Particle Swarms in Sizing and Global Optimization

by
Jaco F. Schutte

A dissertation submitted in partial fulfillment
of the requirements for the degree of

Master of Engineering

in the Department of Mechanical and Aeronautical Engineering,
University of Pretoria

December 2001

Supervisor:
Prof. Albert A. Groenwold

Abstract

Title: Particle Swarms in Sizing and Global Optimization

Author: Jaco. F. Schutte

Supervisor: Prof. A.A. Groenwold

Department: Department of Mechanical Engineering

Degree: Master of Engineering

Keywords: Particle swarm optimization, artificial life, derivative free, global optimization, structural optimization

In this work, the particle swarm optimization algorithm (PSOA) is implemented, evaluated and studied. A number of recently proposed variations on the PSOA are also considered. The algorithm and it's variants are applied to, firstly, an extended Dixon-Szegö bound constrained global test set, and secondly, the sizing design of truss structures.

Using the extended Dixon-Szegö test set, it is shown that the constriction variant as proposed by Clerc, and the dynamic inertia and maximum velocity reduction variant proposed by Fourie and Groenwold, represent the main contenders from a cost efficiency point of view.

In the interests of finding a reliable general purpose 'off-the-shelf' PSOA for global optimization, a parameter sensitivity study is then performed for the constriction and dynamic inertia and maximum velocity reduction variants. In doing so, it is shown that inclusion of dynamic inertia renders the PSOA relatively insensitive to the values of the cognitive and social scaling factors.

The constriction and dynamic inertia and maximum velocity reduction variants are then applied to the optimal sizing design of truss structures. While few results with the PSOA for constrained problems have previously been presented, a simple approach is proposed herein to accommodate the stress and displacement constraints during the initial stages of the swarm searches. Increased social (peer) pressure, at the cost of cognitive learning, is exerted on infeasible birds to increase their rate of migration to feasible regions.

Extensive numerical results are presented for the extended Dixon-Szegö test set, as well as a number of well known truss structures with dimensionality of up to 21. The results indicate the suitability of the gradient free PSOA for the two programming classes considered.

Opsomming

Titel: Partikel Swerms in Afmetingsontwerp en Globale Optimering

Auteur: Jaco. F. Schutte

Leier: Prof. A.A. Groenwold

Departement: Departement van Meganiese Ingenieurswese

Graad: Meester van Ingenieurswese

Sleutelwoorde: Partikel swerm optimimering, kunsmatige lewe, gradiëntlose metodes, globale optimering, strukturele optimering

In hierdie verhandeling word die partikel swerm optimeringsalgoritme (PSOA) geimplementeer, ge-evalueer en bestudeer. Verskeie onlangs voorgestelde variasies op die PSOA word ook beskou. Die algoritme en die variasies daarop word dan toegepas op, eerstens, 'n uitgebreide Dixon-Szegö stel rand begrensde globale toetsprobleme, en tweedens, die afmetingsontwerp van stangstrukture.

Met behulp van die uitgebreide Dixon-Szegö toetsprobleme word aangetoon dat die inkrimping variant, voorgestel deur Clerc, en die dinamiese momentum en maksimum snelheid vermindering variant voorgestel deur Fourie en Groenwold, die belangrikste mededingers is vanuit 'n oogpunt van koste effektiwiteit.

In belang van die formulering van 'n betroubare, algemeen toepasbare PSOA vir globale optimering, word 'n parameter sensitiwiteit studie gedoen vir die inkrimping en die dinamiese momentum en maksimum snelheid vermindering variante. Hierdie studie toon aan dat die insluiting van dinamiese momentum vermindering die PSOA onsensitief maak vir waardes van die kognitiewe en sosiale skaleringsfaktore.

Die inkrimping en dinamiese momentum en maksimum snelheid vermindering variante word dan toegepas op die probleem van optimale ontwerp van stang strukture. Alhoewel min resultate voorheen met die PSOA vir begrensde probleme bereken is, word 'n eenvoudige benadering hierin voorgestel om spanning- en verplasingsbegrensings gedurende die aanvanklike

stadiums van die swerm soektog te hanteer. Verhoogde sosiale druk (groepsdruk) word ten koste van kognitiewe leer op ontoelaatbare partikels toegepas. Hiermee word gepoog om die migrasie tempo na gunstige gebiede te bespoedig.

