionary Programming (EP) [79] [80]

3) Evolution Strategies (ES) [81]

4) Particle Swarm Optimisation (PSO) [32]

Since there are no ES and EP toolboxes available, we adopted the test re-

sults of f_1 - f_{23} from [58] and [59] directly for comparison. In their studies, Yao and Liu proposed Fast EP (FEP) and Fast ES (FES) which replace Guassian mutations of conventional EP (CEP) and conventional ES (CES) with Cauchy mutations. We also employed the publicly available GA and PSO toolboxes in order to compare their accuracy and convergence rate with the GSO algorithm. The GA toolbox we used in our experiments is the Genetic Algorithm and Direct Search Toolbox (GADST) provided by Matlab 7.0. The GA algorithm we executed is real-coded with intermediate crossover and Guaussian mutation. The population of the GA was 50. The reproduction function was conducted using uniform stochastic selection. No subpopulation was used in the GA, therefore the migration rate was set to be 0. All the control parameters, *e.g.* mutation rate and crossover rate, etc., were set to be default. We also employed PSOt - a particle swarm optimisation toolbox for Matlab [82], which includes a standard PSO algorithm and several variants. The PSO algorithm we executed is the standard one. The parameters were given by default setting of the toolbox: the acceleration factors c_1 and c_2 were both 2.0; and a decaying inertia weight ω starting at 0.9 and ending at 0.4 was used. The population of 50 was used in the PSO algorithm.

For the 300 dimensional cases, since there are very few results published at present, besides GADST and PSOt, we also implemented EP and ES for comparison. The implementation of EP was based on the algorithm described in [80] and [83]. The population size and the tournament size for selection were 100 and 10, respectively. The initial standard deviation of the EP algorithm was 3.0. The ES algorithm used in our experiments is a state-of-the-art (μ, λ)-ES algorithm which was implemented according to [81]. The population μ was set to at 30 and the offspring number λ was 200. A standard deviation of 3.0 was adopted. Global intermediate recombination [?] was also employed in the ES algorithm.

Fifty independent runs of the GSO algorithm, GA and PSO were executed on benchmark functions $f_1 \sim f_{23}$. We tabulated the numbers of function evaluations for the 23 benchmark functions in Table 2.4. For 300-dimensional

benchmark functions, 5 independent runs of the GSO algorithm and the other four algorithms were executed to obtain average results. The number of generations for the six 300-dimensional benchmark functions were set to be 75000 for GSO and the other four algorithms. Therefore, 3750000 function evaluations were executed for each algorithm on each function.

Table 2.4: Number of function evaluations for function $f_1 \sim f_{23}$

Function	GSO/GA/PSO	CEP/FEP and CES/FES	Function	GSO/GA/PSO	CEP/FEP and CES/FES
f_1	150,000	150,000	f_{13}	150,000	150,000
f_2	150,000	200,000	f_{14}	7,500	10,000
f_3	150,000	500,000	f_{15}	250,000	400,000
f_4	150,000	500,000	f_{16}	1,250	10,000
f_5	150,000	2,000,000	f_{17}	5,000	10,000
f_6	150,000	150,000	f_{18}	10,000	10,000
f_7	150,000	300,000	f_{19}	4,000	10,000
f_8	150,000	900,000	f_{20}	7,500	20,000
f_9	150,000	500,000	f_{21}	10,000	10,000
f_{10}	150,000	150,000	f_{22}	10,000	10,000
f_{11}	150,000	200,000	f_{23}	10,000	10,000
f_{12}	150,000	150,000			

The experiment included an average test on all the algorithms which run 50 times respectively to get an average result of each algorithm for each benchmark function. In order to further assess the performance of the GSO algorithm in a stochastic search process with a consideration of randomly distributed initial populations, a set of two-tailed t-tests were adopted [58] [84]. The t-test assesses whether the means of two groups of results are statistically different from each other, for which the statistical significance of our experimental results between the GSO and the other four algorithms were measured. In this case, a critical value, t_{crit}, was set up to be ± 2.0 and the level of significance was placed as $\alpha = 0.05$ for a benchmark function, with 49 degrees of freedom at this level. This means if $|t| > 2.0$ the difference between the two means of the two tests is statistically significant.

2.4.3 Uni-modal functions

It is worth mentioning that uni-modal problems can be solved efficiently by many deterministic optimisation algorithms that use gradient information. However, uni-modal functions have been adopted to assess the convergence rates of EAs [59]. We tested the GSO on a set of uni-modal functions in comparison with the other two algorithms. Table 2.5 lists the experimental results (i.e., the mean and standard deviations of the function values found in 50 runs) for each algorithm on uni-modal functions $f_1 \sim f_7$. Figs. 2.4, 2.5, 2.6 and 2.7 show the search progress of the average values found by the three algorithms over 50 runs for functions $f_1 \sim f_7$. The results generated from CEP, FEP, CES and FES are tabulated in Table 2.6 in comparison with the results generated by our GSO algorithm.

From Table 2.5, the GSO generated significantly better results than GA on functions $f_1 \sim f_6$ and yielded a similar result to GA on function f_7. From the comparisons between GSO and the PSO, we can see that, statistically, GSO has similar or significantly better performance on $f_4 \sim f_5$. The GSO algorithm only yielded statistically worst results on $f_1 \sim f_3$ compared to PSO. In summary, the search performance of the three algorithms tested here can

be ordered as PSO > GSO > GA.

It can be found from Table 2.6 that GSO was ranked the third which was outperformed by FEP and FES. However, according to Table 2.4, GSO required much less number of function evaluation than the other 4 algorithms.

Table 2.5: Comparison among GSO with GA and PSO on benchmark functions $f_1 \sim f_7$. All results have been averaged over 50 runs.

	Mean function value (standard deviation)			t-test	
Function	GSO	GA	PSO	GSO-GA	GSO-PSO
f_1	8.4340×10^{-9} (8.6319×10^{-9})	1.2854×10^{-2} (7.1434×10^{-3})	6.2462×10^{-41} (2.7028×10^{-40})	-12.72	6.91
f_2	2.6087×10^{-5} (2.7580×10^{-5})	5.7651×10^{-2} (1.3799×10^{-2})	1.1575×10^{-22} (8.0937×10^{-22})	-29.52	4.77
f_3	33.8763 (17.4544)	51.6702 (18.0901)	1.5435 (1.7091)	-5.22	12.17
f_4	0.1115 (3.5456×10^{-2})	0.5119 (0.1199)	0.4188 (0.2196)	-21.49	-9.79
f_5	48.2122 (29.4532)	113.6078 (176.6329)	39.4321 (33.6186)	-2.35	1.84
f_6	2.0000×10^{-2} (0.1414)	1.2200 (1.6199)	0.2200 (0.5817)	-5.17	-2.33
f_7	6.7302×10^{-2} (3.4435×10^{-2})	30614×10^{-2} (2.0013×10^{-2})	7.1759×10^{-2} (3.2122×10^{-2})	5.97	-0.53

Table 2.6: Comparison among GSO with CEP, FEP, CES and FES on benchmark functions $f_1 \sim f_7$.

	Mean function value (**Rank**) (standard deviation)				
Function	GSO	FEP	CEP	FES	CES
f_1	8.4340×10^{-9} (**1**) (8.6319×10^{-9})	5.7×10^{-4} (**5**) (1.3×10^{-4})	2.2×10^{-4} (**3**) (5.9×10^{-4})	2.5×10^{-4} (**4**) (6.8×10^{-4})	3.4×10^{-5} (**2**) (8.6×10^{-6})
f_2	2.6087×10^{-5} (**1**) (2.7580×10^{-5})	8.1×10^{-3} (**3**) (7.7×10^{-4})	2.6×10^{-3} (**2**) (1.7×10^{-4})	6.0×10^{-2} (**5**) (9.6×10^{-3})	2.1×10^{-2} (**4**) (2.2×10^{-3})
f_3	33.8763 (**5**) (17.4544)	1.6×10^{-2} (**3**) (1.4×10^{-2})	5.0×10^{-2} (**4**) (6.6×10^{-2})	1.4×10^{-3} (**2**) (5.3×10^{-4})	1.3×10^{-4} (**1**) (8.5×10^{-5})
f_4	0.1115 (**2**) (3.5456×10^{-2})	0.3 (**3**) (0.5)	2.0 (**5**) (1.2)	5.5×10^{-3} (**1**) (6.5×10^{-4})	0.35 (**4**) (0.42)
f_5	48.2122 (**5**) (29.4532)	5.06 (**1**) (5.87)	6.17 (**2**) (13.61)	33.28 (**4**) (43.13)	6.69 (**3**) (14.45)
f_6	2.0000×10^{-2} (**3**) (0.1414)	0 (**1**) (0)	577.76 (**5**) (1125.76)	0 (**1**) (0)	411.16 (**4**) (695.35)
f_7	6.7302×10^{-2} (**5**) (3.4435×10^{-2})	7.6×10^{-3} (**1**) (2.6×10^{-3})	1.8×10^{-2} (**3**) (6.4×10^{-3})	1.2×10^{-2} (**2**) (5.8×10^{-3})	3.0×10^{-2} (**4**) (1.5×10^{-2})
Average Rank	3.14	2.42	3.43	2.71	3.14
Final Rank	3	1	5	2	3

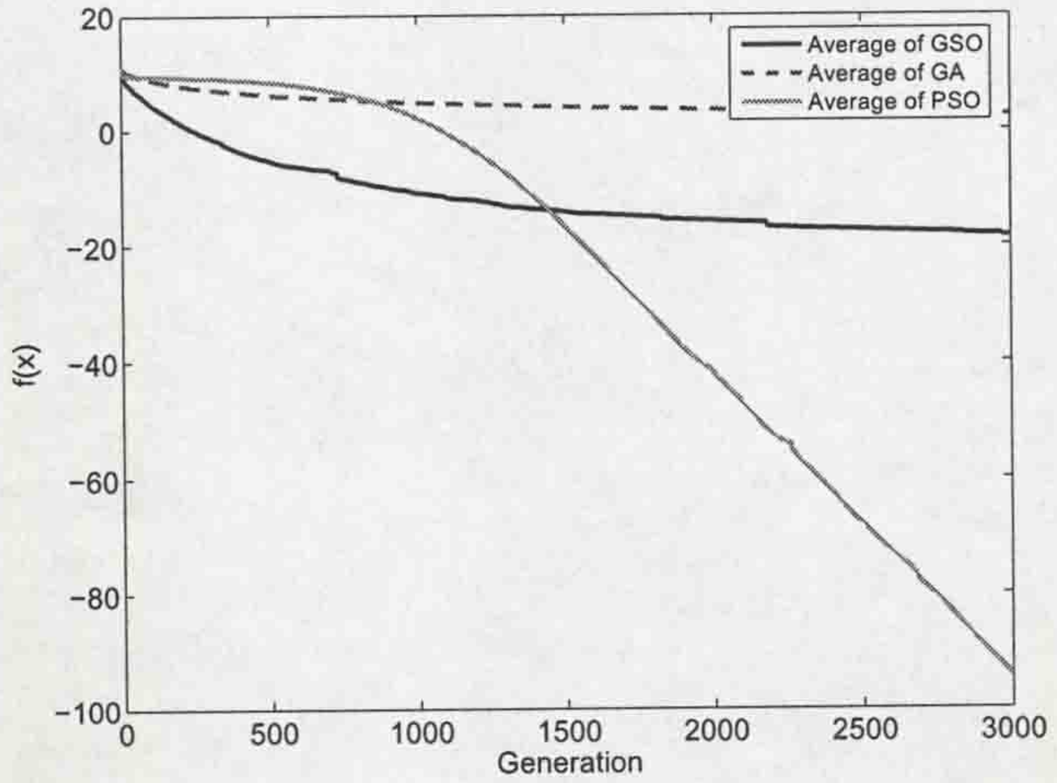
2.4.4 Multi-modal functions

Multi-modal functions with many local minima

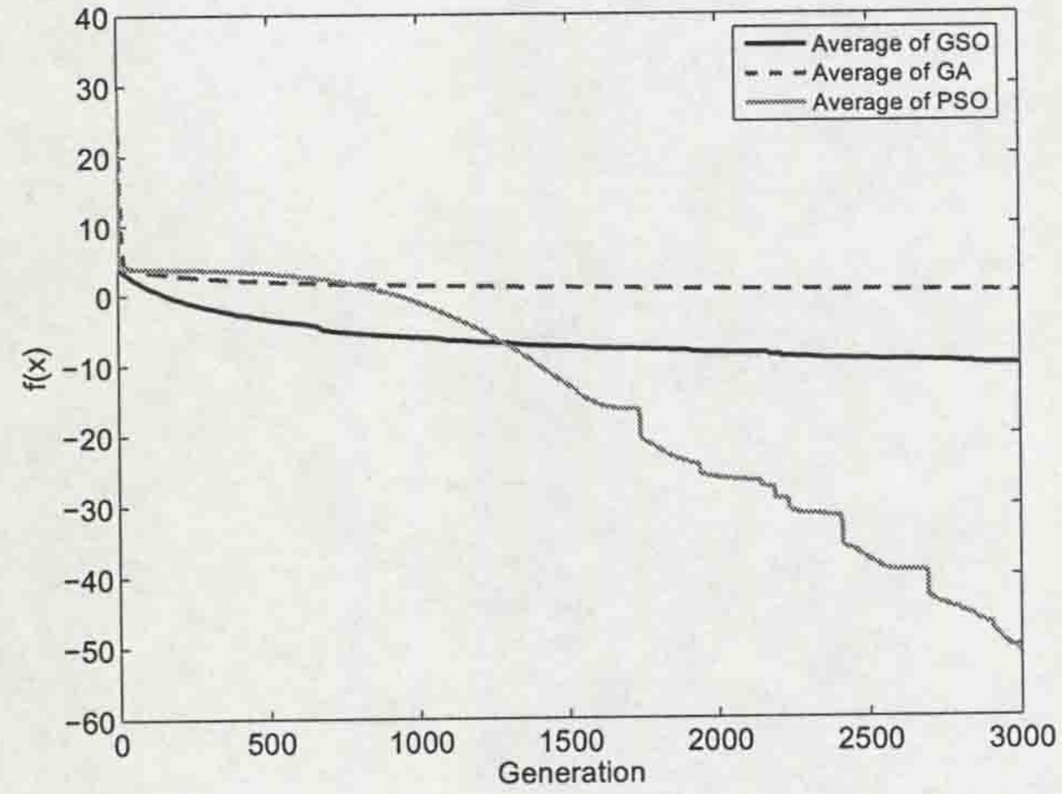
This set of benchmark functions ($f_8 \sim f_{13}$) are regarded as the most difficult functions to optimise since the number of local minima increases exponentially as the function dimension increases [?]. The mean and standard deviations of the function values found in 50 runs for each algorithm on each test function are listed in Table 2.7. Figs. 2.8, 2.9, and 2.10 show the search progress of the average values found by the three algorithms over 50 runs for functions $f_8 \sim f_{13}$. Results adopted from [58] and [59] are tabulated in Table 2.8 in comparison with the results produced by GSO.

From Table 2.7, it is clear to see that for most of the tested benchmark functions, GSO markedly outperformed GA and PSO. For example, on function f_8, GSO found the global minimum almost every time of run while the other four algorithms generated poorer results in this case. GSO generated significant better results than those of PSO on most functions. The only exception is Ackley's function (f_{10}) and Griewank (f_{11}). PSO outperformed GSO statistically. However, according to [85], the regions of the Griewank function's local minima become narrower and narrower as the dimension increases. Consequently, it is much easier for optimisation algorithms to find the global minimum since the local minima are more and more likely to be neglected as the dimension increases. It has been found that local optimisation algorithms, for example the limited memory BFGS (Broyden-Fletcher-Goldfarb-Shanno) algorithm [86], yielded better results than global optimisation algorithms, *e.g.* EAs for the Griewank function in the cases of high dimensions. Therefore, the Griewank function is not a challenging multi-modal benchmark problem for evaluation of global optimisation algorithms. No substantial conclusion can be drawn from the comparisons between GSO and PSO on the Griewank function.

It can be seen from Figs. 2.8 2.9, and 2.10 that on average the GSO algorithm consistently outperformed the other two algorithms for 4 benchmark functions: f_8, f_9, f_{12} and f_{13}, respectively. From our experiments, we also found

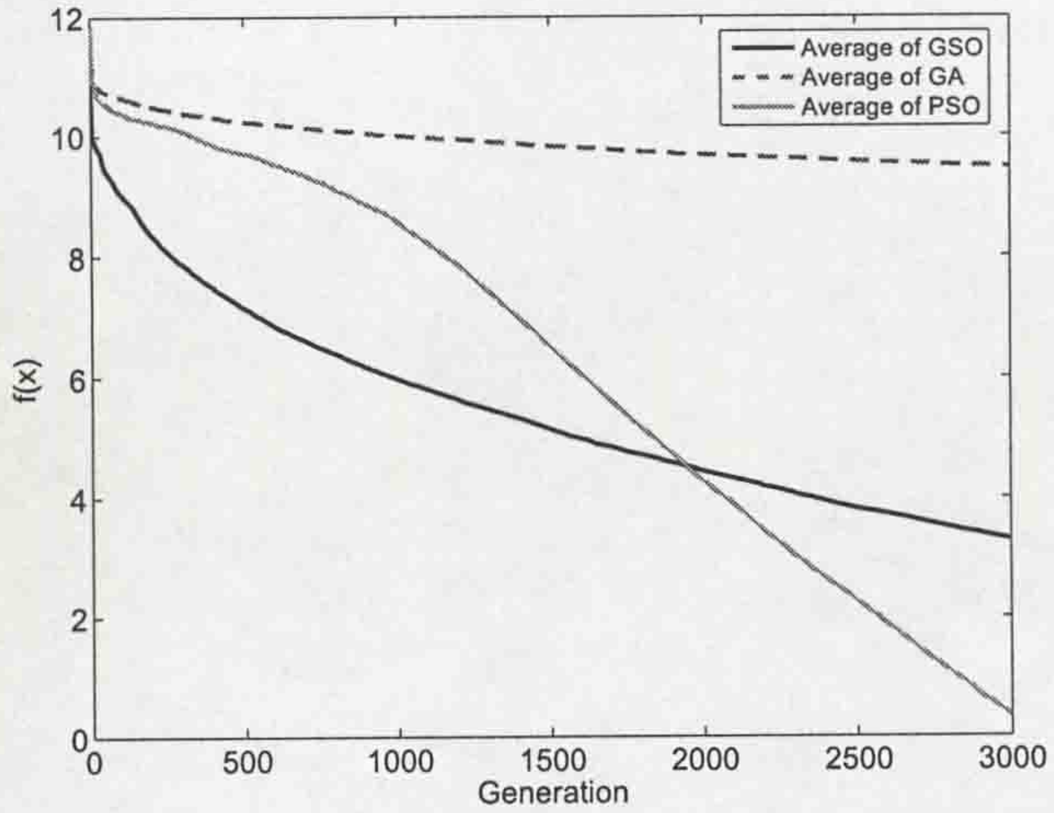


(a)

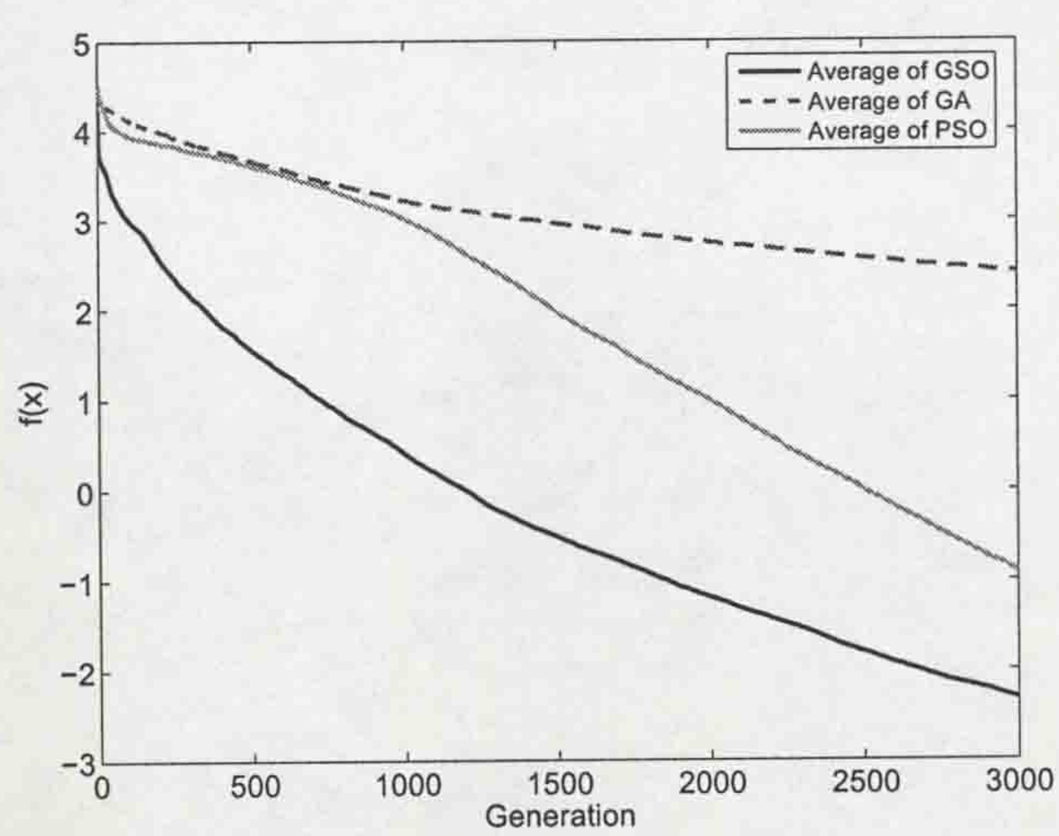


(b)

Figure 2.4: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a) and (b) correspond to functions f_1 and f_2, respectively.

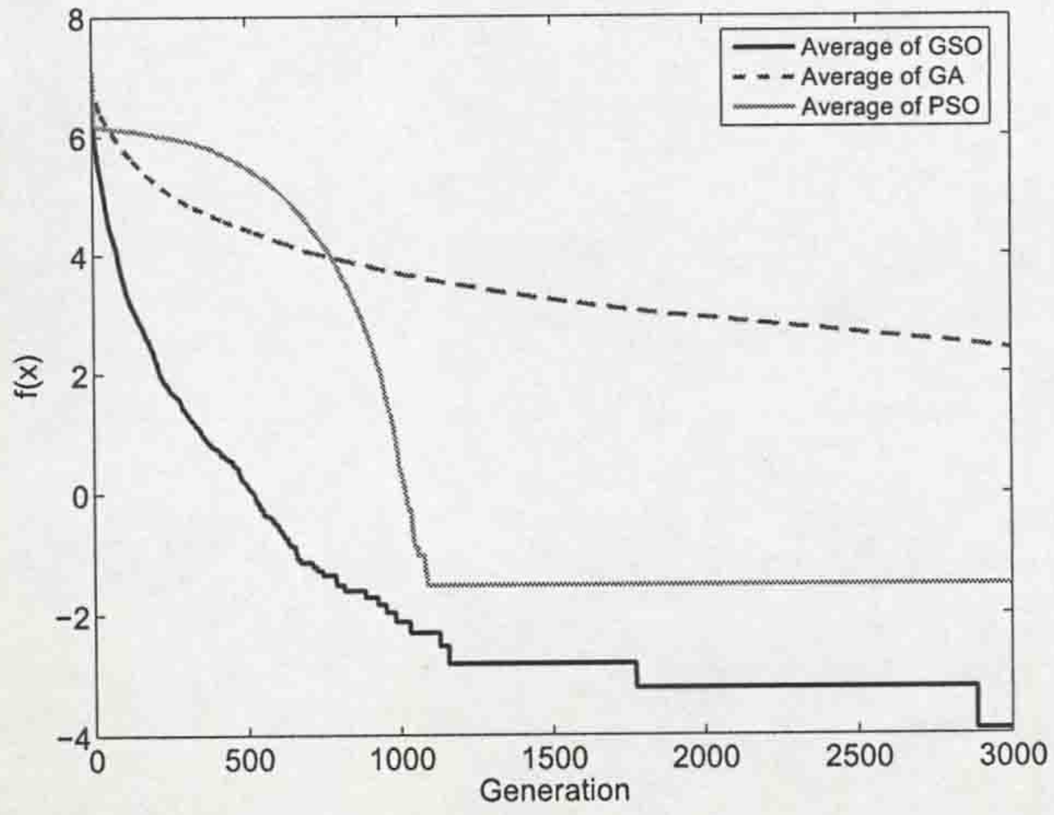


(a)

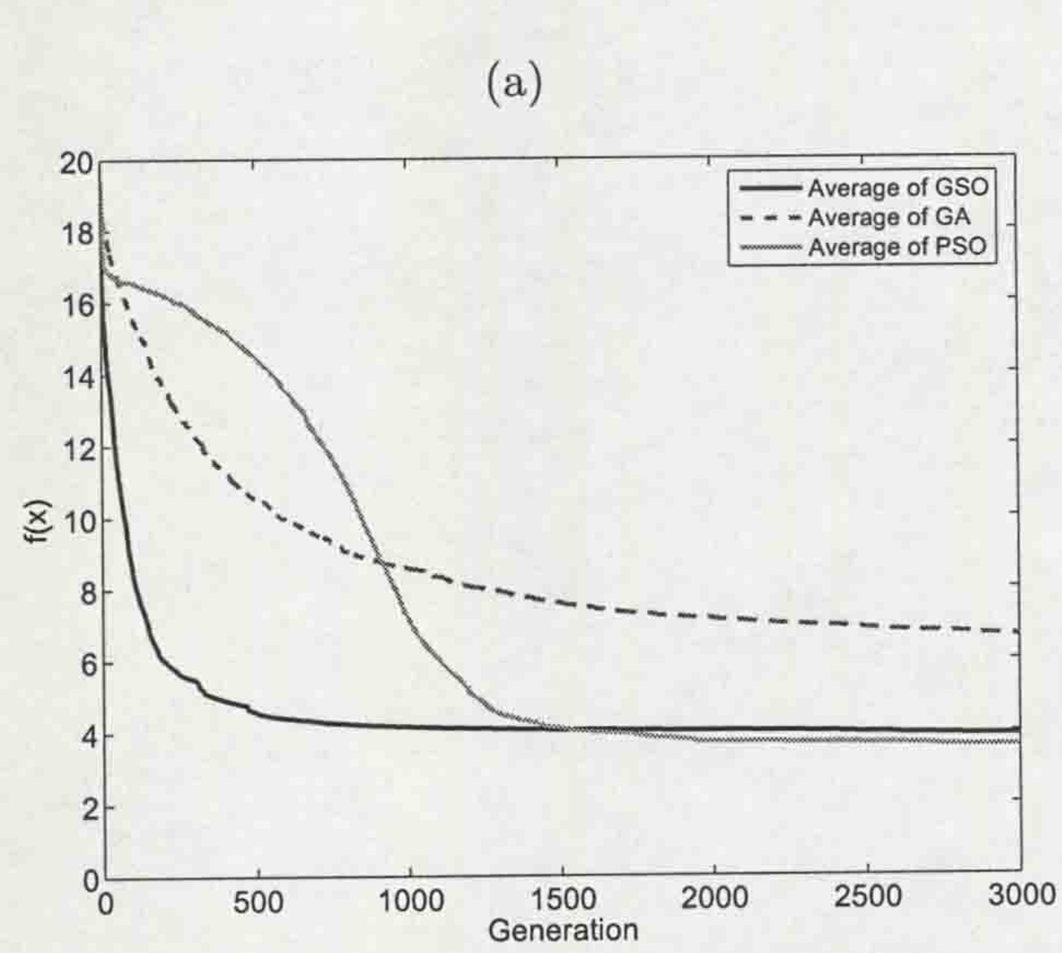


(b)

Figure 2.5: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a) and (b) correspond to functions f_3 and f_4, respectively.

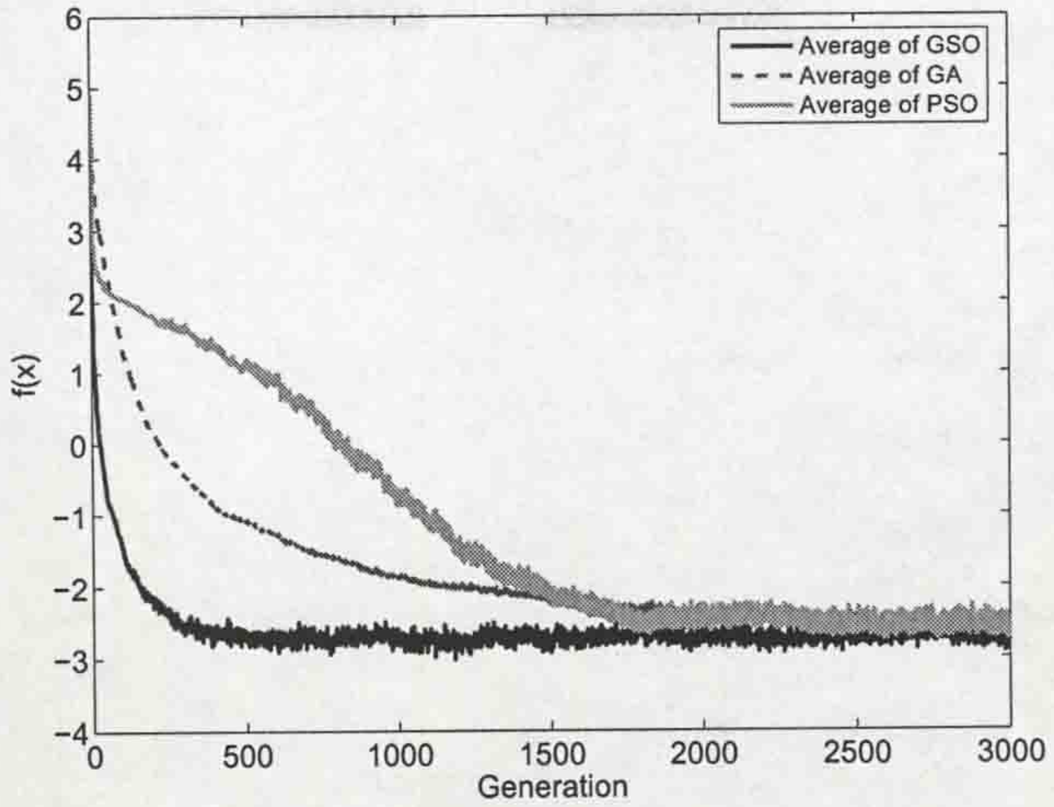


(a)



(b)

Figure 2.6: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a) and (b) correspond to functions f_5 and f_6, respectively.

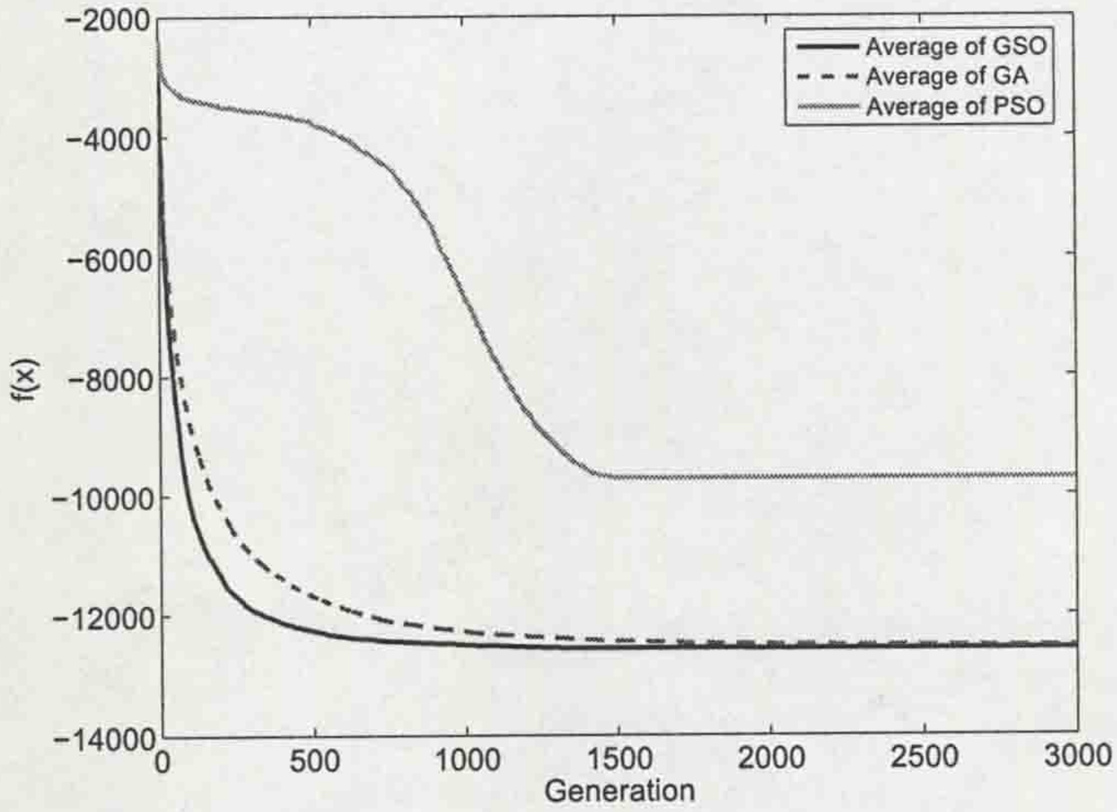


(a)

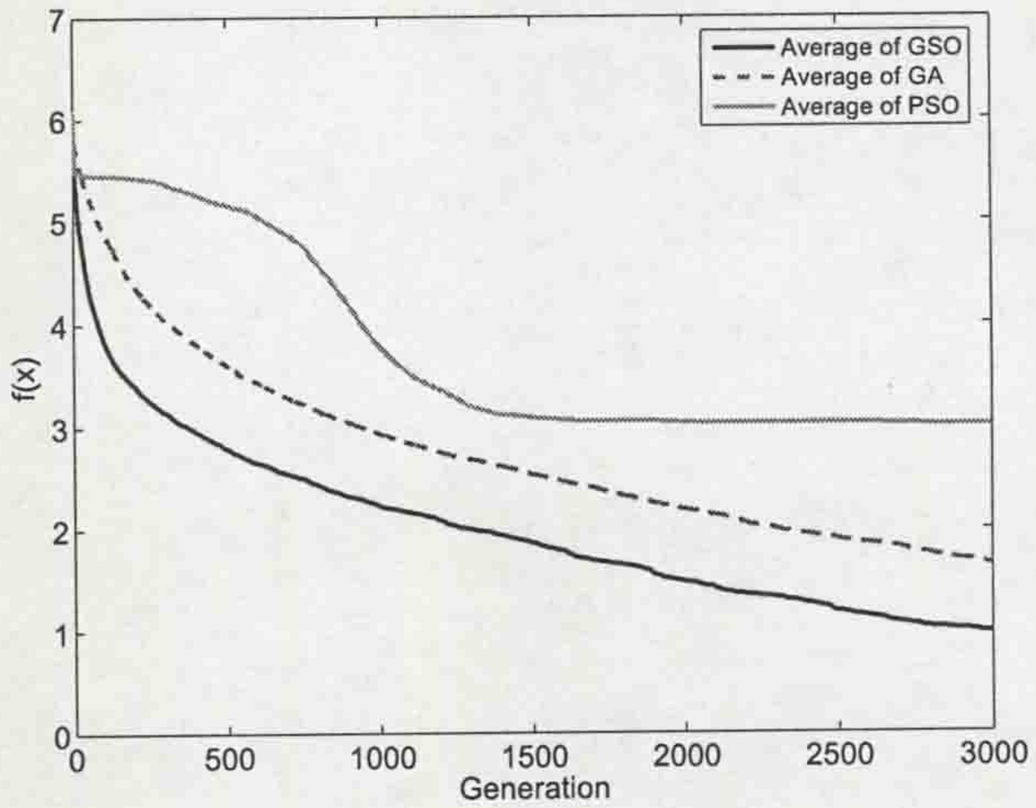
Figure 2.7: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a) correspond to function f_7

that, for functions f_{12} and f_{13}, the best results found by the PSO are better than those found by the GSO in terms of accuracy and convergence speed. However, the average results and the standard deviations generated by PSO indicate that PSO is more likely to be trapped by poor local minima, therefore it leads to inconsistent search performance on these two functions. It can be concluded from Table 2.7 that the order of the search performance of these three algorithms is GSO > PSO > GA.

It can be found from Table 2.8 that, in comparison with CEP, FEP, CES and FES, GSO has the best performance (Rank 1) with less function evaluations. It can also be found from Table 2.8 that, for 4 out of 6 functions, GSO generated better results than the other four algorithms. The two exceptions are Rastrigin (f_9) and Griewank (f_{11}) functions. GSO was outperformed by FEP and FES on Rastrigin function and by FEP on Griewank function, respectively.

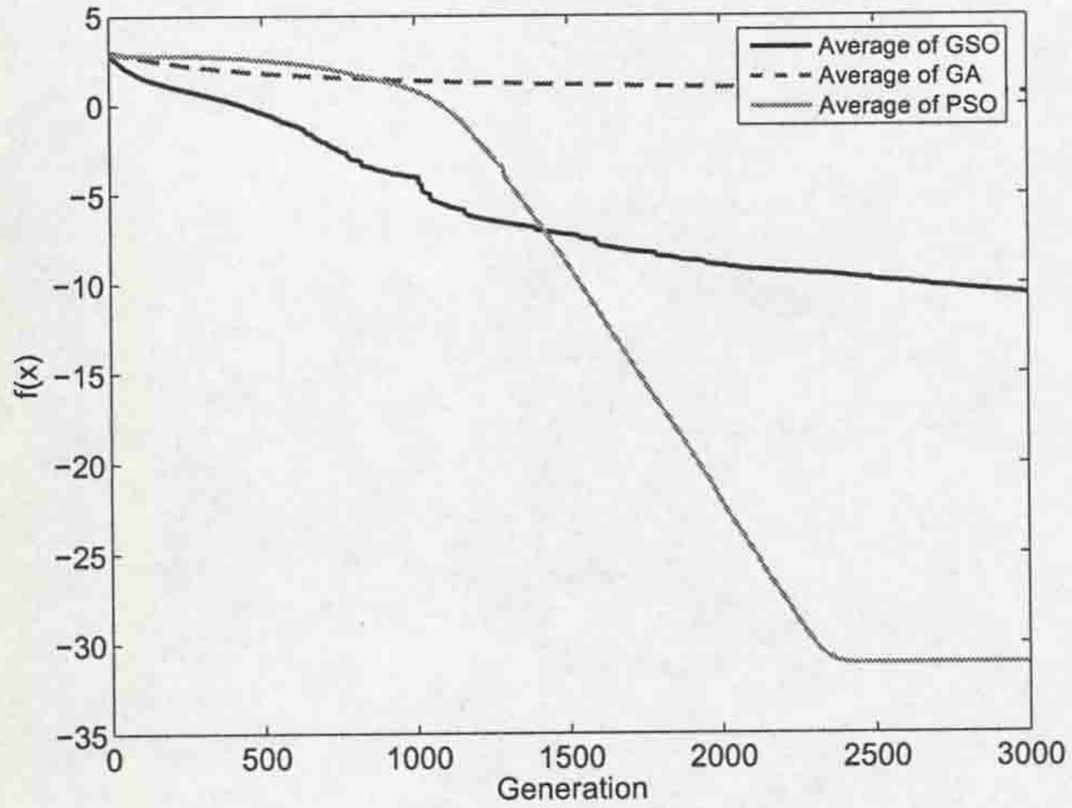


(a)

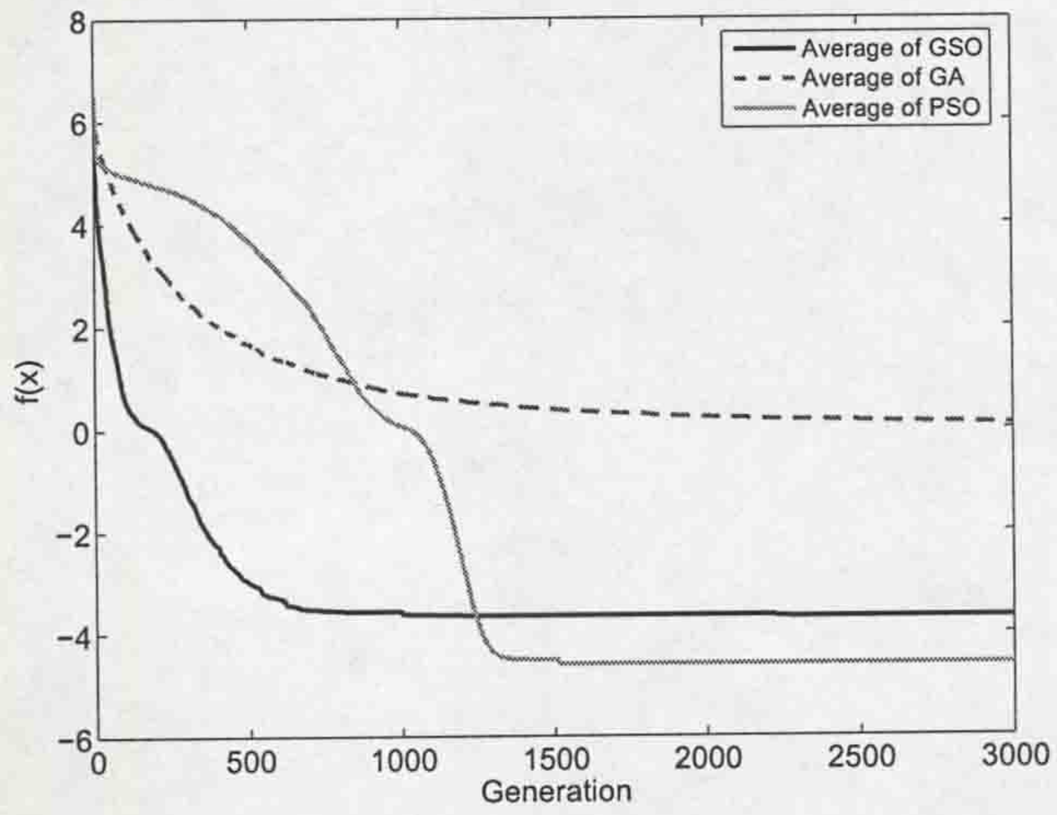


(b)

Figure 2.8: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a) and (b) correspond to functions f_8 and f_9, respectively.