Omvattende numeriese resultate word voorgelê vir die uitgebreide Dixon-Szegö toetsprobleme, asook 'n paar bekende stangstrukture met dimensionaliteit van tot 21. Die resultate toon die toepaslikheid van die gradiëntlose PSOA vir die twee programmeringsklasse onder beskouing aan.

Acknowledgments

I would like to express my sincere gratitude towards the following persons:

- My advisor, Professor Albert Groenwold, for his guidance and willingless to assist. Without his support this work would never have been finished.
- My parents for their patient support during the writing of this thesis.
- My friends Marco and Stefmarie.

Financial support granted by the NRF is gratefully acknowledged.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Traditional single trajectory algorithms based on the simple gradient descent algorithm, when applied to global optimization problems, all have one great common drawback, namely the tendency to become 'stuck' in a local minima during the search. Numerous approaches have been formulated to overcome this drawback. A good review of global optimization algorithms is presented by Törn and Zilinskas [1]. Recently, multi-start methods [2] have been proposed to overcome this drawback, where parallel or sequential searches are executed with each trajectory origin randomly positioned in the search space. This is done in the hope that at least one of the search trajectories will terminate at the global optimum. A major drawback however is that information gathered by each search trajectory is usually unavailable to the others, or discarded in subsequent searches.

With population based methods it is attempted to retain and share the fitness information that has been obtained previously and, by using this information, direct subsequent function evaluations in a more intelligently coordinated search pattern to improve performance and avoid entrapment in local minima.

The most commonly used population-based evolutionary methods are based on, or inspired by, phenomena found in nature. Several different types of evolutionary search methods were developed independently. These include genetic programming (GP) [3], which evolve programs, evolutionary programming (EP) [4], which focus on optimizing continuous functions without recombination, evolutionary strategies (ES) [5], which focus on optimization of continuous functions with recombination, and genetic algorithms (GAs) [6], which focus on optimizing general combinatorial problems. (For a brief description of these and other related methods, see Appendix C).

While population based methods compare badly in terms of cost effectiveness with finely tuned gradient based trajectory methods, they do tend to be more robust (i.e more resistant to being trapped in a local minima) in complex global optimization problems, especially those with excessive numerical 'noise', when the use of gradient methods become an 'art' indeed.

In this dissertation such a recently introduced population based method, namely the particle swarm optimization paradigm, is implemented, analyzed and applied to a number of global and structural optimization problems.

1.2 Objectives

The objective of this work is to contribute to the development of a reliable and efficient global optimization algorithm based on the the particle swarm optimization algorithm. It is also the intention to analyse the various variants which have recently been proposed to improve upon the PSOA's performance on a common test set, as they have previously been studied in isolation by their respective authors, and usually for different problems.

In addition, the developed algorithm is to be applied to an engineering problem of practical importance, namely the optimal design of truss structures. To accomplish this, the accomodation of constraints into the PSOA is to be studied.

The final objective is to obtain a general purpose particle swarm algorithm which will strike an acceptable balance between reliability and cost.

1.3 Thesis overview

In Chapter 2 a brief historical overview of the particle swarm optimization algorithm is presented, followed by a discussion of the benefits of population based algorithms. The formal mathematical formulation of the PSOA is then detailed, with the remainder of the chapter taken up by an analysis and identification of shortcomings of the PSOA.

Chapter 3 is concerned with a number of variants on the original PSOA, which were proposed in order to improve performance in terms of reliability and cost. These variants are formulated and discussed.

Chapter 4 introduces the global programming problem and the extended Dixon-Szegö test set, which is used in a comparative study of the abovementioned variants.

In Chapter 5 a detailed parameter sensitivity study is performed on the most successful of the PSOA modifications.

In Chapter 6 the results of the parameter study are used in applying the selected PSOA variants with optimized parameters to a set of structural truss design problems.

Finally, conclusions and recommendations for future study are made in Chapter 7.