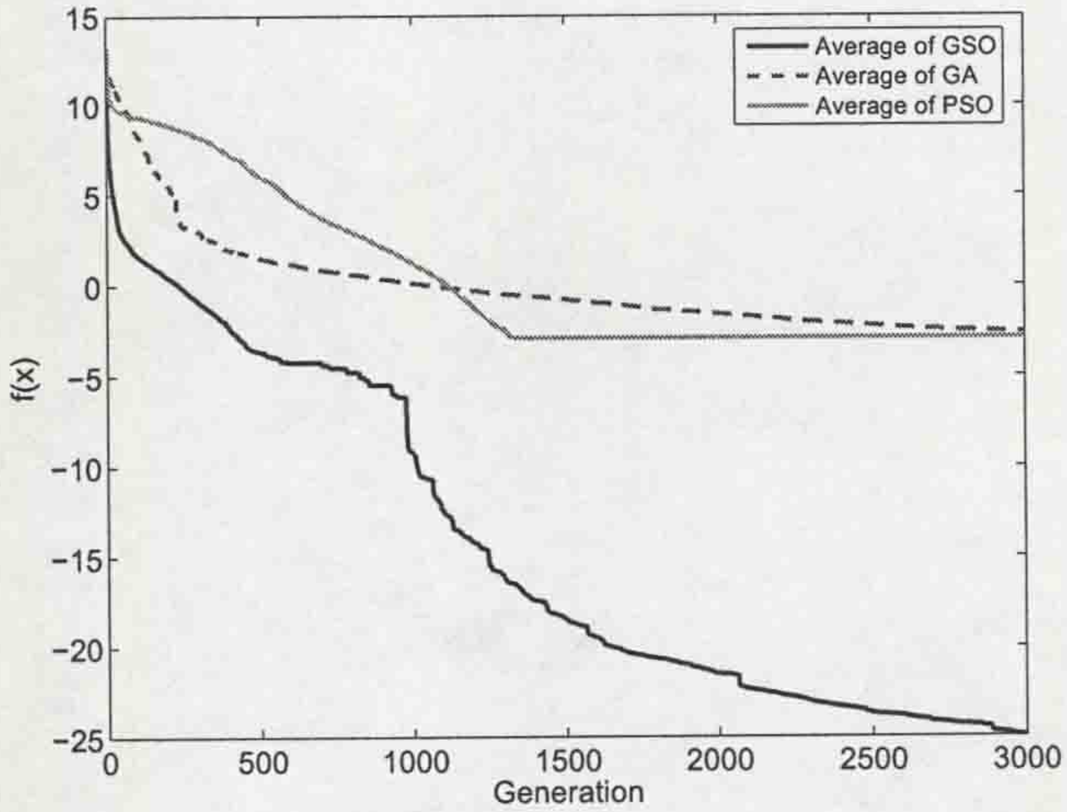


(a)

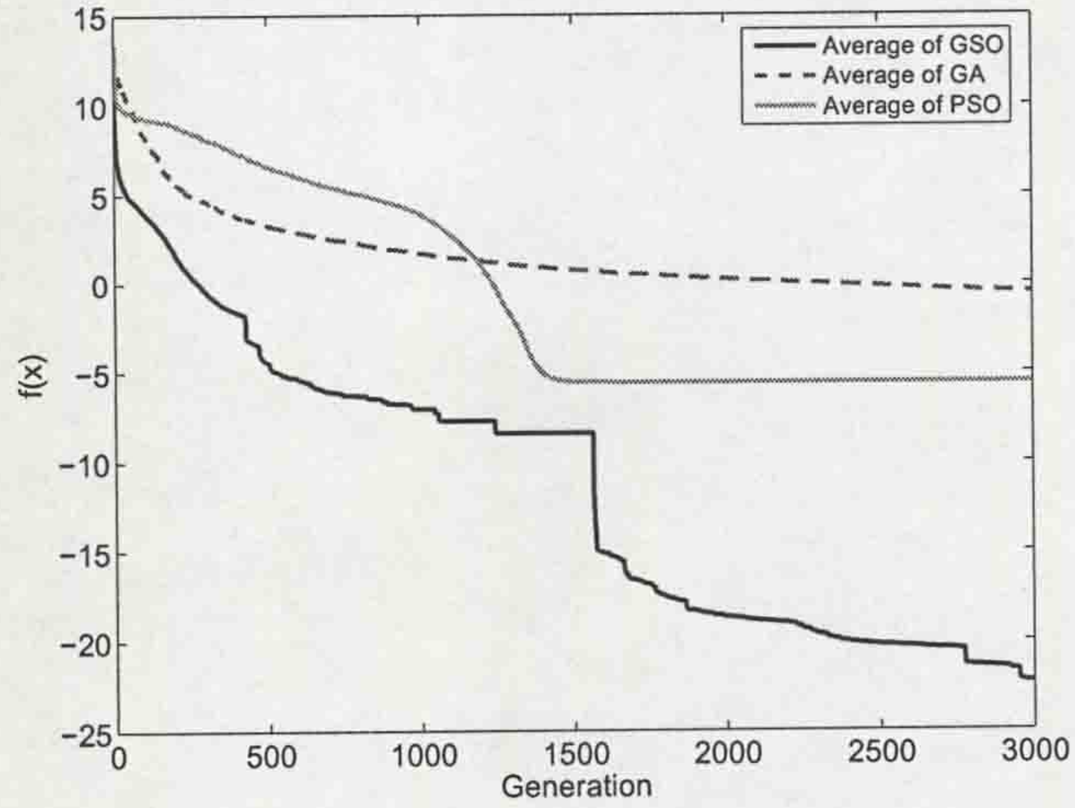


(b)

Figure 2.9: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a) and (b) correspond to functions f_{10} and f_{11}, respectively.



(a)



(b)

Figure 2.10: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a) and (b) correspond to functions f_{12} and f_{13}, respectively.

Table 2.7: Comparison among GSO with GA and PSO on benchmark functions $f_8 \sim f_{13}$. All results have been averaged over 50 runs.

	Mean function value (standard deviation)			t-test	
Function	GSO	GA	PSO	GSO-GA	GSO-PSO
f_8	-12569.4809 (2.6205×10^{-2})	-12533.7800 (12.8206)	-9718.2764 (370.7043)	-19.6882	-54.39
f_9	2.3518 (1.4029)	5.1874 (1.5164)	20.3768 (4.8462)	-8.02	-26.22
f_{10}	3.5711×10^{-5} (4.0282×10^{-5})	1.8865 (0.3652)	3.1157×10^{-14} (8.6202×10^{-15})	-36.52	8.01
f_{11}	2.8396×10^{-2} (2.87567×10^{-2})	1.0972 (3.3138×10^{-2})	1.004×10^{-2} (1.1564×10^{-2})	-68.31	4.09
f_{12}	2.1922×10^{-11} (3.2899×10^{-11})	0.9529 (1.5987)	0.9012 (1.5979)	-4.21	-3.62
f_{13}	3.1144×10^{-10} (3.9485×10^{-10})	0.5759 (0.2546)	3.7167×10^{-3} (5.6463×10^{-3})	-15.9927	-4.65

Table 2.8: Comparison among GSO with CEP, FEP, CES and FES on benchmark functions $f_8 \sim f_{13}$.

	Mean function value (**Rank**) (standard deviation)				
Function	GSO	FEP	CEP	FES	CES
f_8	-12569.4809 (**1**) (2.6205×10^{-2})	-12554.5 (**3**) (52.6)	-7917.1 (**4**) (634.5)	-12556.4 (**2**) (32.53)	-7549.9 (**5**) (631.39)
f_9	2.3518 (**3**) (1.4029)	4.6×10^{-2} (**1**) (1.2×10^{-2})	89.0 (**5**) (23.1)	0.16 (**2**) (0.33)	70.82 (**4**) (21.49)
f_{10}	3.5711×10^{-5} (**1**) (4.0282×10^{-5})	1.8×10^{-2} (**3**) (2.1×10^{-2})	9.2 (**5**) (2.8)	1.2×10^{-2} (**2**) (1.8×10^{-3})	9.07 (**4**) (2.84)
f_{11}	2.8396×10^{-2} (**2**) (2.87567×10^{-2})	1.6×10^{-2} (**1**) (2.2×10^{-2})	8.6×10^{-2} (**4**) (0.12)	3.7×10^{-2} (**3**) (5.0×10^{-2})	0.38 (**5**) (0.77)
f_{12}	1.3893×10^{-11} (**1**) (1.9517×10^{-11})	9.2×10^{-6} (**2**) (6.1395×10^{-5})	1.76 (**5**) (2.4)	2.8×10^{-2} (**3**) (8.1×10^{-11})	1.18 (**4**) (1.87)
f_{13}	2.0247×10^{-10} (**1**) (3.0132×10^{-10})	1.6×10^{-4} (**3**) (7.3×10^{-5})	1.4 (**5**) (3.7)	4.7×10^{-5} (**2**) (1.5×10^{-5})	1.39 (**4**) (3.33)
Average Rank	1.5	2.17	4.67	2.22	4.33
Final Rank	1	2	5	3	4

Multi-modal functions with a few local minima

This set of benchmark functions $f_{14} \sim f_{23}$ are multi-modal but in low dimensions ($n \leq 6$) and they have only a few local minima. Compared to the multi-modal functions with many local minima ($f_8 \sim f_{13}$), this set of functions are not challenging: some of them can even be solved efficiently by deterministic algorithms [87] [88].

From Table 2.9, we can see in comparison to GA, GSO achieved better results on all benchmark functions. Two-tailed t-test also indicated that for 7 out of 10 benchmark functions, GSO statistically outperformed GA. For the rest 3 benchmark functions, no statistically significant difference can be found between GSO and GA. In comparison with PSO, it can be seen that GSO has a better performance on most of the functions except the Kowalik's function (f_{15}) and Shekel's family functions (f_{21}, f_{22} and f_{23}) where PSO generated better average results than those of GSO. From the Two-tailed t-test, it can be found that, statistically, GSO outperformed PSO on functions $f_{14} \sim f_{15}$ and achieved similar results on functions f_{14}, f_{15} and f_{21}. The search progress of the average values and the best solutions found by the three algorithms on functions $f_{14} \sim f_{23}$ are shown in Figs. 2.11, 2.12, 2.13, 2.14 and 2.15. From Table 2.9 we can see that the order of the search performance of these three algorithms is GSO > PSO > GA.

Table 2.10 reveals that GSO ranked the first in comparison with CEP, FEP, CES and FES. For function f_{14} to f_{19}, GSO has the best performance. However, it was outperformed by the other four algorithms on Hartman's Function f_{20} and Shekel's family functions (f_{21}, f_{22} and f_{23}).

Table 2.9: Average fitness values of benchmark functions $f_{14} \sim f_{23}$. All results have been averaged over 50 runs.

	Mean function value (standard deviation)			t-test	
Function	GSO	GA	PSO	GSO-GA	GSO-PSO
f_{14}	0.9980 (0)	1.1149 (0.2596)	1.0239 (0.1450)	-3.18	-1.26
f_{15}	3.7713×10^{-4} (2.5973×10^{-4})	1.0525×10^{-3} (3.0211×10^{-4})	3.8074×10^{-4} (2.5094×10^{-4})	−9.54	−0.08
f_{16}	-1.031628 (0)	-0.9139 (0.1773)	-1.0160 (1.2786×10^{-2})	-4.69	-8.62
f_{17}	0.3979 (0)	0.4226 (3.1834×10^{-2})	0.4040 (6.8805×10^{-2})	-5.49	-6.23
f_{18}	3.0 (0)	3.002 (0.1484)	3.0005 (1.2117×10^{-3})	-4.70	-3.30
f_{19}	-3.8628 (3.8430×10^{-6})	-3.8545 (1.0254×10^{-5})	-3.8582 (3.2127×10^{-3})	-5.69	-10.06
f_{20}	-3.2697 (5.9647×10^{-2})	-3.2534 (5.7947×10^{-2})	-3.1846 (6.1053×10^{-2})	-1.37	-6.70
f_{21}	-6.09 (3.4563)	-5.0537 (2.6997)	-7.5439 (3.0303)	-1.64	0.30
f_{22}	-6.5546 (3.2443)	-5.1411 (3.0715)	-8.3553 (2.0184)	-2.11	3.15
f_{23}	-7.4022 (3.2131)	-6.5887 (3.2851)	-8.9439 (1.6304)	-1.06	3.1551

Table 2.10: Comparison among GSO with CEP, FEP, CES and FES on benchmark functions $f_{14} \sim f_{23}$.

	Mean function value (**Rank**) (standard deviation)				
Function	GSO	FEP	CEP	FES	CES
f_{14}	0.9980 (**1**) (0)	1.22 (**3**) (0.56)	1.66 (**4**) (1.19)	1.20 (**2**) (0.63)	2.16 (**5**) (1.82)
f_{15}	4.1687×10^{-4} (**1**) (3.1238×10^{-4})	5.0×10^{-4} (**2**) (3.2×10^{-4})	4.7×10^{-4} (**2**) (3.0×10^{-4})	9.7×10^{-4} (**4**) (4.2×10^{-4})	1.2×10^{-3} (**5**) (1.6×10^{-5})
f_{16}	-1.031628 (**1**) (0)	-1.03 (**2**) (4.9×10^{-4})	-1.03 (**2**) (4.9×10^{-4})	-1.0316 (**4**) (6.0×10^{-7})	-1.0316 (**4**) (6.0×10^{-7})
f_{17}	0.3979 (**1**) (0)	0.398 (**4**) (1.5×10^{-7})	0.398 (**4**) (1.5×10^{-7})	0.398 (**2**) (6.0×10^{-8})	0.398 (**2**) (6.0×10^{-8})
f_{18}	3.0 (**1**) (0)	3.02 (**5**) (0.11)	3.0 (**1**) (0)	3.0 (**1**) (0)	3.0 (**1**) (0)
f_{19}	-3.8628 (**1**) (1.8150×10^{-5})	-3.86 (**2**) (1.4×10^{-5})	-3.86 (**5**) (1.4×10^{-2})	-3.86 (**4**) (4.0×10^{-3})	-3.86 (**3**) (1.4×10^{-5})
f_{20}	-3.2697 (**3**) (5.9644×10^{-2})	-3.27 (**2**) (5.9×10^{-2})	-3.28 (**1**) (5.8×10^{-2})	-3.23 (**5**) (0.12)	-3.24 (**4**) (5.7×10^{-2})
f_{21}	-5.9911 (**3**) (3.2948)	-5.52 (**5**) (1.59)	-6.86 (**2**) (2.67)	-5.54 (**4**) (1.82)	-6.96 (**1**) (3.10)
f_{22}	-6.5866 (**4**) (3.2443)	-5.52 (**5**) (2.12)	-8.27 (**2**) (2.95)	-6.76 (**3**) (3.01)	-8.31 (**1**) (3.10)
f_{23}	-7.3021 (**4**) (3.6133)	-6.57 (**5**) (3.14)	-9.10 (**1**) (2.92)	-7.63 (**3**) (3.27)	-8.50 (**2**) (1.25)
Average Rank	2	3.5	2.4	3.2	2.8
Final Rank	1	5	2	4	3

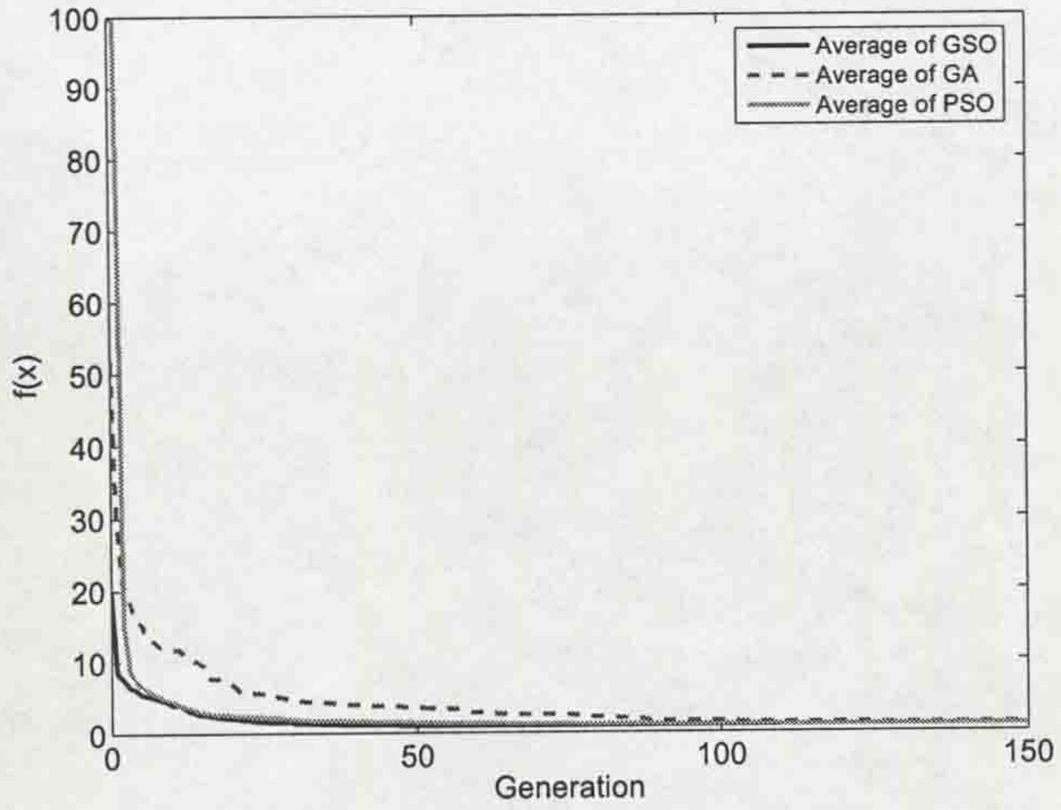
300-dimensional multi-modal functions

Many real-world optimisation problems usually involve hundreds or even thousands variables. However, previous studies showed that although some algorithms generated good results on relatively low-dimensional ($n \leq 30$) benchmark problems, they do not perform satisfactorily for some large-scale problems [89]. Therefore, in order to assess the scalability of our GSO algorithm, which is crucial for its applicability to real-world problems, a set of multi-modal benchmark functions (f_8 to f_{13}) were extended to 300-dimensions and used in our experimental studies as high-dimensional benchmark functions. The results are presented in Table 2.11.

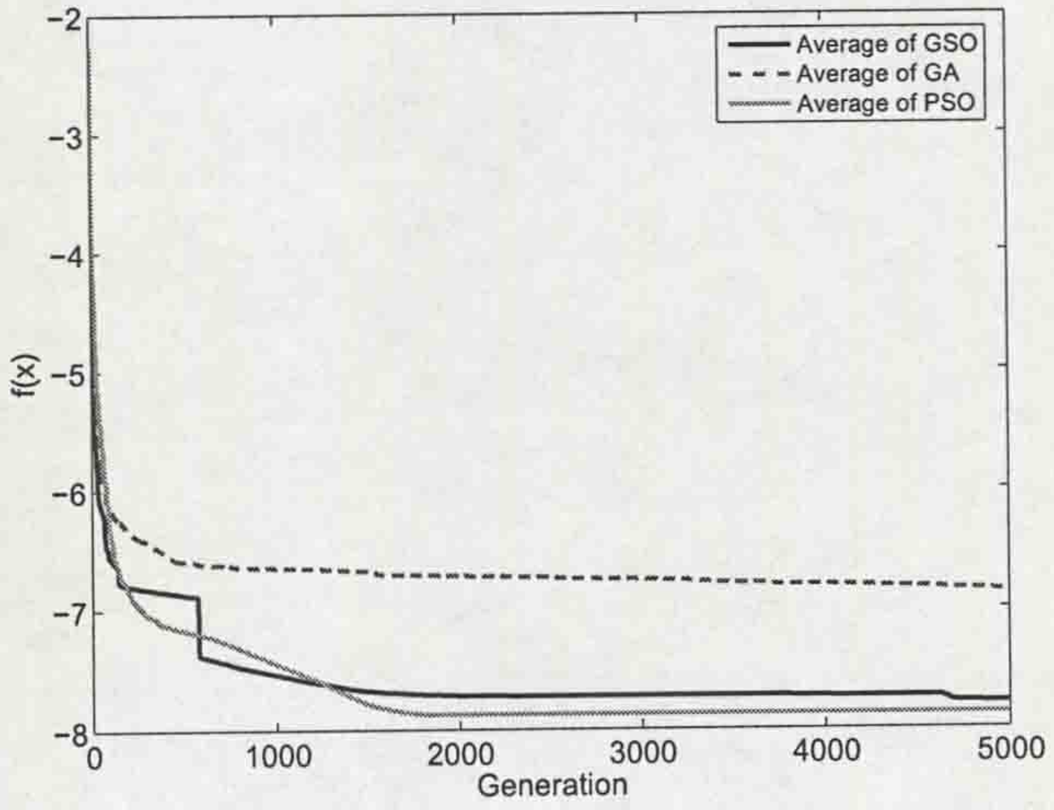
From Table 2.11, it can be seen that in terms of final average results, GSO markedly outperformed the other algorithms. For the six problems we tested, the GSO algorithm converged to good near optimal solutions. It can also been seen that although PSO achieved satisfactory results in 30-dimensional multi-modal benchmark problems (see Table 2.7), it cannot be scaled up to handle most of the 300-dimensional cases except $f_{10}(x)^{300}$.

A limited scale of research scalability of EAs has been found [89] [90]. In [89], four EP algorithms, namely CEP, FEP, Improved FEP (IFEP) [58] and a Mixed EP (MEP) [89] were studied. The benchmark functions used in their studies were a uni-modal function f_1 (Sphere function) and a multi-modal function f_{10} (Ackley's function) with dimensions ranged from 100 to 300. It was found that CEP and FEP failed to converge on function f_{10}. The average results generated by IFEP and MEP on function f_{10} in 300 dimensions were 7.6×10^{-2} and 5.5×10^{-2} respectively, which are both worse than that generated by our GSO algorithm. Liu and Yao also improved FEP with cooperative coevolution [90] by decoupling the whole optimisation function to a set of coordinates of populations. Eight functions, including four uni-modal and four multi-modal functions were used as benchmark functions in their studies. The results presented in their paper were excellent, *e.g.*, the result on 300-dimensional f_{10} was 3.6×10^{-4}. In this case, it is unfair to compare our current GSO algorithm with their algorithm without population decoupling.

Shan He

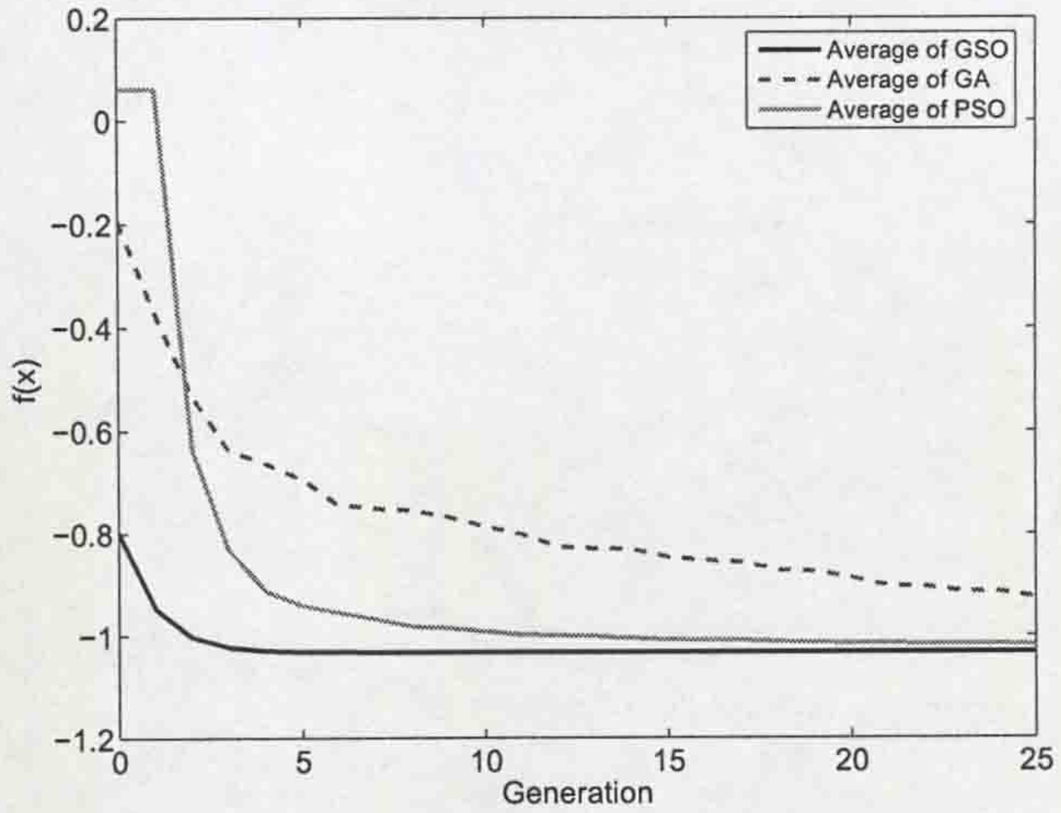


(a)

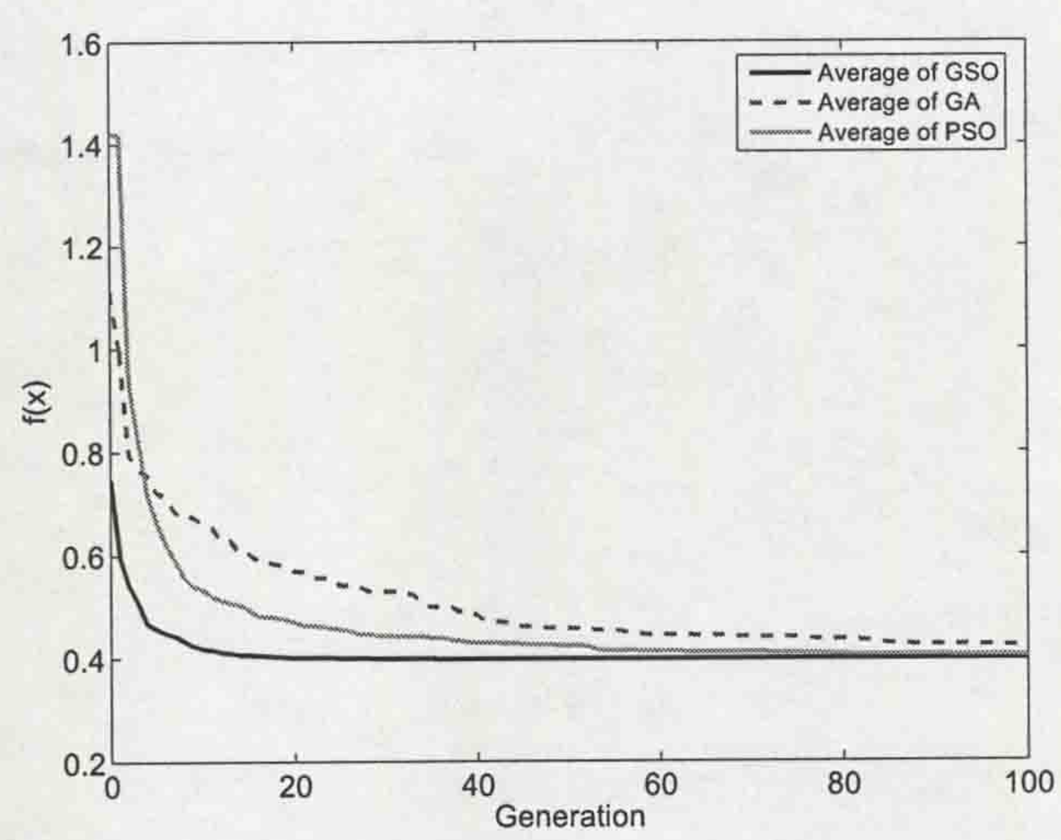


(b)

Figure 2.11: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a)-(b) correspond to functions f_{14}-f_{15}, respectively.

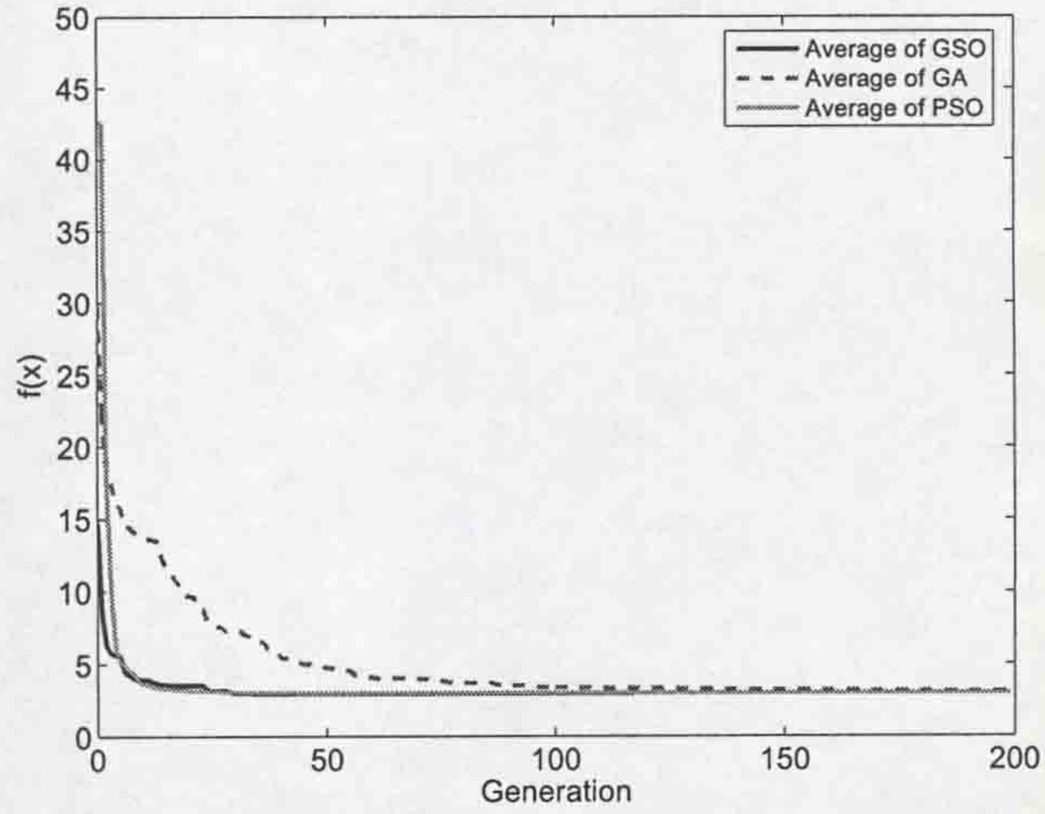


(a)

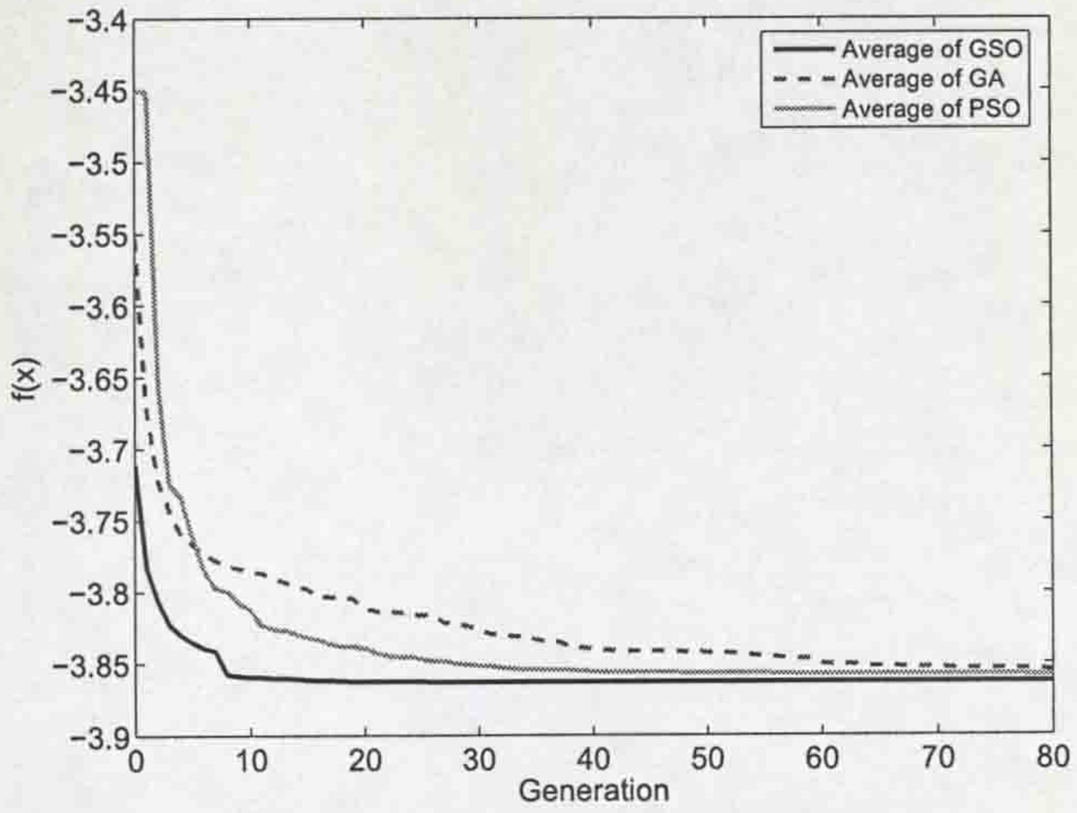


(b)

Figure 2.12: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a)-(b) correspond to functions f_{16}-f_{17}, respectively.

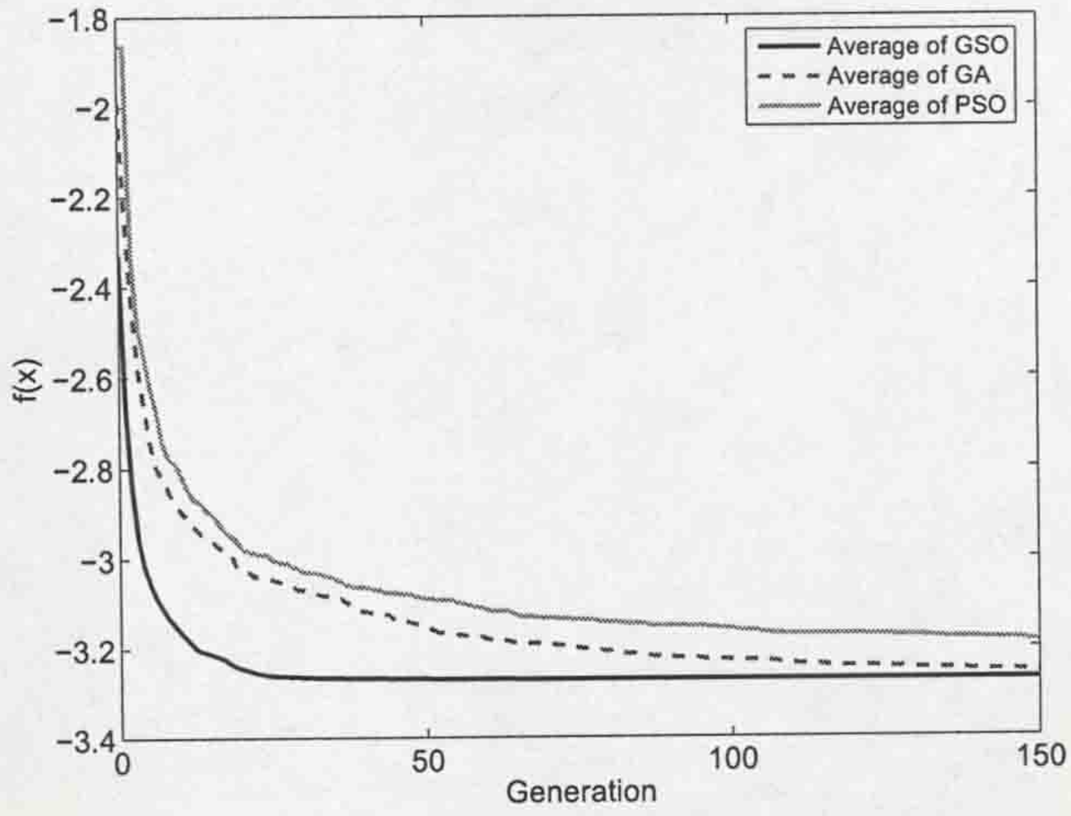


(a)

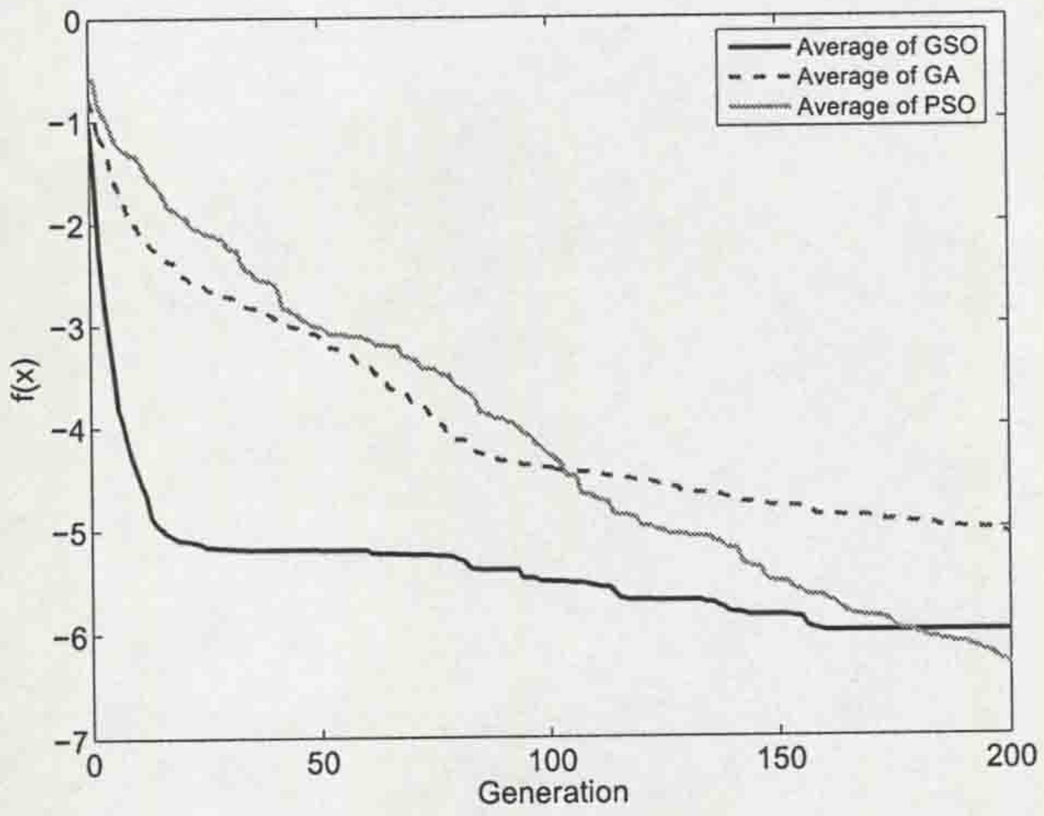


(b)

Figure 2.13: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a)-(b) correspond to functions f_{18}-f_{19}, respectively.

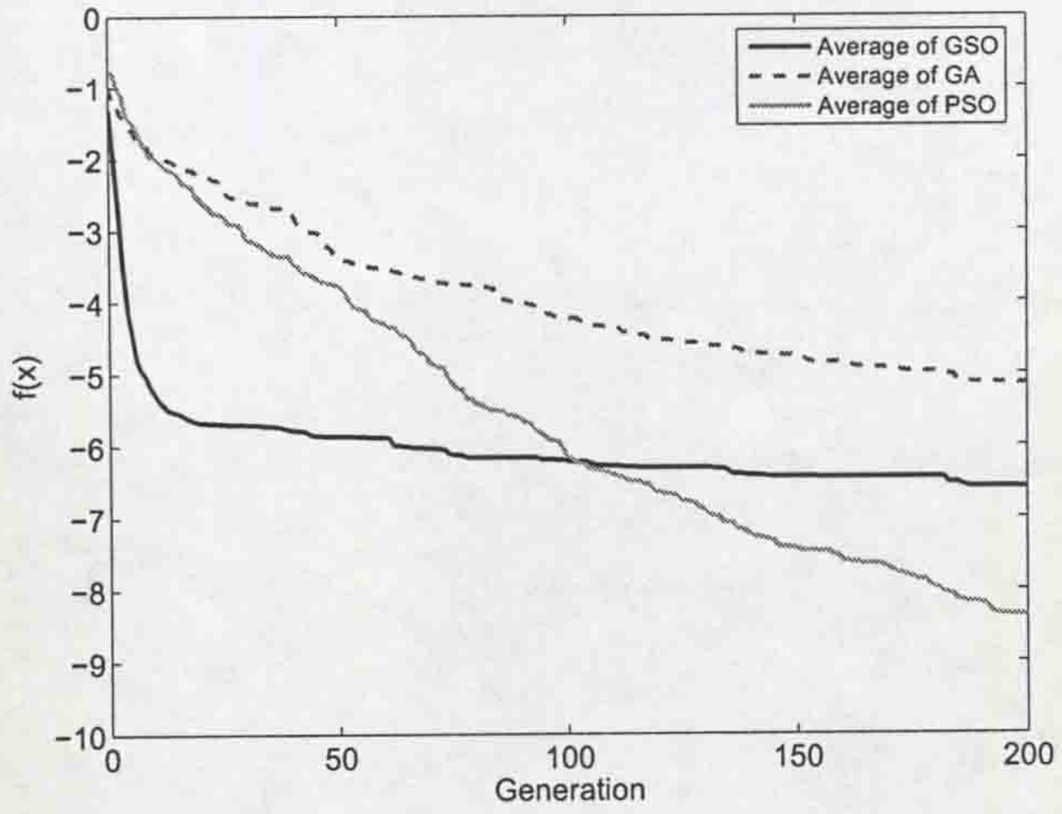


(a)

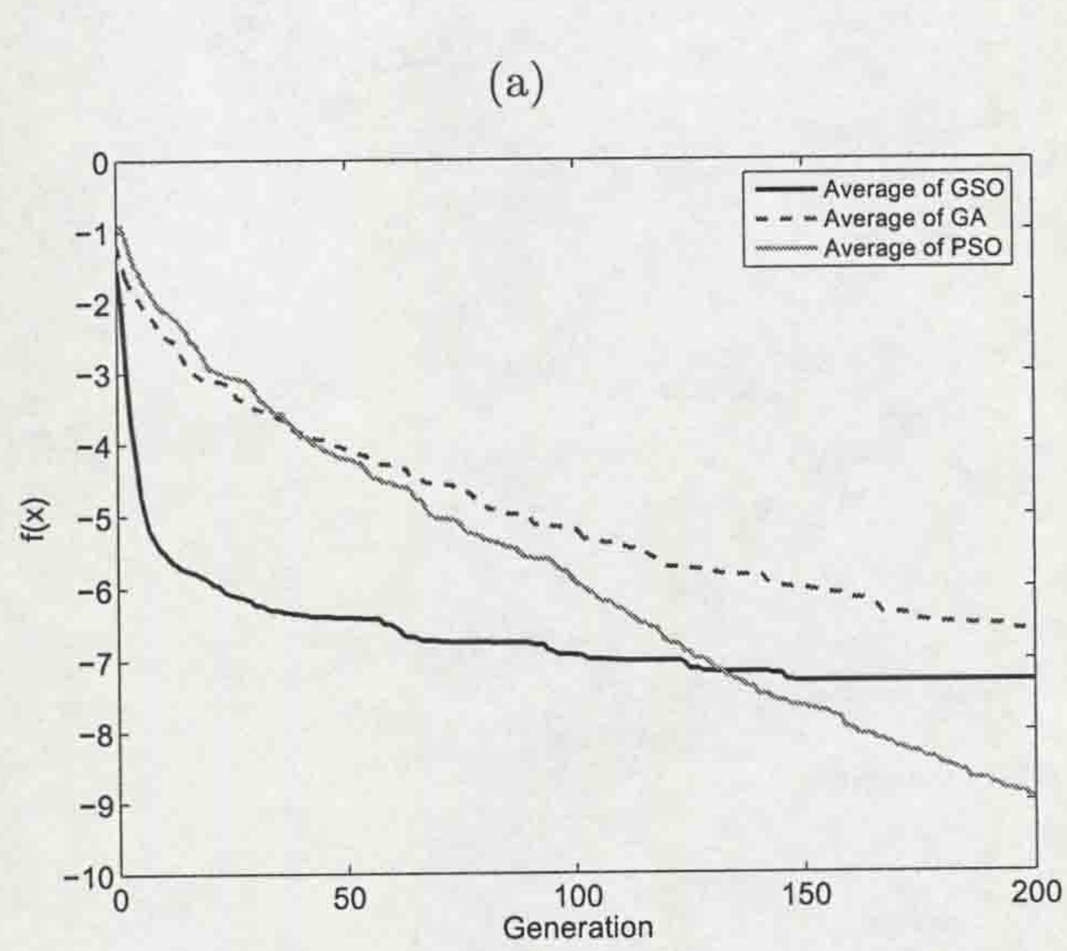


(b)

Figure 2.14: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a)-(b) correspond to functions f_{20}-f_{21}, respectively.



(a)



(b)

Figure 2.15: Convergence results of GSO, GA and PSO. The results were averaged over 50 runs. (a)-(b) correspond to functions f_{22}-f_{23}, respectively.

Table 2.11: Comparison among GSO with GA, PSO, EP and ES on benchmark functions $f_8(x)^{300}$ $f_{13}(x)^{300}$.

	Mean function value				
Function	GSO	GA	PSO	EP	ES
$f_8(x)^{300}$	-125351.2	-78088.1	-87449.2	-78311.9	-66531.3
$f_9(x)^{300}$	98.9	260.8	427.1	383.3	583.2
$f_{10}(x)^{300}$	1.3527×10^{-3}	11.37	3.9540×10^{-6}	0.2946	9.6243
$f_{11}(x)^{300}$	1.8239×10^{-7}	2.234	1.81	2.8244×10^{-2}	0.1583
$f_{12}(x)^{300}$	8.2582×10^{-8}	49.06	14.56	39.3	3093.2
$f_{13}(x)^{300}$	2.0175×10^{-7}	578.5	549.2	738.2	2123.2

2.5 Discussion

Currently, there are only a few optimisation algorithms inspired by animal behavior. The most notable and successful one is ACO. Although both GSO and ACO drew inspiration from animal social animal foraging behavior, there are many obvious differences. The most distinct one is that ACO was inspired specifically by behavior of ant colonies: by laying pheromone trails, ants collectively establish the shortest path between their colony to feeding sources. The GSO algorithm was inspired by general animal searching behavior and a generic social foraging model, *e.g.* Producer-Scrounger model. Another difference is that ACO was proposed primarily for combinatorial optimisation problems while at present GSO is more applicable to continuous function optimisation problems.

PSO is another newly emerged optimisation algorithm inspired by animal behavior. Like GSO, it was also proposed for continuous function optimisation problems. However, it is not difficult to note that there are some major differences between GSO and PSO. First and the most fundamental one is that PSO was originally developed from the models of coordinated animal motion such as Reynolds's Boids [91] and Heppner and Grenander's model [92]. Animal swarm behavior, mainly bird flocking and fish schooling, serves as the

metaphor for the design of PSO. The GSO algorithm was inspired by general animal searching behavior. A generic social foraging model, *e.g.*, Producer-Scrounger model, was employed as the framework to derive GSO. Secondly, although the producer of GSO is quite similar to the global best particle of PSO, the major difference is that the producer performs producing, which is a searching strategy differs from the strategies performed by the scroungers and the dispersed members. While in PSO, each individual performs the same searching strategy. Thirdly, in GSO, individuals do not posses memory. However, in PSO each individual maintains memory to remember the best place it visited. Finally, unlike GSO, there is no dispersed group members which perform ranging strategy in PSO.

Although the EAs and GSO were inspired by completely different disciplines, as a population-based algorithm, GSO shares some similarities with other EAs. For example, they both use the concept of fitness to guide search towards better solutions; the scrounging behavior of scroungers is similar to the crossover operator, *e.g.*, extended intermediate crossover [93] of real-coded GA; and the ranging behavior of dispersed members can be regarded as EAs' mutation operator which both produce new solutions by perturbation. However, under millions even billions of years of natural selection, animal behavior, especially searching behavior, has been honed and sharpened by evolution. Research in animal behavior provides many off-the-shelf searching strategies to be incorporated into GSO to solve different hard optimisation problems. For example, animal's strategies to deal with resources changing over time (*e.g.*, leaving patch when profitability declines) could be employed by GSO to solve optimisation problems in dynamic environments. This is our work in the future.

2.6 Conclusions

We have proposed a novel optimisation algorithm — GSO, which is based on animal searching behavior and group living theory. This algorithm is concep-

tually simple and easy to implement. It has only one parameter (percentage of dispersed members) to tune and can handle a variety of optimisation problems (including large-scale), which makes it particularly attractive for real-world applications.

A set of 23 benchmark functions have been used to test GSO in comparison with GA, PSO, CEP, FEP, CES and FES, respectively. For the uni-modal functions, the results show that the GSO does not possess an obvious advantage to PSO but has a better performance to that of GA in terms of accuracy and convergence rate. Compared to CEP, FEP, CES and FES, GSO was outperformed by FEP an FES. For most of the multi-modal benchmark functions with many local minima, GSO is able to statistically find better average results than those generated by the GA and PSO and find better average results than the other four algorithms. The test results obtained from the multi-modal benchmark functions, which have a few local minima, GSO also outperformed the other six algorithms. We have also evaluated the GSO on a set of multi-modal functions in 300 dimensions. In these cases, the GSO appeared to be an overpowering winner compared with the GA, PSO, EP and ES.

A new paradigm of swarm intelligence, GSO, has been presented in this chapter. One of the most significant merits of GSO is that provides an open framework to utilize research in animal behavioral ecology to tackle hard optimisation problems.

Chapter 3

Improve PSO with Passive Congregation

This chapter presents a particle swarm optimiser (PSO) with passive congregation to improve the performance of standard PSO (SPSO). Passive congregation is an important biological force preserving swarm integrity. By introducing passive congregation to PSO, information can be transferred among individuals of the swarm. A particle swarm optimiser (PSO) with passive congregation (PSOPC) is tested with a set of 10 benchmark functions with 30 dimensions and compared to a global version of SPSO (GSPSO), a local version of SPSO (LSPSO), and PSO with a constriction factor (CPSO) respectively. Experimental results indicate that the PSO with passive congregation improves the search performance on the benchmark functions significantly.

3.1 Introduction

The particle swarm optimiser (PSO) is a population-based algorithm that was invented by [32], which was inspired by the social behavior of animals such as fish schooling and bird flocking. Similar to other population-based algorithms such as evolutionary algorithms, PSO can solve a variety of difficult optimisation problems but has shown a faster convergence rate than other

evolutionary algorithms on some problems ([21]). Another advantage of PSO is that it has very few parameters to adjust which makes it particularly easy to implement.

[94] pointed out that although PSO may outperform other evolutionary algorithms in the early iterations, its performance may not be competitive as the number of generations is increased. Recently, several investigations have been undertaken to improve the performance of standard PSO (SPSO). [95] presented a hybrid PSO model with breeding and subpopulations. [96] investigated the impacts of population structures to the search performance of SPSO. Other investigations on improving PSO's performance were undertaken using cluster analysis [97] and fuzzy adaptive inertia weight [98].

The foundation of PSO is based on the hypothesis that social sharing of information among conspecifics offers an evolutionary advantage [32]. The SPSO model is based on the following two factors [32]:

1) The autobiographical memory, which remembers the best previous position of each individual (P_i) in the swarm;

2) The publicized knowledge, which is the best solution (P_g) found currently by the population.

Therefore the sharing of information among conspecifics is achieved by employing the publicly available information P_g, shown in Fig. 3.1. There is no information sharing among individuals except that P_g broadcasts the information to the other individuals. Therefore, the population may lose diversity and is more likely to confine the search around local minima if committed too early in the search to the global best found so far.

Biologists have proposed four types of biological mechanisms that allow animals to aggregate into groups: passive aggregation, active aggregation, passive congregation, and social congregation [30]. There are different information sharing mechanisms inside these forces. We found that the passive congregation model is suitable to be incorporated in the SPSO model. Inspired by this research, we propose a hybrid model of PSO with passive congregation.

Section 3.2 introduces several animal aggregation models. A PSO algorithm

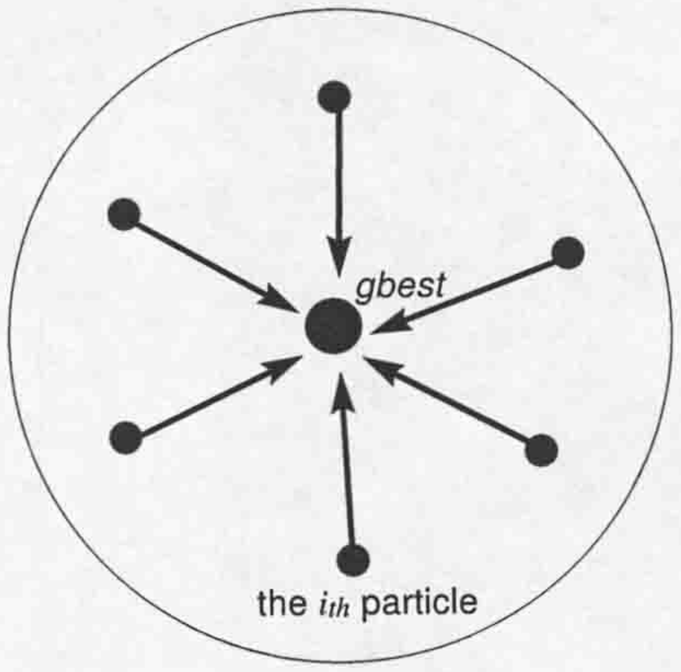


Figure 3.1: Interaction between particles and the best particle *gbest*.

with passive congregation is presented in section 3.3. In section 4.3, we describe the test functions, experimental settings, and the experimental results. The discussions are given in section 3.5. This chapter is concluded in section 6.5.

3.2 Biological Forces Behind Animal Aggregations

The PSO algorithm is inspired by social behaviors such as spatial order, more specially, aggregation such as bird flocking, fish schooling, or swarming of insects. Each of these cases has stable spatio-temporal integrities of the group of organisms: the group moves persistently as a whole without losing the shape and density. For example, the individual fish in the school do not appear to act selfishly but rather they seem to behave and interact for the benefit of the school as a whole. If the individuals within a school did not look and behave similarly, then the primary anti-predatory advantage associated with schooling could not exist. Indeed, cohesion and coherence are hallmarks of many types of animal aggregation [30]. By natural selection, behavioural patterns emphasize similarity and uniformity within a group [30]. In order to study these patterns behind animal aggregations, in this section we will presents some background concepts about animal aggregations.

For animal groups, different biological forces are essential for preserving the group's integrity. [30] proposed mathematical models of the spatial structure of animal groups to show how animals organize themselves. In these models, aggregation sometimes refers to a grouping of the organisms by non-social, external, physical forces. There are two types of aggregation: passive aggregation and active aggregation. Passive aggregation is a passive grouping by physical processes. One example of passive aggregation is the dense aggregation of plankton in open water, in which the plankton are not attracted actively to the aggregation but are transported passively there via physical forces such as water currents. Active aggregation is a grouping by attractive resource, such as food or space, with each member of the group recruited to a specific location actively. In these situations the aggregation will disperse if the attractive source wanes. Individuals in the aggregation also may continuously join and leave as discussed in Chapter 2, rather than remain continuous members. As results, turn over in the aggregation may be high even the size, density or shape of the aggregation remain fairly constant. Repulsion also plays a crucial role in determining group structure [99]. Repulsion helps an aggregation to avoid unmitigated attraction which may result in the costs of individual members outweighing the benefits. The combination of attractive and repulsive forces defines the physical attributes of the group.

Congregation, which is different from aggregation, is a grouping by social forces, that is the source of attraction is the group itself. Examples of animal congregations include flocks of birds, schools of fish, swarms of insects. The forces shape congregations include internal, *i.e.*, member-derived, forces, external forces, and frictional forces.

Congregation can be classified into passive congregation and social congregation. Passive congregation is an attraction of an individual to other group members but where there is no display of social behavior. There are very little genetic relation to each other in a passive congregation, and thus they display low fidelity to the group and no reciprocal altruism. The flocks of birds, schools of fish, swarms of insects should be classified as passive congre-

gation. Social congregations usually happen in a group where the members are related (sometimes highly related). A variety of inter-individual behaviors are displayed in social congregations, necessitating active information transfer [30]. For example, ants use antennal contacts to transfer information about individual identity or location of resources [100].

Many animal congregations share one or more of the following properties [30]:

1. Edges. Many types of animal congregations have very distinct edges. The change in density from inside to outside is abrupt. When an animal congregation moves or changes shape, the edges remain intact.

2. Uniform densities. This property can be found in many types of animal congregations, particularly when on the move, *e.g.*. herds, flocks, schools. Some types of animal congregations, *e.g.* midge swarms, may have a broader distribution of densities most of the time but retain the ability to assemble into a more uniform mass.

3. Polarized. Animal congregations with uniform density are often also polarized. In such a polarized congregation, all members face in the same direction.

4. Freedom to move. Whether in a polarized group on not, individuals within the group have the freedom to move with respect to their neighbours. In a resting group individuals can shift positions constantly, even if the position or shape of the congregation remains static. In moving groups individuals can re-sort without disturbing the integrity of the group.

5. Coordinated movement patterns. These almost ballet-like movement patterns can be found in many animal congregations. For example, flocks of birds appear to turn simultaneously. Fish in schools display a fountain-like pattern in response to an attack by a predator, completing the move by re-aggregating behind the predator.

3.3 Particle Swarm Optimiser with Passive Congregation

From the definitions in 3.2, the third part of equation (1): $c_2r_2(P_g^k - Xi^k)$ can be classified as either active aggregation or passive congregation. But since P_g is the best solution the swarm has found so far, which can be regarded as the place with most food, we argue that it is better to classify $c_2r_2(P_g^k - Xi^k)$ as active aggregation.

It has been discovered that in spatially well-defined congregations, such as fish schools, individuals may have low fidelity to the group because the congregations may be composed of individuals with little to no genetic relation to each other [101]. Schooling fish are generally considered a "selfish herd" [102], in that each individual attempts to take the sweeping generalization advantage from group living, independent of the fates of neighbors [103]. In these congregations, information may be transferred passively rather than actively [104]. Such asocial types of congregations can be referred to as passive congregation. Because PSO is inspired by fish schooling, it is therefore natural to ask if a passive congregation model can be employed to increase the performance of SPSO. Here we do not consider other models such as passive aggregation, because PSO is not aggregated passively via physical processes. And social congregation usually happens when group fidelity is high, such that the chance of each individual meeting any of the others is high [105]. Social congregations frequently display a division of labor. In a social insect colony such as an ant colony, large tasks are accomplished collectively by groups of specialized individuals, which is more efficient than performing sequentially by unspecialized individuals [17]. The concept of labor division can be employed by data clustering, sorting [106] and data analysis [107].

Group members in an aggregation can react without direct detection of an incoming signal from the environment, because they can get necessary information from their neighbors [30]. Individuals need to monitor both environment and their immediate surroundings, such as the bearing and speed of their

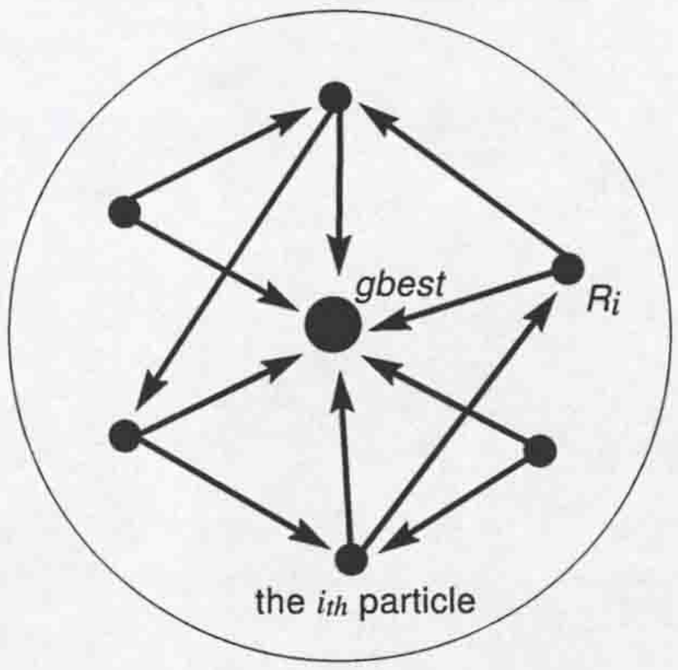


Figure 3.2: Interactions of particles with passive congregation

neighbors [30]). Therefore each individual in an aggregation has a multitude of potential information from other group members that may minimize the chance of missed detection and incorrect interpretations [30]. Such information transfer can be employed in the model of passive congregation. Inspired by this result, and to keep the model simple and uniform with SPSO, we propose a hybrid PSO with passive congregation:

$$V_i^{k+1} = \omega V_i^k + c_1 r_1 (P_i^k - X_i^k) + c_2 r_2 (P_g^k - X_i^k) + c_3 r_3 (R_i^k - X_i^k) \quad (3.3.1)$$

$$X_i^{k+1} = X_i^k + V_i^{k+1} \quad (3.3.2)$$

where R_i is a particle selected randomly from the swarm, c_3 is the passive congregation coefficient, and r_3 is a uniform random sequence in the range (0,1): $r_3 \sim U(0,1)$. The interactions between individuals of PSOPC are shown in Fig. 3.2. The search directions of standard PSO and PSOPC are shown in Fig. 3.4 and Fig. 3.3, respectively. The pseudocode for PSOPC is listed in Table 3.1. We implemented the PSOPC algorithm in MATLAB 6.5 and executed it on a Pentium 4, 2 GHz machine.

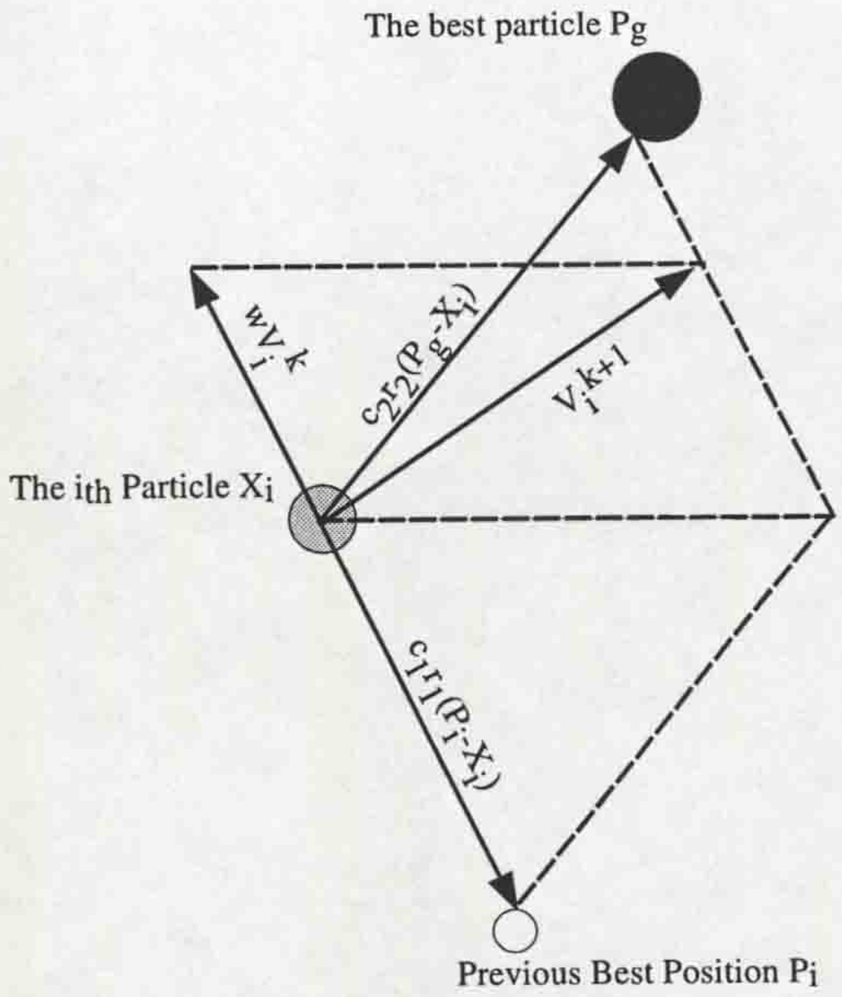


Figure 3.3: Search direction of the ith particle in PSO

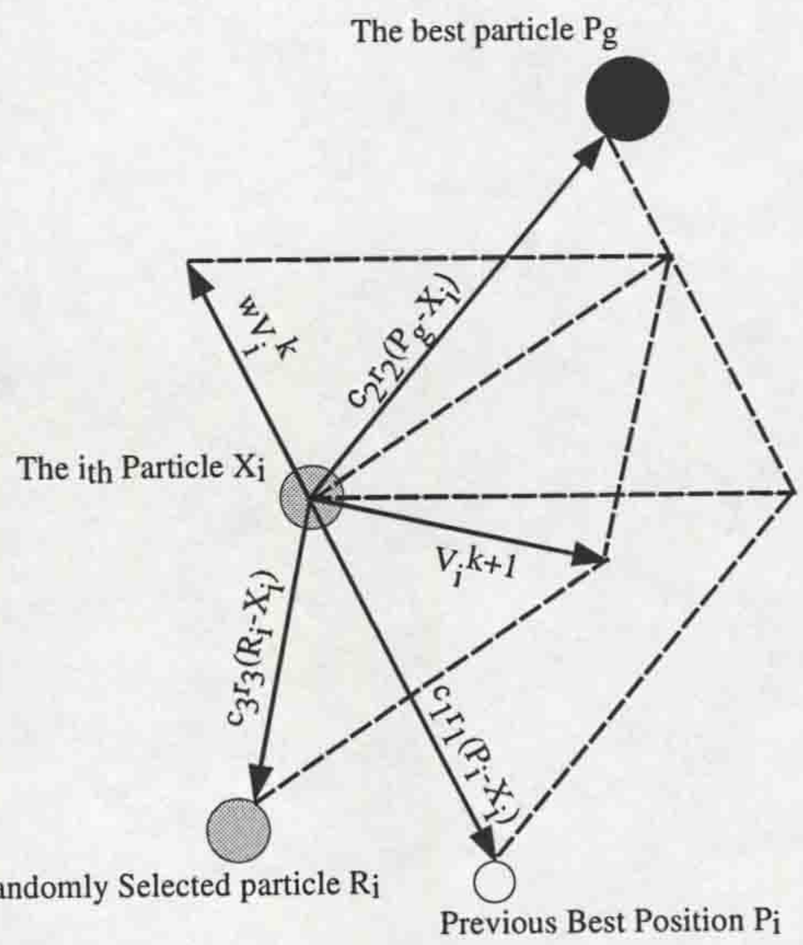


Figure 3.4: Search direction of the ith particle in PSOPC

Table 3.1: Pseudocode for the PSOPC algorithm.

Set $k := 0$;

Randomly initialize positions and velocities of all particles;

WHILE (the termination conditions are not met)

FOR (each particle i in the swarm)

Calculate fitness: Calculate the fitness value of current particle: $f(X_i)$;

Update *pbest*: Compare the fitness value of *pbest* with $f(X_i)$. If $f(X_i)$ is better than the fitness value of *pbest*, then set *pbest* to the current position X_i;

Update *gbest*: Find the global best position of the swarm. If $f(X_i)$ is better than the fitness value of *gbest*, then *gbest* is set to the position of the current particle X_i;

Update R_i: Randomly select a particle from the swarm as R_i;

Update velocities: Calculate velocities V_i using equation (3.3.1). If $V_i > V_{\max}$ then $V_i = V_{\max}$. If $V_i < V_{\min}$ then $V_i = V_{\min}$;

Update positions: Calculate positions X_i using equation (3.3.2);

END FOR

Set $k := k + 1$;

END WHILE

3.4 Experimental Studies

3.4.1 Test functions

In our experimental studies a set of 10 benchmark functions were employed to evaluate the PSOPC algorithm in comparison with others.

Sphere Model:

$$f_1(x) = \sum_{i=1}^{30} x_i^2$$

Schwefel's Problem 1.2:

$$f_2(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2$$

Schwefel's Problem 2.21:

$$f_3(x) = \max_i \{|x_i|, 1 \leq i \leq 30\}$$

Generalized Rosenbrock's Function:

$$f_4(x) = \sum_{i=1}^{29} (100(x_{i+1} - x_i^2)^2 + (x_i - 1))^2$$

Generalized Schwefel's Problem 2.26:

$$f_5(x) = -\sum_{i=1}^{30} \left(x_i \sin\left(\sqrt{|x_i|} \right) \right)$$

Generalized Rastrigin's Function:

$$f_6(x) = \sum_{i=1}^{30} (x_i^2 - 10\cos(2\pi x_i) + 10)^2$$

Ackley's Function:

$$f_7(x) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30} x_i^2} \right) - \exp\left(\frac{1}{30}\sum_{i=1}^{30} \cos 2\pi x_i \right) + 20 + e$$

Generalized Griewank Function:

$$f_8(x) = \frac{1}{4000}\sum_{i=1}^{30}(x_i - 100)^2 - \prod_{i=1}^{30}\cos(\frac{x_i - 100}{\sqrt{i}}) + 1$$

Generalized Penalized Functions:

$$f_9 = \frac{\pi}{30}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{30} u(x_i, 10, 100, 4) \quad (3.4.1)$$

and

$$f_{10} = 0.1\left\{\sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} + \sum_{i=1}^{30} u(x_i, 10, 100, 4) \quad (3.4.2)$$

where

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \le x_i \le a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

The above benchmark functions were tested widely by [58], [108], and [109]. They can be grouped as unimodal (function f_1 to f_4) and multimodal functions (function f_5 to f_{10}) where the number of local minima increases exponentially with the problem dimension. The dimension of each function n, feasible solution space, and $f_{\min}$ are listed in Table 3.2.

3.4.2 Experimental setting

To evaluate the performance of the proposed PSOPC, three variants of standard PSO were used for comparisons: global version of standard PSO

Table 3.2: Basic characters of the test functions.

Function	n	Feasible solution space	$f_{\min}$
f_1	30	$[-100, 100]^n$	0
f_2	30	$[-100, 100]^n$	0
f_3	30	$[-100, 100]^n$	0
f_4	30	$[-30, 30]^n$	0
f_5	30	$[-500, 500]^n$	-12569.5
f_6	30	$[-5.12, 5.12]^n$	0
f_7	30	$[-32, 32]^n$	0
f_8	30	$[-600, 600]^n$	0
f_9	30	$[-50, 50]^n$	0
f_{10}	30	$[-50, 50]^n$	0

(GSPSO), local version of standard PSO (LSPSO), and constriction factor version of PSO (CPSO). The parameters used for these three standard PSO were recommended from [21], [36], [35], [34], and [96], or hand selected.

The population size of all algorithms used in our experiments was set at 100. The maximum velocity $V_{\max}$ and minimum velocity $V_{\min}$ for GSPSO, LSPSO and CPSO were set at half value of the upper bound and lower bound respectively. $V_{\max}$ and $V_{\min}$ for PSOPC was set to the upper bound and lower bound respectively. The acceleration constants c_1 and c_2 for GSPSO and LSPSO were both 2.0 [21]. For CPSO, a setting of $c_1 = c_2 = 2.05$ was adopted [36]. The acceleration constants $c_1 = c_2 = 0.5$ were used in PSOPC.

The inertia weight ω is critical for the convergence behavior of GSPSO and LSPSO. A suitable value for the inertia weight ω usually provides a balance between global and local exploration abilities and consequently results in a better optimum solution. Initially, the inertia weight was constant. However, experimental results indicated that it is better to initially set the inertia to a large value in order to promote global exploration of the search space and decrease it to get more refined solutions [35]. Therefore, a decaying inertia weight starting at 0.9 and ending at 0.4 following [34], was used for GSPSO

and LSPSO. The inertia weight for PSOPC started at 0.9 and ended at 0.7. For CPSO, the constriction factor was calculated with equation (1.2.6), that is $\chi = 0.73$. The neighborhood size of LSPSO was set to be 2 [96].

The newly introduced passive congregation coefficient c_3 is important for the search performance of PSOPC. Experiments were executed to select a proper value of c_3. Four benchmark functions: f_1 (Sphere function), f_5 (Rosenbrock function), f_9 (Rastrigin function), and f_{11} (Griewank function) were tested with different values of c_3. The average test results obtained from 25 runs are listed in Table 3.3. When $c_3 = 0.6$, PSOPC generated good results on functions f_1 and f_6. For functions f_4 and f_8, the best results were generated at the point $c_3 = 0.8$. With $c_3 \geq 0.9$ the search performance of PSOPC on function f_1, f_4, and f_8 is deteriorated. For functions f_6, c_3 should be equal or smaller than 0.6 otherwise PSOPC will not converge in 2000 generations. Therefore, a generic c_3 for all functions should be equal or smaller than 0.6.

It is our interest to investigate whether PSOPC with a linear increasing c_3 generates better results on the benchmark functions than PSOPC with a fixed value of c_3. Therefore, f_9 (Rastrigin function) was selected and tested with different ranges of linearly increasing c_3. The results are tabulated in Table 3.4. The best result was generated by PSOPC with a linearly increasing passive congregation coefficient c_3, which started at 0.4 and ended at 0.6.

The parameters setting for all algorithms are summarized in Table 3.5.

All experiments were repeated for 50 runs. A fixed number of maximum generations 2000 was applied to all algorithms.

3.4.3 Experimental results and comparison

The experimental results (*i.e.*, the mean and the standard deviations of the function values found in 50 runs) for each algorithm on each test function are listed in Table 3.6. To measure the statistical significance of our experimental results between PSOPC and other three standard PSO variants, a set of two-tailed tests were adopted. The results are listed in Table 3.7. The critical value with 49 degrees of freedom at $\alpha = 0.05$ is 2.0, which means if $|t| > 2.0$ the

Table 3.3: Average fitness values of functions f_1, f_4, f_6 and f_8 with different c_3.

	Function			
c_3	f_1	f_4	f_6	f_8
0.0	5.6×10^{-12}	69.65	29.02	1.2×10^{-2}
0.1	1.4×10^{-8}	56.39	23.33	7.1×10^{-3}
0.2	9.6×10^{-11}	49.95	12.04	1.3×10^{-2}
0.3	6.5×10^{-14}	31.58	6.87	5.3×10^{-3}
0.4	4.2×10^{-18}	30.21	4.15	7.4×10^{-3}
0.5	1.1×10^{-20}	35.83	3.45	3.9×10^{-3}
0.6	5.8×10^{-28}	33.70	2.95	4.1×10^{-3}
0.7	2.7×10^{-21}	34.53	167.86	3.3×10^{-3}
0.8	4.6×10^{-6}	29.70	231.14	9.2×10^{-4}
0.9	4510.26	1.4×10^{6}	273.78	29.72
1.0	20336.59	1.4×10^{7}	300.20	129.17

Table 3.4: Average fitness value of Rastrigin (f9) function with different linearly increasing c_3.

	$c_{3\,\min}$						
$c_{3\,\max}$	0.1	0.2	0.3	0.4	0.5	0.6	0.7
0.1	23.33	NA	NA	NA	NA	NA	NA
0.2	22.54	12.04	NA	NA	NA	NA	NA
0.3	22.33	11.49	6.87	NA	NA	NA	NA
0.4	20.60	10.35	5.43	4.15	NA	NA	NA
0.5	21.04	11.75	5.23	4.24	3.45	NA	NA
0.6	17.71	8.91	4.43	2.66	2.79	2.95	NA
0.7	17.51	6.52	6.87	3.72	107.72	156.69	167.86
0.8	31.55	212.03	219.24	230.91	233.38	232.10	229.56
0.9	148.70	257.99	264.46	267.02	269.83	274.41	270.42
1.0	202.57	296.50	295.15	301.24	301.71	301.25	302.82

Table 3.5: Parameter Setting.

	PSOPC	GSPSO	LSPSO	CPSO
Population Size	100	100	100	100
Neighborhood Size	100	100	2	100
ω	[0.7, 0.9]	[0.4, 0.9]	[0.4, 0.9]	NA
χ	NA	NA	NA	0.73
c_1	0.5	2.0	2.0	2.05
c_2	0.5	2.0	2.0	2.05
c_3	[0.4, 0.6]	NA	NA	NA

difference between two means is statistically significant.

From Table 3.6, PSOPC outperformed the other three standard PSO algorithms significantly for most of the benchmark functions. The two exceptions are f_2 and f_{10}. For function f_2, the result generated by PSOPC is better than those generated by GSPSO and LSPSO but slightly worse than the result of CPSO. For function f_{10}, GSPSO slightly outperformed PSOPC while the result of PSOPC is far better than LSPSO and CPSO. However from Table 3.7, for functions f_2 and f_{10}, the results generated respectively by CPSO and GSPSO are not significantly better than PSOPC. For function f_9, the results obtained from PSOPC do not differ significantly from those generated by GSPSO. For functions f_4 and f_{10}, the differences between the results generated by PSOPC and CPSO are not statistically significant. It can be concluded that PSOPC significantly outperforms LSPSO on all the tested benchmark functions.

The performance of CPSO is better than GSPSO on all the unimodel benchmark functions (functions f_1 to f_4). But GSPSO results in good performance on the multimodel benchmark functions (functions f_5 to f_{10}). Although it is believed that LSPSO is able to "flow around" local optima [96], our experimental results have indicated that GSPSO and CPSO exhibit better global convergence performance. The search performance of 4 algorithms tested here can be ordered as PSOPC > GSPSO $\sim$ CPSO > LSPSO.

Figures 3.5 to 3.14 show the search progress of the average values and the

best solutions found by the 4 algorithms over 50 runs for functions f_1 to f_{10}. From these figures, for most of the benchmark functions, PSOPC quickly found the near optima in the early search process.

For unimodel functions (function f_1 to f_4), the convergence rates are more important than the final results of optimisation as there are other methods such as gradient-based search methods that are designed specially to optimise unimodal functions [58]. From Figs. 3.5 to 3.8, it can be seen that PSOPC has a faster convergence rate than other three algorithms.

Functions f_5 to f_{10} are multimodal functions that are very difficult to optimise since the number of local minima increases exponentially as the function dimension increases ([?] and [?]). The search process of four algorithms for f_5 to f_{10} are shown by Figs. 3.9 to 3.14. According to these figures, for most of the functions (f_5, f_6, f_8, f_9 and f_{10}), PSOPC converges near global minima while the other three algorithms were trapped by poor local minima and then stagnated. The only exception is about function f_7, for which GSPSO and CPSO did not fully converge when the maximum generations was reached.

Figure 3.5: f_1 (Sphere function)

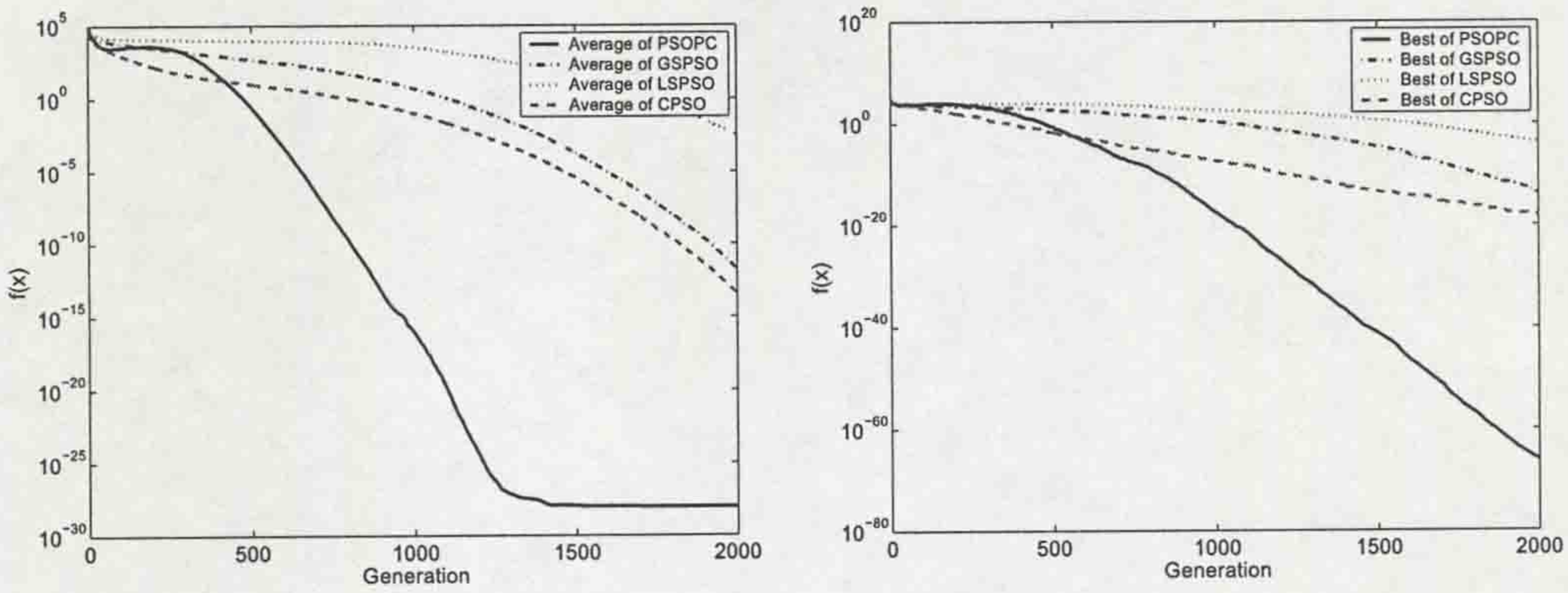
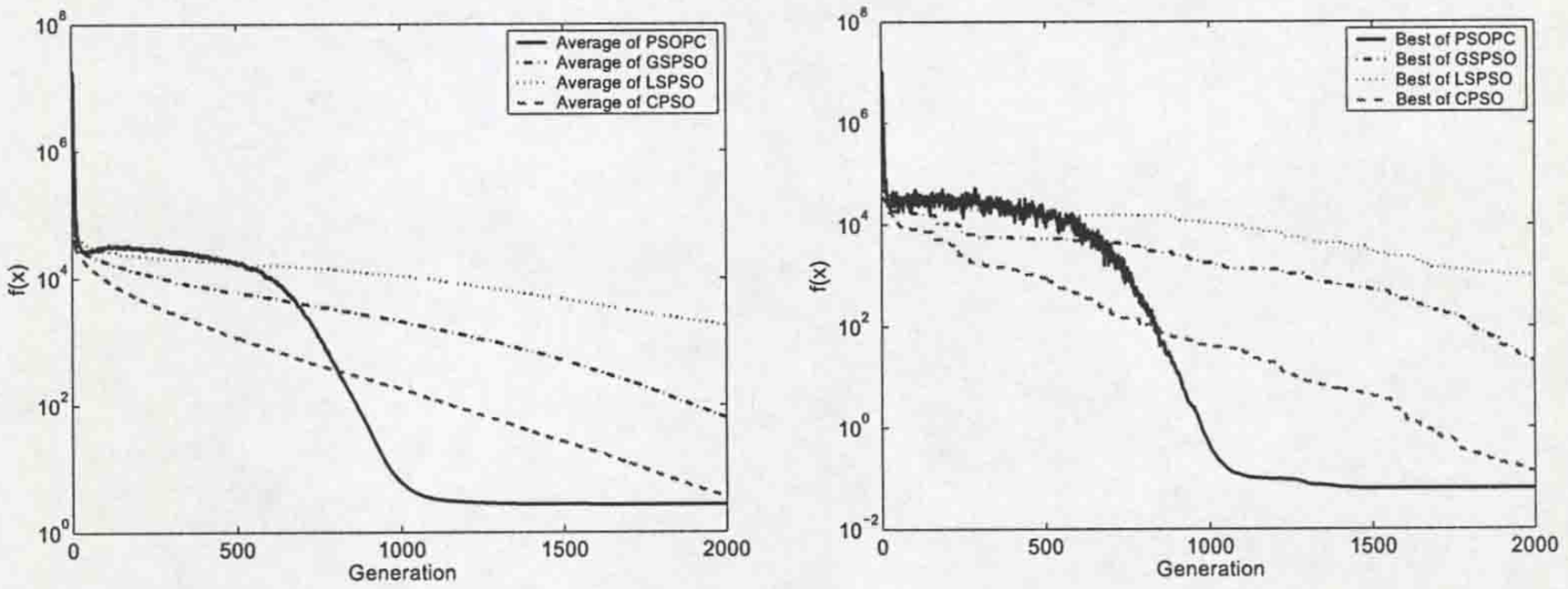


Figure 3.6: f_2 (Schwefel's Problem 1.2)



3.5 Discussion

Arithmetically, this passive congregation operator can be regarded as a stochastic variable that introduces perturbations to the search process. [110] also introduced a stochastic variable into the standard PSO, which is referred to *turbulence* in their paper. The velocity-updating equation is given by

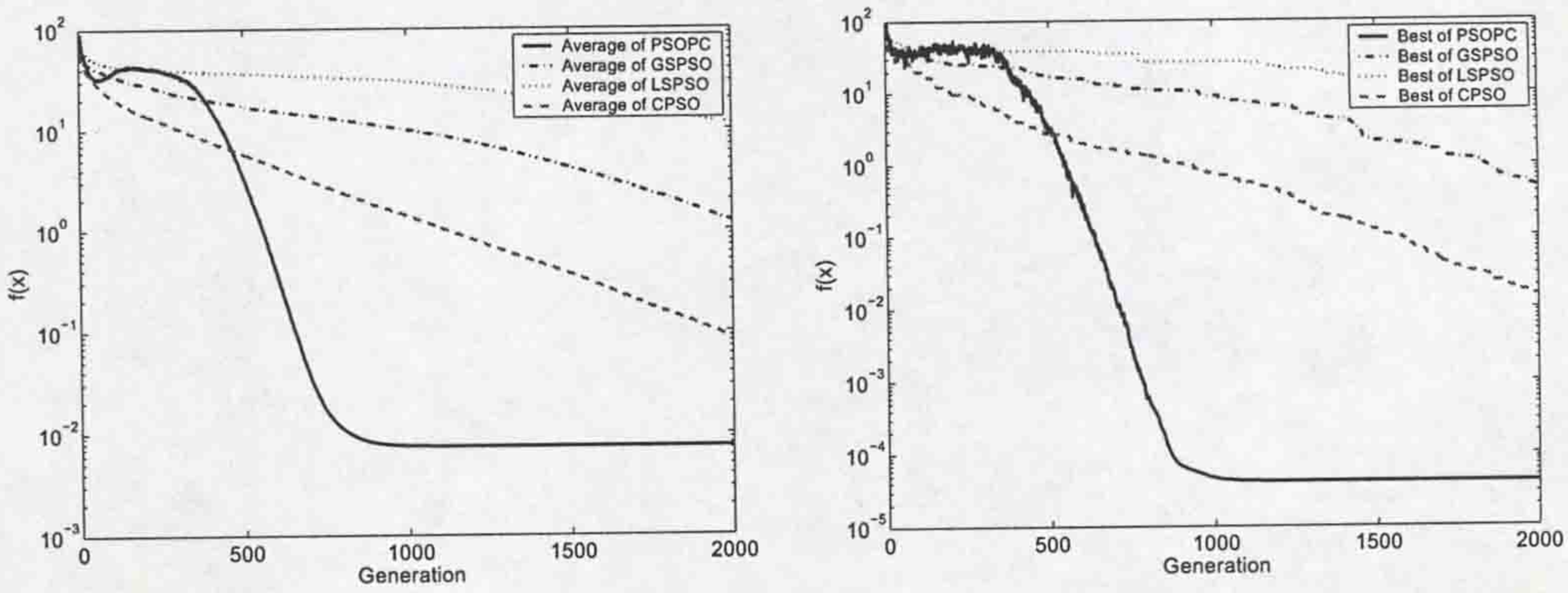
$$V_i^{k+1} = \omega V_i^k + c_1 r_1 (P_i^k - X_i^k) + c_2 r_2 (P_g^k - X_i^k) + r_3 \qquad (3.5.1)$$

where r_3 is a random variable $r_3 \sim U(0, 0.1R)$, and R is the absolute range of the model parameter.

From our experience, a large R will help the swarm escape local minima but may also cause the search process to diverge. A too small R may have no impact on search performance. The value of R is also problem-specific, *e.g.*, a suitable R for some benchmark functions will deteriorate the search performance on other functions. Therefore, finding a proper value of R is necessary for the best solution of an optimisation problem.