Chapter 2

Particle swarm optimization

2.1 Particle swarm optimization: A brief history

Computer simulations of the movement and behavior patterns of bird flocks and fish schools were first presented by Reynolds [7], and Heppner (a zoologist) and Grenander [8]. These simulations attempted to define the underlying rules of the movement dynamics of bird flocking and fish schooling, and were mainly reliant on the manipulation of inter-individual distances. These studies were the precursors to the particle swarm paradigm.

The particle swarm optimization algorithm (PSOA), was first introduced by Kennedy and Eberhart in 1995 [9, 10], and, compared to other well established population based evolutionary methods such as genetic algorithms, is still in it's infancy. It finds its roots in a variety of fields, which, among others, include artificial life and collective intelligence, chaos theory, fuzzy computing, sociobiology and, interestingly enough, psychology.

Kennedy and Eberhart inferred a likeness of the swarm behavior to human social behavior. This deduction followed on the observation that, like individual fish or birds adjust their movement patterns to maintain their position in a school or flock, humans tend to adjust their beliefs and attitudes to conform to their peers.

The PSOA models the exploration of a problem space by a population of agents or particles; the agents' success history influence their own search patterns and those of their peers. The search is focused toward promising regions by biasing each particle's velocity vector toward their own remembered best position as well as the communicated best ever swarm location. The importance of these two positions are weighed by two factors, aptly called the cognitive and social scaling parameters [11]. These two components are the among the main governing parameters of swarm behavior (and algorithm efficiency), and have been the subject of extensive study [12, 13, 14].

As various studies [11, 15, 16] have revealed, there are a number of shortcomings and limitations to the 'standard' PSOA as first proposed by Kennedy and Eberhart. Subsequently the PSOA has undergone rapid development, with several adaptations to improve performance and to apply the algorithm to other types of problems have been proposed. These

adaptations or variants will be discussed and analyzed in Chapter 3.

Particle swarm optimization, while also being a population based method, differs from other similar methods such as genetic algorithms, evolutionary programming and evolutionary strategies, (for a brief discussion of these and other related approaches see Appendix C), in the respect that the primary operator which drives the algorithm is not evolutionary based, but rather a set of rules which dictate the social interaction between members of the swarm. This interaction takes the form of the exchange of information regarding the fitness history of the swarm which is used to influence decisions relating to areas to be explored. There are however a number of similarities between evolutionary approaches and the PSOA, which were studied by Angeline [17] and Eberhart [18].

Furthermore, the PSOA is simpler, both in formulation and computer implementation, than the GA. In addition, the PSOA seems to outperform the GA for a number of difficult programming classes, notably the unconstrained global optimization problem [12, 16].

Previously, the PSOA has been applied to analytical test functions, mostly univariate or bivariate without constraints, by Shi and Eberhart [15] and Kennedy [19]. Kennedy also applied the algorithm to multimodal problem generators and used the PSOA as an optimization paradigm to simulate the ability of human societies to process knowledge [12].

Notwithstanding it's recent popularity, the PSOA has a number of drawbacks, one of which is the presence of problem dependent parameters. Previously, a number of workers have attempted to find 'universal' values for the PSOA parameters, the most recent being Carlisle and Dozier [20].

2.2 PSOA Applications

2.2.1 Neural network training via particle swarm optimization

The PSOA has been applied with success in the field of neural network (NN) training, with this type of problem among one of the first to be addressed by Kennedy and Eberhart [9]. The "training" of a neural network involves the minimization of the fitness error in the forward propagated result through the network by adjusting the weights of the network components.

Extensive research by others have been done in this field, some of the more notable the work by Van den Bergh and Engelbrecht who have applied several modifications to the algorithm which involve dividing the swarm into sub-components and using them in a cooperative manner to solve the NN training problem [21, 22]. Other examples where the PSOA has been used in a neural network context include the training of a NN to identify the presence of Parkinson's disease in patients [23] and the extraction of rules from a fuzzy neural network [24].

2.2.2 Structural optimization

Lately, the PSOA was successfully applied to the optimal shape and size design of structures by Fourie and Groenwold [25, 26], where the design variables represent geometric properties of the structure and certain constraints are enforced (e.g. displacement limits or maximum allowed stress). The optimal topological design of problems is also addressed by Fourie and Groenwold [27].