Compared with the turbulence factor r_3, the passive congregation operator $c_3 r_3 (R_i^k - X_i^k)$ is more adaptive to different optimisation problems. For each individual, the turbulence (perturbation) is proportional to the distance

Figure 3.7: f_3 (Schwefel's Problem 2.21)

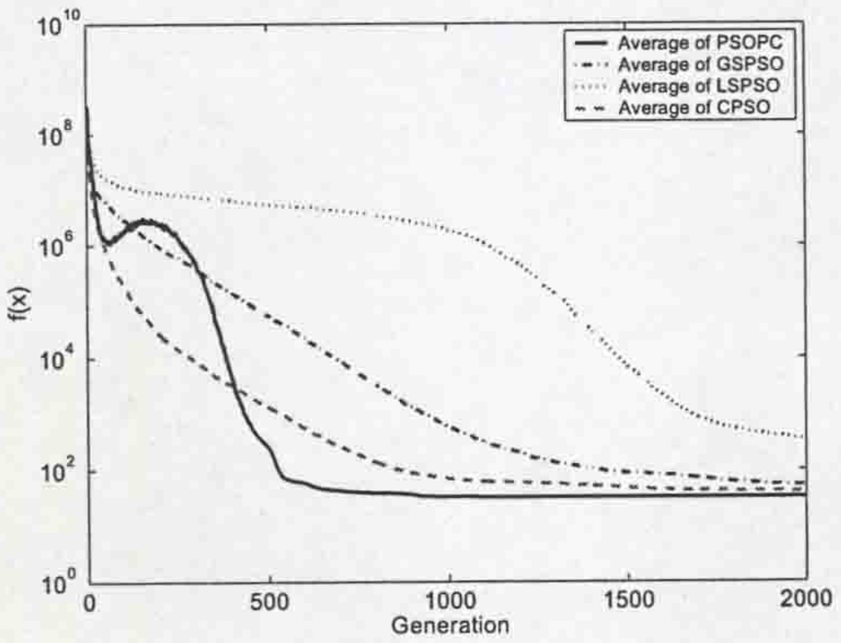


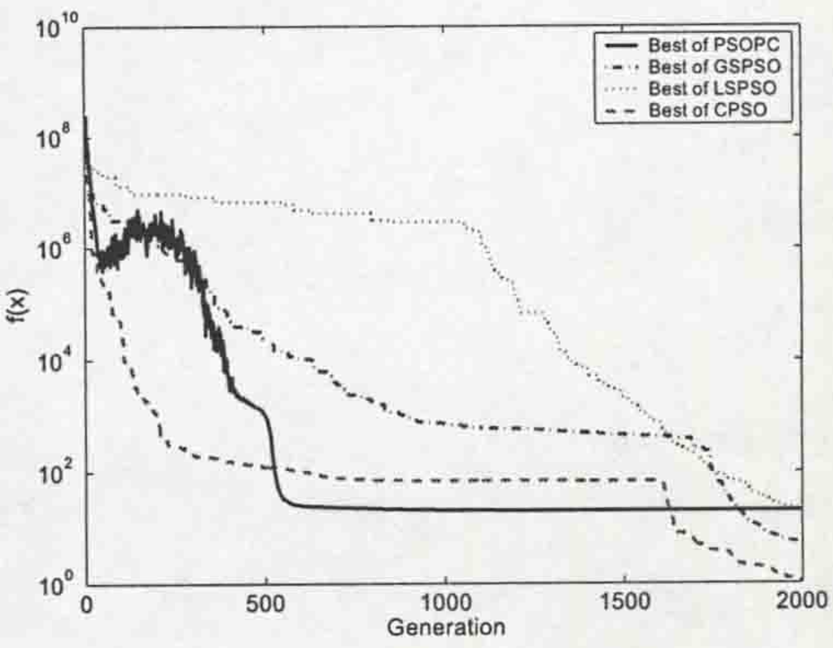
between itself and a randomly selected neighborhood rather than an external random number. In the early search process, the distances between individuals are large, therefore the turbulence is large, which may allow the swarm to avoid converging to a poor local minimum. As the generations increase, the distances between individuals become smaller, therefore the turbulence becomes smaller, which enables the swarm to refine solutions.

[96] investigated population topologies of PSO systematically. In their study, two sociometric variables, the number of neighbors for each node in the population k and the number of neighbors in common C, were varied to generated different topologies. One experiment, called random graphs, is implemented to generate different topologies by randomly initialized different k, C, standard deviation of $k(stdk)$, and standard deviation of $C(stdC)$, and then optimised by a method with a cooling mechanism that was inspired by simulated annealing. Since the PSO algorithm used in their work is the CPSO as defined in equation (1.2.5), the only factor affected by $k, C, stdk$, and $stdC$ is P_g. Therefore the essential result of their experiment is most likely finding a proper selection scheme for P_g rather than introducing a new information sharing mechanism into swarms as PSOPC does.

Shan He

Figure 3.8: f_4 (Generalized Rosenbrock function)



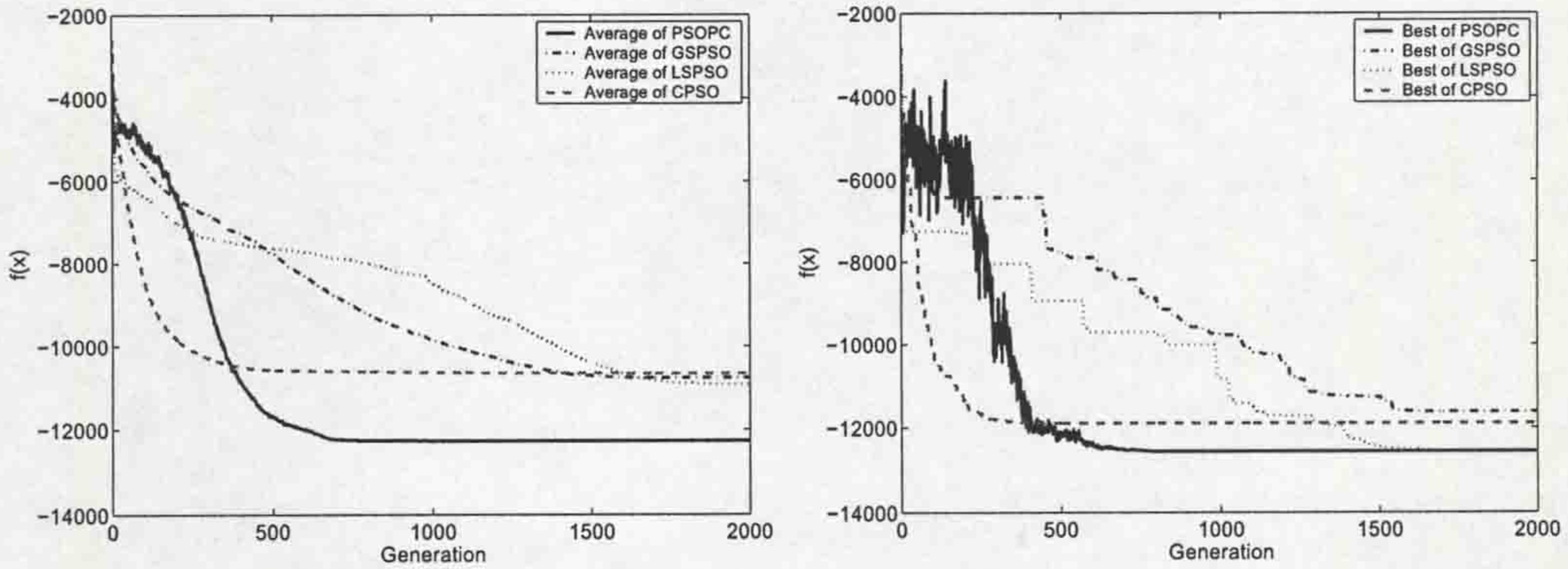


3.6 Conclusions

In this chapter a new PSO with passive congregation (PSOPC) has been presented based on the standard PSO. By introducing passive congregation, information can be transferred among individuals that will help individuals avoid misjudging information and becoming trapped by poor local minima. The only coefficient introduced into the standard PSO is the passive congregation coefficient c_3. A generic value of c_3 was selected by experiments.

A set of 10 benchmark functions have been used to test PSOPC in comparison with GSPSO, LSPSO, and CPSO. Among them, four functions were unimodal and six were multimodal. For the multimodal benchmark functions, PSOPC found better results on functions f_5 to f_9 than those generated by the other three standard PSO variants. For the unimodal functions, of which the convergence rate is more important than the final results, our PSOPC outperformed the other three algorithms in terms of accuracy and convergence rate on 3 out of 4 benchmark functions: f_1, f_3 and f_4. We also applied two-tailed tests to evaluate the statistical significance of differences between PSOPC and the other three algorithms. The results indicated that for 6 out of 10 benchmark functions, PSOPC performed significantly better than all other three standard

Figure 3.9: f_5 (Generalized Schwefel's Problem 2.26)



PSO variants.

Figure 3.10: f_6 (Generalized Rastrigin's function)

Figure 3.11: f_7 (Ackley's function)

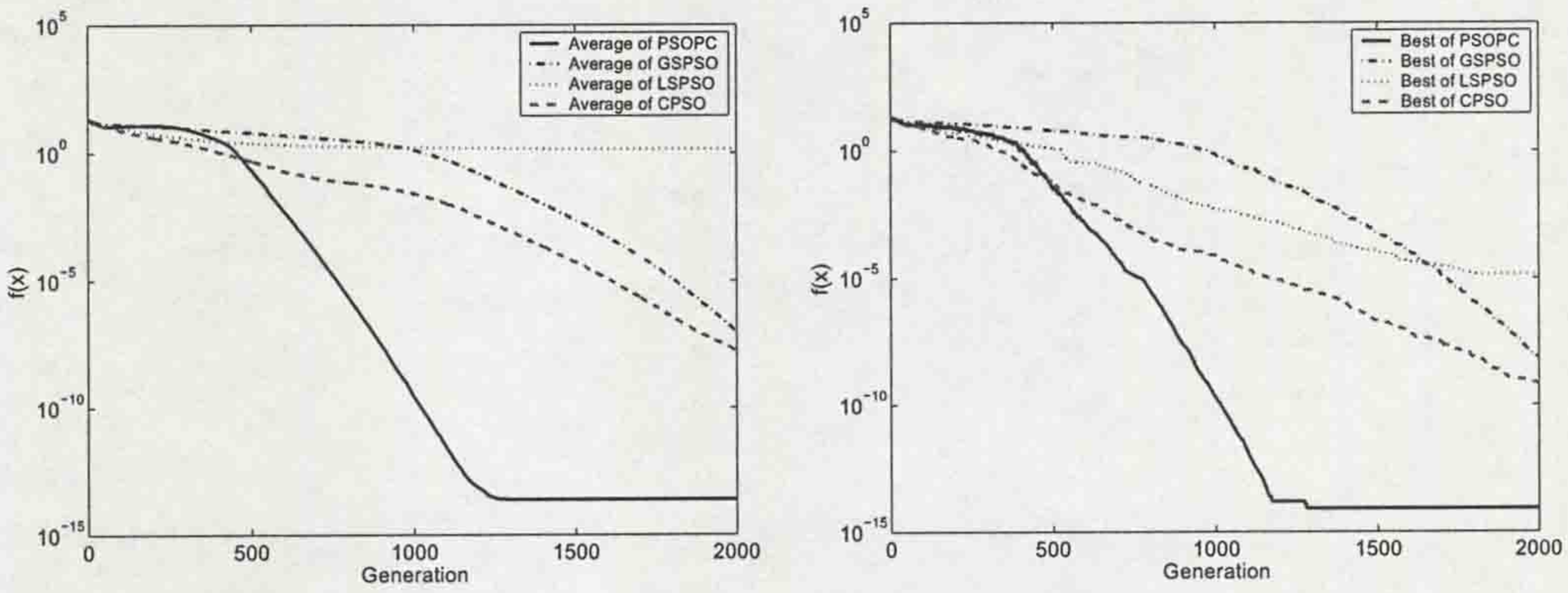


Table 3.6: Comparison between PSOPC, GSPSO, LSPSO, and CPSO.

	Mean function value (standard deviation)			
Function	PSOPC	GSPSO	LSPSO	CPSO
f_1	9.5×10^{-29} (5.9×10^{-28})	1.9×10^{-12} (3.7×10^{-12})	3.0×10^{-3} (5.3×10^{-3})	2.3×10^{-15} (4.9×10^{-15})
f_2	2.71 (2.79)	62.33 (34.69)	1805.03 (458.70)	2.29 (1.99)
f_3	7.5×10^{-3} (1.0×10^{-2})	1.2 (5.1×10^{-1})	11.37 (1.93)	6.1×10^{-2} (4.2×10^{-2})
f_4	32.44 (23.48)	52.83 (38.38)	347.92 (417.85)	39.70 (31.51)
f_5	-12267.77 (-451.86)	-10768.82 (-417.20)	-10928.65 (-1013.97)	-10443.47 (-618.94)
f_6	2.91 (1.65)	21.56 (5.12)	59.07 (10.00)	43.76 (12.13)
f_7	2.3×10^{-14} (2.2×10^{-14})	9.0×10^{-8} (9.3×10^{-8})	1.41 (8.6×10^{-1})	1.6×10^{-8} (1.8×10^{-8})
f_8	3.2×10^{-3} (5.6×10^{-3})	1.4×10^{-2} (1.6×10^{-2})	9.6×10^{-3} (1.0×10^{-2})	1.9×10^{-2} (2.0×10^{-2})
f_9	4.5×10^{-26} (3.2×10^{-25})	2.0×10^{-3} (1.5×10^{-2})	1.06 (9.2×10^{-1})	3.9×10^{-2} (8.6×10^{-2})
f_{10}	1.1×10^{-3} (3.3×10^{-3})	8.84×10^{-4} (3.0×10^{-3})	13.35 (21.57)	3.8×10^{-1} (1.8)

Table 3.7: Two-tailed test on PSOPC, GSPSO, LSPSO, and CPSO. The value of t with 49 degree of freedom is significant at $\alpha = 0.05$ by a two-tailed test and $t_{0.025} = 2.0$.

	t		
Function	PSOPC-GSPSO	PSOPC-LSPSO	PSOPC-CPSO
f_1	−3.57	−3.90	−3.28
f_2	−12.46	−27.77	0.85
f_3	−16.50	−41.71	−8.65
f_4	−3.25	−5.29	−1.17
f_5	−15.76	−8.05	−17.87
f_6	−24.14	−40.23	−23.86
f_7	−6.82	−11.62	−6.42
f_8	−4.88	−3.80	−5.14
f_9	−1.00	−8.09	−3.24
f_{10}	0.37	−4.37	−1.44

Figure 3.12: f_8 (Generalized Griewank function)

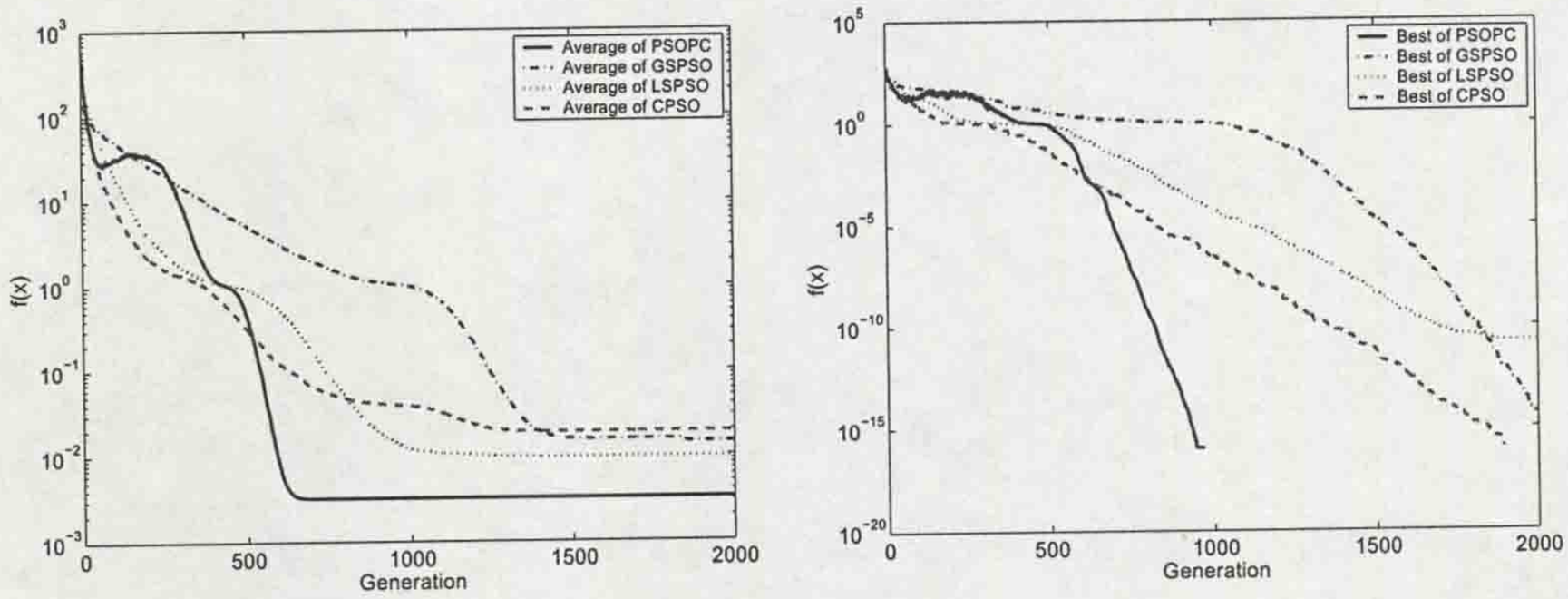


Figure 3.13: f_9 (Penalized function P8)

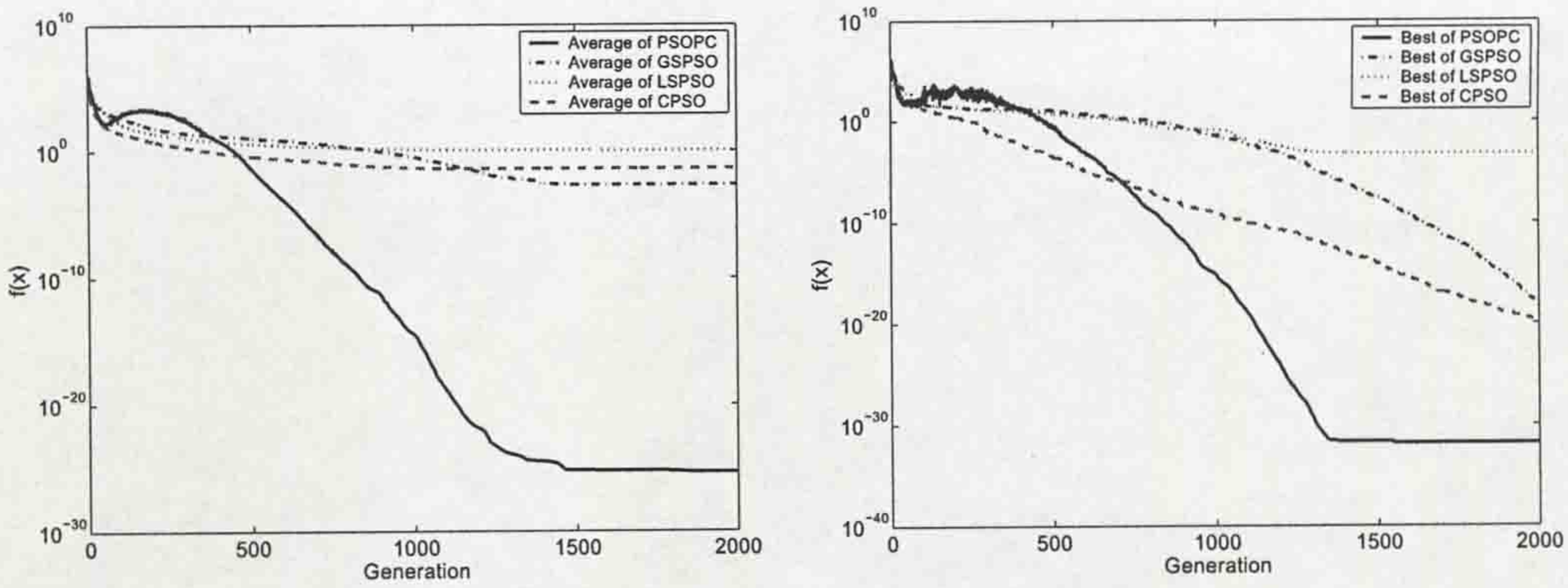
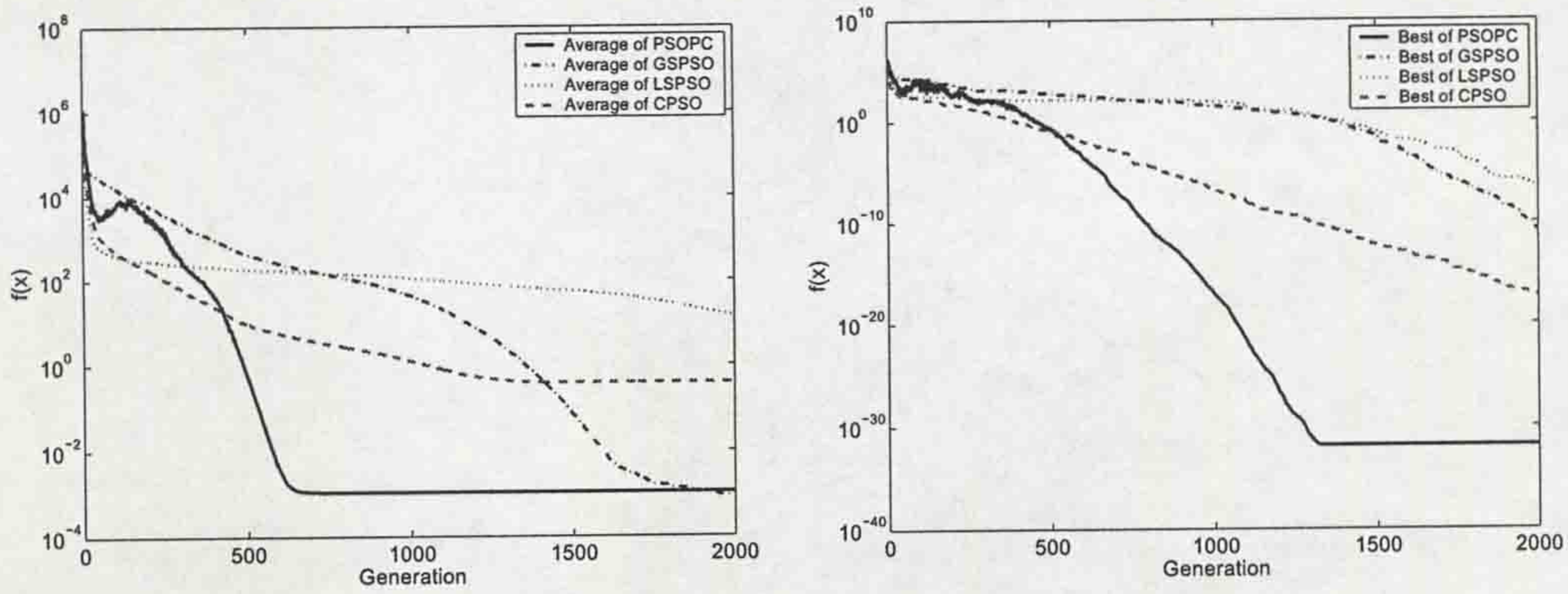


Figure 3.14: f_{10} (Penalized function P16)



Part 2

Applications of Animal Behaviour Inspired Optimisation Algorithms to Real-world Problems

Chapter 4

Neural Networks Training using Group Search Optimiser

In Chapter 1, a novel optimisation algorithm: GSO has been proposed. The superiority of GSO has been demonstrated by solving a large set of benchmark functions including a set of large-scale multi-modal functions. In this chapter, we apply GSO to Artificial Neural Network (ANN) training to further investigate its applicability to real-world problems. The ANN training process can be regarded as a hard optimisation problem because the search space is high-dimensional multi-modal and is usually polluted by noises and missing data. The most popular training algorithm is back-propagation (BP) algorithm. However, this gradient-based algorithm usually gets stuck in local minima and therefore the trained ANNs yield poor results. Here we proposed an ANN training algorithm based on the GSO algorithm. Parameters of a 3-layer feed-forward ANN, including connection weights and biases are tuned by our GSO algorithm. 4 real-world classification problems and 1 forecasting problem have been used as benchmarks to assess the performance of our GSO-based trained ANN (GSOANN). We also implemented other 5 training algorithms for comparison. GSOANN achieved better results than those generated by the other 5 training algorithms. Due to GSO's superior global search performance, GSOANN even has a better generalization performance than those of other

sophisticated ANNs, including some ANN ensembles on several benchmark problems.

4.1 Introduction

Artificial Neural Networks (ANNs) as a kind of computational intelligence technique have been widely applied to many problem domains such as pattern recognition [111] and control [112] since their renaissance in the mid-80's. Various ANN architectures and training algorithms have been proposed. Among them, the most popular ANN architecture and training algorithm are feed-forward ANNs and the BP training algorithm, respectively. However, the gradient based BP training algorithm is easy to be trapped by local minima and therefore deteriorates the performance of ANNs. Moreover, designing a near optimal ANN architecture to achieve good generalization performance is a hard optimisation problem.

In the past two decades, Evolutionary Algorithms (EAs) have been introduced to ANNs to perform various tasks, such as connection weight training, architecture design, learning rule adaption, input feature selection, connection weight initialization, rule extraction from ANN, etc.[113]. The combinations of ANNs and EAs are usually referred to as Evolutionary ANNs (EANNs). The earliest attempt to combine EAs and ANNs can be traced back to late 80s [114] [115]. Since then, the successful marriage of ANNs and EAs has attracted more and more attentions [116] [117]. We direct interested readers to an excellent review [113] of research on EANNs before 1999. In this section, we will briefly introduce some related works on EANNs in the last 5 years.

In [118], an improved genetic algorithm was used to tune the structure and parameters of a neural network. In order to tuning the structure of ANN in a simple way, link switches were incorporated into a three layer neural network. By introducing link switches, a given fully connected feed-forward neural network may become a partially connected network after leaning [118]. An improved Genetic Algorithm (GA) with new genetic operators was introduced to

train the proposed ANN. Two application examples, sunspots forecasting and associative memory tuning, were solved in their study.

Palmes *et al.* proposed a mutation-based genetic neural network (MGNN) [119]. A simple matrix encoding scheme was used to represent an ANN's architecture and weights. The neural network utilized the mutation strategy of local adaption of evolutionary programming to evolve network structure and connection weights dynamically. As classified in their paper, the MGNN falls into the category of "invasive" EANNs, where the ANN system uses EA for ANN's weights and structure evolution without the employment of BP or other gradient training [119]. Three classification problems, namely iris classification, wine recognition problem, and Wisconsin breast cancer diagnosis problem were used in their paper as benchmark functions.

In [120], an EAs, differential evolution (DE), was applied to train feed-forward ANNs' weights. A curve fitting problem and three classification problems (no details about these problems were given) were studied. However, the DE seems not to provide any distinct advantage in terms of learning rate or solution quality. For three out of the four problems tested in [120], the DE based ANN even yielded poorer results than those of ANN trained by a BP training algorithm.

Cantú-Paz and Kamath presented an empirical evaluation of eight combinations of EAs and ANNs on 11 well studied real-world benchmaks and 4 synthetic problems [121]. The algorithms they used included binary-encoded, real-encoded GAs, and the BP algorithm. The tasks performed by these algorithms and their combinations included searching for weights, designing architecture of ANNs, and selecting feature subsets for ANN training. Although the authors successfully applied EANNs to identify bent-double galaxies from FIRST (Faint Images of the Radio Sky at Twenty-cm) survey [122], the conclusion in [121] is somewhat surprising: in most cases, the combinations of EAs perform equally well on the problems and were not more accurate than hand-designed ANNs trained with the simple BP algorithm.

Combining ANN ensembles and EAs is becoming increasingly popular in

the past five years. There are lots of real-world problems that are too large and too complex for a single ANN, even a single EANN, to solve alone. ANN ensembles, which consists of several individual ANNs, were proposed to tackle these large-scale and complex real-world problems. Under the umbrella of the divide-and-conquer strategy, ANN ensembles subdivide a task and thereby solve it more efficiently and elegantly. Due to their superior generalization performance, ANN ensembles are enjoying and will continue to enjoy their successes. However, designing ANN ensemble is a tough task which heavily relies on human experts and prior knowledge about the problem [113]. In case of lacking human experts and prior knowledge, tedious trial-and-error processes are often required to design ANN ensembles in practice. Recently, EAs have been applied to address the issues of automatic designing of ANN ensembles [123] [124] [125] [126]. By employing population information, EAs trained ANN ensembles provide even better generalization performance on many problems [127].

Since we proposed GSO for continuous function optimisation problems, it is quite natural to apply the GSO algorithm to ANN weight training. The ANN weight training process can be regarded as a hard continuous optimisation problem because the search space is high-dimensional multi-modal and is usually polluted by noises and missing data. The objective of ANN weight training process is to minimize the error function. However, it has been pointed out that minimizing the error function is different from maximizing generalization [128]. The error on training set may be driven to a very small value by minimizing the error function, however, as a side effect, sometimes the overfitting problems will occur, that is, when test data are presented to the trained ANN, the error is still large. Therefore, to improve ANN's generalization performance, in this study, early stopping is introduced. The error rates of validation sets were monitored during the training processes. When the validation error increases for a specified number of iterations, the training will stop. Our experimental results on the five benchmark functions show that the GSO-based ANN (GSOANN) has a superior generalization performance to

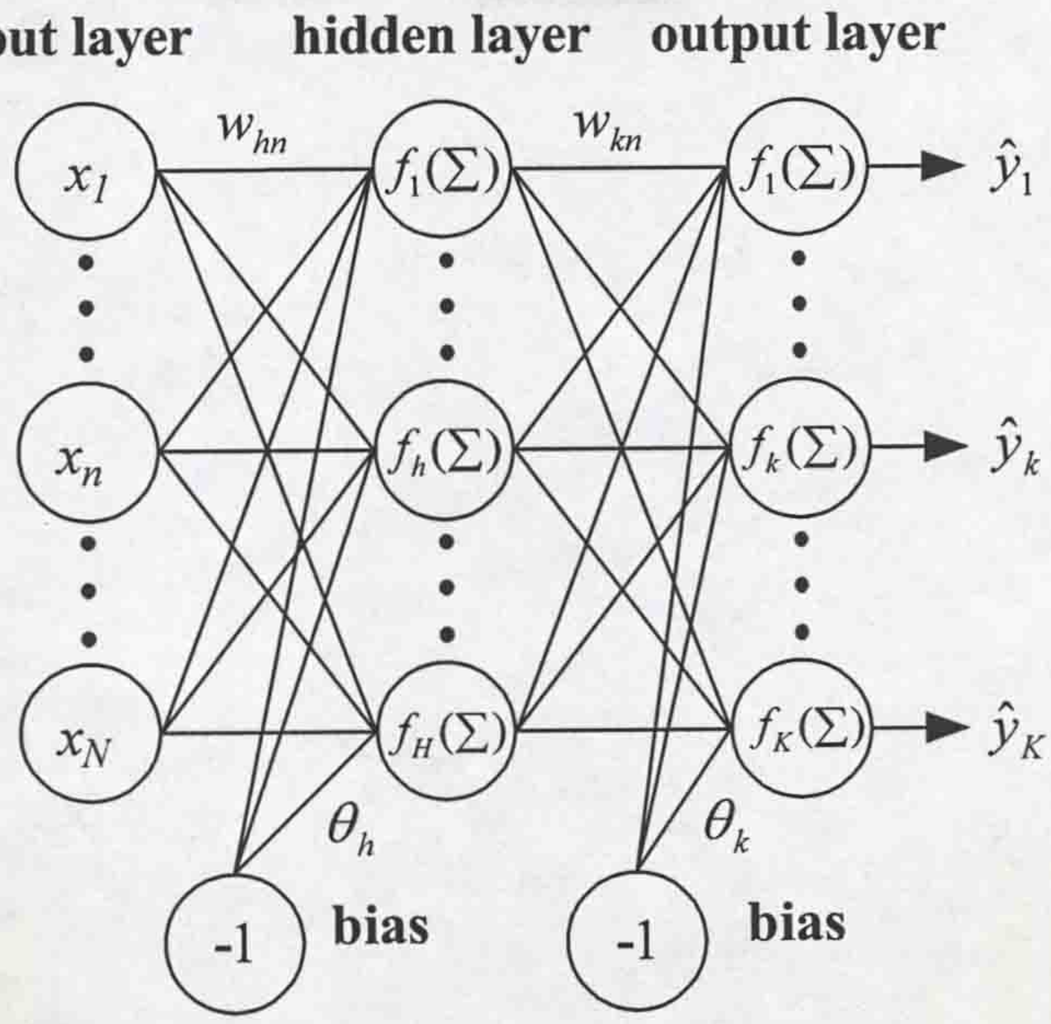


Figure 4.1: A three-layer feed-forward ANN.

those of many sophisticated ANNs and even some ANN ensembles.

The rest of the chapter is organized as follows. In Section 4.2, GSOANN will be introduced and the details of implementation will be given. In Section 4.3, we describe the benchmark functions, experimental settings and the experimental results. The chapter is concluded in Section 4.4.

4.2 GSO Based Training Algorithm for Neural Networks

Figure 4.1 presents the three-layer feed-forward ANN tuned by our GSO algorithm. The ANN consists three layers, namely, input, hidden, and output layers. The nodes in each layer receive input signals from the previous layer and pass the output to the subsequent layer. The nodes of the input layer supply respective elements of the activation pattern (input vector), which constitute the input signals from outside system applied to the nodes in the hidden layer by the weighted links. The output signals of the nodes in the output layer of

the network constitute the overall response of the network to the activation pattern supplied by the source nodes in the input layer. The subscripts n, h, and k denote any node in the input, hidden, and output layers, respectively. The net input u is defined as the weighted sum of the incoming signal minus a bias term. The net input of node h , u_h , in the hidden layer is expressed as follows:

$$u_h = \sum^{n} w_{hn} y_n - \theta_h$$

where y_n is the output of node n in the input layer, w_{hn} represents the connection weight from node n in the input layer to node h in the hidden layer, and θ_h is the bias of node h in the hidden layer. The activation function used in the proposed ANN is the sigmoid function. Therefore, in the hidden layer, the output y_h of node h, can be expressed as

$$y_h = f_h(u_h) = \frac{1}{1 + e^{u_h}}$$

The output of node k in the output layer can be also described as

$$y_k = f_k(u_k) = \frac{1}{1 + e^{u_k}} \tag{4.2.1}$$

where

$$u_k = \sum_{h} w_{kh} y_h - \theta_k$$

where θ_k is the bias of node k in the output layer.

The parameters (connection weights and bias terms) are tuned by the our GSO algorithm as shown in Figure 4.2. In the GSO-based training algorithm, each member of the population is a vector comprises connection weights and bias terms. Without loss of generality, we denote W_1 as the connection weight matrix between the input layer and the hidden layer, Θ_1 as the bias terms to the hidden layer, W_2 as the one between the hidden layer and the output layer, and Θ_2 as the bias terms to the output layer for the ANN structures established in the study. The i_{th} member in the population can be represented as: $X_i = [W_1\ \Theta_1\ W_2\ \Theta_2]$. The fitness function assigned to the i_{th} individual is the least-squared error function defined as follows:

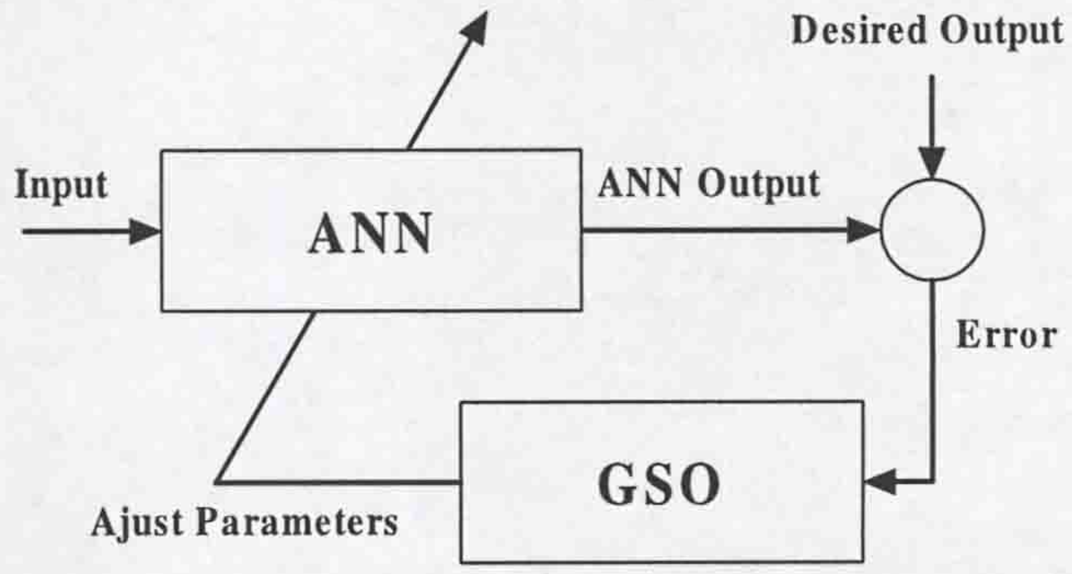


Figure 4.2: Schematic diagram of GSO based ANN.

$$F_i = \frac{1}{2}\sum_{p=1}^{P}\sum_{k=1}^{K}(d_{kp} - y_{kp})^2 \tag{4.2.2}$$

where y_{kp} indicates the k_{th} computed output in equation (4.2.1) of the ANN for the p_{th} sample vector; P denotes the total number of sample vectors; and d_{kp} is the desired output in the k_{th} output node.

4.3 Experimental Studies

In order to evaluate GSOANN's performance, several well-studied benchmark functions, including 4 classification problems, and one time series prediction problem were tested. The classification problems tested here are from the UCI machine learning repository: Wisconsin breast classification data; Pima Indian diabetes data; Cleveland heart disease data; and Australian credit card assessment data. They are all real-world problems which are investigated by human experts in practice. The data sets of these problems are usually contain missing attribute values and are usually polluted by noise. Therefore, they represent some of the most challenging problems in machine learning field [117]. The time series prediction problem studied here is sunspot number forecasting problem.

For comparison reason, we also implemented a modified back-propagation training algorithm: gradient descent with momentum and adaptive learning

rate; and four EAs based training algorithms, namely, Simple Genetic Algorithm (SGA) [78] based algorithm; Evolutionary Programming (EP) [79] [80] based algorithm; Evolution Strategies (ES) [81] based algorithm; Constriction Particle Swarm Optimiser (CPSO) [32] based algorithm. Although the GSOANN proposed here is relatively simple so it is not fair to compare the results of GSOANN to those of other sophisticated ANNs, it is our interest to compared the results we have obtained with the latest paper published in the literature.

4.3.1 Experimental setting

The parameter setting of the GSO algorithm is as same as the setting used in [129]. The initial population of GSO is generated uniformly at random in the search space. The initial head angle φ^0 of each individual is set to be $\frac{\pi}{4}$. The constants a is given by $round(\sqrt{n+1})$. The maximum pursuit angle $\theta_{\max}$ is $\frac{\pi}{a^2}$. The maximum turning angle α is set to be $\frac{\pi}{2a^2}$. The maximum pursuit distance $l_{\max}$ is calculated from:

$$l_{\max} = \parallel U_i - L_i \parallel = \sqrt{\sum_{i=1}^{n}(U_i - L_i)^2}$$

where L_i and U_i are the lower and upper bounds for the i_{th} dimension. The parameter need to be tuned is the percentage of rangers; our recommended percentage of rangers is 20%, which was used throughout all our experiments. The population size of the GSO algorithm was set to at 50.

The SGA algorithm we executed in our experiments is a real-coded simple genetic algorithm. The population of SGA was 50. The crossover and mutation rate was set to be 0.9 and 0.1, respectively. Stochastic universal sampling selection was used. The implementation of EP was based on the algorithm described in [80]. The population size and the tournament size for selection were 100 and 10, respectively. The initial standard deviation of the EP algorithm was 3.0. The ES algorithm used in our experiments is a state-of-the-art (μ, λ)-ES algorithm which was implemented according to [81]. The population

μ was set to at 200 and the offspring number λ was 30. A standard deviation of 3.0 was adopted. Global intermediate recombination was also employed in the ES algorithm. We also implemented a constriction factor approach PSO (CPSO), which is an improved PSO algorithm. The population of 50 was used in the CPSO algorithm. The constriction factor χ was 0.73 and the acceleration factors c_1 and c_2 were both 2.05 which followed the recommendations from [36].

For GSOANN and the other four EAs trained ANNs, the maximum epoch was set to be 300.

4.3.2 The classification problems

The Wisconsin Breast Cancer Data Set

The breast cancer data set was obtained by W. H. Wolberg *et al.* at the University of Wisconsin Hospitals, Madison, based on cell descriptions gathered by microscopic examination. The data set currently contains 9 integer-valued attributes and 699 instances of which 458 are benign and 241 are malignant examples. In order to train ANNs to classify a tumor as either benign or malignant, we partitioned this data set into three sets: a training set which contains the first 349 examples, a validation set which contains the following 175 examples, and a test set which contains the final 175 examples.

Results from GSO and the other 5 ANNs trained by EAs and BP algorithms are listed in Table 4.1. It can be seen that GSOANN produced the best average testing result. Although the other ANNs yielded reasonable best results, *e.g.*, 4 ANNs generated a testing error rate of 0%, the worst results found by these ANNs greatly deteriorated their overall performance, *e.g.*, the worst results found by PSOANN and BPANN are 11.43% and 28.57%, respectively. Figure 4.3 shows the evolution of the mean training error of the five ANNs over 30 runs.

The comparison between results produced by GSOANN and those of 10 other algorithms was tabulated in Table 4.2. Among these algorithms, MGNN

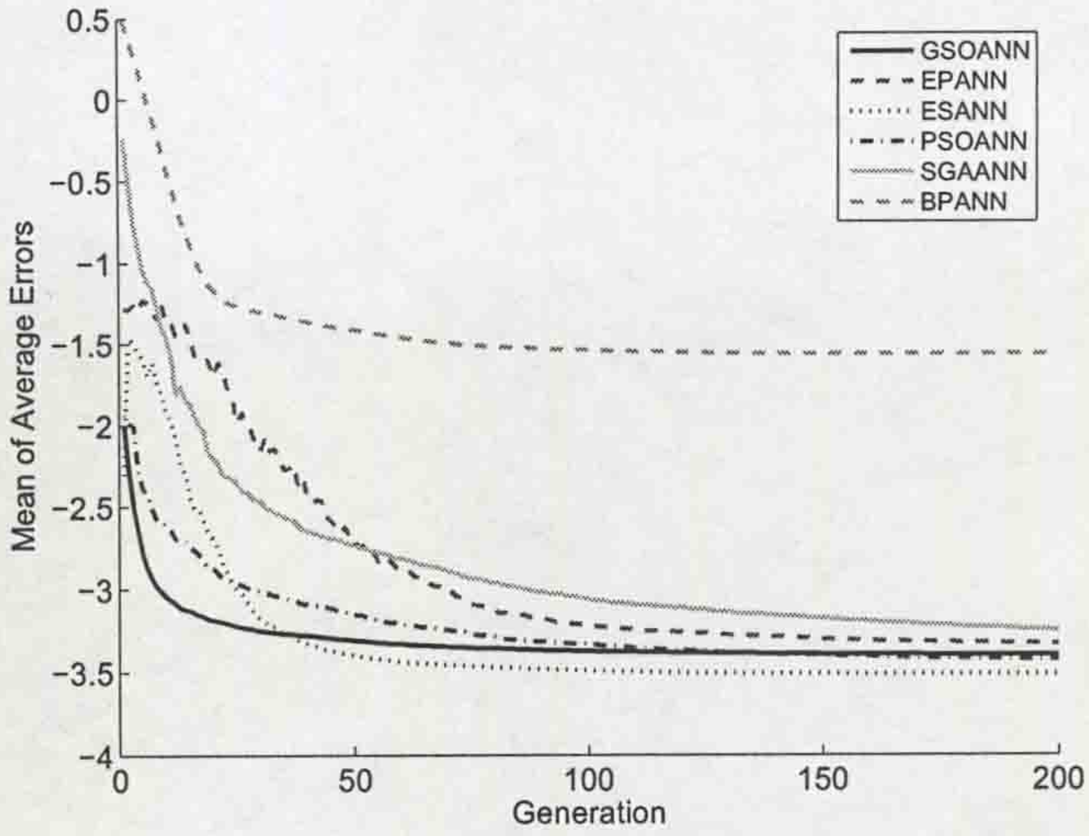


Figure 4.3: Evolution of ANNs' accuracy for the Wisconsin breast cancer data set.

[119] and EPNet [117] evolve ANN structure as well as connection weights; COOP [126] is an evolutionary ANN ensemble evolved by cooperative coevolution; CNNE [130] is a constructive algorithm for training cooperative ANN ensembles. CCSS [131], OC1-best [132] and EDTs [133] are state-of-the-art decision tress classifiers, including decision tree ensembles [131] [133] and hybrid evolutionary decision tree [132]; GANet-best is the best result from [121], which was generated by an EANN based on a real-encoded EA [134] to evolve connection weights; SVM-best is the best result of 8 least squares SVM classifiers [135]. It is worth to mention that the decision trees [131] [133] [121] and SVM [135] techniques used k-fold cross-validation which generated more optimistic results.

Compared with the sophisticated classifiers mentioned above, we can find that this simple GSOANN produced the best average result from Table 4.2.

Table 4.1: Error rate of GSOANN of the Wisconsin breast cancer data set.

Method	Training Set				Validation Set				Test Set			
	Mean	SD	Min	Max	Mean	SD	Min	Max	Mean	SD	Min	Max
GSOANN	3.35	0.09	3.26	3.56	2.17	0.21	1.93	2.89	**0.65**	0.25	0.00	1.14
SGAANN	3.88	0.63	3.04	5.63	3.86	1.14	2.59	7.82	1.50	0.72	0.00	2.85
EPANN	3.58	0.63	3.03	6.18	3.30	1.45	1.85	8.99	1.54	1.16	0.57	6.29
ESANN	2.98	0.11	2.73	3.16	2.70	0.39	2.14	3.52	0.95	0.66	0.00	2.86
PSOANN	3.26	0.24	2.92	3.80	2.37	0.43	1.37	3.35	1.24	2.02	0.00	11.43
BPANN	17.55	11.42	3.94	45.69	13.34	9.05	1.78	30.70	11.28	8.56	0.00	28.57

Table 4.2: Comparison between GSOANN and other approaches in terms of average testing error rate (%) on the Wisconsin breast cancer data set.

Algorithm	GSOANN	GANet-best[121]	COOP [126]	CNNE [130]	EPNet [117]
Test error rate (%)	**0.65**	1.06	1.23	1.20	1.38

Algorithm	MGNN [119]	SVM-best [135]	CCSS [131]	OC1-best [132]	EDTs [133]
Test error rate (%)	3.05	3.1	2.72	3.9	2.63

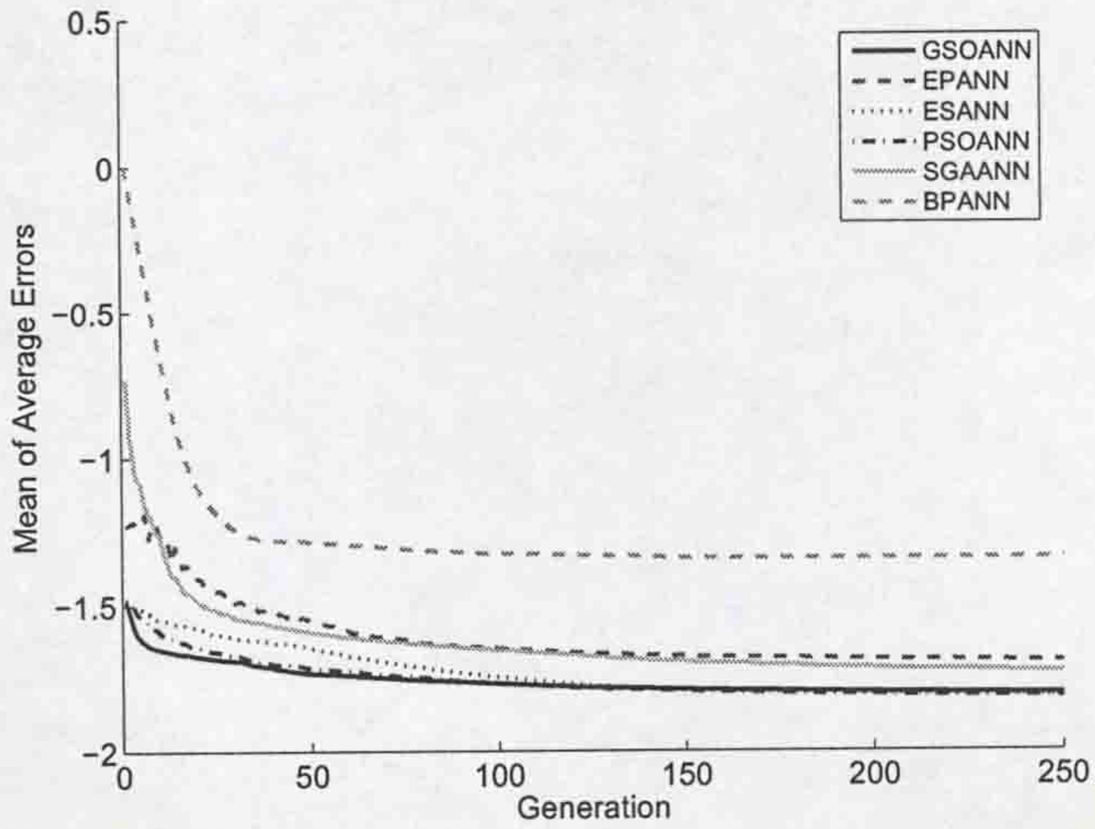


Figure 4.4: Evolution of ANNs' accuracy for the Pima Indian diabetes data set.

The Pima Indian Diabetes Data Set

The Pima Indian diabetes data were originally donated by Vincent Sigillito at the Johns Hopkins University. The data was gathered from a group of female patients of over 21 years old and of Pima Indian heritage living near Phoenix, Arizona, USA. The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to World Health Organization criteria. There are eight numeric-valued attributes and 768 instances. The data set contains 500 instances of patients with signs of diabetes and 268 instances of patients without. The data set was partitioned: the first 384 instances were used as the training set, the following 192 instances as the validation set, and the final 192 instances as the test set.

We tabulated the results generated by GSOANN and the other five ANNs in Table 4.3. Again, GSOANN yielded the best average result over 50 runs. The evolution of the mean training error of the five ANNs over 30 runs is presented in Figure 4.4.

This problem is one of the most difficult problems since the data set is relatively small and was heavily polluted by noise. Results from other state-of-

the-art classifiers are tabulated in Table 4.4. COVNET [125] is a cooperative coevolutionary model for evolving artificial neural networks. EENCL is evolutionary ensembles with negative correlation learning presented in [123]. 12-fold cross-validation was used by EENCL. GANet-best is the best result produced an ANN trained by a subset of features selected by binary-encoded GA [121].

Referring to Table 4.4, it can be seen that GSOANN is outperformed by COOP [126] and CNNE [130] which are both ANN ensembles. However, GSOANN produced better results than the rest classifiers including evolutionary ANN ensembles COVNET [125] and EENCL [123].

Table 4.3: Error rate of GSOANN of the Pima diabetes disease data set.

Method	Training Set				Validation Set				Test Set			
	Mean	SD	Min	Max	Mean	SD	Min	Max	Mean	SD	Min	Max
GSOANN	16.43	0.21	15.97	16.80	14.82	0.21	14.37	15.21	**19.79**	0.96	17.19	21.88
SGAANN	17.73	0.96	16.05	20.67	16.48	1.23	14.61	19.44	24.46	3.75	20.31	35.94
EPANN	18.38	1.56	16.28	21.34	17.18	1.87	14.75	20.55	25.75	4.89	18.23	36.46
ESANN	15.85	0.28	15.32	16.37	14.26	0.35	13.34	16.51	20.93	1.76	17.19	25.52
PSOANN	16.25	0.19	15.76	16.77	14.74	0.47	14.20	16.51	20.99	1.47	18.23	23.96
BPANN	25.48	4.24	19.59	42.02	22.00	3.30	18.03	34.76	35.98	4.56	28.13	52.08

Table 4.4: Comparison between GSOANN and other approaches in terms of average testing error rate (%) on the Pima diabetes disease data set.

Algorithm	GSOANN	GANet-best [121]	COOP [126]	CNNE [130]	COVNET [125]
Test error rate (%)	19.79	24.70	**19.69**	19.60	19.90
Algorithm	EENCL [123]	EPNet [117]	SVM-best [135]	CCSS [131]	OC1-best [132]
Test error rate (%)	22.1	22.38	22.7	24.02	26.0

The Cleveland Heart Disease Data Set

This data set comes from the Cleveland Clinic Foundation and was supplied by Robert Detrano of the V.A. Medical Center, Long Beach, CA. The goal of this data set is to predict the presence of absence of heart disease based on the data collected from various medical tests carried out on a patient. The database contains 13 attributes, which have been extracted from a larger set of 75. The original data set had five classes, considering four degrees of heart disease. The database originally contained 303 instances but six of them had missing values and 27 of the remaining were retained in case of dispute, leaving a final total of 270. The total 270 instances were partitioned into the training set of 134 instances, the validation set of 68 instances, and the testing set of the final 68 instances.

Table 4.5 compares GSOANN's result against those of other ANNs trained by EAs and BP algorithms. In terms of testing error rate, GSOANN generated the best average result. The best error rate was produced by ESANN which also had the best average training and validation error rates. However, ESANN yielded far worst testing result than that generated by GSOANN. The evolution of the mean training error of the five ANNs over 30 runs is presented in Figure 4.5.

In the machine learning literature, the Cleveland heart disease problem has been studied by researchers on data sets of either 303 or 270 instances. The outcomes from different data sets are quite different. Therefore, to compare fairly with other methods, we only listed studies which carried on the data set of 270 instances in Table 4.6 in comparison to our GSOANN. From this table, we can see that GSOANN generated worse result than those of COOP, CNNE and COVNET which are all ANN ensembles. However, the result is better than those of the rest 6 classifiers including EPNetEn which evolves ANN ensembles of EPNets[117].

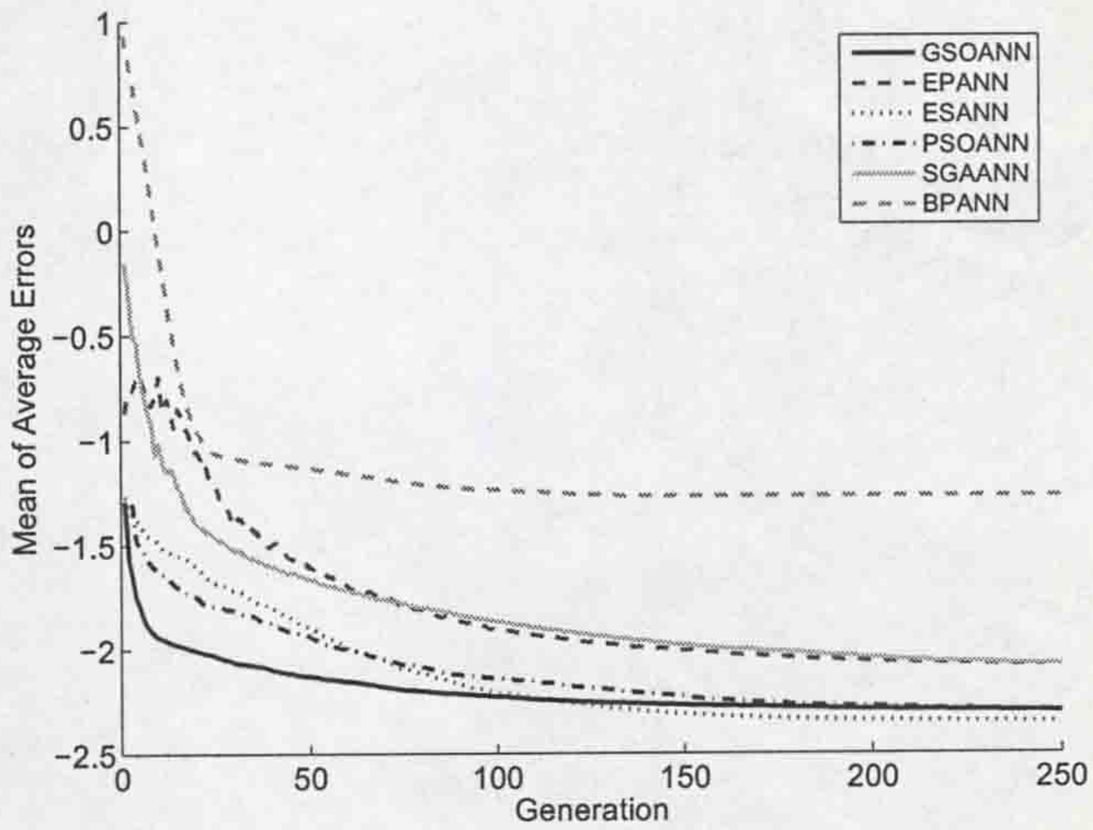


Figure 4.5: Evolution of ANNs' accuracy for the Cleveland heart disease data set.

Table 4.5: Error rate of GSOANN of the Cleveland heart disease data set.

Method	Training Set				Validation Set				Test Set			
	Mean	SD	Min	Max	Mean	SD	Min	Max	Mean	SD	Min	Max
GSOANN	10.03	0.44	9.26	11.67	12.66	0.70	11.51	14.52	**14.51**	1.48	11.76	17.65
SGAANN	12.53	2.35	8.89	18.21	17.88	4.21	12.04	34.07	20.88	5.04	14.71	33.82
EPANN	12.54	3.28	9.72	20.89	17.86	7.37	11.97	46.33	21.37	8.43	13.24	45.58
ESANN	8.70	0.67	7.44	10.27	15.49	3.55	11.28	28.52	16.86	3.08	10.29	23.53
PSOANN	10.03	1.23	8.46	14.73	13.65	1.18	11.55	16.40	16.08	3.30	11.76	25.00
BPANN	29.23	9.96	11.19	51.75	30.56	10.02	11.76	54.70	43.28	17.64	13.24	77.94

Table 4.6: Comparison between GSOANN and other approaches in terms of average testing error rate (%) on the Cleveland heart disease data set.

Algorithm	GSOANN	COOP [126]	CNNE [130]	COVNET [125]	EPNetEn [127]
Test error rate (%)	14.51	**11.96**	13.84	14.26	15.1

Algorithm	EPNet [117]	SVM-best [135]	CCSS [131]	CCMDT [136]	EDTs [133]
Test error rate (%)	16.77	15.1	16.04	16.17	20.45

The Australian Credit Card Assessment Data Set

The purpose of this data set is to assess applications to an Australian bank for a credit card based on a number of attributes. It is also from the UCI Machine Learning Repository. There are two classes, meaning whether the application was granted (44.5% of the instances) or denied (55.5% of the instances). Each record has 14 attributes of which the names and values have been changed to meaningless symbols to protect confidentiality of the data. This data set is very difficult to classify because it contains many missing values (there are of missing values in 37 cases of the records). Moreover, the attributes are mixed: there are 5 continuous, 4 binary and 5 nominal. The whole data was partitioned randomly into a training data set which contains 346 instances, a validation set which contains 172 instances and a testing data of 172 instances.

We list the training accuracy, validation accuracy and test accuracy of GSOANN along with other five algorithms in Table 4.7. It can be seen from the table that GSOANN achieved the best average training error rate, 9.48%. Not surprisingly, GSOANN produced a far better test error rate than those from the other five ANNs. To illustrate the training process of ANNs, we present the evolution of the mean training error of the six ANNs over 30 runs in Figure 4.6.

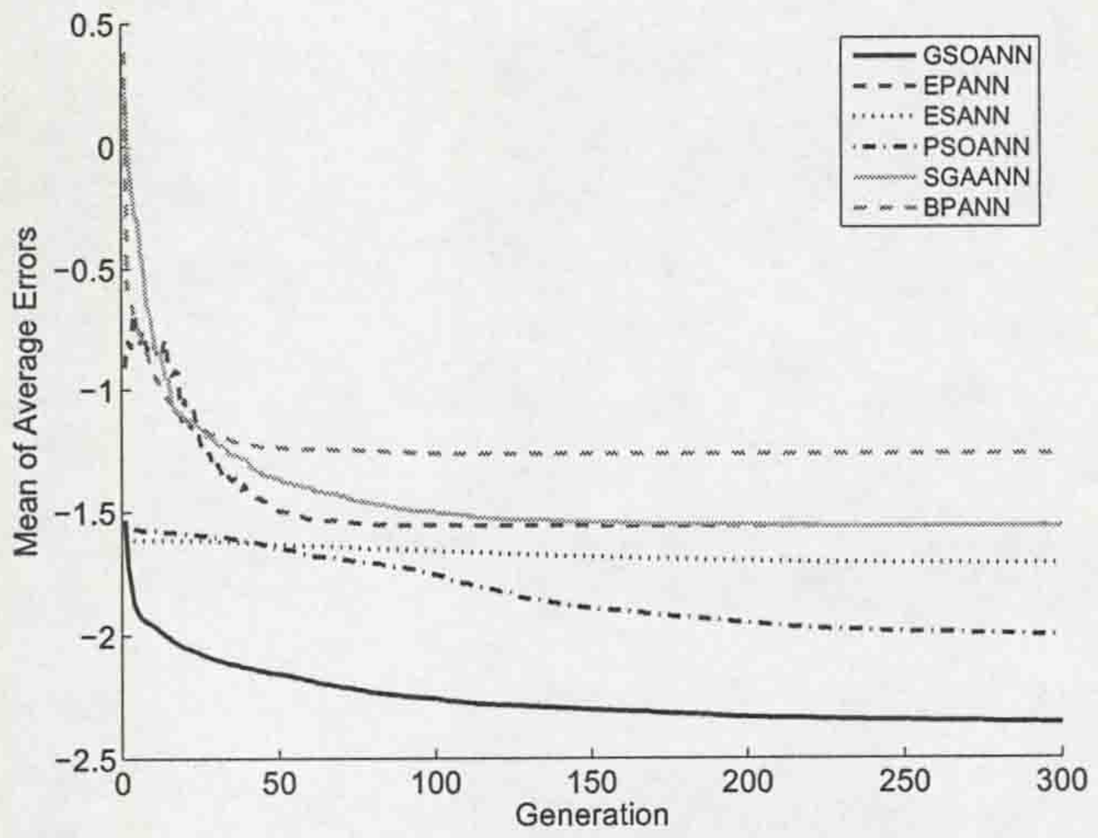


Figure 4.6: Evolution of ANNs' accuracy for the Australian credit card assessment data set.

Table 4.7: Error rate of GSOANN of the Australian credit card assessment data set.

Method	Training Set				Validation Set				Test Set			
	Mean	SD	Min	Max	Mean	SD	Min	Max	Mean	SD	Min	Max
GSOANN	9.48	0.67	8.16	11.25	11.01	2.22	6.06	14.88	**13.79**	2.35	8.72	18.60
SGAANN	21.04	1.99	16.69	26.84	24.78	10.42	17.75	76.01	33.28	5.12	23.26	45.93
EPANN	21.06	1.12	19.19	24.72	22.48	2.21	16.97	29.07	32.71	3.74	26.74	45.93
ESANN	18.06	1.49	13.69	20.30	19.95	2.99	11.14	25.34	27.51	5.70	16.27	38.95
PSOANN	13.51	4.11	7.41	20.38	15.85	5.76	7.07	25.23	21.63	8.62	10.47	47.67
BPANN	23.56	4.96	15.64	42.48	23.75	6.41	12.36	50.57	41.40	12.31	20.34	74.41

As listed in Table 4.8, we also compare the result from GSOANN to those adopted from several papers which include 2 studies on evolutionary ANNs (GANet-best [121] and EPNet [117]), 5 papers of ANN ensembles (COOP [126], CNNE [130], COVNET [125], EENCL [123], EPNetEn [127]) and the studies on SVMs (SVM-best [135]) and decision trees ensembles (CCSS [131]). The best result adopted from [121] was generated from an ANN whose architecture was designed by GAs. Although our GSOANN has been proved to be an overpowering winner when compared to the other five ANNs we implemented, the average testing error rate generated by GSOANN is not as good as those of other sophisticated approaches. GSOANN generated a better result than those of GANet-best and CCSS but is outperformed by EPNet, SVMs and all the ANN ensembles.

Table 4.8: Comparison between GSOANN and other approaches in terms of average testing error rate (%) on the Australian credit card assessment data set.

Algorithm	GSOANN	GANet-best [121]	COOP [126]	CNNE [130]	COVNET [125]
Test error rate (%)	13.79	13.91	11.96	**9.10**	11.57

Algorithm	EENCL [123]	EPNetEn [127]	EPNet [117]	SVM-best [135]	CCSS [131]
Test error rate (%)	13.5	9.3	11.5	13.00	14.32

4.3.3 The forecasting problems

Forecasting of the Sunspot Number

Sunspot series is a record of the activity of the surface of the sun. It is known that sunspot activity is a precursor to periods of active solar flares. A sufficiently large solar flare ejects coronal material from the core of the sun, and this material disrupts the operation of satellites. Therefore, predicting the sunspot is becoming more and more important especially in our modern world where people heavily rely on satellite communication. However, the sunspot series is nonlinear, non-stationary and non-Gaussian and is a well-known challenging task for time series analysis.

The data set used in our experiment was included in MATLAB environment which recorded the sunspot activity over the last 300 years. The sunspot cycles from 1700 to 1987 are shown in Fig. 4.7. It can be seen from this figure that the sunspot activity is cyclical, reaching a maximum about every 11 years. The first 180 year (1700 – 1987) were used as the training set to train the proposed GSOANN. Following [118], the inputs x_i of the GSOANN consists of three past data points: $x_1 = y_1^d(t-1)$, $x_2 = y_1^d(t-2)$, and $x_3 = y_1^d(t-3)$, where t denotes the year and $y_i^d(t)$ denotes the sunspot number at the year t. The output is the prediction of the sunspot number at year t: $\hat{y}_1(t)$. The performance (forecasting error rate) of the trained GSOANN can be calculated from:

$$\text{err} = \sum_{t=1885}^{1980} \left(\left| \frac{|y_1^d(t) - \hat{y}_1(t)}{96} \right| \right)$$

We tabulated the results of training errors and forecasting errors in Table 4.9. From the table, it can be seen that although GSOANN yielded slightly worse mean training error than that of PSOANN, it generated the best mean forecasting error. We can also find that the best (minimum) forecasting result found in the 30 runs by GSOANN is similar to or slightly worse than those of the other five ANNs. However, the worst (maximum) forecasting error generated by GSOANN is the smallest among the worst forecasting errors. It can be concluded that although GSOANN could not find the best forecasting error,

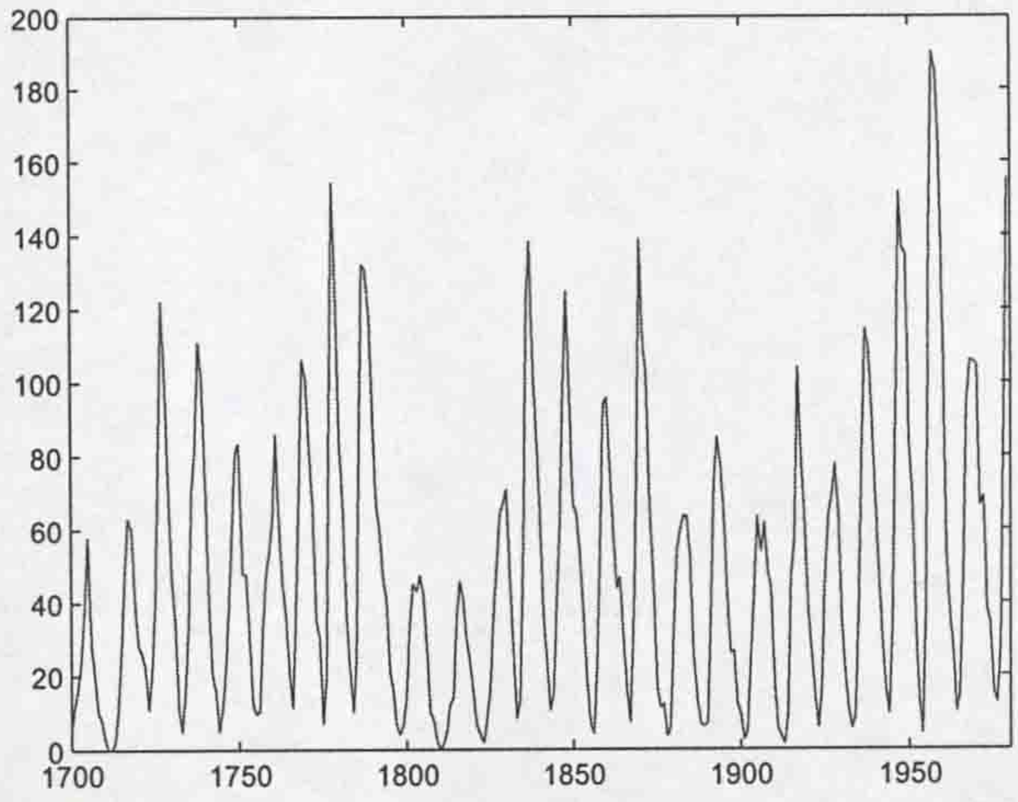


Figure 4.7: Sunspot cycles from 1700 to 1987.

the superior global search performance of GSO guaranteed the search was not trapped by poor local minima as other algorithms did, therefore yield more robust forecasting results.

This problem has been used as a benchmark by Leung *et al.* [118] to evaluate the performance of their ANN based on an improved GA. The best result obtained in their study is an ANN with six hidden nodes. The training error and the forecasting error are 11.5730 and 14.0933, respectively. It can be seen that although the training error obtained by their ANN is better than

Table 4.9: Accuracies of GSOANN of the sunspot forecasting problem.

Method	Training Error				Forecasting Error			
	Mean	SD	Min	Max	Mean	SD	Min	Max
GSOANN	12.34	0.28	11.92	13.02	**13.37**	0.53	12.40	14.62
SGAANN	13.41	0.91	11.89	15.17	16.68	7.46	11.83	43.76
EPANN	13.22	0.49	12.53	14.52	14.77	1.73	12.05	22.60
ESANN	12.52	0.55	11.64	13.55	14.22	1.05	12.64	16.34
PSOANN	11.88	0.23	11.53	12.44	13.55	0.75	12.22	15.59
BPANN	13.36	1.17	11.79	16.22	14.40	0.90	12.79	16.17

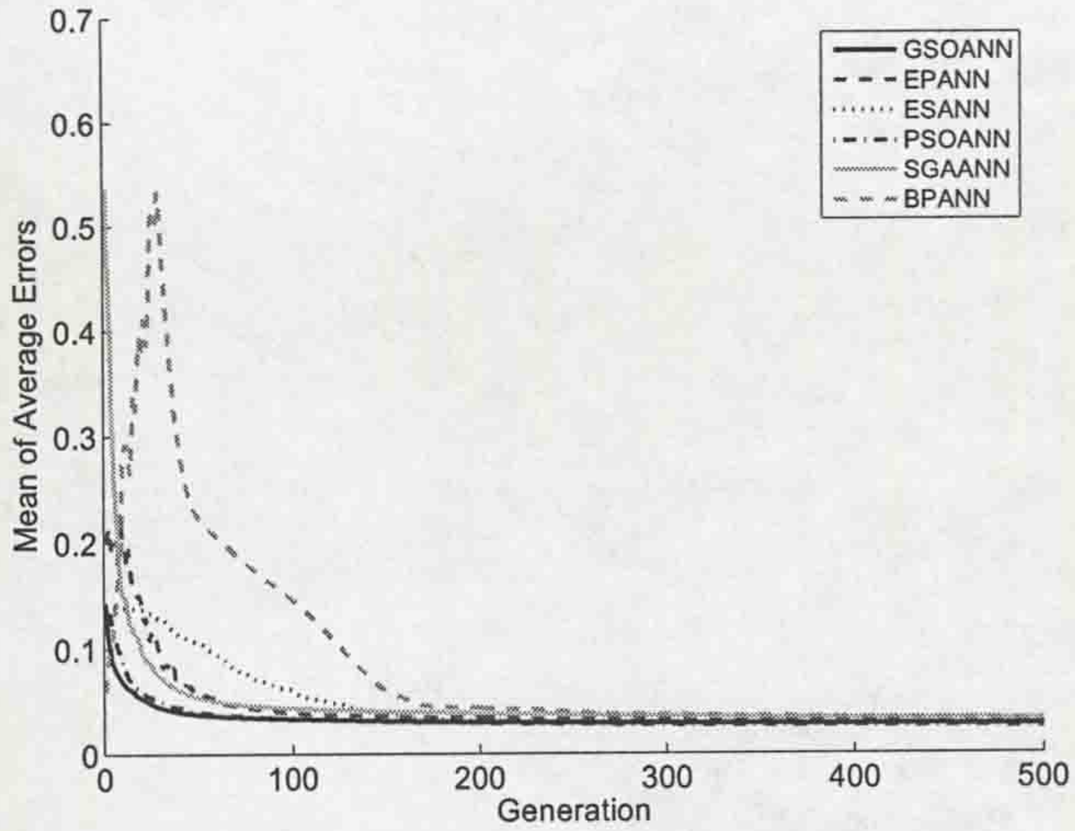


Figure 4.8: Evolution of ANNs' accuracy for the forecasting of the sunspot number.

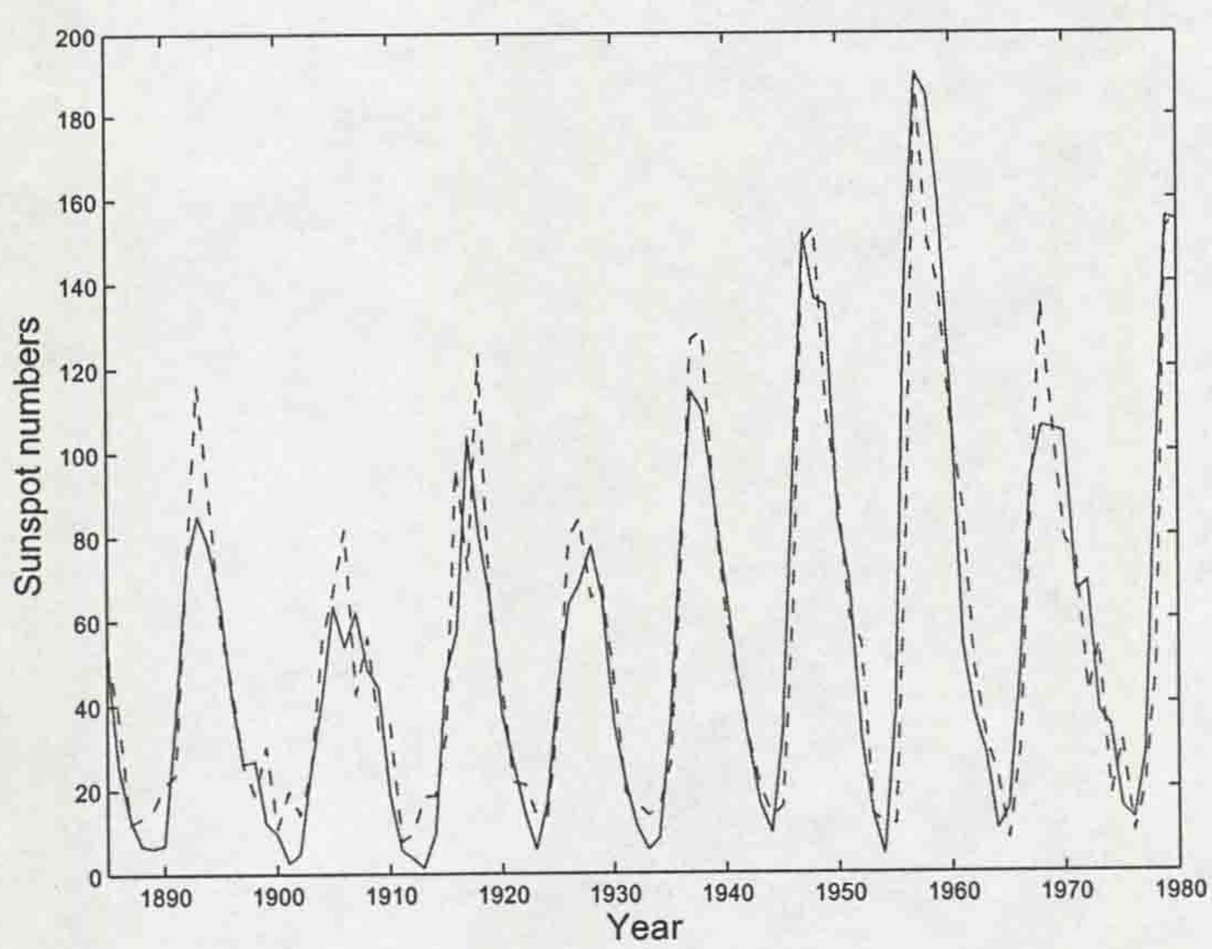


Figure 4.9: Simulation results of a 96-year prediction using GSOANN (dashed line) and the actual numbers (sold line).

Shan He

those of all ANNs tested here, including GSOANN, the forecasting error is worse than GSOANN and PSOANN.

4.4 Conclusions

In this chapter, the GSO algorithm has been applied to train ANN's connection weights. Our initial goal was not to propose a sophisticated ANNs which can achieve the best generalization performance. Instead, we aimed to access the global search performance of GSO by applying it to train ANN's weights since the training process can be regarded as a hard continuous optimisation problem. It has been pointed out that minimizing the error function is different from maximizing generalization performance [128]. However, optimisation algorithms with better global search performance are more capable of steering away from poor local minima of the error function therefore improving generalization performance.

In order to further investigate the generalization property of GSOANN, we also implemented five other EA-based and gradient-based training algorithms. Compared to the other implemented ANNs, GSOANN yields the best average test and forecasting error rates on all five benchmark problems. It can also be seen that although the average training error rates generated by GSOANN are outperformed by those of ESANN and PSOANN, the validation error rates on 3 out of 4 classification problems are better than those produced by all the other ANNs. We also noticed that the GSOANN yields smaller standard deviations than those of other ANNs which means GSOANN is more robust than the other ANN. Although the GSOANN proposed here is relatively simple so it is not fair to compare the results of GSOANN to those of other sophisticated ANNs, surprisingly, compared to those generated by some sophisticated ANN ensembles, this simple GSO based ANN training algorithm provides relatively good results on the five benchmark problems.

Chapter 5

Application of PSO to Mechanical Design Optimisation Problems

This chapter presents an improved particle swarm optimiser (PSO) for solving mechanical design optimisation problems involving problem-specific constraints and mixed variables such as integer, discrete and continuous variables. A constraint handling method called the "fly-back-mechanism" is introduced to maintain a feasible population. The standard PSO algorithm is also extended to handle mixed variables using a simple scheme. Five benchmark problems commonly used in the literature of engineering optimisation and nonlinear programming are successfully solved by the proposed algorithm. The proposed algorithm is easy to implement, and the results and the convergence performance of the proposed algorithm are better than other techniques such as GAs.

5.1 Introduction

In the past few decades, many optimisation algorithms have been applied to solve mechanical design optimisation problems. Among them, evolutionary algorithms (EAs) such as Genetic Algorithms (GAs), Evolutionary Program-

ming (EP) and Evolution Strategies (ES) become attractive because they do not apply mathematical assumptions to the optimisation problems and have better global search abilities over conventional optimisation algorithms[137]. Many successful applications of evolutionary algorithms have been reported to solve engineering problems such as power system dispatch [138] [139] and mechanical optimal design problems[140] [141]. Recently a new evolutionary algorithm called Particle Swarm Optimiser (PSO) has been proposed [32]. PSO is a population based optimisation algorithm which was inspired by the social behaviour of animals such as fish schooling and bird flocking. Similar to other evolutionary algorithms, it can solve a variety of hard optimisation problems but with a faster convergence rate [21]. Another advantage is that it requires only few parameters to be tuned making it attractive from an implementation view point.

Most mechanical optimal design problems are hard to solve for both conventional optimisation algorithms and EAs, because they involve problem-specific constraints. To handle these constraints, many different approaches have been proposed. The most common approach in the EAs community is to make use of penalty functions. However, the major drawback of using penalty functions is that they require additional tuning parameters. In particular the penalty coefficients have to be fine tuned in order to balance the objective and penalty functions. Inappropriate penalty coefficients will make the optimisation problem intractable [142] [143]. Other approaches to handle constraints, according to [144], include rejection of infeasible individuals, maintaining feasible population, repair of infeasible individuals, separation of individuals and constraints, replacement of individuals by their repaired versions and use of decoders. The standard PSO is usually applied to solve unconstrained optimisation problems. In this chapter, the standard PSO algorithm is extended to solve constrained mechanical design optimisation problems using preserving feasible population methods.

Mechanical optimal design problems may contain integer, discrete and continuous variables, which are referred to as mixed-variable nonlinear optimisa-

tion problems. To solve them, Sandgren [145] and Hajela [146] have proposed nonlinear branch and bound algorithms based on integer programming. Cao and Wu developed mixed variable evolutionary programming (MVEP) [140] with different mutation operators associated with different types of variables. Deb and Goyal [141] presented a combined genetic search technique (GeneAS) which combined binary and real-coded GAs to handle mixed variables. Originally PSO was proposed to handle continuous optimisation problems. Recently, PSO has been applied to Integer Programming by Parsopoulos [147] by simply truncating the real values to integers which does not effect significantly the search performance. In this chapter, the standard PSO is extended to handle mixed-variable nonlinear optimisation problems more effectively.

This chapter is organized as follows: Section 5.2 introduces the formulation of mechanical design optimisation problems. A modified version of the PSO algorithm to handle constraints with mixed variables is proposed in Section 5.3. The proposed PSO has been tested on five examples which are commonly used in the mechanical design optimisation literature. Experimental results and discussions are given in Section 5.4. The chapter is concluded in Section 6.5.

5.2 Formulation of Mechanical Design Optimisation Problems

Mechanical design optimisation problems can be formulated as a nonlinear programming (NLP) problem. Unlike generic NLP problems which only contain continuous or integer variables, mechanical design optimisations usually involve continuous, binary, discrete and integer variables. The binary variables are usually involved in the formulation of the design problem to select alternative options. The discrete variables are used to represent standardization constraints such as the diameters of standard sized bolts. Integer variables usually occur when the numbers of objects are design variables, such as the number of gear teeth. Considering the mixed variables, the formulation can be

expressed as follows:

$$\min f(X) \tag{5.2.1}$$

subject to:

$$h_i(X) = 0 \quad i = 1, 2, \cdots, m$$
$$g_i(X) \geq 0 \quad i = m+1, \cdots, p$$

where $f(X)$ is the scalar objective function, and $h_i(X)$ and $g_i(X)$ are the equality and inequality constraints, respectively.

The variables vector $X \in \mathbb{R}^N$ represents a set of design variables which can be written as:

$$X = \begin{pmatrix} X^C \\ X^B \\ X^I \\ X^D \end{pmatrix} = [x_1^C, \cdots, x_{n_C}^C, x_1^B, \cdots, x_{n_B}^B, x_1^I, \cdots, x_{n_I}^I, x_1^D, \cdots, x_{n_D}^D]^T$$

where

$$\begin{aligned} x_i^{Cl} \leq x_i^C \leq x_i^{Cu}, &\quad i = 1, 2, \cdots, n_C \\ x_i^B \in \{x_i^{Bl}, x_i^{Bu,}\}, &\quad i = 1, 2, \cdots, n_B \\ x_i^{Il} \leq x_i^I \leq x_i^{Iu}, &\quad i = 1, 2, \cdots, n_I \\ x_i^{Dl} \leq x_i^D \leq x_i^{Du}, &\quad i = 1, 2, \cdots, n_D \end{aligned} \tag{5.2.2}$$

where $X^C \in \mathbb{R}^{n_C}, X^B \in \mathbb{R}^{n_B}, X^I \in \mathbb{R}^{n_I}$ and $X^D \in \mathbb{R}^{n_D}$ denote feasible subsets of comprising continuous, binary, integer and discrete variables, respectively. $x_i^{Cl}, x_i^{Bl}, x_i^{Il}$ and x_i^{Dl} are the lower bounds of the i_{th} variables of X^C, X^B, X^I and X^D, respectively. $x_i^{Cu}, x_i^{Zu}, x_i^{Iu}$ and x_i^{Du} are the upper bounds of the i_{th} variables of X^C, X^B, X^I and X^D, respectively. n_C, n_B, n_I and n_D are the numbers of continuous, binary, integer and discrete variables, respectively. The total number of variables is $N = n_C + n_B + n_I + n_D$.

5.3 Improved Particle Swarm Optimiser

As mentioned in the introduction, the difficulties in using EAs to solve mechanical optimisation problems come from problem-specific constraints and mixed variables. Little work has been done for solving constrained mixed-variable optimisation problems with PSO. In this section, the use of the PSO techniques to handle mixed variables and constraints are proposed.

5.3.1 Mixed-variable handling methods

Originally, most of the EAs were proposed to handle continuous variables. In the last decade, GAs [148], ESs [149], EPs [140] have been extended to handle mixed variables.

In its basic form, PSO can only handle continuous variables. To handle integer variables, simply truncating the real values to integers to calculate fitness value will not affect the search performance significantly [147]. The truncation is only performed in evaluating the fitness function. That is, the swarm will "fly" in a continuous search space regardless of the variable type. For binary variables, since they can be regarded as integer variables within the range of $[0, 1]$, we do not consider them separately.

For the ith particle X_i contains n_C continues variables and n_D discrete variables, and the j_{th} discrete variable which consists of m_j discrete values is expressed as:

$$X_{i,j}^D = [x_{i,j,l}^d, \cdots, x_{i,j,l}^d, \cdots, x_{i,j,m_j}^d] \tag{5.3.1}$$

For the j_{th} discrete variable, a fictitious real variable x is used instead of the discrete variable $x_{i,j}^D$, where $x \in [1, m_j + 1]$ and it is updated directly in the same way as does it for continues variables in the GSO algorithm. Then, the index l is determined by setting $l = INT(x)$, where $INT(x)$ denotes the greatest integer less than the real value x, to select a discrete value $x_{i,j,l}^d$ of the j_{th} discrete variable $X_{i,j}^D$, before involving it in the function evaluation.

Hence, the fitness function of the ith member X_i can be expressed as follows:

$$f(X_i) \qquad i = 1, \cdots, M \tag{5.3.2}$$

where

$$\begin{aligned} X_i = & \left\{ x^c_{i,j}, x^d_{i,j,l} INT(x_{i,j}) \middle| x^c_{i,j} \in X^C_i, j = 1, \cdots, n_C, \right. \\ & x^d_{i,j,l} \in X^D_{i,j}, l \in [1, m_j],\ j = 1, \cdots, n_D, \\ & \left. x^I_{i,j} \in X^I_{i,j},\ j = 1, \cdots, n_I \right\} \end{aligned} \tag{5.3.3}$$

where $X^C_i \in \mathbb{R}^{n_C}$, $X^D_i \in \mathbb{R}^{\sum_{j=1}^{n_D} m_j}$ and $X^I_i \in \mathbb{R}^{n_I}$ and X^C_i, X^D_i and X^I_i denote the feasible subsets of comprising continuous, discrete and integer variables of member X_i, respectively.

5.3.2 Constraint handling methods

Evolutionary Algorithms (EAs) are heuristic optimisation techniques which have been successfully applied to various optimisation problems. However they are not able to handle constrained optimisation problems directly [150]. In the past few years, much work has been done to improve EAs performance to deal with constrained optimisation problems. Penalty functions are commonly used to incorporate constraints into the fitness function. Other techniques developed to handle the constraints, reported in [137] and [144], include rejection of infeasible individuals, maintaining feasible population, repair of infeasible individuals, and multi-objective optimisation techniques.

The PSO algorithms have been applied to constrained optimisation problems. EI-Gallad et al. [151] proposed a constraint handling technique based on maintaining a feasible population. However our experimental results indicate that such a technique will lower the efficiency of the standard PSO. Their technique resets the infeasible particles to their previous best positions *pbest* which will sometimes prevent the search reaching a global minimum. Hu [152] also proposed a constraint handling technique based on preserving feasible population. The algorithm starts from a feasible initial population. During the search process, only feasible particles are counted when calculating the value of the previous best position *pbest* and global best position *gbest*. Parsopoulos et al. [153] incorporated a non-stationary multi-stage assignment penalty function into PSO. In their paper, a set of 6 benchmark functions were tested. However

some of their solutions are not feasible. Other attempts include applying a multi-objective optimisation technique to handle constraints [154].

In this study, the technique of maintaining a feasible population is investigated. The technique starts from a feasible initial population. A closed set of operators is used to maintain the feasibility of the solutions. Therefore, the subsequent solutions generated at each iteration are also feasible. Algorithms based on this technique are much more reliable than those based on a penalty approach [144]. For mechanical design problems, reliability is crucial since most of the constraints need to be satisfied. The concept of maintaining a feasible population is suitable for incorporation into the standard PSO algorithm for solving mechanical design problems.

For the PSO algorithm, the intuitive idea to maintain a feasible population is for a particle to fly back to its previous position when it is outside the feasible region. This is the so called "fly back mechanism". Since the population is initialized in the feasible region, flying back to previous position will guarantee the solution to be feasible. From our experience, the global minima of mechanical optimal design problems are usually close to the boundaries of the feasible space, as shown in Fig. 5.1. Flying back to its previous position when a particle violates the constraints will allow a new search closer to the boundaries. Fig. 5.2 and Fig. 5.3 illustrate the search process of the "fly back mechanism". In Fig. 5.2, the i^{th} particle would fly into the infeasible search space at the k^{th} iteration. At the next iteration as shown in Fig. 5.3, this particle is set back to its previous position X_i^{k-1} and starts a new search. Assuming that the global best particle P_g stays in the same position, the direction of the new velocity V_i^{k+1} will still point to the boundary but closer to P_g. Since P_g is inside the feasible space and ωV_i^k is smaller than V_i^k, the chance of particle X_i flying outside the boundaries at the next iteration will be decreased. This property makes the particles more likely to explore the feasible search space near the boundaries. Therefore, such a "fly back mechanism" is suitable for mechanical design problems. Moreover our experimental results show that this technique can find better minima with less iterations compared with other techniques.