2.2.3 Other applications

The PSOA has also been applied to a variety of other types of problems, among others the optimization of reactive power and voltage control in electrical distribution networks [28] and the practical distribution state estimation thereof [29]. It has also been applied successfully to the field of process biochemistry [30].

2.3 Particle swarm optimization algorithm formulation

We will now formulate the particle swarm algorithm as proposed by Kennedy and Eberhart [9, 10]. The algorithm is constructed as follows: Let us consider a flock of p particles or birds, each representing a possible solution point in the problem space D. For each particle i, Kennedy and Eberhart originally proposed that the position $\boldsymbol{x}^i$ is updated in the following manner:

$$\boldsymbol{x}^i_{k+1} = \boldsymbol{x}^i_k + \boldsymbol{v}^i_{k+1}, \tag{2.1}$$

with the velocity $\boldsymbol{v}^i$ calculated as follows:

$$\boldsymbol{v}^i_{k+1} = \boldsymbol{v}^i_k + c_1 r_1(\boldsymbol{p}^i_k - \boldsymbol{x}^i_k) + c_2 r_2(\boldsymbol{p}^g_k - \boldsymbol{x}^i_k). \tag{2.2}$$

Here, subscript k indicates an (unit) pseudo-time increment. $\boldsymbol{p}^i_k$ represents the best ever position of particle i at time k, with $\boldsymbol{p}^g_k$ representing the global best position in the swarm at time k. r_1 and r_2 represent uniform random numbers between 0 and 1. Kennedy and Eberhart proposed that the cognitive and social scaling parameters c_1 and c_2 are selected such that $c_1 = c_2 = 2$, in order to allow a mean of 1 (when multiplied by the random numbers r_1 and r_2). The result of using these proposed values is that the particles overshoot the target half the time.

Let us denote the best ever fitness value of a particle at $\boldsymbol{p}^i_k$ as f^i_{best} and the best ever fitness value of a particle at $\boldsymbol{p}^g_k$ as f^g_{best}.

The particle swarm optimization algorithm is now outlined as follows:

1. Initialize
   (a) Set constants k_{max}, c_1, c_2.
   (b) Randomly initialize particle positions $\boldsymbol{x}^i_0 \in \boldsymbol{D}$ in $\mathbb{R}^n$ for $i = 1, ..., p$.

(c) Randomly initialize particle velocities $0 \leq \boldsymbol{v}_0^i \leq \boldsymbol{v}_0^{max}$ for $i = 1, ..., p$.

(d) Set $k = 1$

2. Optimize

(a) Evaluate function value f_k^i using design space coordinates $\boldsymbol{x}_k^i$.

(b) If $f_k^i \leq f_{best}^i$ then $f_{best}^i = f_k^i$, $\boldsymbol{p}_k^i = \boldsymbol{x}_k^i$.

(c) If $f_k^i \leq f_{best}^g$ then $f_{best}^g = f_k^i$, $\boldsymbol{p}_k^g = \boldsymbol{x}_k^i$.

(d) If stopping condition is satisfied then goto 3.

(e) Update all particle velocities $\boldsymbol{v}_k^i$ for $i = 1, ..., p$ with rule (2.1).

(f) Update all particle positions $\boldsymbol{x}_k^i$ for $i = 1, ..., p$ with rule (2.2).

(g) Increment k.

(h) Goto 2(a).

3. Terminate

The above algorithm is also represented by the flow diagram depicted in Figure 2.1.

2.4 Analysis of velocity rule

The manner in which the velocity rule influences an individual particle's position can be explained at the hand of Figures 2.2, 2.3, 2.4 and 2.5. If we examine the velocity rule (2.2), we note that the cognitive contribution to calculating the velocity is:

$$c_1 r_1 (\boldsymbol{p}_k^i - \boldsymbol{x}_k^i), \tag{2.3}$$

with c_1 the cognitive parameter, r_1 a random number between 0 and 1, and $\boldsymbol{p}_k^i$ and $\boldsymbol{x}_k^i$ the best fitness and current positions of particle i respectively.