Shan He

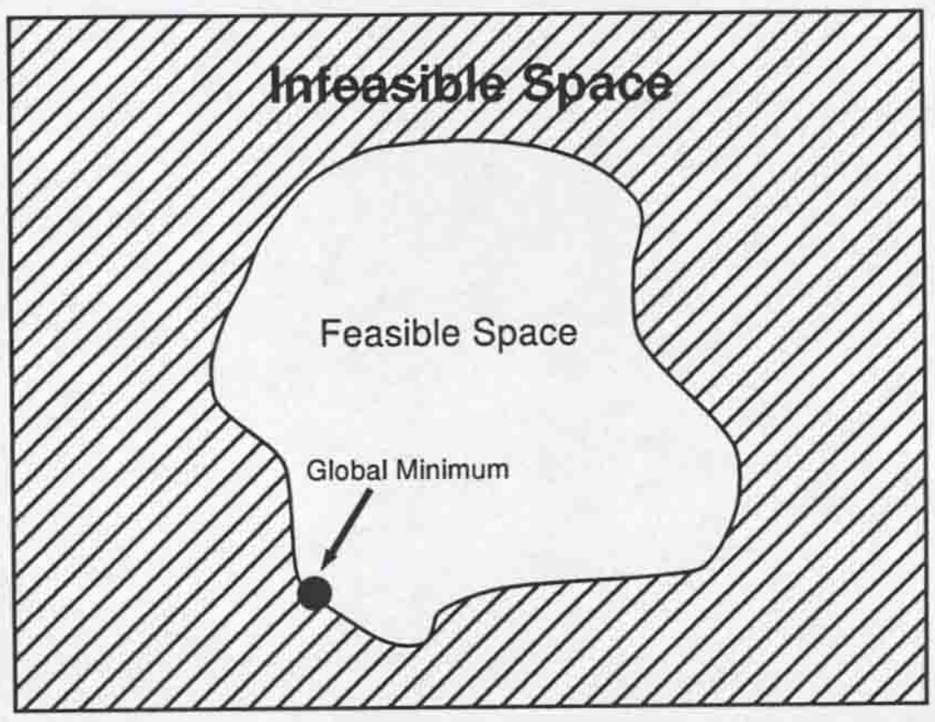


Figure 5.1: Global minimum in the feasible space.

5.3.3 Improved particle swarm optimiser algorithm

Regarding the proposed constraint handling technique described in section 5.3.2, the improved PSO requires a feasible initial population to guarantee that the solutions of successive generations are feasible. To do so, an extra loop at the beginning of the algorithm is required to keep randomly re-initializing infeasible particles to ensure that they stay inside the feasible search space. Our experience indicates that this simple method is sufficiently good enough for most mechanical design problems since their feasible search spaces are usually large and feasible particles can be easily generated. Small size populations are preferred to minimize the time to find a feasible initial population.

The improved PSO algorithm is given in Table 5.1.

5.4 Numerical Examples

In this section, five numerical examples have been used to test our new PSO algorithm. The first example is a classical benchmark problem in nonlinear constrained optimisation. Four other examples are taken from the mechanical design optimisation literature. All these problems have linear and nonlinear constraints and have been investigated by various EAs or traditional

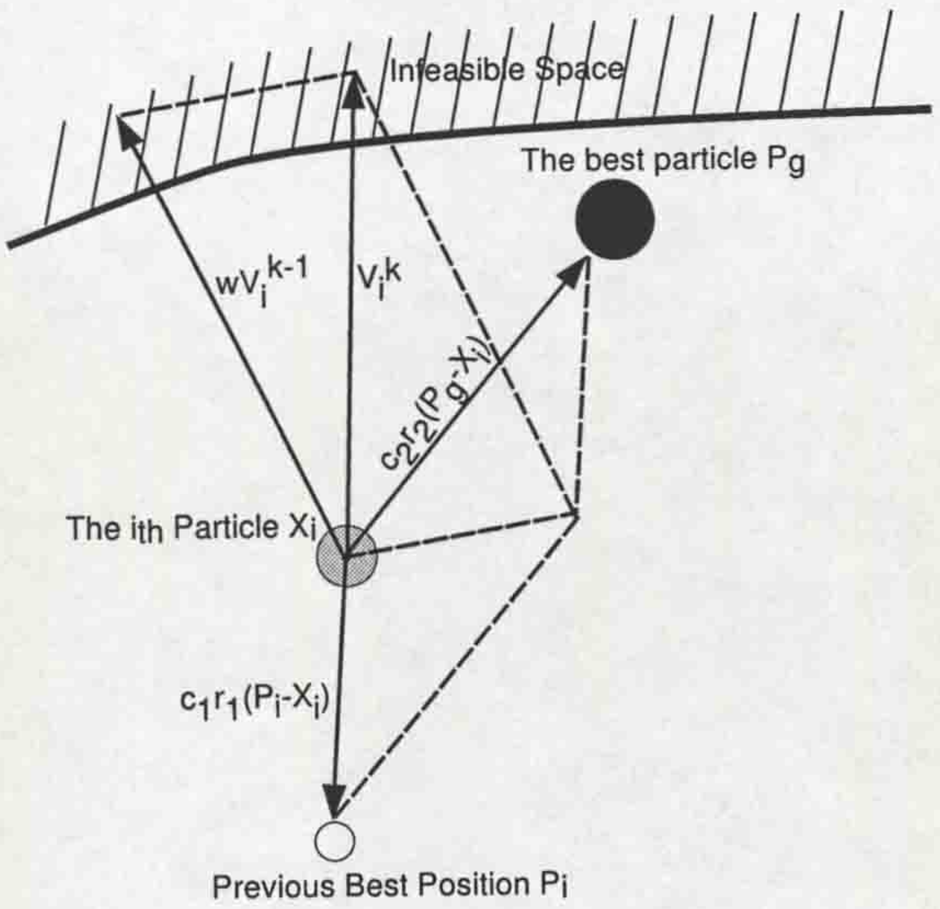


Figure 5.2: X_i at iteration k would fly outside the feasible search space.

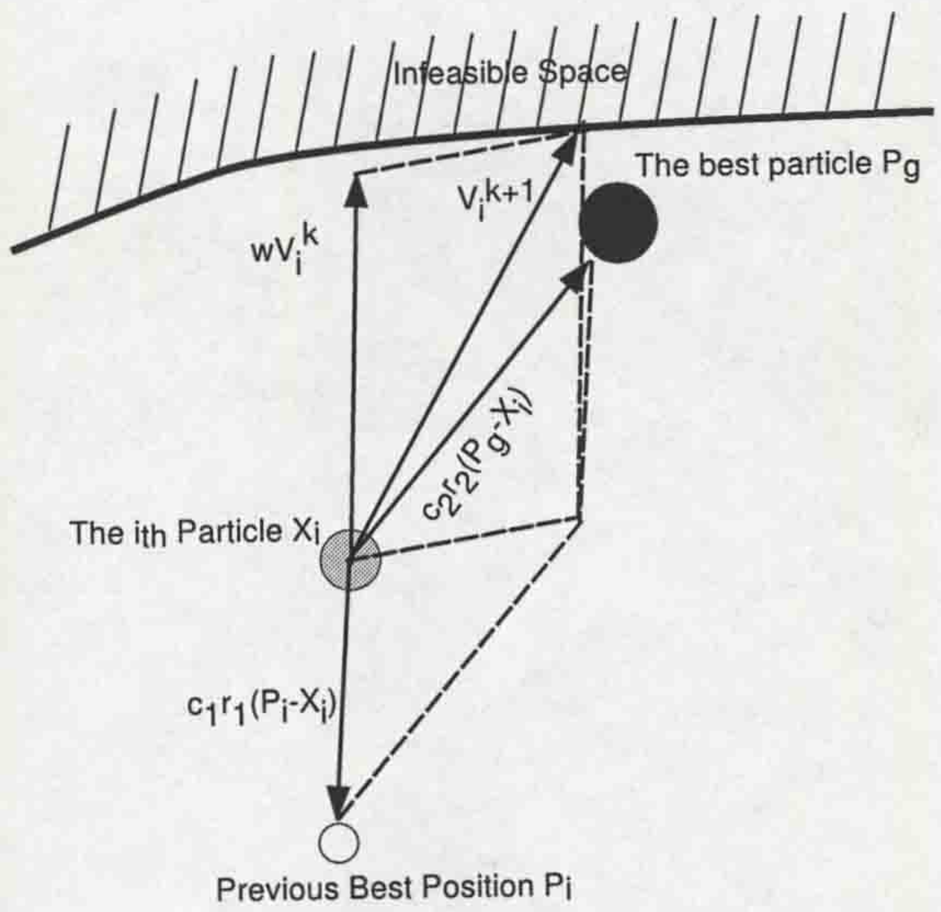


Figure 5.3: X_i flies back to its previous position and starts a new search.

techniques.

For all problems a population of 30 individuals is used. Although a time decreasing inertia weight was suggested to be better than a fixed one [35], the experimental results suggested that for these five examples, a fixed inertia weight $\omega = 0.8$ can produce better results. The default values of acceleration constants c_1, c_2 typically are set to 2.0. However with a setting of $c_1 = c_2 = 0.5$ better results were obtained. For each problem, 100 independent runs were carried out. The proposed algorithm was implemented in MATLAB 6.5 and executed on a Pentium 4, 2 GHz machine.

5.4.1 Example 1: Himmelblau's function

This problem, proposed by Himmelblau [155], is a common benchmark function for nonlinear constrained optimisation problems. We adopted this problem to test our PSO algorithm which has an improved constraint handling capability. The problem including 5 design variables and 6 nonlinear constraints is as follows:

Minimize

$$f(X) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \quad (5.4.1)$$

subject to:

$$0 \leq g_1(X) \leq 92 \quad (5.4.2)$$

$$90 \leq g_2(X) \leq 110 \quad (5.4.3)$$

$$20 \leq g_3(X) \leq 25 \quad (5.4.4)$$

where

$$g_1(X) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 \quad (5.4.5)$$

$$g_2(X) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 \quad (5.4.6)$$

$$g_3(X) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 \quad (5.4.7)$$

and

$$78 \leq x_1 \leq 102,\ 33 \leq x_2 \leq 45,\ 27 \leq x_3 \leq 45,\ 27 \leq x_4 \leq 45,\ \text{and}\ 27 \leq x_5 \leq 45$$

Himmelblau [155] used the Generalized Reduced Gradient method (GRG) to solve this problem. This problem was also tackled by Gen and Cheng [156] using a GA based on both local and global references. Philip and Yao [157] proposed an ES with stochastic ranking to solve this problem.

For Himmelblau's function, all the results obtained from the methods mentioned above are listed in Table 5.2 and are compared against those obtained with the proposed PSO. Other researchers have also proposed different approaches to solve this problem and produced good results. For example, Koziel and Michalewicz [158] proposed a new approach to solve this problems based on incorporating a homomorphous mapping between n-dimensional cube and a feasible search space. The best result they obtained was -30664.5. Parsopoulos [153] reported a best result of -31528.289, which is not feasible. The best solution reported by Hu [152] was -30665.5. Since the design variables were not included in their papers, we could not list their solutions in Table 5.2.

The maximum number of generations, used in the proposed PSO, was 3000 with 90000 function evaluations. The average execution time required for finding a feasible initial population and 90000 function evaluations was 36.5 s of CPU time. From Table 5.2 it can be seen that the proposed PSO has found the same optimum. The mean value for 100 independent runs is -30643.989 with a standard deviation of 70.043, which is worse than the mean value of -30665.539 reported by Philip and Yao [157]. However, it is worth mentioning that the number of function evaluations of their stochastic ranking technique was 350000. The proposed PSO has a much faster performance.

5.4.2 Example 2: spring design

In this section we will investigate two cases of a compression spring design problem. They both have 3 design variables: the wire diameter $d = x_1$, the mean coil diameter $D = x_2$ and the number of active coils $N = x_3$ as shown in Figure 5.4. The data type of design variables, objective function and constraints of these two cases are different.

Case 1

Case 1 is a real-world optimisation problem which involves discrete, integer and continuous design variables. It is aimed to minimize the volume of a compression spring under static loading. The 3 design variables are mixed: D is continuous, N is an integer, and d is a discrete variable having 42 possible value as shown in Table 5.3. The problem is formulated as follows:

Minimize

$$f(X) = \frac{\pi^2 x_2 x_1^2 (x_3 + 2)}{4} \tag{5.4.8}$$

subject to:

$$g_1(X) = \frac{8C_f F_{\max} x_2}{\pi x_1^3} - S \leq 0 \tag{5.4.9}$$

$$g_2(X) = l_f - l_{\max} \leq 0 \tag{5.4.10}$$

$$g_3(X) = d_{\min} - x_1 \leq 0 \tag{5.4.11}$$

$$g_4(X) = x_2 - D_{\max} \leq 0 \tag{5.4.12}$$

$$g_5(X) = 3.0 - \frac{x_2}{x_1} \leq 0 \tag{5.4.13}$$

$$g_6(X) = \sigma_p - \sigma_{pm} \leq 0 \tag{5.4.14}$$

$$g_7(X) = \sigma_p + \frac{(F_{\max} - F_p)}{K} + 1.05(x_3 + 2)x_1 - l_f \leq 0 \tag{5.4.15}$$

$$g_8(X) = \sigma_w - \frac{(F_{\max} - F_p)}{K} \leq 0 \tag{5.4.16}$$

where

$$C_f = \frac{4(x_2/x_1) - 1}{4(x_2/x_1) - 4} + \frac{0.615 x_1}{x_2} \tag{5.4.17}$$

$$K = \frac{Gx_1^4}{8x_3x_2^3} \tag{5.4.18}$$

$$\sigma_p = \frac{F_p}{K} \tag{5.4.19}$$

$$l_f = \frac{F_{\max}}{K} + 1.05(x_3 + 2)x_1 \tag{5.4.20}$$

Other specifications are: the maximum work load $F_{\max} = 1000.0$ lb; the maximum free length $l_{\max} = 14.0$ inch; the minimum wire diameter $d_{\min} = 0.2$ inch; the allowable maximum shear stress $S = 189000.0$ psi; the maximum outside diameter of the spring $D_{\max} = 3.0$ inch; the preload compression force $F_p = 300.0$ lp; the allowable maximum deflection under preload $\sigma_{pm} = 6.0$ inch; the deflection from preload position to maximum load position $\sigma_w = 1.25$ inch; the shear modulus of the material $G = 11.5 \times 10^6$ psi;

The design variables are limited as follows:

$$0.2 \leq x_1 \leq 1,\ 0.6 \leq x_2 \leq 3,\ 1 \leq x_3 \leq 70$$

This problem was investigated by Sandgren [145]. Deb [141] applied Genetic Adaptive Search (GeneAS) to solve this problem. Other attempts included a mixed-variable Differential Evolution (DE) algorithm [159].

The maximum number of generations, used in the proposed PSO, was fixed to 500 with 15000 function evaluations. The best solution for 100 runs is listed and it is compared to the results obtained by the other techniques mentioned above, which are listed in Table 5.4. It can be seen that PSO found the same global optimum as DE. It is worth mentioning that the maximum number of generations of DE was 650 generations corresponding to 26000 function evaluations[159].

The mean value for the 100 runs performed was 2.738024 with a standard deviation of 0.107061. The average time required for a single run was 5.8 s of CPU time.

Case 2

This problem was first investigated by Belegundu [160] and Arora [161], it aims to minimize the weight of a tension/compression spring. All three design

variables are continuous. There are four constraints which relate to minimum deflection, shear stress, surge frequency, and limits on outside diameter and design variables [161]. The mathematical model of the problem can be expressed as follows:

Minimize

$$f(X) = (x_3 + 2)x_2x_1^2 \tag{5.4.21}$$

subject to:

$$g_1(X) = 1 - \frac{x_2^3 x_3}{71785x_1^4} \leq 0 \tag{5.4.22}$$

$$g_2(X) = \frac{4x_2^2 - x_1x_2}{12566(x_2x_1^3 - x_1^4)} + \frac{1}{5108x_1^2} - 1 \leq 0 \tag{5.4.23}$$

$$g_3(X) = 1 - \frac{140.45x_1}{x_2^2x_3} \leq 0 \tag{5.4.24}$$

$$g_4(X) = \frac{x_2 + x_1}{1.5} - 1 \leq 0 \tag{5.4.25}$$

And the boundaries of design variables are given as follows:

$$0.05 \leq x_1 \leq 2,\ 0.25 \leq x_2 \leq 1.3,\ 2 \leq x_3 \leq 15$$

Arora [161] proposed an optimisation technique called Constraint Correction at constant Cost (CCC) to deal with this problem. Coello [162] investigated this problem with a GA with a self-adaptive penalty approach to handle constraints. This problem was also tackled by Ray and Liew using an EA inspired by a formal society and the civilization model [163].

The maximum number of generation was 500 corresponding to 15000 fitness function evaluations. The average execution time required for a single run was 5.2 s of CPU time. Table 5.5 lists the best solutions for 100 runs of our PSO and the techniques mentioned above. From Table 5.5, it can be noticed that Arora's technique is not applicable because the first constraint is violated. It can also be seen that our proposed approach was able to find the best solution.

The mean value for the 100 runs performed was 0.01270233 with a standard deviation of 4.124390×10^{-5}. Ray [163] reported a mean from 50 runs of

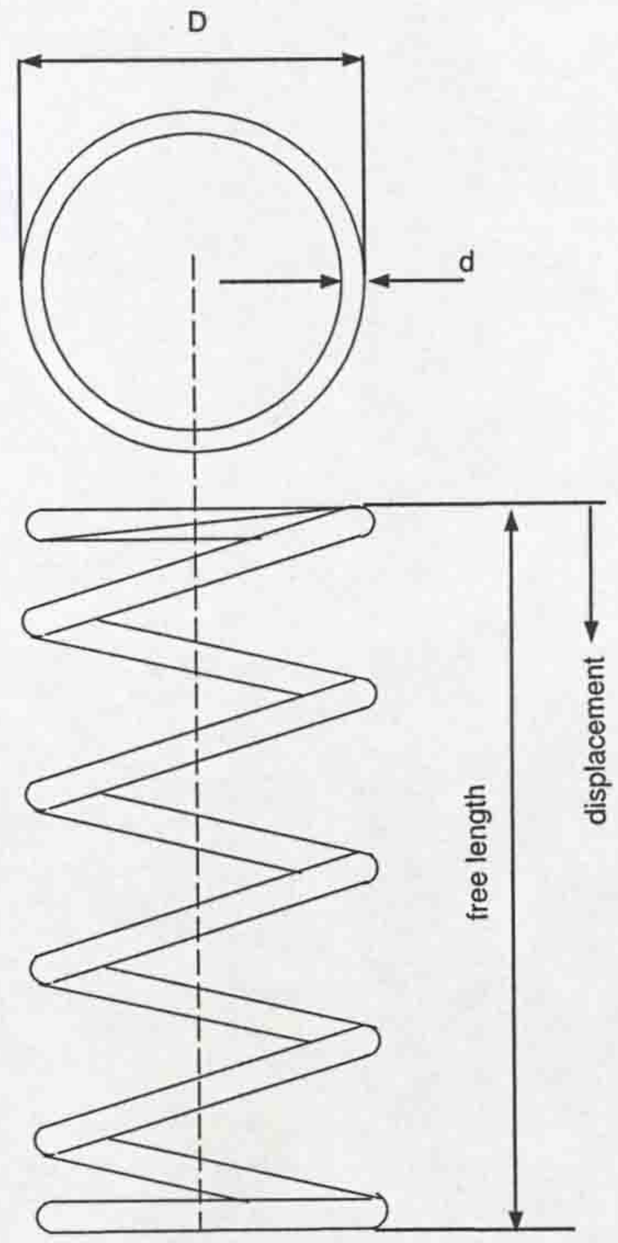


Figure 5.4: Spring design.

0.012922669 which is worse than that obtained by our proposed technique. The number of fitness function evaluations of Ray's algorithm was 25167.

5.4.3 Example 3: pressure vessel design

The pressure vessel design problem, shown in Figure 5.5, was introduced by Sandgren [145]. The objective of this problem is to minimize the total cost of materials, forming and welding of the pressure vessel. There are four design variables: the shell thickness $T_s = x_1$, the thickness of the head $T_h = x_2$, the inner radius $R = x_3$ and the length of the cylindrical section of the vessel $L = x_4$. T_s and T_h are discrete values which are integer multiples 0.0625 inch, in accordance with the availabe thickness of rolled steel plates, R and L are continuous. The optimisation problem can be expressed as follows:

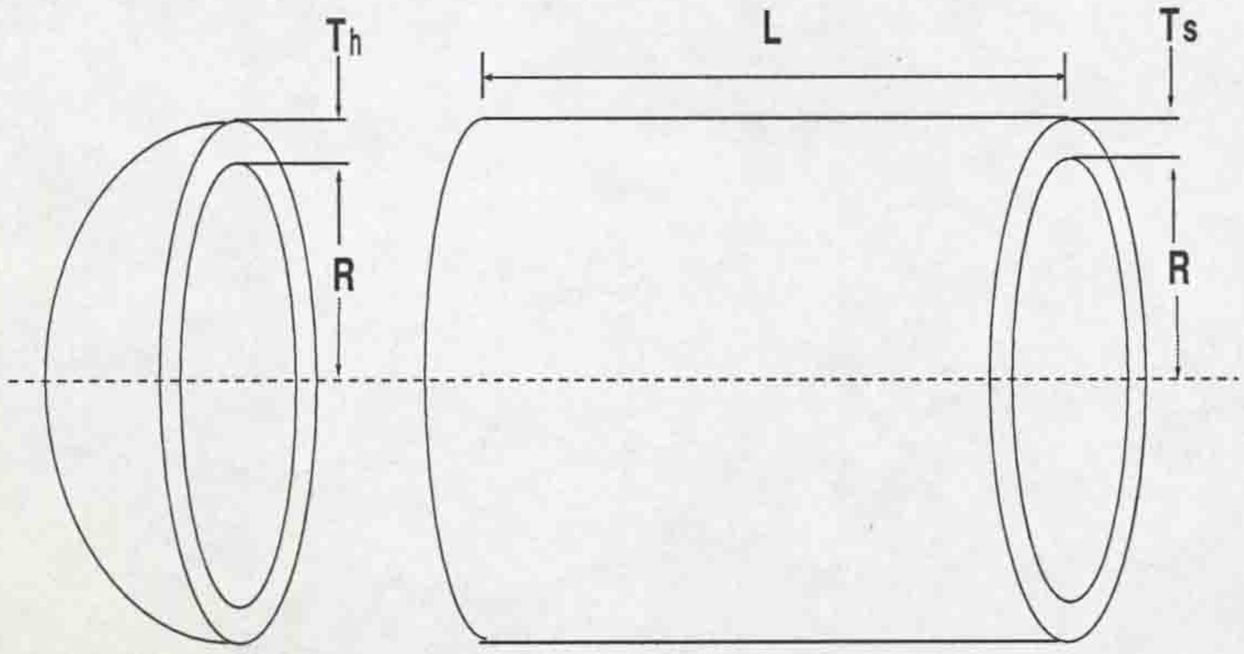


Figure 5.5: Pressure vessel design.

Minimize

$$f(X) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3; \qquad (5.4.26)$$

subject to:

$$\begin{aligned} g_1(X) &= 0.0193x_3 - x_1 \leq 0 && (5.4.27)\\ g_2(X) &= 0.00954x_3 - x_2 \leq 0 && (5.4.28)\\ g_3(X) &= 1,296,000 - \pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 \leq 0 && (5.4.29)\\ g_4(X) &= x_4 - 240 \leq 0 && (5.4.30) \end{aligned}$$

where the design variables have to be in the following ranges:

$$0.0625 \leq x_1 \leq 6.1875,\ 0.0625 \leq x_2 \leq 6.1875,\ 10 \leq x_3 \leq 200,\ 10 \leq x_4 \leq 200.$$

This problem was dealt with by Coello [164] using GA with a dominance-based tournament selection scheme (GADTS) to handle constraints. This problem was also investigated previously by Deb using Genetic Adaptive Search (GeneAS) [165]. It has also been tackled by Cao and Wu [140] using mixed-variables evolutionary programming (MVEP).

The maximum number of generations of the proposed PSO was set to 1000, corresponding to 30000 fitness function evaluations. The algorithm undertook

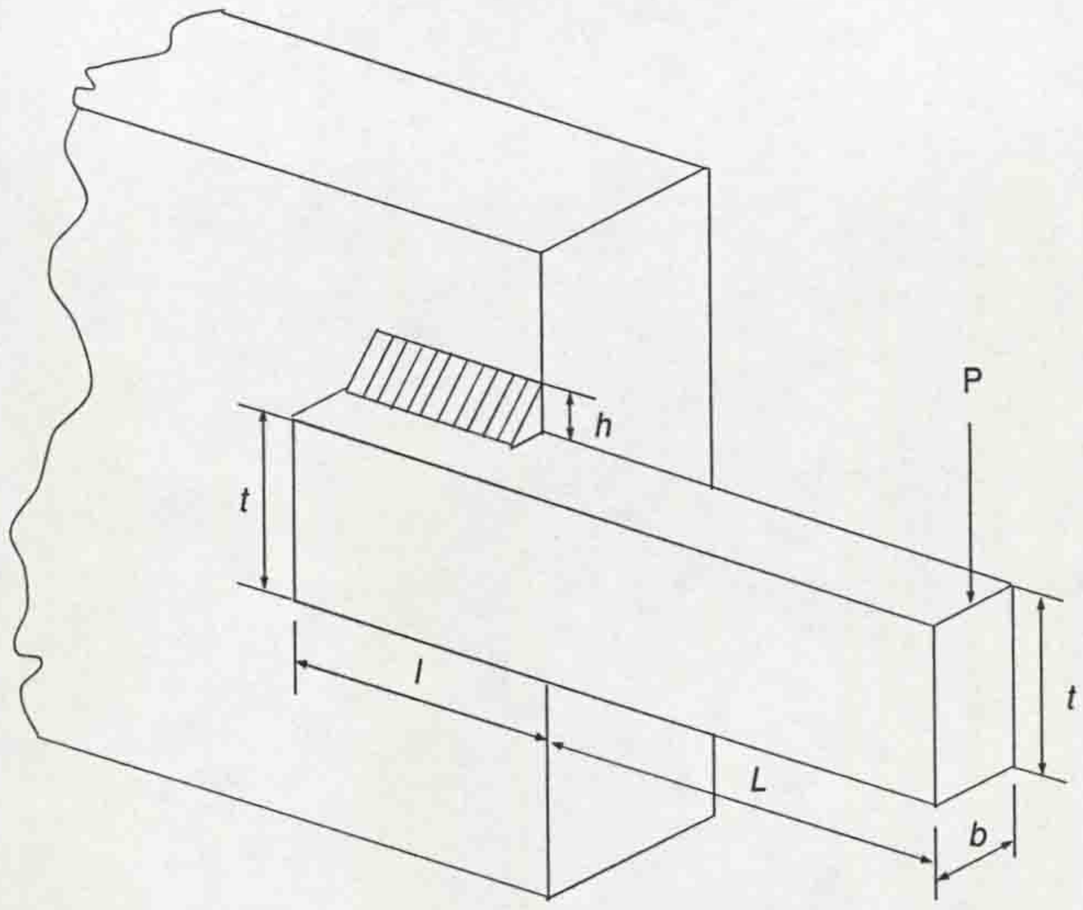


Figure 5.6: Welded beam design.

100 runs and the best result is listed in Table 5.6. The average CPU time required was 8.2 s for a single run. Table 5.6 also lists the best results produced by the other methods. Clearly, the new PSO gives better results than the other techniques.

The mean fitness value was $f(x) = 6289.92881$ with a standard deviation of 305.78, which is worse than the mean value of 6177.253268 produced by GADTS [164]. However, it is worth to mention that the number of fitness function evaluations of GADTS was 80000.

5.4.4 Example 4: welded beam design

As shown in Figure 5.6, a rectangular beam is designed as a cantilever beam to carry a certain load with minimum overall cost of fabrication. The problem involves four design variables: the thickness of the weld $h = x_1$, the length of the welded joint $l = x_2$, the width of the beam $t = x_3$ and the thickness of the beam $b = x_4$. The values of x_1 and x_2 are coded with integer multiples of 0.0065. There are seven constraints, which involve shear stress (τ), bending stress in the beam (σ), buckling load on the bar (P_c), deflection of the beam

(δ) and side constraints [166]. The welded beam problem is stated as follows:

Minimize

$$f(X) = 1.10471x_1^2x_2 + 0.04811x_3x_4(14.0 + x_2) \tag{5.4.31}$$

subject to:

$$\begin{aligned}
g_1(X) &= \tau(X) - \tau_{\max} \leq 0 && (5.4.32)\\
g_2(X) &= \sigma(X) - \sigma_{\max} \leq 0 && (5.4.33)\\
g_3(X) &= x_1 - x_4 \leq 0 && (5.4.34)\\
g_4(X) &= 0.10471x_1^2 + 0.04811x_3x_4(14.0 + x_2) - 5 \leq 0 && (5.4.35)\\
g_5(X) &= 0.125 - x_1 \leq 0 && (5.4.36)\\
g_6(X) &= \delta(X) - \delta_{\max} \leq 0 && (5.4.37)\\
g_7(X) &= P - P_c(X) \leq 0 && (5.4.38)
\end{aligned}$$

where

$$\begin{aligned}
\tau(X) &= \sqrt{(\tau')^2 + 2\tau'\tau''\frac{x_2}{2R} + (\tau'')^2} && (5.4.39)\\
\tau' &= \frac{P}{\sqrt{2}x_1x_2} && (5.4.40)\\
\tau'' &= \frac{MR}{J}, M = P\left(L + \frac{x_2}{2}\right) && (5.4.41)\\
R &= \sqrt{\frac{x_2^2}{4} + \left(\frac{x_1 + x_3}{2}\right)^2} && (5.4.42)\\
J &= 2\left\{\frac{x_1x_2}{\sqrt{2}}\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\} && (5.4.43)\\
\delta(X) &= \frac{4PL^3}{Ex_3^3x_4}, \sigma(X) = \frac{6PL}{x_4x_3^2} && (5.4.44)\\
P_c(X) &= \frac{4.013\sqrt{\frac{EGx_3^2x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right) && (5.4.45)\\
P &= 6000\text{lb}, L = 14\text{in}, E = 30 \times 10^6\text{psi}, G = 12 \times 10^6\text{psi} && (5.4.46)\\
\tau_{\max} &= 13,600 \text{ psi}, \sigma_{\max} = 30,000 \text{ psi}, \delta_{\max} = 0.25 \text{ in} && (5.4.47)
\end{aligned}$$

Shan He

The ranges for the design variables are given as follows:

$$0.1 \leq x_1 \leq 2.0,\ 0.1 \leq x_2 \leq 10,\ 0.1 \leq x_3 \leq 10,\ 0.1 \leq x_4 \leq 2.0.$$

This problem was investigated by Ragsdell [167] using a geometric programming. Deb [168] proposed a simple genetic algorithm (SGA) with binary representation and a traditional penalty function to solve this problem. The best-known result was also obtained by Deb using an real parameter GA [169]. Ray et al. tackled this problem using a society and civilization algorithm [163].

The best solution for 100 runs of the proposed PSO and those produced by the methods mentioned above are listed in Table 5.7. However, we could not list the best-known result of 2.38119 in this table, because the design variables were not presented in [169]. We can see that the new PSO algorithm provides even better results, which were obtained with the maximum number of generations set to 1000 and the total number of fitness function evaluations performed set to 30000. The average CPU time required for one execution of the proposed algorithm was 10.2 s.

The mean value of the objective function obtained from 100 runs was 2.381932, with a standard deviation 5.239371×10^{-3}. The number of fitness function evaluations of Deb's technique was 40080.

5.4.5 Example 5: hydrostatic thrust bearing design

The thrust bearing design problem was also proposed by Siddall [170]. This problem aims to minimize power loss associated with the bearing while satisfying several constraints. Four design variables are used: the bearing step radius R, recess radius R_0, oil viscosity μ and flow rate Q. There are seven constraints which limit load-carrying capacity, inlet oil pressure, oil temperature rise, oil film thickness and some physical requirements. The optimisation problem can be formulated as follows:

Minimize:

$$F(X) = \frac{QP_0}{0.7} + E_f \tag{5.4.48}$$

subject to:

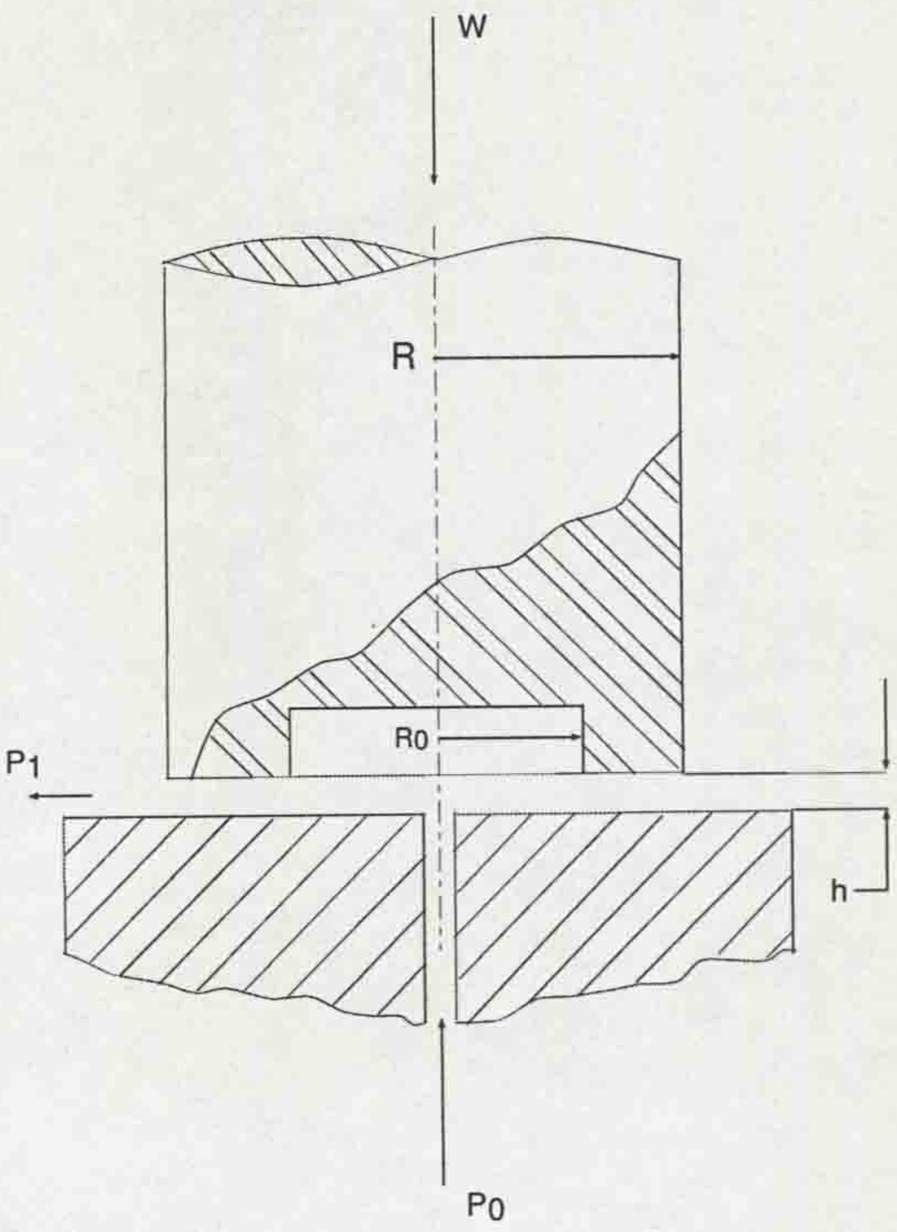


Figure 5.7: Thrust bearing design.

$$g_1(X) = W - W_s \leq 0 \tag{5.4.49}$$

$$g_2(X) = P_{\max} - P_0 \leq 0 \tag{5.4.50}$$

$$g_3(X) = \triangle T_{\max} - P_0 \leq 0 \tag{5.4.51}$$

$$g_4(X) = h - h_{\min} \leq 0 \tag{5.4.52}$$

$$g_5(X) = R - R_0 \leq 0 \tag{5.4.53}$$

$$g_6(X) = 0.001 - \frac{\gamma}{gP_0}\left(\frac{Q}{2\pi Rh}\right) \leq 0 \tag{5.4.54}$$

$$g_7(X) = 5000 - \frac{W}{\pi(R^2 - R_0^2)} \leq 0 \tag{5.4.55}$$

where W is the load carrying capacity which is given by:

$$W = \frac{\pi P_0}{2}\frac{R^2 - R_0^2}{\ln(R/R_0)} \tag{5.4.56}$$

and P_0 is the inlet pressure which can be defined as:

$$P_0 = \frac{6\mu Q}{\pi h^3}\ln\frac{R}{R_0} \tag{5.4.57}$$

and E_f is the friction loss:

$$E_f = 9336Q\gamma C\triangle T \tag{5.4.58}$$

where $\gamma = 0.0307$ lb/ is the weight density of oil and specific heat of oil $C = 0.5$ Btu/lb °F. And $\triangle T$ is the temperature which can be estimated by

$$\triangle T = 2(10^P - 559.7) \tag{5.4.59}$$

where

$$P = \frac{\log_{10}\log_{10}(8.122 \times 10^6\mu + 0.8) - C_1}{n} \tag{5.4.60}$$

and n and C_1 are constants for a given oil. Table 5.8 gives n and C_1 for various grades of oil. In this example, SAE 20 grade oil is chosen. Therefore, $n = 10.04$ and $C_1 = -3.55$. The film thickness can be calculated from the friction loss E_f from following equation:

$$h = \left(\frac{2\pi N}{60}\right)^2 \frac{2\pi\mu}{E_f}\left(\frac{R^4}{4} - \frac{R_0^4}{4}\right) \tag{5.4.61}$$

Other specifications of design are: weight of generator: $W_s = 101000$ lb (45804.99 Kg), maximum pressure available: $P_{\max} = 1000$ psi (6.89655×10^6 Pa), maximum temperature rise $\triangle T_{\max} = 50°\text{F}$ (10 °C), minimum oil thickness $h_{\min} = 0.001$ in (0.00254 cm), $g = 32.3 \times 12 = 386.4$ in/seg^2 (981.465cm/seg^2) and angular speed of shaft $N = 750$ RPM.

The following ranges were used for the design variables:

$$1.000 \leq R \leq 16.000,\ 1.000 \leq R_0 \leq 16.000,$$

$$1.0 \times 10^{-6} \leq \mu \leq 16 \times 10^{-6},\ 1.000 \leq Q \leq 16.000.$$

This problem was tackled by Siddall [170] using ADRANS (Gall's adaptive random search with a penalty function). Deb and Goyal [141] used GeneAS (Genetic Adaptive Search) to deal with this problem. Coello [171] proposed a novel constraint handling technique to solve this problem; GASO, which treats constraints as objective functions and solves them with a multiobjective technique.

It is worth noting that there are several discrepancies of unit and design specifications between Deb and Coello's papers [141] [171] and Siddall's book [170]. The first one is the absolute temperature (°F degrees Rankine) of ambient. Deb and Coello used 560.0 while Siddall used 559.7 in equation (5.4.59). In Siddall's book, the fourth constraint (g_4) and the sixth one (g_6) are multiplied by 10^8, and the fifth constraint and the third one are multiplied by 10^5 and 2000, respectively. The unit of fitness value from Deb and Coello's papers is foot-pounds per second while Siddall used inches-pounds per second. Due to these differences, we adopted two experiments: Case 1 and Case 2, with different unit and design specifications. The results are compared against those of Deb and Coello's, and Siddall's, respectively. Each experiment was performed 100 runs. The best solutions for Case 1 and for Deb and Coello's papers are listed in Table 5.9. The best solutions for Case 2 and Siddall's book are listed in Table 5.10.

Shan He

The maximum numbers of generations for both cases were set to 3000 with 90000 evaluations of the fitness function. The average execution time required for both cases were 52.7 s and 48.8 s of CPU time, respectively. The average fitness value from the proposed PSO for Case 1 is 1757.376840 with a standard deviation of 316.851024 which is better than most of the best results reported by other techniques depicted in Table 5.9. The average fitness value for Case 2 is 22874.674800 with a standard deviation of 3140.292915, which is better than the best result reported by Siddall [170].

In order to further illustrate the superiority of our algorithm, both in terms of accuracy and convergent rate, Case 1 of Example 5 is used to compare the proposed algorithm with the modified PSO algorithm of EI-Gallad [151] and a standard PSO with a static penalty given in [172]. The average solutions of the three algorithms were obtained after 100 runs where the maximum generation was set to 3000. The major drawback of [172] is that the static penalty coefficient r_g requires to be fine tuned in order to generate an acceptable result. For EI-Gallad's PSO and the standard PSO, the average solutions were 1877.195620 and 2939.070620, respectively, which are worse than the average result of 1757.37684 found by the proposed algorithm. The search processes of these three algorithms are shown in Figure 5.8. Clearly, from this figure one can see that our algorithm converges more quickly than the algorithms given in [151] and [172].

5.5 Conclusions

In this chapter, the standard PSO algorithm has been extended to handle mixed variables and constraints. The proposed method is relatively simple and easy to implement. A "fly back mechanism" is proposed to preserve feasible individuals. Compared to other constraint handling techniques based on penalty functions, this method is simpler, faster and provides more reliable solutions without any violation of the constraints.

The proposed PSO algorithm has been applied to solve a mathematical

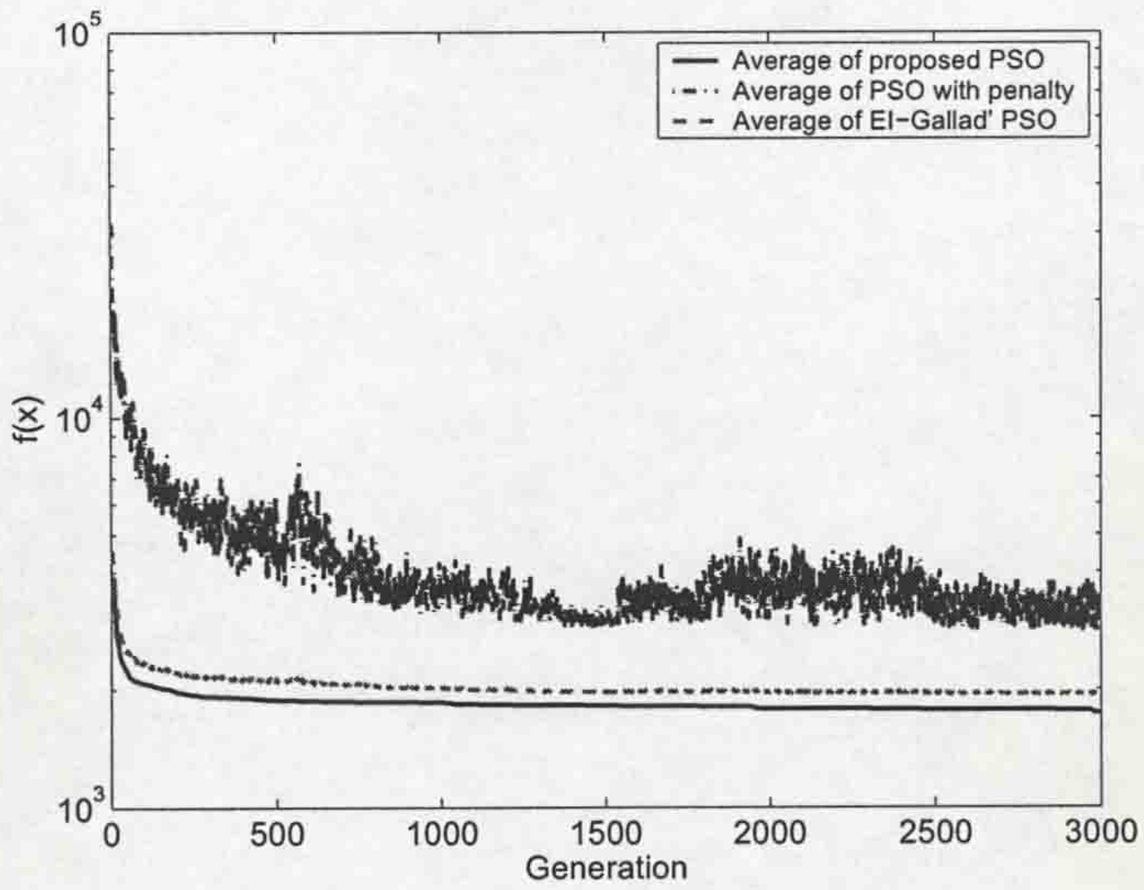


Figure 5.8: Search processes of three algorithms for thrust bearing design Case 1.

benchmark function and four mechanical design optimisation problems. The numerical results obtained by the proposed algorithm are better than or equal to other existing methods. Moreover, for most of our numerical examples, the PSO algorithm with "fly back mechanism" converges to the global minima within a few hundred iterations and its computational time is far less than the other PSO algorithms.

A drawback of the proposed PSO is that the constraint handling method requires a feasible initial population. For some problems, finding a feasible solution is NP-hard [173], and even impossible for the problems with conflicting constraints. Future work should extend the proposed PSO to tackle the initial population problem.

Table 5.1: Pseudo code for the improved PSO algorithm.

Set $k = 1$;

Randomly initialize positions and velocities of all particles;

FOR (each particle i in the initial population)

  WHILE (the constraints are violated)

    Randomly re-initialize current particle X_i

  END WHILE

END FOR

WHILE (the termination conditions are not met)

  FOR (each particle i in the swarm)

    Check feasibility: Check the feasibility of the current particle. If X_i^k is outside the feasible region, then reset X_i^k to the previous position X_i^{k-1};

    Calculate fitness: Calculate the fitness value $f(X_i^k)$ of current particle using equation (5.3.3);

    Update *pbest*: Compare the fitness value of *pbest* with $f(X_i^k)$. If $f(X_i^k)$ is better than the fitness value of *pbest*, then set *pbest* to the current position X_i^k;

    Update *gbest*: Find the global best position of the swarm. If the $f(X_i^k)$ is better than the fitness value of *gbest*, then *gbest* is set to the position of the current particle X_i^k;

    Update velocities: Calculate velocities V_i^k using equation (1.2.1);

    Update positions: Calculate positions X_i^k using equation (1.2.2);

  END FOR

  Set $k = k + 1$;

END WHILE

Table 5.2: Optimal solution of Himmelblau's function.

Design Variables	**Best solution found**			
	PSO	**Philip** [157]	**GRG** [155]	**Gen** [156]
x_1	78.0000	78.0000	78.6200	81.490
x_2	33.0000	33.0000	33.4400	34.0900
x_3	29.995256025682	29.995256025682	31.0700	31.2400
x_4	45.0000	45.0000	44.1800	42.2000
x_5	36.775812905789	36.775812905788	35.2200	34.3700
$g_1(X)$	92.0000	92.0000	91.7927	91.7819
$g_2(X)$	98.8405	98.8405	98.8929	99.3188
$g_3(X)$	20.0000	20.0000	20.1316	20.0604
$f(X)$	**-30665.539**	**-30665.539**	**-30373.949**	**-30183.576**

Table 5.3: Possible spring steel wire diameters.

Wire diameters (in inch)						
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

Table 5.4: Optimal solution of spring design for Case 1.

Design Variables	**Best solution found**			
	PSO	**Sandgren** [145]	**GeneAS** [141]	**DE** [159]
$x_1(d)$	0.283	0.283	0.283	0.283
$x_2(D)$	1.223041010	1.180701	1.226	1.223041010
$x_3(N)$	9	10	9	9
$g_1(X)$	-1008.8114	-54309	-713.510	-1008.8114
$g_2(X)$	-8.9456	-8.8187	-8.933	-8.9456
$g_3(X)$	-0.083	-0.08298	-0.083	-0.083
$g_4(X)$	-1.777	-1.8193	-1.491	-1.777
$g_5(X)$	-1.3217	-1.1723	-1.337	-1.3217
$g_6(X)$	-5.4643	-5.4643	-5.461	-5.4643
$g_7(X)$	0.0000	0.0000	0.0000	0.0000
$g_8(X)$	0.0000	0.0000	-0.009	0.0000
$f(X)$	**2.65856**	**2.7995**	**2.665**	**2.65856**

Table 5.5: Optimal solution of spring design for Case 2.

Design Variables	**Best solution found**			
	PSO	**Ray** [163]	**SAPA** [162]	**Arora** [161]
$x_1(d)$	0.05169040	0.0521602	0.051480	0.053396
$x_2(D)$	0.35674999	0.36815870	0.351661	0.399180
$x_3(N)$	11.28712599	10.64844226	11.632201	9.185400
$g_1(X)$	-0.00000449	-0.00000001	-0.002080	0.000019
$g_2(X)$	0.00000000	-0.00000000	-0.000110	-0.000018
$g_3(X)$	-4.05382661	-4.075805	-4.026318	-4.123832
$g_4(X)$	-0.72770641	-0.71978739	-4.02638	-0.698283
$f(X)$	**0.0126652812**	**0.0126692493**	**0.0127047834**	**0.0127302737**

Table 5.6: Optimal solution of pressure vessel design.

Design Variables	**Best solution found**			
	PSO	**GADTS** [164]	**GeneAS** [165]	**MVEP** [140]
$x_1(T_s)$	0.81250000	0.8125	0.9345	1.000
$x_2(T_h)$	0.43750000	0.4375	0.5000	0.625
$x_3(R)$	42.09844560	40.097398	48.3290	51.1958
$x_4(L)$	176.63659584	176.654047	112.6790	90.7821
$g_1(X)$	0.00000000	-0.000020	-0.004750	-0.0119
$g_2(X)$	-0.03588083	-0.035891	-0.038941	-0.1366
$g_3(X)$	0.00000000	-27.886075	-3652.876838	-13584.5631
$g_4(X)$	-63.36340416	-63.345953	-127.321000	-149.2179
$f(X)$	**6059.7143**	**6059.946341**	**6410.3811**	**7108.6160**

Table 5.7: Optimal solution of welded beam design.

Design Variables	**Best solution found**			
	PSO	**Ray** [163]	**Ragsdell** [167]	**Deb** [168]
$x_1(h)$	0.24436898	0.244438276	0.2455	0.2489
$x_2(l)$	6.21751974	6.237967234	6.1960	6.1730
$x_3(t)$	8.29147139	8.288576143	8.2730	8.1789
$x_4(b)$	0.24436898	0.244566182	0.2455	0.2533
$g_1(X)$	-5741.17693313	-5760.11047125	-5743.826517	-5758.603777
$g_2(X)$	-0.00000067	-3.24542756	-4.71509720	-255.576901
$g_3(X)$	0.00000000	-0.00012790	0.00000000	-0.004400
$g_4(X)$	-3.02295458	-3.02005520	-3.02028858	-2.982866
$g_5(X)$	-0.11936898	-0.11943827	-0.12050000	-0.123900
$g_6(X)$	-0.23424083	-0.23423703	-0.23420813	-0.234160
$g_7(X)$	-0.00030900	-13.07930496	-74.27685602	-618.81849251
$f(X)$	**2.3809565827**	**2.3854347**	**2.385937**	**2.433116**

Table 5.8: Values of n and C_1 for various grades of oil.

Oil	C_1	n
SAE 5	10.85	-3.91
SAE 10	10.45	-3.72
SAE 20	10.04	-3.55
SAE 30	9.88	-3.48
SAE 40	9.83	-3.46
SAE 50	9.82	-3.44

Table 5.9: Optimal solution of thrust bearing design for Case 1, Coello and Deb's papers.

Design Variables	**Best solution found**			
	PSO	**GASO** [171]	**GeneAS** [141]	**BGA** [141]
$x_1(R)$	5.956868685	6.271	6.778	7.7077
$x_2(R_0)$	5.389175395	12.901	6.234	6.549
$x_3(\mu) \times 10^{-6}$	5.40213310	5.605	6.096	6.619
x_4 (Q)	2.30154678	2.938	3.809	4.849
$g_1(X)$	22.01094912	2126.86734	8329.7681	1440.6013
$g_2(X)$	0.00000000	68.0396	177.3527	297.1495
$g_3(X)$	0.58406092	3.705191	10.684543	17.353800
$g_4(X)$	0.00033480	0.000559	0.000652	0.000891
$g_5(X)$	0.56769329	0.666000	0.544000	0.528000
$g_6(X)$	0.00083138	0.000805	0.000717	0.000624
$g_7(X)$	7.61684431	849.718683	83.618221	467.686527
$f(X)$	**1632.2149**	**1950.2860**	**2161.4215**	**2296.2119**

Table 5.10: Optimal solution of thrust bearing design for Case 2 and Siddall's book.

Design Variables	**Best solution found**	
	PSO	**Siddall** [170]
$x_1(R)$	5.956048839021	7.1550805
$x_2(R_0)$	5.388766560465	6.6886822
$x_3(\mu) \times 10^{-6}$	6.001637904878	8.3207655
x_4 (Q)	2.778703032216	9.1684614
$g_1(X)$	0.00000074	71.040915
$g_2(X)$	0.00234129	328.27277
$g_3(X)$	17605.54663647	68912.380
$g_4(X)$	47175.71419475	144524.15
$g_5(X)$	56728.22785557	46639.822
$g_6(X)$	79777.35289299	17724.808
$g_7(X)$	4.54730416	17.287484
$f(X)$	**20374.684**	**29221.321**

Chapter 6

Solving Optimal Power Flow Problems with PSOPC and GSO

In this chapter, Optimal Power Flow (OPF) problems will be investigated. Essentially, OPF problems are a kind of mixed-variable constrained optimisation problem. Traditionally, OPF problems have been tackled by gradient-based optimisation methods. Here we applied two novel ABO algorithms we developed, namely, PSOPC in Chapter 3 and GSO in Chapter 2, to OPF problems. Numerical experiments were carried out on an IEEE 30-bus for three different fuel cost minimization problems. In order to evaluate its performance on real-world power systems, a practical IEEE 118-bus system is also employed for the GSO algorithm. So far, both algorithms provides better results than those obtained from the other optimisation techniques in terms of accuracy and convergence speed.

6.1 Nomenclature

θ_{ij} voltage angle difference between buses i and j (rad)

B_{ij} transfer susceptance between bus i and j (p.u.)

G_{ij} transfer conductance between bus i and j (p.u.)

g_k conductance of branch k (p.u.)

N_0 set of numbers of total buses excluding slack bus

N_B set of numbers of total buses

N_C set of numbers of shunt compensators

N_D set of numbers of power demand buses

N_E set of numbers of network branches

N_G set of numbers of generator buses

N_i set of numbers of buses adjacent to bus i, including bus i

N_{PQ} set of numbers of PQ buses

N_{PV} set of numbers of PV buses

$N_Q^{\lim}$ set of numbers on buses on which injected reactive power outside limits

N_T set of numbers of transformer branches

$N_V^{\lim}$ set of numbers on buses on which voltages outside limits

P_{D_i} demanded active power at bus i (p.u.)

P_{G_i} injected active power at bus i (p.u.)

Q_{C_i} reactive power source installation at bus i (p.u.)

Q_{D_i} demanded reactive power at bus i (p.u.)

Q_{G_i} injected reactive power at bus i (p.u.)
V_{G_i} voltage vectors of PQ buses (p.u.)
T_i tap position at transformer i
V_i voltage magnitude at bus i (p.u.)
S_k apparent power flow in branch k (p.u.)

6.2 Introduction

The optimal power flow (OPF) problem [174] aims to achieve an optimal solution of a specific power system objective function, such as fuel cost, by adjusting the power system control variables, while satisfying a set of operational and physical constraints. The OPF problem has been intensively studied and widely used in power system operation and planning [175]. It can be formulated as a nonlinear constrained optimisation problem. The control variables include the generator active power, the generator bus voltages, the tap ratios of transformer and the reactive power generations of VAR sources. State variables are slack bus power, load bus voltages, generator reactive power outputs, and network power flows. The constrains include inequality ones which are the limits of control variables and state variables; and equality ones which are the power flow equations.

In order to solve the OPF problem, a number of conventional optimisation techniques have been applied. They include nonlinear programming (NLP) [176], quadratic programming (QP) [177], linear programming (LP) [178], and interior point methods [179]. All these techniques are gradient-based deterministic optimisation algorithms and usually rely on the existence of the gradients of variables, to find the global minimum. However, the OPF problem is very complex, considering the various constraints, mixed-variables and high-dimensionality. These constraints lead to the non-differentiable, nonlinear and non-convex nature of the OPF problem. Therefore the methods rely on the gradient information and the convexity property of the objective function, which do not guarantee to find the global optimum for the OPF problem. These conventional techniques also suffer from bad starting points and frequently

converge to local minima or even diverge.

On the other hand, evolutionary algorithms (EAs), such as Genetic Algorithms (GA), Evolutionary Programming (EP), and Evolution Strategies (ES), have been developed in the past a few decades. Their applications to global optimisation problems become attractive because they have better global search abilities over conventional optimisation algorithms. The OPF problem has been solved with Evolutionary Programming (EP) [139]. The EP based OPF was evaluated on an IEEE 30-bus system and the results were compared with those obtained using a conventional gradient-based method. In [180] an enhanced EP with the use of gradient information was applied to the IEEE 30-bus system under different generator input-output conditions. A GA with adaptive crossover and mutation, based on the fitness statistics of population, was applied to minimize the active power loss in transmission networks [138]. Another enhanced GA was also applied to solve OPF problem [3] in which advanced genetic operators such as fitness scaling, elitism and hill climbing and other problem-specific operators were employed to improve the efficiency of the simple GA. Recently, a novel Particle Swarm Optimiser with Passive Congregation (PSOPC) [181] was applied to solve the OPF problem [44]. Numerical experiments were also carried out on an IEEE 30-bus for three different fuel cost minimization problems. However, these evolutionary algorithms were originally evaluated within a 30-dimensional space, and assumed it would be appropriate to be used for very high-dimensional optimisation problems. Therefore, the scalability of these algorithms to practical power systems, which usually consist hundreds control variables, is questionable.

We first apply the GSO algorithm as introduced in Chapter 2 for the solution of the OPF problem. We test the GSO algorithms on the standard IEEE 30-bus power system in three cases: (1) minimization of fuel cost, (2) voltage profile improvement and (3) voltage stability enhancement, and also in comparison with GA and PSO respectively. The simulation study is also carried out based on a practical 118-bus system for the GSO algorithm. In Chapter 3, passive congregation, a concept from biology, was introduced to the standard

PSO to improve its search performance. In this chapter, we also present the PSO with passive congregation (PSOPC) algorithm for the solution of OPF.

6.3 Optimal Power Flow Problem Formulation

The OPF problem can be formulated as a constrained optimisation problem as follows:

$$\min \quad f(\mathbf{x}, \mathbf{u}) \tag{6.3.1}$$

$$\text{s.t.} \quad g(\mathbf{x}, \mathbf{u}) = 0 \tag{6.3.2}$$

$$h(\mathbf{x}, \mathbf{u}) \leq 0 \tag{6.3.3}$$

where $\mathbf{x}$ is the vector of dependent variables such as slack bus power P_{G_1}, load bus voltage V_L, generator reactive power outputs Q_G and apparent power flow S_k. $\mathbf{x}$ can be expressed as

$$\mathbf{x}^T = [P_{G_1}, V_{L_1} \cdots V_{L_{N_L}}, Q_{G_1} \cdots Q_{G_{N_G}}, S_1 \cdots S_{N_E}] \tag{6.3.4}$$

$\mathbf{u}$ is a set of the control variables such as generator active power outputs P_G except the slack bus P_{G_1}, generator voltages V_G, transformer tap setting T, number of load buses N_L and reactive power generations of VAR sources Q_c. Therefore, $\mathbf{u}$ can be expressed as

$$\mathbf{u}^T = [P_{G_2} \cdots P_{G_{N_G}}, V_{G_1} \cdots V_{G_{N_G}}, T_1 \cdots T_{N_T}, Q_{c_1} \cdots Q_{c_{N_C}}] \tag{6.3.5}$$

The equality constraints $g(\mathbf{x}, \mathbf{u})$ are the nonlinear power flow equations which are formulated as follows:

$$0 = P_{G_i} - P_{D_i} - V_i \sum_{j \in N_i} V_j (G_{ij} \cos\theta_{ij} + B_{ij} \sin\theta_{ij})$$

$$i \in N_0 \tag{6.3.6}$$

$$0 = Q_{G_i} - Q_{D_i} - V_i \sum_{j \in N_i} V_j (G_{ij} \sin\theta_{ij} + B_{ij} \cos\theta_{ij})$$
$$i \in N_{PQ} \qquad (6.3.7)$$

And the inequality constraints $h(\mathbf{x}, \mathbf{u})$ are the limits of control variables and state variables which can be formulated as:

$$\begin{aligned}
P_{G_i}^{\min} \le P_{G_i} \le P_{G_i}^{\max} \quad & i \in N_G \\
Q_{G_i}^{\min} \le Q_{G_i} \le Q_{G_i}^{\max} \quad & i \in N_G \\
Q_{C_i}^{\min} \le Q_{C_i} \le Q_{C_i}^{\max} \quad & i \in N_C \\
T_k^{\min} \le T_k \le T_k^{\max} \quad & i \in N_T \\
V_i^{\min} \le V_i \le V_i^{\max} \quad & i \in N_B \\
|S_k| \le S_k^{\max} \quad & i \in N_E
\end{aligned} \qquad (6.3.8)$$

To solve a nonlinear constrained optimisation problem, the most common method uses penalty functions to transfer a constrained optimisation problem into an unconstrained one. The objective function equation (6.3.1), is generalized as follows:

$$F = f + \sum_{i \in N_V^{\lim}} \lambda_{V_i} (V_i - V_i^{\lim})^2 +$$
$$\sum_{i \in N_Q^{\lim}} \lambda_{G_i} (Q_{G_i} - Q_{G_i}^{\lim})^2 + \sum_{i \in N_E^{\lim}} \lambda_{S_i} (|S_i| - S_i^{\max})^2 \qquad (6.3.9)$$

where λ_{V_i}, λ_{G_i} and λ_{S_i} are the penalty factors. $V_i^{\lim}$, $Q_{G_i}^{\lim}$ are defined as

$$V_i^{\lim} = \begin{cases} V_i^{\max} & \text{if} \quad V_i > V_i^{\max} \\ V_i^{\min} & \text{if} \quad V_i < V_i^{\min} \end{cases} \qquad (6.3.10)$$

$$Q_{G_i}^{\lim} = \begin{cases} Q_{G_i}^{\max} & \text{if} \quad Q_{G_i} > Q_{G_i}^{\max} \\ Q_{G_i}^{\min} & \text{if} \quad Q_{G_i} < Q_{G_i}^{\min} \end{cases} \qquad (6.3.11)$$

where max and min denote the maximum and minimum values of the variables, respectively.

6.4 Numerical Results

For all problems a population of 50 individuals is used. A time decreasing inertia weight ω which starts from 0.9 and ends at 0.4 was used for the PSOPC algorithm. The default value of acceleration constants c_1, c_2 typically are set to 2.0. However with a setting of $c_1 = c_2 = 0.5$ better results were obtained.

For the GSOOPF algorithm, the initial head angle φ^0 of each individual is set to be $\frac{\pi}{4}$. The constant a is given by $round(\sqrt{n+1})$. The maximum pursuit angle $\theta_{\max}$ is $\frac{\pi}{a^2}$. The maximum turning angle α is set to be $\frac{\pi}{2a^2}$. The maximum pursuit distance $l_{\max}$ is calculated from:

$$l_{\max} = \parallel U_i - L_i \parallel = \sqrt{\sum_{i=1}^{n}(U_i - L_i)^2}$$

where L_i and U_i are the lower and upper bounds for the i_{th} dimension, respectively. The parameter need to tune is the percentage of rangers; our recommended percentage of rangers is 20%, which was used throughout all our experiments. For each experiment, 100 independent runs were carried out. The maximum generation was set to 500.

Table 6.1: The best values of GSOOPF, PSOPCOPF, GAOPF, PSOOPF, IEA [2], and EGA [3] for Case 1.

	GSOOPF	PSOPCOPF	GAOPF	PSOOPF	IEP	EGA
Fuel cost ($/h)	**802.0453**	802.0477	804.349	802.41	802.465	802.6087
$\sum$ voltage deviations	0.8259	0.8089	0.9023	0.8765	-	0.8073
$L_{\max}$	0.1382	0.1383	0.1398	0.1384	-	0.1394

6.4.1 IEEE 30-Bus system

The standard IEEE 30-bus test system as shown in Fig. 6.1 was employed to evaluate our PSOPCOPF and GSOOPF algorithms. The system line and bus data for 30-bus system were adopted from [176]. The system consists of 48 branches, 6 generator-buses, and 22 load-buses. The generators are at bus 1, 2, 5, 8, 11 and 13. Branches (6,9), (6,10), (4,12) and (27,28), contain transformers with off-nominal tap ratios. The transformer tap setting can take 17 discrete values in the range of [0.9 1.1] with the step size of 0.0125. The bus shunt admittances are also discrete variables in the interval of [0.0 0.05] p.u. and the step size is 0.01 p.u. In total, there are 24 control variables.

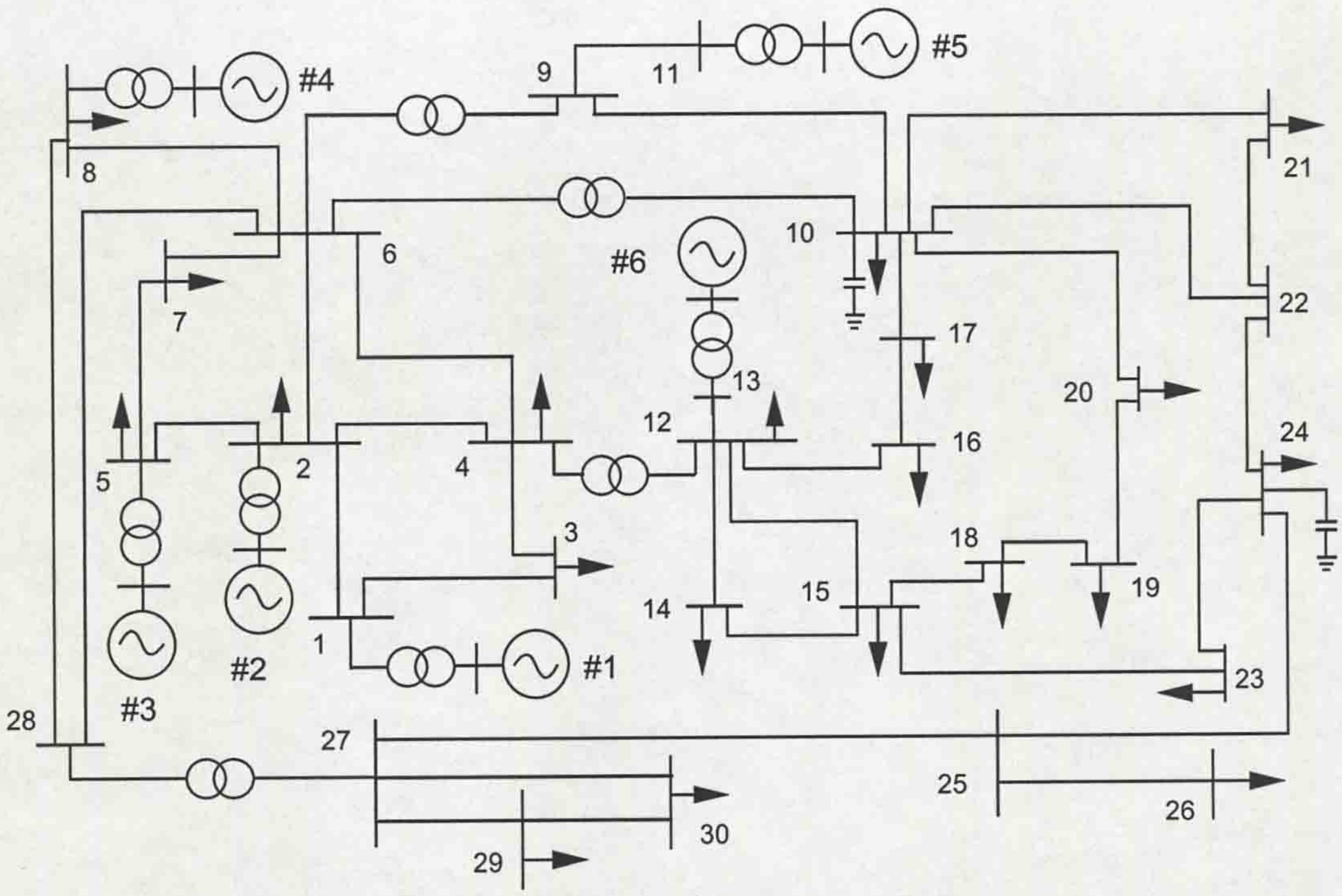


Figure 6.1: IEEE 30-bus System

Table 6.2: Optimal control variables.

	Case 1	Case 2	Case 3
P_1	176.0951	176.1143	176.1971
P_2	48.8271	49.0286	48.8318
P_5	21.5123	21.7131	21.5216
P_8	22.1133	21.5152	22.0664
P_{11}	12.2255	12.5905	12.2704
P_{13}	12.0012	12.6071	12.0129
V_1	1.0500	1.0272	1.0499
V_2	1.0377	1.0180	1.0359
V_5	1.0105	1.0195	1.0071
V_8	1.0182	1.0016	1.0138
V_{11}	1.0847	1.0560	1.0842
V_{13}	1.0703	1.0093	1.0852
T_{11}	1.0250	1.0750	0.9750
T_{12}	0.9250	0.9000	0.9500
T_{15}	1.0000	0.9750	1.0000
T_{36}	0.9500	0.9500	0.9250
Q_{c10}	0.05	0.05	0.01
Q_{c11}	0.04	0.00	0.01
Q_{c15}	0.04	0.00	0.04
Q_{c17}	0.05	0.00	0.02
Q_{c20}	0.04	0.05	0.03
Q_{c21}	0.05	0.05	0.04
Q_{c23}	0.03	0.05	0.01
Q_{c24}	0.05	0.04	0.00
Q_{c29}	0.02	0.01	0.00

Case 1: Minimization of fuel cost

The objective of this case is to minimize the total fuel cost:

$$J = \sum_{i=1}^{N_G} f_i \tag{6.4.1}$$

where f_i is the fuel cost (\$ /h) of the i_{th} generator:

$$f_i = a_i + b_i P_{G_i} + c_i P_{G_i}^2$$

a_i, b_i and c_i are fuel cost coefficients, P_{G_i} is the real power output generated by the i_{th} generator.

The optimal control variables obtained by the GSOOPF from 100 runs for this case are tabulated in Table 6.2. This problem was also tackled using a gradient based optimisation method [176]. An improved Evolutionary Programming (IEA) was applied to solve this problem [2]. The best-known result was obtained by Bakirtzis et al. [3] using an enhanced GA (EGA). They designed a set of advanced and problem-specific genetic operators, for example, Gene Swap Operator, Gene Inverse Operator, etc., to solve OPF problems.

In Table 6.1, we tabulate the results obtained from the techniques mentioned above in comparison with the result generated by PSOPCOPF and GSOOPF. We also implemented a GA based OPF (GAOPF) algorithm using GADST toolbox and a PSO based OPF (PSOOPF) algorithm using PSOt toolbox. The search process of our GSOOPF algorithm is shown in Fig. 6.2. It is worth mentioning that, as different programming environments and power flow calculation methods were used in [176], [2], [3] and this research, it is not easy to compare the computation time required by each algorithm. However, by comparing the computation time used by the implemented algorithms, e.g., PSOOPF and GAOPF, we found that our GSOOPF obtained the best results in the shortest time.

Case 2: Voltage profile improvement

This example aims at minimizing fuel cost together with a flatter voltage profile. The objective function is modified to minimize the fuel cost while at

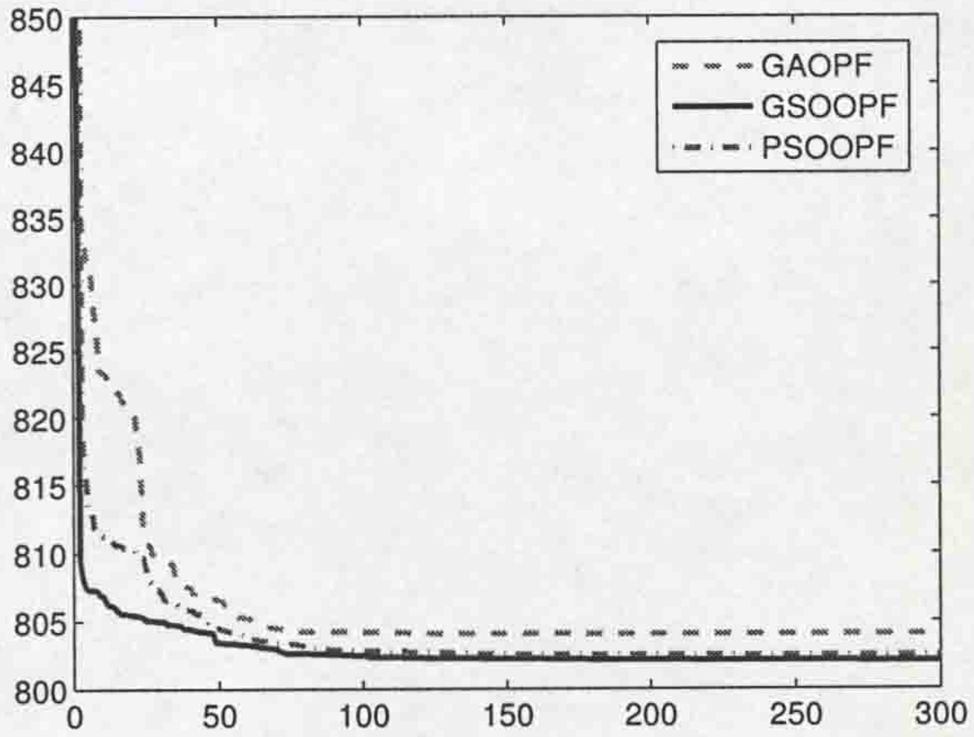


Figure 6.2: Search process of GSOOPF for Case 1

the same time to improve voltage profile by minimizing the load bus voltage deviations from 1.0 per unit. The objective function can be express as:

$$J = \sum_{i=1}^{N_G} f_i + \omega \sum_{i \in NL} |V_i - 1.0| \tag{6.4.2}$$

where ω is the weighting factor.

Table 6.2 tabulates the optimal control variables of the GSOOPF obtained from 100 runs. The best result of the GSOOPF and PSOPCOPF from 100 runs is also tabulated in comparison to PSOOPF and GAOPF in Table 6.3. It can be seen from the table that, for GSOOPF, the voltage variation has been reduced from 0.8259 in Case 1 to 0.0926 in Case 2. The reduction ratio is 88.79%. For PSOPCOPF, the voltage variation has been reduced from 0.8089 in Case 1 to 0.0954 in Case 2. The reduction ratio is 88.20%. The system voltage profile obtained by GSOOPF of this case is compared to that of Case 1 in Fig. 6.3. The search process of our GSOOPF algorithm is shown in Fig. 6.4 in comparison to the search processes of the other two algorithms.

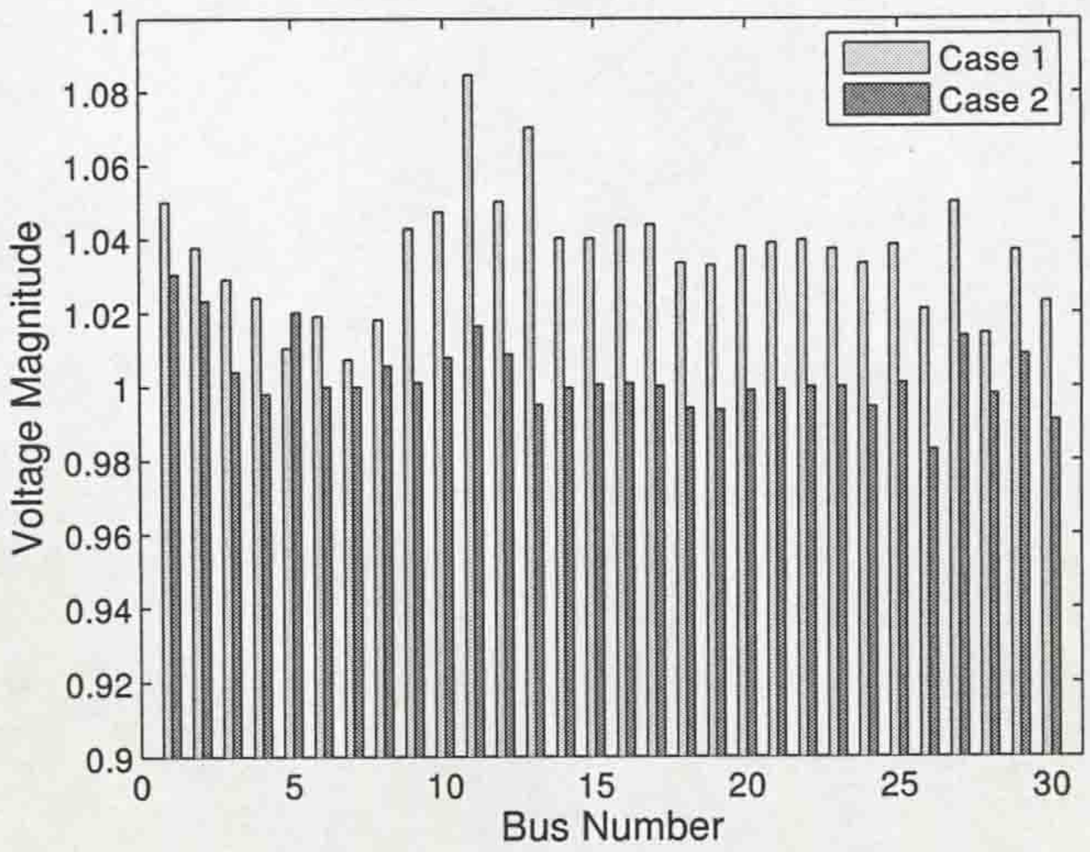


Figure 6.3: System voltage profile

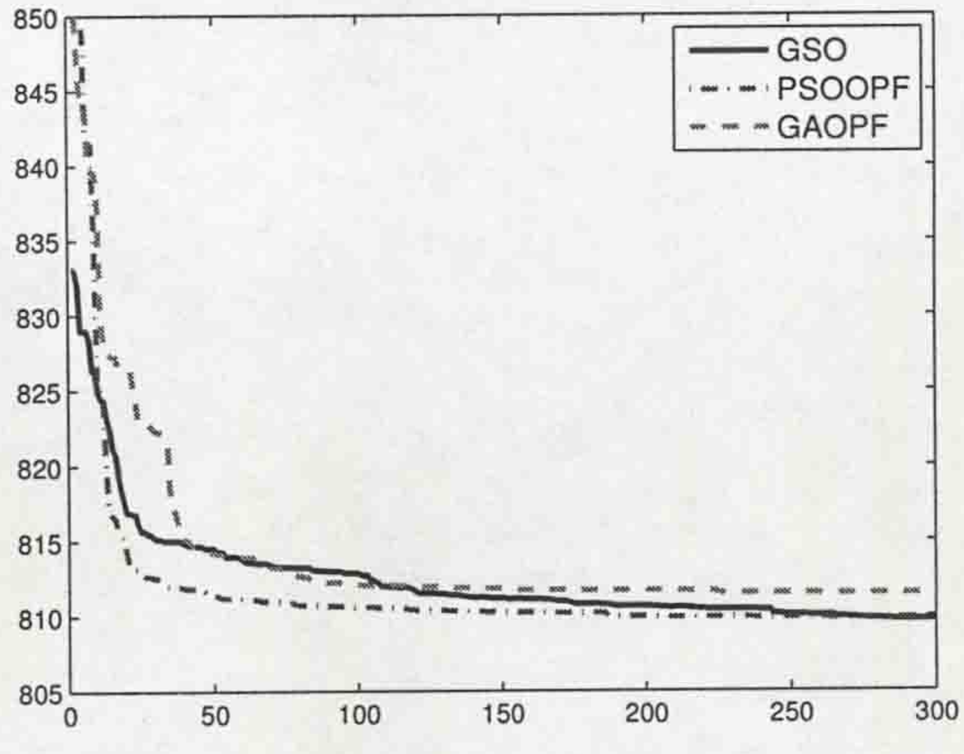


Figure 6.4: Search process of GSOOPF for Case 2

Table 6.3: The best values of GSOOPF for Case 2

	GSOOPF	PSOPCOPF	GAOPF	PSOOPF
Fuel cost ($/h)	804.0996	**804.0650**	806.2321	804.1426
$\sum$ voltage deviations	**0.0926**	0.0954	0.1423	0.1011
$L_{\max}$	0.1473	0.1481	0.1488	0.1479

Case 3: Voltage stability enhancement

In this example, we aim to minimize fuel cost and enhances voltage stability profile through out the whole network. L is the stability indicators at every bus of the system and $L_{\max}$ is the maximum value of L-index defined as [182]:

$$L_{\max} = \max\{L_k, K = 1, \cdots, NL\} \tag{6.4.3}$$

And L can be calculated from the following equation:

$$L_j = \left|1 + \frac{V_{0j}}{V_j}\right| = \left|\frac{S_j^+}{Y_{jj}^{+*} \cdot V_j^2}\right| \tag{6.4.4}$$

where Y_{jj}^+ is the transformed admittance, $Y_{jj}^+ = 1/Z_{jj}$; V_j is the consumer node voltage; S_j^+ is the transformed power $S_j^+ = S_j + S_j^{\text{cor}}$; and S_j^{cor} is given by:

$$S_j^{\text{cor}} = \left[\sum_{i \in \alpha} \left(\frac{Z_{ji}^*}{Z_{jj}^*}\right) \cdot \left(\frac{S_i}{V_i}\right)\right] \cdot V_j \tag{6.4.5}$$

and Z_{ji} and Z_{jj} are the off-diagonal and diagonal elements of the impedance matrices, and α_L is the set of consumer nodes.

One way of determining L is:

$$L = \max_{j \in \alpha_L} \left|1 - \frac{\sum_{i \in \alpha_G} F_{ij} \cdot V_i}{V_j}\right| \tag{6.4.6}$$

where α_L is the set of load buses; α_G is the set of generator buses. V_j is the voltage at load bus j; V_i is the complex voltage at generator bus i; F_{ij} is the element of matrix $[F]$ determined by

$$[F] = -\frac{[Y_{LL}]}{[Y_{LG}]} \tag{6.4.7}$$

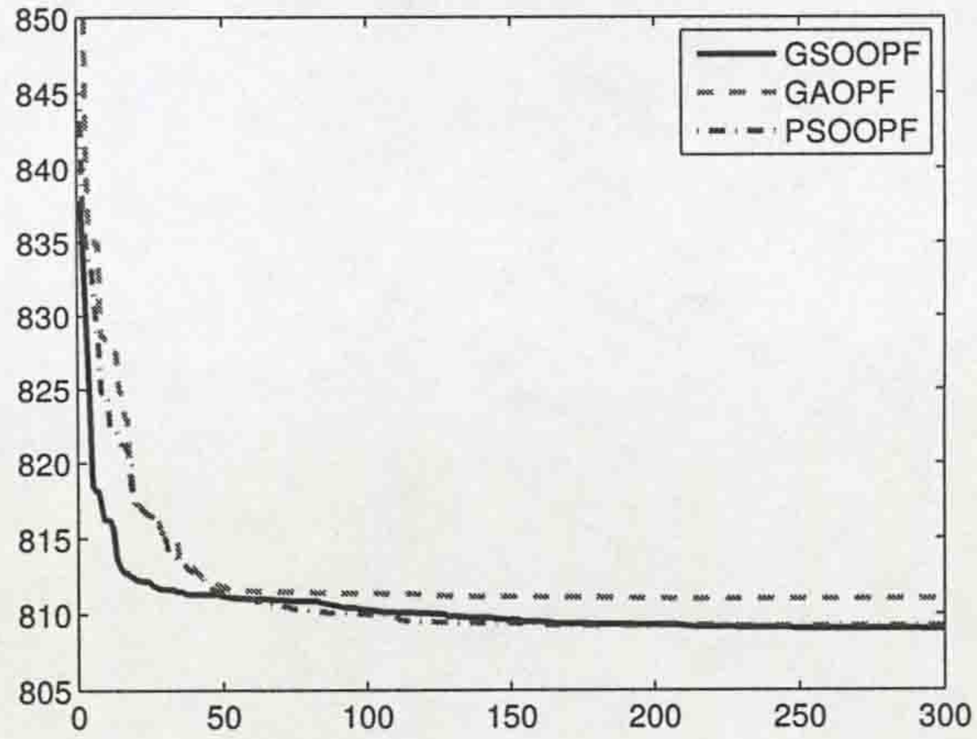


Figure 6.5: Search process of GSOOPF for Case 3

where $[Y_{LL}]$ and $[Y_{LG}]$ are sub-matrices of the Y-bus matrix.

The objective function can be expressed as:

$$J = \sum_{i=1}^{N_G} f_i + \omega L_{\max}$$

The optimal control variables of the GSOOPF for this case from 100 runs is tabulated in Table 6.2. The best results of the GSOOPF and PSOPCOPF obtained from 100 runs are tabulated in Table 6.4. The search processes of our GSOOPF algorithm and other 2 algorithms are shown in Fig. 6.5. In this case, the three indices, fuel case, voltage profile and voltage stability have been minimized by GSOOPF and PSOPCOPF and they are much smaller than that obtained by GAOPF and PSOOPF.

Table 6.4: The best values of GSOOPF for Case 3

	GSOOPF	PSOPCOPF	GAOPF	PSOOPF
Fuel cost ($/h)	**802.0520**	802.0638	803.9216	802.1190
$\sum$ voltage deviations	**0.8191**	0.8232	0.8877	0.8567
$\boldsymbol{L_{\max}}$	0.1381	**0.1379**	0.1434	0.1382

Table 6.5: Best values of GSOOPF GAOPF and PSOOPF for the IEEE 118-bus system

	GSOOPF	GAOPF	PSOOPF
Fuel cost ($/h)	**15863.3475**	18981.6442	16012.3433

6.4.2 IEEE 118-Bus system

From Chapter 2, we can see the GSO algorithm is capable of handling high-dimensional optimisation problems. This feature makes it possible to solve practical optimal flow problems. We employ the IEEE 118-bus systems [183] to evaluate the performance of GSOOPF. The IEEE 118-bus system consists of 181 transmission elements, 17 generators for AVR control, 9 transformers with off-nominal tap ratio, and 14 shunt admittances. The number of total control variables is 130. We tabulated the results from GSOOPF, GAOPF and PSOOPF algorithms in Table 6.5, which shows that GSOOPF is able to obtain a better optimisation result in comparison with the others.

6.5 Conclusions

In this chapter, we have applied GSO and PSOPC to tackle OPF problems. These two new approaches utilizes the superior global searching ability of GSO and PSOPC. Numerical experiments were carried out on an IEEE 30-bus for three different OPF problems which include minimizing the fuel cost, improving the voltage profile and enhancing the voltage stability. We have also employed a practical IEEE 118-bus system to evaluate the GSOOPF algorithm. Our algorithm provides better results than those obtained from the other optimisation techniques in terms of accuracy and convergence speed.

Chapter 7

Conclusions

7.1 Introduction

This chapter concludes the thesis and summarises the major achievements of the work in the multi-disciplinary research between swarm intelligence and animal behaviour. Suggestions for future research are listed at the end.

7.2 Summary of Results

This study primarily aims at developing Animal Behaviour inspiration Optimisation (ABO) algorithms by transferring knowledge from the research of animal behaviour. As results, a novel ABO algorithm, GSO, has been developed. The study has also improved standard PSO with an animal congregation model: passive congregation. The standard PSO has also been extended to handle mix-variable constrained optimisation problems. Besides of the development of ABO algorithms, the ABO algorithms developed in this study has also been successfully applied to real-world problems.

In the preceding chapters, the following work and results were presented.

The background of animal behaviour was given in Chapter 1. Then Natural Computation and Swarm Intelligence (SI) were introduced, followed by a discussion on the relationship between SI, self-organisation and animal behaviour.

The chapter went on to proposed a new definition of SI and the definition of ABO. Details of Particle Swarm Opitimiser was given. The motivation behind this study was discussed in this chapter. The outline and major contributions of this thesis were also presented.

The first part of this thesis, *e.g.*, Chapters 2 and 3, devotes to the algorithm developments of ABO. In Chapter 2, a novel ABO algorithm, Group Search Optimiser (GSO), developed in this study was introduced. The inspiration behind this algorithm, animal group searching behaviour, was explained in details. Then the Producer-Scrounger model, which is a generic animal social foraging model, was presented. Once the theoretical foundation has been laid, the details of the GSO algorithm were given. A large set of benchmark functions were employed to evaluate the performance of the GSO algorithm. From the obtained results, the performance of the GSO algorithm is seen to greatly outperform other EAs and PSO on multi-modal functions while remaining similar performance on uni-modal functions. Finally in this chapter, discussion of the differences between GSO and other EAs and SI algorithms was also presented.

Chapter 3 described an improved PSO algorithm with passive congregation (PSOPC). An animal congregation model, passive congregation, which is an important biological force preserving swarm integrity, was introduced to the standard PSO algorithm. A set of 10 benchmark functions were used to evaluate the performance of the PSOPC algorithm. In comparison to the other standard PSO variants, the search performance of the PSOPC algorithms is better in terms of accuracy and convergence speed.

In order to evaluate the performance of GSO, in Chapter 4, the GSO algorithm has been applied to train ANNs. This chapter begins with a brief introduction to ANNs, especially evolutionary ANNs, followed by the details of the 3-layer feed-forward ANN used in the study. The training algorithm, GSOANN, then was presented. A set of machine learning benchmark problems, including 4 classification problems and 1 forecasting problem, were employed to access the performance of the proposed training algorithm. Among these

---

problems, the proposed GSOANN achieved the best results on the Wisconsin breast cancer diagnosis problem and the sun spot forecasting problem in the literature. For the rest problems, the GSOANN algorithm also achieved satisfactory results compared with other sophisticated ANN training algorithm.

The PSO algorithm has also been extended to handle mixed-variables and constrains in order to solve real-world engineering optimisation problems. In Chapter 5, a simple truncation scheme was introduced to the standard PSO algorithm to handle mixed variables. In order to deal with problem specific constrains, a simple so called "fly-back mechanism" was employed. Then the extended PSO algorithms was applied to solve mechanical design optimisation problems. Four mechanical design problems, typically employed by the literature as benchmark functions were solved successfully by the extended PSO algorithm. The results obtained are better than many other algorithms and many results are the best in the literature.

Chapter 6 begins with a brief introduction to Optimal Power Flow (OPF) problems followed by the formulation of OPF problems. Then the PSOPC and GSO algorithms were employed to solved the OPF problems on an IEEE 30-bus test system with 3 different cases which minimize fuel cost, improve voltage profile and enhance voltage stability, respectively. An practical IEEE 118-bus system was also employed to evaluate the performance of the GSO algorithm. The results obtained by the two algorithms were compared to the standard PSO, GA and the results obtained from the current literature. The comparison verifies the superior search performance of the two algorithms on real-world optimisation problems.

Conclusively, from the successful developments of the two novel ABO algorithms, this thesis demonstrates the power of multi-disciplinary study in optimisation and animal behaviour: knowledge from animal behaviour can provide new thinking to to solve optimisation problems. The thesis also demonstrates the outstanding performance of ABO algorithms by solving real-world problems.

Shan He

7.3 Suggestions for Future Work

In this section, we list several points that deserve further investigations to develop and improve the algorithms and applications described in this thesis.

1. From the discussion in Chapter 1, the most important feature which distinguishes SI from other novel computing paradigms is self-organisation. The GSO algorithm also displays some self-organising features, for example, the complex searching process of a global optimum is achieved by a population of simple agents interact using three simple searching strategies: producing, scrounging and ranging. However, these features are not sufficient enough to characterise GSO as a self-organising (swarm) optimisation algorithm, *e.g.*, similar to the global PSO algorithm, the selection of the producer does not emerge from local interactions between members. Further work need to be done to develop GSO as a SI algorithm.

2. From the experiments, we found that although the computational time required by the ABO algorithms, *e.g.*, PSOPC and GSO, is less or similar to other Natural computational algorithms, *e.g.*, EAs, compared with traditional gradient-based optimisation algorithms, they are still too slow, which might hamper their applications to some large-scale real-world problems. Parallelisation of these algorithms is worthy to be investigated to overcome this drawback.

3. Coevolution, especially cooperative coevolution has been incorporated into EAs to improved their performance. It is also interesting to investigate multi-group GSO with cooperative strategies. The research in animal cooperation can be incorporated into the GSO framework naturally. Combine with parallelisation, it is expected that this future work can not only improve the speed but also the search performance.

4. As discussed in the previous chapters, the GSO algorithm is similar to the memetic algorithms. It will be interesting to incorporate other local

search strategies for the producer to improve the performance of GSO on a certain set of problems.

Appendix A

Global Optimisation Benchmark Functions

Table A.1: The 23 benchmark functions, where n is the dimension of the function, S is the feasible search space, and $f_{\min}$ is the global minimum value of the function.

	Test function	n	S	$f_{\min}$
$f_1(x)$	Sphere Model	30	$[-100, 100]^n$	0
$f_2(x)$	Schwefel's Problem 2.22	30	$[-10, 10]^n$	0
$f_3(x)$	Schwefel's Problem 1.2	30	$[-100, 100]^n$	0
$f_4(x)$	Schwefel's Problem 2.21	30	$[-100, 100]^n$	0
$f_5(x)$	Generalized Rosenbrock's Function	30	$[-30, 30]^n$	0
$f_6(x)$	Step Function	30	$[-100, 100]^n$	0
$f_7(x)$	Quartic Function with Noise	30	$[-1.28, 1.28]^n$	0
$f_8(x)$	Generalized Schwefel's Problem 2.26	30	$[-500, 500]^n$	-12569.5
$f_9(x)$	Generalized Rastrigin's Function	30	$[-5.12, 5.12]^n$	0
$f_{10}(x)$	Ackley's Function	30	$[-32, 32]^n$	0
$f_{11}(x)$	Generalized Griewank Function	30	$[-600, 600]^n$	0
$f_{12}(x)$	Generalized Penalized Function 1	30	$[-50, 50]^n$	0
$f_{13}(x)$	Generalized Penalized Function 2	30	$[-50, 50]^n$	0
$f_{14}(x)$	Shekel's Foxholes Function	2	$[-65.536, 65.536]^n$	1
$f_{15}(x)$	Kowalik's Function	4	$[-5, 5]^n$	0.0003075
$f_{16}(x)$	Six-hump Camel-Back Function	2	$[-5, 5]^n$	-1.0316285
$f_{17}(x)$	Branin Function	2	$[-5, 10] \times [0, 15]$	0.398
$f_{18}(x)$	Goldstein-Price Function	2	$[-2, 2]^n$	3
$f_{19}(x)$	Hartman's Function 1	3	$[0, 1]^n$	-3.86
$f_{20}(x)$	Hartman's Function 2	6	$[0, 1]^n$	-3.32
$f_{21}(x)$	Shekel's Family 1	4	$[0, 10]^n$	-10
$f_{22}(x)$	Shekel's Family 2	4	$[0, 10]^n$	-10
$f_{23}(x)$	Shekel's Family 3	4	$[0, 10]^n$	-10

Sphere Model:

$$f_1(x) = \sum_{i=1}^{30} x_i^2$$

Schwefel's Problem 2.22:

$$f_2(x) = \sum_{i=1}^{30} |x_i| + \prod_{i=1}^{30} |x_i|$$

Schwefel's Problem 1.2:

$$f_3(x) = \sum_{i=1}^{30} \left(\sum_{j=1}^{i} x_j \right)^2$$

Schwefel's Problem 2.21:

$$f_4(x) = \max_i \{|x_i|, 1 \leq i \leq 30\}$$

Generalized Rosenbrock's Function:

$$f_5(x) = \sum_{i=1}^{29} (100(x_{i+1} - x_i^2)^2 + (x_i - 1))^2$$

Step Function:

$$f_6(x) = \sum_{i=1}^{30} (\lfloor x_i + 0.5 \rfloor)^2$$

Quartic Function with Noise:

$$f_7(x) = \sum_{i=1}^{30} i x_i^4 + \text{random}[0, 1)$$

Shan He

Generalized Schwefel's Problem 2.26:

$$f_8(x) = -\sum_{i=1}^{30}\left(x_i \sin\left(\sqrt{|x_i|}\right)\right)$$

Generalized Rastrigin's Function:

$$f_9(x) = \sum_{i=1}^{30}(x_i^2 - 10\cos(2\pi x_i) + 10)^2$$

Ackley's Function:

$$f_{10}(x) = -20\exp\left(-0.2\sqrt{\frac{1}{30}\sum_{i=1}^{30}x_i^2}\right) - \exp\left(\frac{1}{30}\sum_{i=1}^{30}\cos 2\pi x_i\right) + 20 + e$$

Generalized Griewank Function:

$$f_{11}(x) = \frac{1}{4000}\sum_{i=1}^{30}(x_i - 100)^2 - \prod_{i=1}^{30}\cos(\frac{x_i - 100}{\sqrt{i}}) + 1$$

Generalized Penalized Functions:

$$\begin{aligned} f_{12} &= \frac{\pi}{30}\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{29}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_n - 1)^2\right\} \\ &+ \sum_{i=1}^{30}u(x_i, 10, 100, 4) \end{aligned}$$

and

$$\begin{aligned} f_{13} &= 0.1\left\{\sin^2(\pi 3x_1) + \sum_{i=1}^{29}(x_i - 1)^2[1 + \sin^2(3\pi x_{i+1})]\right. \\ &+ \left.(x_n - 1)^2[1 + \sin^2(2\pi x_{30})]\right\}\sum_{i=1}^{30}u(x_i, 5, 100, 4) \end{aligned}$$

Shan He

where

$$u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a \\ 0, & -a \leq x_i \leq a \\ k(-x_i - a)^m, & x_i < -a \end{cases}$$

$$y_i = 1 + \frac{1}{4}(x_i + 1)$$

Shekel's Foxholes Function:

$$f_{14}(x) = \left[\frac{1}{500} + \sum_{j=1}^{25} \frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right]^{-1}$$

$$\text{where } (a_{ij}) = \begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \cdots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \cdots & 32 & 32 & 32 \end{pmatrix}$$

Kowalik's Function:

$$f_{15}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1(b_i^2 + b_i x_2)}{b_i^2 + b_i x_3 + x_4}\right]^2$$

Six-hump Camel-Back Function:

$$f_{16}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

Branin Function:

$$\begin{aligned} f_{17}(x) &= \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos x_1 + 10 \\ &\quad -5 \leq x_1 \leq 10,\ 0 \leq x_2 \leq 15 \\ x_{\min} &= (-3.142, 12.275),\ (3.142, 2.275),\ (9.425, 2.425) \\ \min(f_{17}) &= 0.398 \end{aligned}$$

Table A.2: Kowalik's Function f_{15}

i	a_i	b_i^{-1}
1	0.1957	0.25
2	0.1947	0.5
3	0.1735	1
4	0.1600	2
5	0.0844	4
6	0.0627	6
7	0.0456	8
8	0.0342	10
9	0.0323	12
10	0.0235	14
11	0.0246	16

Goldstein-Price Function:

$$f_{18} = [1 + (x_1 + x_2 + 1)^2(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)]$$
$$\times[30 + (2x_1 - 3x_2)^2(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]$$

$$-2 \leq x_i \leq 2 \quad \min(f_{18}) = f_{18}(0, -1) = 3$$

Hartman's Function:

$$f(x) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{n} a_{ij}(x_j - p_{ij})^2\right]$$

with n=3,6 for $f_{19}(x)$ and $f_{20}(x)$, respectively. The coefficients are defined by Tables and , respectively.

$$0 \leq x_j \leq 1$$

$$\min(f_{19}) = f_{19}(0.114, 0.556, 0.852) = -3.86$$

$$\min(f_{20}) = f_{20}(0.201, 0.150, 0.477, 0.275, 0.311, 0.657) = -3.32$$

Table A.3: Hartman's Function f_{19}

i	$a_{ij},\ j=1,2,3$			c_i	$p_{ij},\ j=1,2,3$		
1	3	10	30	1	0.3689	0.1170	0.2673
2	0.1	10	35	1.2	0.4699	0.4387	0.7470
3	3	10	30	3	0.1091	0.8732	0.5547
4	0.1	10	35	3.2	0.038150	0.5743	0.8828

Table A.4: Hartman's Function f_{20}

i	$a_{ij},\ j=1,2,3,4,5,6$						c_i	$p_{ij},\ j=1,2,3,4,5,6$					
1	10	3	17	3.5	1.7	8	1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.05	10	17	0.1	8	14	1.2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	3	3.5	1.7	10	17	8	3	0.2348	0.1415	0.3522	0.2883	0.3047	0.6650
4	17	8	0.05	10	0.1	14	3.2	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

Shekel's Family:

$$f(x) = -\sum_{i=1}^{m}[(x-a_i)(x-a_i)^T + c_i]^{-1}$$

with $m = 5, 7$ and 10 for $f_{21}(x)$, $f_{22}(x)$ and $f_{23}(x)$, respectively. $0 \leq x_j \leq 10$. $x_{\text{local-opt}} \simeq a_i$ and $\min(f_{x_{\text{local-opt}}}) \simeq 1/c_i$ for $1 \leq i \leq m$.

Table A.5: Shekel's Family f_{21},f_{22},f_{23}

i	a_{ij}, $j = 1,2,3,4$				c_i
1	4	4	4	4	0.1
2	1	1	1	1	0.2
3	8	8	8	8	0.2
4	6	6	6	6	0.4
5	3	7	3	7	0.4
6	2	9	2	9	0.1
7	5	5	3	3	0.1
8	8	1	8	1	0.1
9	6	2	6	2	0.1
10	7	3.6	7	3.6	0.1

References

[1] J. W. Bell. *Searching Behaviour - The Behavioural Ecology of Finding Resources.* Chapman and Hall Animal Behaviour Series. Chapman and Hall, 1990.

[2] W. Ongsakul and T. Tantimaporn. Optimal power flow by improved evolutionary programming. *Electric Power Components and Systems*, 34(1):79–95, Jan. 2006.

[3] A.G. Bakirtzis, P.N. Biskas, C.E. Zoumas, and V. Petridis. Optimal power flow by enhanced genetic algorithm. *IEEE Transactions on Power Systems*, 17(2):229–236, MAY 2002.

[4] N. Tinbergen. On aims and methods of ethology. *Z Tierpsychol*, 20:410–433, 1963.

[5] Wikipedia. Comparative psychology — wikipedia, the free encyclopedia, 2005. [Online; accessed 21-JULY-2006].

[6] J. R. Krebs and N. Davies. *An Introduction to Behavioural Ecology.* Blackwell publishing, third edition edition, 2004.

[7] D. McFarland. *Animal Behaviour.* Longman, third edition edition, 1999.

[8] J. R. Krebs and N. Davies. *Behavioural Ecology: an evolutionary approach.* Blackwell publishing, fourth edition edition, 1997.

[9] G. A. Parker and J. Maynard Smith. Optimality theory in evolutionary biology. *Nature*, 348(1):27–33, Nov. 1990.

[10] R. A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2:14–23, 1985.

[11] R. C. Arkin. *Behavior-Based Robotics.* MIT Press, 1998.

[12] Wikipedia. Natural computation — wikipedia, the free encyclopedia, 2006. [Online; accessed 23-JULY-2006].

[13] H. Lipson and J. B. Pollack. Automatic design and manufacture of robotic lifeforms. *Nature*, 406:974–978, August 2000.

[14] D. B. Fogel. The advantages of evolutionary computation. In D. Lundh, B. Olsson, and A. Narayanan, editors, *Bio-Computing and Emergent Computation 1997*, pages 1–11. World Scientific Press, 1997.

[15] G. Beni and J. Wang. Swarm intelligence. In *Seventh Annual Meeting of the Robotics Society of Japan*, pages 425–428, Tokio, Japan, 1989. RSJ press.

[16] Wikipedia. Swarm intelligence — wikipedia, the free encyclopedia, 2005. [Online; accessed 21-JULY-2006].

[17] E. Bonabeau, M. Dorigo, and G. Theraulaz. *Swarm Intelligence: From Natural to Artificial Systems.* Oxford University Press, 1999.

[18] E. Bonabeau and C. Meyer. Swarm intelligence: A whole new way to think about business. *Harvard Business Review*, 79:106–114, 2001.

[19] Wikipedia. Self-organization — wikipedia, the free encyclopedia, 2006. [Online; accessed 21-Sep-2006].

[20] Nigel R. Franks James Sneyd Guy Theraulaz Scott Camazine, Jean-Louis Deneubourg and Eric Bonabeau. *Self-Organization in Biological Systems.* Princeton University Press, 2003.

[21] J. Kennedy, R. C. Eberhart, and Y. H. Shi. *Swarm Intelligence.* Morgan Kaufmann Publishers, 2001.

---

[22] Self-organizing systems (sos) faq.

[23] D. Whitley. Cellular genetic algorithms. In S. Forrest, editor, *Proceedings of the 5th ICGA*, volume 658. Morgan-Kaufmann, CA, 1993.

[24] J.M. Bishop. Stochastic searching networks. In *Proc. 1st IEE Conf. on Artificial Neural Networks*, pages 329–331, London, 1989.

[25] K. C. Tsui and J. Liu. Evolutionary diffusion optimization, part 1: Description of the algorithm (cec2002). In X. Yao, editor, *Proc. Congr. Evolutionary Computation*, pages 169–174, 2002.

[26] A. Linhares. Synthesizing a predatory search strategy for vlsi layouts. *IEEE Trans. on Evolutionary Computation*, 3(2):147–152, 1999.

[27] T. Ray and K. M. Liew. Society and civilization: An optimization algorithm based on the simulation of social behavior. *IEEE Trans. on Evolutionary Computation*, 7(4):386–396, Aug. 2003.

[28] S. D. Muller, J. Marchetto, S. Airaghi, and P. Koumoutsakos. Optimization based on bacterial chemotaxis. *IEEE Trans. on Evolutionary Computation*, 6(1):16–29, Feb. 2002.

[29] M. Laumanns, G. Rudolph, and H. P. Schwefel. A spatial predator-prey approach to multi-objective optimization: A preliminary study. In *LECTURE NOTES IN COMPUTER SCIENCE*, volume 1498 of *PARALLEL PROBLEM SOLVING FROM NATURE - PPSN V*, pages 241–249, 1998.

[30] J. K Parrish and W. M. Hamner. *Animal Groups in Three Dimensions.* Cambridge University Press, Cambridge, UK, 1997.

[31] D.G. Kerlick. Moving iconic objects in scientific visualization. In *Proc First 90 IEEE Conf Visualization Visualization 90*, pages 124–130, Oct. 1990.

Shan He

[32] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *IEEE international Conference on Neural Networks*, volume 4, pages 1942–1948. IEEE Press, 1995.

[33] Y. Shi and R. C. Eberhart. A modified particle swarm optimiser. In *Proc. IEEE Inc. Conf. on Evolutionary Computation*, pages 303–308, 1997.

[34] Y. Shi and R. C. Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII (1998), Lecture Notes in Computer Scdience 1447*, pages 591–600. Springer, 1998.

[35] R. C. Eberhart and Y. Shi. Particle swarm optimization: developments, applications and resources. In *Proc. IEEE Int. Conf. on Evolutionary Computation*, pages 81–86, 2001.

[36] M. Clerc and J. Kennedy. The particle swarm: Explosion, stability, and convergence in a multi-dimensional complex space. *IEEE Trans. on Evolutionary Computation*, 6(1):58–73, 2002.

[37] R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proc. of the Congress on Evolutionary Computation (CEC2000)*, pages 84–88, 2000.

[38] F. van den Bergh and A.P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):225– 239, 2004.

[39] A. Ratnaweera, S.K. Halgamuge, and H.C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 8(3):240 – 255, 2004.

[40] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.

[41] C. A. C. Coello, G. T. Pulido, and M. S. Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, 2004.

[42] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi. A particle swarm optimization for reactive power and voltage control considering voltage security assessment. *IEEE Transactions on Power Systems*, 15(4):1232–1239, 2000.

[43] M. A. Abido. Optimal power flow using particle swarm optimization. *International Journal of Electrical Power and Energy*, 24(7):563–571, October 2002.

[44] S. He, J. Y. Wen, E. Prempain, Q. H. Wu, J. Fitch, and S. Mann. An improved particle swarm optimization for optimal power flow. In *2004 International Conference on Power System Technology*, Nov. 2004.

[45] Z. L. Gaing. Particle swarm optimization to solving the economic dispatch considering the generator constraints. *IEEE Transactions on Power Systems*, 18(3):1187–1195, 2003.

[46] I. N. Kassabalidis, M. A. El-Sharkawi, R. J. I. Marks, L. S. Moulin, and A. P. Alves da Silva. Dynamic security border identification using enhanced particle swarm optimization. *IEEE Transactions on Power Systems*, 17(3):723–729, 2002.

[47] S. Naka, T. Genji, T. Yura, and Y. Fukuyama. A hybrid particle swarm optimization for distribution state estimation. *IEEE Transactions on Power Systems*, 18(1):60–68, 2003.

[48] N. Higashi and H. Iba. Particle swarm optimization with gaussian mutation. In *Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE*, pages 72–79, April 2003.

[49] A. Stacey, M. Jancic, and I. Grundy. Particle swarm optimization with mutation. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 2, pages 1425–1430, Dec. 2003.

[50] T. Krink and M. Løvbjerg. The lifecycle model: Combining particle swarm optimisation, genetic algorithms and hillclimbers. In *Proceedings of Parallel Problem Solving from Nature VII (PPSN-2002)*, pages 621–630, April 2002.

[51] J. M. McNamara, A. I. Houston, and E. J. Collins. Optimality models in behavioral biology. *SIAM Review*, 43(3):413–466, 2001.

[52] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report 826, California Institute of Technology, Pasadena, California, USA, 1989.

[53] G. M. Morris, D. S. Goodsell, R. S. Halliday, R. Huey, W. E. Hart, R. K. Belew, and A. J. Olson. Automated docking using a lamarkian genetic algorithm and an empirical binding free energy function. *J Comp Chem*, 14:1639–1662, 1998.

[54] Wikipedia. Lamarckism — wikipedia, the free encyclopedia, 2006. [Online; accessed 21-JULY-2006].

[55] J. G. Vlachogiannis. Constricted local-neighborhood particle swarm optimization with passive congregation applied in reactive power and voltage control. *ELECTRIC POWER COMPONENTS AND SYSTEMS*, 34(5):509–520, May 2006.

[56] E. Bonabeau, M. Dorigo, and G. Theraulza. Inspiration for optimization from social insect behaviour. *Nature*, 406:39–42, July 2000.

[57] C. J. Barnard and R. M. Sibly. Producers and scroungers: a general model and its application to captive flocks of house sparrows. *Animal Behaviour*, 29:543–550, 1981.

---

[58] X. Yao, Y. Liu, and G. Liu. Evolutionary programming made faster. *IEEE Trans. on Evolutionary Computation*, 3(2):82–102, 1999.

[59] X. Yao and Y. Liu. Fast evolution strategies. In P. J. Angeline, R. G. Reynolds, J. R. McDonnell, and R. Eberhart, editors, *Evolutionary Programming VI*, pages 151–161, Berlin, 1997. Springer.

[60] G. M. Viswanathan, S. V. Buldyrev, S. Havlin, M. G. da Luz, E. Raposo, and H. E. Stanley. Optimizing the success of random searches. *Nature*, 401(911-914), 1999.

[61] H. R. Pulliam and G. E. Millikan. Social organization in the non-reproductive season. *Animal Behaviour*, 6(169-197), 1983.

[62] J. Brockmann and C. J. Barnard. Kleptoparasitism in birds. *Animal Behaviour*, 27(546-555), 1979.

[63] L-A. Giraldeau and G. Beauchamp. Food exploitation: searching for the optimal joining policy. *Trends in Ecology & Evolution*, 14(3):102–106, March 1999.

[64] C. W. Clark and M. Mangel. Foraging and flocking strategies: information in an uncertain environment. *Am. Nat.*, 123:626–641, 1984.

[65] I.D. Couzin, J. Krause, N.R. Franks, and S.A. Levin. Effective leadership and decision-making in animal groups on the move. *Nature*, 434:513–516, Feb. 2005.

[66] L-A. Giraldeau and L. Lefebvre. Exchangeable producer and scrounger roles in a captive flock of feral pigeons - a case for the skill pool effect. *Animal Behaviour*, 34(3):797–803, Jun 1986.

[67] R. H. S. Carpenter. *Eye Movements.* Macmilan, London, 1991.

[68] S. P. Liversedge and J. M. Findley. Saccadic eye movements and cognition. *Trends in Cognitive Sciences*, 4:6–14, 2000.

---

[69] J. Najemnik and W. S. Geisler. Optimal eye movement strategies in visual search. *Nature*, 434:387–391, March 2005.

[70] T. Caraco. Time budgeting and group size: a test of theory. *Ecology*, 60:618–627, 1979.

[71] D. G. C. Harper. Competitive foraging in mallards: 'ideal free' ducks. *Animal Behaviour*, 30:575–584, 1988.

[72] T. H. Waterman. *Animal Navigation.* Scientific American Library, NY., 1989.

[73] D. B. Dusenbery. Ranging strategies. *Journal of Theoretical Biology*, 136:309–316, 1989.

[74] W. J. O'Brien, B. I. Evans, and G. L. Howick. A new view of the predation cycle of a planktivorous fish, white crappie (pomoxis annularis). *Can. J. Fish. Aquat. Sci.*, 43:1894–1899, 1986.

[75] C. L. Higgins and R. E. Strauss. Discrimination and classification of foraging paths produced by search-tactic models. *Behavioral Ecology*, 15(2):248–254, 2003.

[76] A. F. G. Dixon. An experimental study of the searching behaviour of the predatory coccinellid beetle adalia decempunctata. *J. Anim. Ecol.*, 28:259–281, 1959.

[77] D. H. Wolpert and W. G. Macready. No free lunch theorems for search. *IEEE Trans. on Evolutionary Computation*, 1(1):67–82, 1997.

[78] J. H. Holland. *Adaption in Natural and Artificial Systems.* Ann Arbor, 1975.

[79] Lawrence J. Fogel, Alvin J. Owens, and Michael J. Walsh. Artificial intelligence through a simulation of evolution. In M. Maxfield, A. Callahan, and L. J. Fogel, editors, *Biophysics and Cybernetic Systems: Proc. of the*

2nd Cybernetic Sciences Symposium, pages 131–155, Washington, D.C., 1965. Spartan Books.

[80] D. B. Fogel. *Evolutionary computation: toward a new philosophy of machine intelligence.* IEEE Press, New York, 1995.

[81] H-P Schwefel. *Evolution and optimum seeking.* Wiley, New York, 1995.

[82] B. Birge. Psot - a particle swarm optimization for use with matlab. In *SIS '03. Proceedings of the 2003 IEEE*, Swarm Intelligence Symposium, pages 182–186, April 2003.

[83] J. Biethahn and V. Nissen. *Evolutionary Algorithms in Management Applications.* Springer-Verlag, Berlin, 1995.

[84] D. C. Montgomery. *Statistical Quality Control.* Wiley, New York, 1996.

[85] M. Locatelli. A note on the Griewank test function. *Journal of Global Optimization*, 25(2):169–174, 2003.

[86] J. Nocedal. Updating quasi-newton matrices with limited storage. *Math.Comput.*, 35:773–782, 1980.

[87] J. B. Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[88] J. Barhen, V. Protopopescu, and D. Reister. Trust: A deterministic algorithm for global optimization. *Science*, 276:1094–1097, May 1997.

[89] X. Yao and Y. Liu. Scaling up evolutionary programming algorithms. In *Evolutionary Programming VII: Proc. of the Seventh Annual Conference on Evolutionary Programming (EP98)*, Lecture Notes in Computer Science, pages 103–112, Berlin, 1998. Springer-Verlag.

[90] Y. Liu, X. Yao, Q. Zhao, and T. Higuchi. Scaling up fast evolutionary programming with cooperative coevolution. In *Proceedings of the 2001 Congress on Evolutionary Computation*, pages 1101–1108, Piscataway, NJ, USA, 2001. IEEE Press.

[91] C. W. Reynolds. Flocks, herds and schools: a distributed behavioral model. *Computer Graphics*, 21(4):25–34, 1987.

[92] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In S. Krasner, editor, *The Ubiquity of Chaos*. AAAS Publications, Washington, DC., 1990.

[93] H. Mühlenbein and D. Schlierkamp-Voosen. Predictive models for the breeder genetic algorithm, I.: continuous parameter optimization. *Evolutionary Computation*, 1(1):25–49, 1993.

[94] P. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance difference. In *Proc. Of Evolutionary Programming conference, San Diago, USA*, 1998.

[95] M. Løbjerg, T. K. Rasmussen, and K. Krink. Hybrid particle swarm optimiser with breeding and subpopulations. In *In: Proceedings of the third Genetic and Evolutionary Computation Conference (GECCO-2001)*, volume 1, pages 469–476, 2001.

[96] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 1671–1676. IEEE Press, 2002.

[97] J. Kennedy. Stereotyping: improving particle swarm performance with cluster analysis. In *Pro. IEEE int. Conf. On Evolutionary Computation*, pages 1507–1512, 2000.

[98] Y. Shi and R. C. Eberhart. Fuzzy adaptive particle swarm optimization. In *Proc. IEEE Int. Conf. on Evolutionary Computation*, pages 101–106, 2001.

[99] A. Okubo. Diffusion and ecological problems: Methematical models. *Lecture Notes in Biomathematics*, 54:456–477, 1980.

[100] D. M. Gordon, R. E. Paul, and K. Thorpe. What is the function of encounter pattern in ant colonies? *Animal Behaviour*, (45):1083–1100, 1993.

[101] R. Hilborn. Modelling the stability of fish schools: exchange of individual fish between schools of skipjack tuna (katsuwonus pelamis). *Canadian Journal of Fisheries and Aquatic Sciences*, (48):1080–1091, 1991.

[102] W. D. Hamilton. Geometry for the selfish herd. *Journal of Theoretical Biology*, (31):295–311, 1971.

[103] T. J. Pitcher and J. K. Parrish. Functions of shoaling behaviour in teleosts. In T. J. Pitcher, editor, *Behaviour of Teleost Fishes*, pages 363–439. London: Chapman and Hall, 1993.

[104] A. E. Magurran and A. Higham. Information transfer across fish shoals under predator threat. *Ethology*, (78):153–158, 1988.

[105] R. D. Alexander. The evolution of social behaviour. *Annual Review of Ecology and Systematics*, (5):325–383, 1974.

[106] J. L. Deneubourg, S. Goss, N. Franks, A. Sendova-Franks, C. Detrain, and L. Chretien. The dynamics of collective sorting: robot like ant and ant-like robot. In *Proc. First Conference on Simulation of Adaptive Behavior: From Animals to Animals*, pages 356–365, 1991.

[107] E. Lumer and B. Faieta. Diversity and adaptation in population of clustering ants. In *Proceedings Third International Conference on Simulation of Adaptive Behavior: From Animals to Animals*, pages 499–508, 1994.

[108] K. Chellapilla. Combining mutation operators in evolutionary programming. *IEEE Trans. on Evolutionary Computation*, 2(3):91–96, Sept. 1998.

[109] D. B. Fogel. *System Identification Through Simulated Evolution: A machine Learning Approach to modeling.* Ginn Press, 160 Gould Street, Needham Heights, MA 01294, 1991.

Shan He

[110] J. E. Fieldsend and S. Singh. A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence. In *Proceedings of UK Workshop on Computational Intelligence (UKCI'02)*, pages 37–44, 2002.

[111] S. B. Thrun, J. Bala, E. Bloedorn, I. Bratko, B. Cestnik, J. Cheng, K. De Jong, S. Džeroski, S. E. Fahlman, D. Fisher, R. Hamann, K. Kaufman, S. Keller, I. Kononenko, J. Kreuziger, R. S. Michalski, T. Mitchell, P. Pachowicz, Y. Reich, H. Vafaie, W. Van de Welde, W. Wenzel, J. Wnek, and J. Zhang. The MONK's problems: A performance comparison of different learning algorithms. Technical Report CS-91-197, Pittsburgh, PA, 1991.

[112] Q. H. Wu, B. W. Hogg, and G. W. Irwin. A neural network regulator for turbogenerators. *IEEE Trans. on Neural Networks*, 3(1):95–100, 1992.

[113] X. Yao. Evolving artificial neural networks. *Proceeding of the IEEE*, 87(9):1423–1447, Sep. 1999.

[114] D. J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. In *Proc. Eveventh Int. Joint Conf. Artificial Intelligence*, pages 762–767, San Mateo, CA, 1989.

[115] T. P. Caudell and C. P. Dolan. Parametric connectivity: Training of constrained networks using genetic genetic algorithms. In J. D. Schaffer, editor, *Pro. Third Int. Conf. Genetic Algorithms*, pages 370–374, San Mateo, CA, 1989.

[116] D. B. Fogel, L. J. Fogel, and V. W. Porto. Evolving neural networks. *Biol. Cybern.*, 63:487–493, 1990.

[117] X. Yao and Y. Liu. A new evolutionary system for evolving artificial neural networks. *IEEE Trans. on Neural Networks*, 8(3):694–713, May 1997.

---

[118] F. H. F. Leung, H. K. Lam, S. H. Ling, and P. K. S. Tam. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. *IEEE TRANSACTIONS ON NEURAL NETWORKS*, 14(1):79–88, Jan. 2003.

[119] P. P. Palmes, T. Hayasaka, and S. Usui. Mutation-based genetic neural network. *IEEE Trans. on Neural Networks*, 16(3):587–600, MAY 2005.

[120] J. Ilonen, J. K. Kamarainen, and J. Lampinen. Differential evolution training algorithm for feed-forward neural networks. *Neural Processing Letters*, 17(1):93–105, 2003.

[121] E. Cantu-Paz and C. Kamath. An empirical comparison of combinations of evolutionary algorithms and neural networks for classification problems. *IEEE Transactions on Systems, Man, and Cybernetics-Part B: Cybernetics*, 35(5):915–927, 2005.

[122] E. Cantu-Paz and C. Kamath. Evolving neural networks to identify bent-double galaxies in the first survey. *Neural Networks*, 16(3-4):507–517, 2003.

[123] Y. Liu and X. Yao. Evolutionary ensembles with negative correlation learning. *IEEE Trans. on Evolutionary Computation*, 4(4):380–387, 2000.

[124] Z.H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: many could be better than all. *Artificial Intelligence*, 137(1-2):239–263, 2002.

[125] N. Garcia-Pedrajas, C. Hervas-Martinez, and J. Munoz-Perez. Covnet: a cooperative coevolutionary model for evolving artificial neural networks. *IEEE Trans. on Neural Networks*, 14(3):575–596, May 2003.

[126] N. Garcia-Pedrajas, C. Hervas-Martinez, and D. Ortiz-Boyer. Cooperative coevolution of artificial neural network ensembles for pattern classification. *IEEE Trans. on Evolutionary Computation*, 9(3):271–302, 2005.

---

[127] X. Yao and Y. Liu. Making use of population information in evolutionary artificial neural networks. *IEEE Trans. on Systems, Man and Cybernetics, Part B: Cybernetics*, 28(3):417–425, 1998.

[128] D. H. Wolpert. A mathematical theory of generalization. *Complex Systems*, 4(2):151–249, 1990.

[129] S. He, Q. H. Wu, and J. R. Saunders. Group search optimizer - an optimization algorithm inspired by animal searching behavior. *Submitted to IEEE Trans. on Evolutionary Computation.*

[130] M. Islam, X. Yao, and K. Murase. A constructive algorithm for training cooperative neural network ensembles. *IEEE Trans. on Neural Networks*, 14(4):820–834, 2003.

[131] S. Dzeroski and B. Zenko. Is combining classifiers with stacking better than selecting the best one? *Machine Learning*, 54(3):255–273, 2004.

[132] E. Cantu-Paz and C. Kamath. Inducing oblique decision trees with evolutionary algorithms. *IEEE Trans. on Evolutionary Computation*, 7(1):54–68, 2003.

[133] T. G. Dirtterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(12):139–157, 2000.

[134] K. Deb, A. Anand, and D. Joshi. A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, 10(4):371–395, 2002.

[135] T. Van Gestel, J. Suykens, B. Baesens, S. Viaene, J. Vanthienen, G. Dedene, B. De Moor, and J. Vandewalle. Benchmarking least squares support vector machine classifiers. *Machine Learning*, 54(1):5–32, 2004.

[136] L. Todorovski and S. Dzeroski. Combining classifiers with meta decision trees. *Machine Learning*, 50(3):223–249, 2003.

---

[137] C. A. Coello Coello. Theoretical and numerical constraint-handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191(11-12):1245–1287, January 2002.

[138] Q. H. Wu, Y. J. Cao, and J. Y. Wen. Optimal reactive power dispatch using an adaptive genetic algorithm. *Electrical Power and Energy Systems*, 20(8):563–569, 1998.

[139] Q. H. Wu and J. T. Ma. Power system optimal reactive power dispatch using evolutionary programming. *IEEE Trans on Power Syst.*, 10(3):1243–1249, 1995.

[140] Y. J. Cao and Q. H. Wu. A mixed variable evolutionary programming for optimisation of mechanical design. *International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications*, 7(2):77–82, 1999.

[141] K. Deb and M. Goyal. Optimizing engineering designs using a combined genetic search. In In Thomas Back, editor, *the Seventh International Conference on Genetic Algorithms*, pages 512–528, 1997.

[142] L. Davis. *Genetic Algorithms and Simulated Annealing.* Pitman, London, 1987.

[143] R. G. Le Riche, C. Knopf-Lenoir, and R. T. Haftka. A segregated genetic algorithm for constrained structural optimization. In *Sixth International Conference on Genetic Algorithms*, pages 558–565, University of Pittsburgh, 1995. Morgan Kaufmann.

[144] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evolutionary Computation*, 4(1):1–32, 1996.

[145] E. Sandgren. Nonlinear integer and discrete programming in mechanical design optimization. *Journal of Mechanical Design*, (112):223–229, 1990.

---

[146] P. Hajela and C. Shih. Multiobjective optimum design in mixed-integer and discrete design variable problems. *AIAA Journal*, 28(4):670–675, 1989.

[147] K. E. Parsopoulos and M. N. Vrahatis. Recent approaches to global optimization problems through particle swarm optimization. *Natural Computing*, (1):235–306, 2002.

[148] J. L. Chen and Y. C. Tsao. Optimal design of machine elements using genetic algorithms. *Journal of the Chinese Society of Mechanical Engineering*, 12(2):193–199, 1993.

[149] G. Thierauf and J. Cai. Evolution strategies - parallelisation and application in engineering optimization. In B. H. V. Topping, editor, *Parallel and Distributed Processing for Computational Mechanics*. Saxe-Coburg Publications, 1997.

[150] M. Tahk and B. C. Sun. Co-evolutionary augmented lagrangian methods for constrained optimization. *IEEE Transactions on Evolutionary Computation*, 4(2):114–124, July 2000.

[151] A. I. EI-Gallad, M. E. EI-Hawary, and A. A. Sallam. Swarming of intelligent particle for solving the nonlinear constrainted optimization problem. *International Journal of Engineering Intelligent Systems for Electrical Engineering and Communications*, 9(3):155–163, 2001.

[152] X. Hu and R. C. Eberhart. Solving constrained nonlinear optimization problems with particle swarm optimization. In *the Sixth World Multiconference on Systemics, Cybernetics and Informatics 2002 (SCI 2002)*, Orlando, USA, 2002.

[153] K.E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method for constrained optimization problems. In V. Kvasnicka J. Pospichal P. Sincak, J. Vascak, editor, *Intelligent Technologies - Theory and Applications: New Trends in Intelligent Technologies*, volume 76

of *Frontiers in Artificial Intelligence and Applications*, pages 214–220. IOS Press, 2002.

[154] T. Ray and K. M. Liew. A swarm metaphor for multiobjective design optimization. *Engineering Optimization*, 32(2):141–153, 2002.

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

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

[157] T. P. Runarsson and X. Yao. Stochastic ranking for constrained evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 4(3):284–294, 2000.

[158] S Koziel and Z. Michalewicz. Evolutionary algorithms, homomorphous mappings, and constrained parameter optimization. *Evolutionary Computation*, 7(1):19–44, 1999.

[159] J. Lampinen and I Zelinka. Mixed integer-discrete-continuous optimization by differential evolution. In *Proceedings of the 5th International Conference on Soft Computing*, pages 71–76, 1999.

[160] A. D. Belegundu. A study of mathematical programming methods for structural optimization. Technical report, University of Iowa, 1982.

[161] J. S. Arora. *Introduction to Optimun Design*. McGraw-Hill New York, 1989.

[162] C. A. Coello Coello. Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, 41(2):113–127, January 2000.

[163] T. Ray and K. M. Liew. Society and civilization: An optimization algorithm based on the simulation of social behavior. *IEEE Transactions on Evolutionary Computation*, 7(4):386–396, 2003.

---

[164] C. A. Coello Coello and E. Mezura Montes. Use of dominance-based tournament selection to handle constraints in genetic algorithms. In Joydeep Ghosh Mark J. Embrechts Okan Erson Cihan H. Dagli, Anna L. Buczak and Stephen Kercel, editors, *Intelligent Engineering Systems through Artificial Neural Networks (ANNIE'2001)*, volume 11, pages 177–182, St. Louis Missouri, July 2001. ASME Press.

[165] K. Deb. Geneas: A robust optimal design technique for mechanical component design. In Dipankar Dasgupta and Zbiginew Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications*, pages 497–514. Springer-Verlag, 1997.

[166] S. S. Rao. *Engineering Optimization.* John Wiley and Sons, 1996.

[167] K. M. Ragsdell and D. T. Phillips. Optimal design of a class of welded structure using geometric programming. *ASME Journal of Engineering for Industries*, 98(3):1021–1025, 1976.

[168] K. Deb. Optimal design of a welded beam via genetic algorithms. *AIAA journal*, 29(11):2013–2015, November 1991.

[169] K. Deb. An efficient constraint handling method for genetic algorithms. *Comput. Methods Appl. Mech. Eng.*, 186(2-4):311–338, 2000.

[170] J. N. Siddall. *Optimal Engineering Design.* Marcel Dekker, 1982.

[171] C. A. Coello Coello. Treating constraints as objectives for single-objective evolutionary optimization. *Engineering Optimization*, 32(3):275–308, February 2000.

[172] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* Wiley, New-York, 1968.

[173] A. E. Smith and D. W. Coit. Constraint handling techniques - penalty functions. In T. Back, D. B. Fogel, and Z. Michalewicz, editors, *Handbook*

of Evolutionary Computation. Oxford University Press and Institute of Physics Publishing, 1997.

[174] J. Carpentier. Contribution to the economic dispatch problem. *Bull. Soc. Franc. Elect.*, 8(3):431–447, 1962.

[175] J. A. Momoh, R. J. Koessler, M. S. Bond, B. Scott, D. Sun, A. Papalexopoulos, and P. Ristanovic. Challenges to optimal power flow. *IEEE Trans. Power Syst.*, 12:444–455, Feb. 1997.

[176] O. Alsac and B. Scott. Optimal load flow with steady state security. *IEEE Trans. on Power Appara. Syst.*, PAS-93:745–751, May-June 1974.

[177] G. F. Reid and L. Hasdorf. Economic dispatch using quadratic programming. *IEEE Trans. on Power Appara. Syst.*, PAS-92:2015–2023, 1973.

[178] B. Stott and E. Hobson. Power system security control calculation using linear programming. *IEEE Trans. on Power Appara. Syst.*, PAS-97:1713–1731, 1978.

[179] J. A. Momoh and J. Z. Zhu. Improved interior point method for opf problems. *IEEE Transactions on Power Systems*, 14:1114–1120, 1999.

[180] J. Yuryevich and K.P. Wong. Evolutionary programming based optimal power flow algorithm. *IEEE Transactions on Power Systems*, 14(4):1245–1250, 1999.

[181] S. He, Q. H. Wu, J. Y. Wen, J. R. Saunders, and R. C. Paton. A particle swarm optimizer with passive congregation. *BioSystems*, 78(1-3):135–147, Dec. 2004.

[182] P Kessel and H. Glavitch. Estimating the voltage stability ofa power system. *IEEE Transaction on Power Delivery*, 3(1):346–354, 1986.

[183] University of Washington. Power systems test case archive, 2006. http://www.ee.washington.edu/research/pstca/.