If we consider a 2-dimensional search space (Figure 2.2) we can determine the search area to which the particle can possibly move in the next update of (2.1). Replacing the time increment k with a dimensional index for the moment, we see that the distances between $\boldsymbol{p}^i$ and $\boldsymbol{x}^i$ are $(p_1^i - x_1^i)$ and $(p_2^i - x_2^i)$ for dimension 1 and 2 respectively. By virtue of difference calculated in (2.3), the direction the particle will move will always be toward $\boldsymbol{p}^i$ if we neglect the previous velocity v_k. Since r_1 varies between 0 and 1, the maximum possible travel distance (with $v_k = 0$) for the particle during a single timestep for both dimensions are $c_1(p_1^i - x_1^i)$ and $c_1(p_2 - x_2)$ as indicated. The possible positions that can be occupied by the particle during the next timestep will form a line from the current particle position ($r_1 = 0$) toward $\boldsymbol{p}^i$ as r_1 is increased. This line will extend beyond $\boldsymbol{p}^i$ if $c_1 > 1$, allowing for the possibility of the particle overshooting.

Similarly, if we consider the social component of (2.2):

$$c_2 r_2 (\boldsymbol{p}_k^g - \boldsymbol{x}_k^i), \tag{2.4}$$

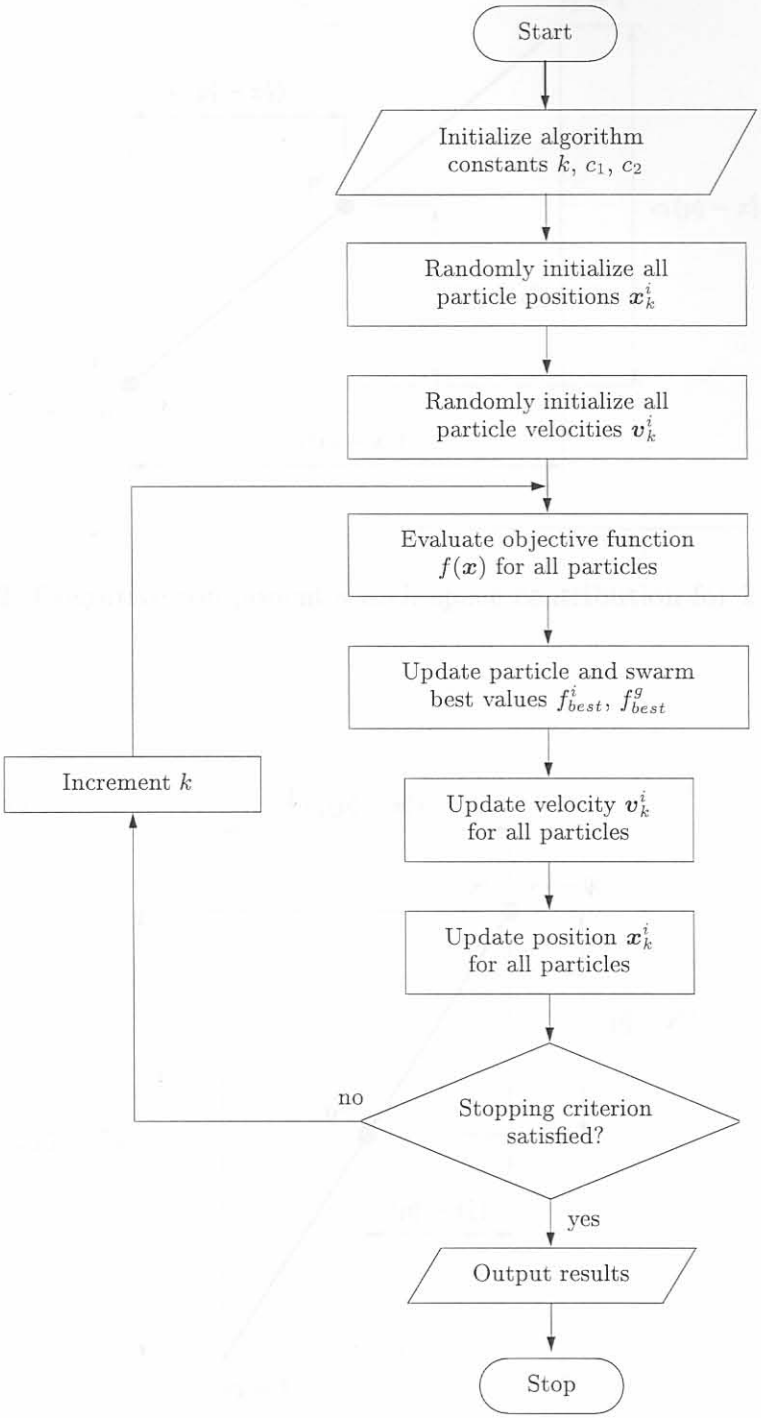


Figure 2.1: Flow analysis of the classical particle swarm optimization algorithm

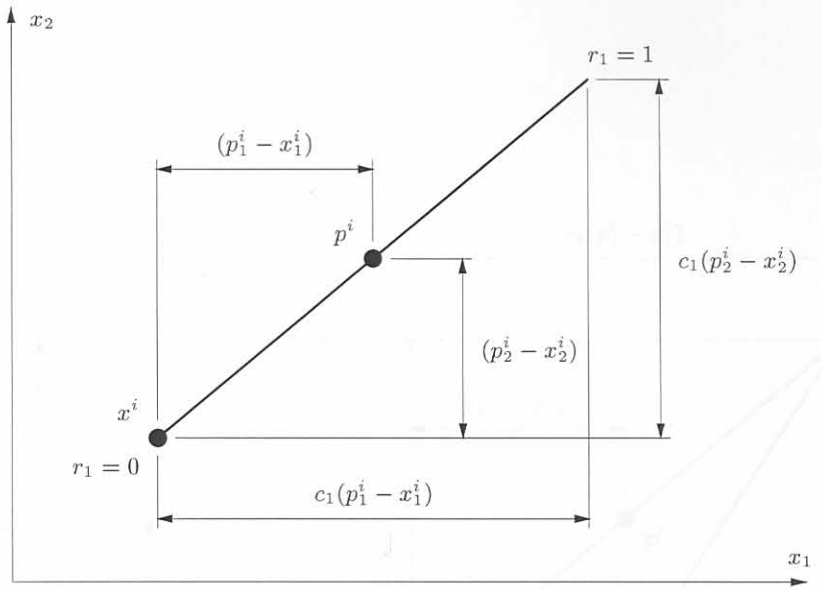


Figure 2.2: Cognitive component search space contribution for 2-D problem

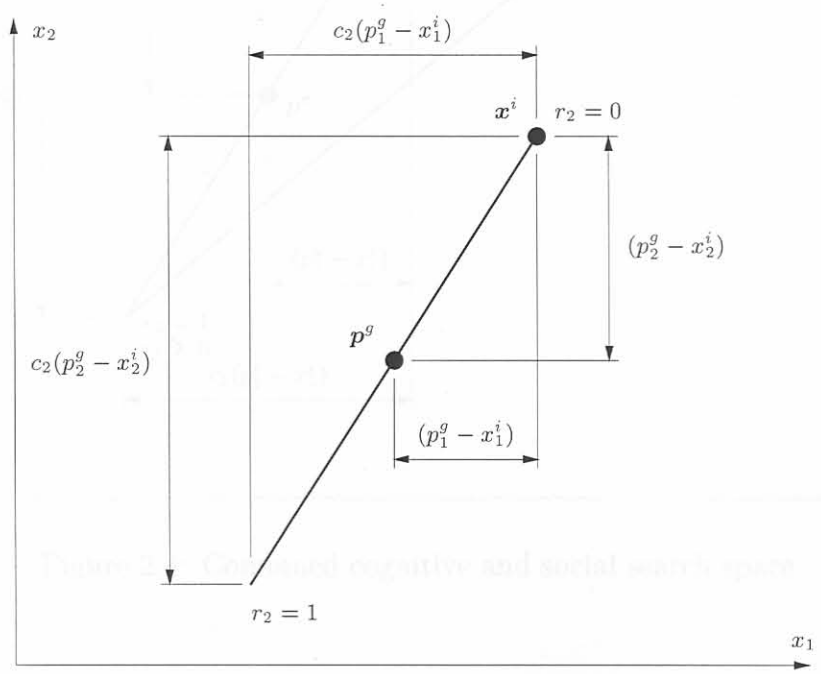


Figure 2.3: Social component search space contribution for 2-D problem

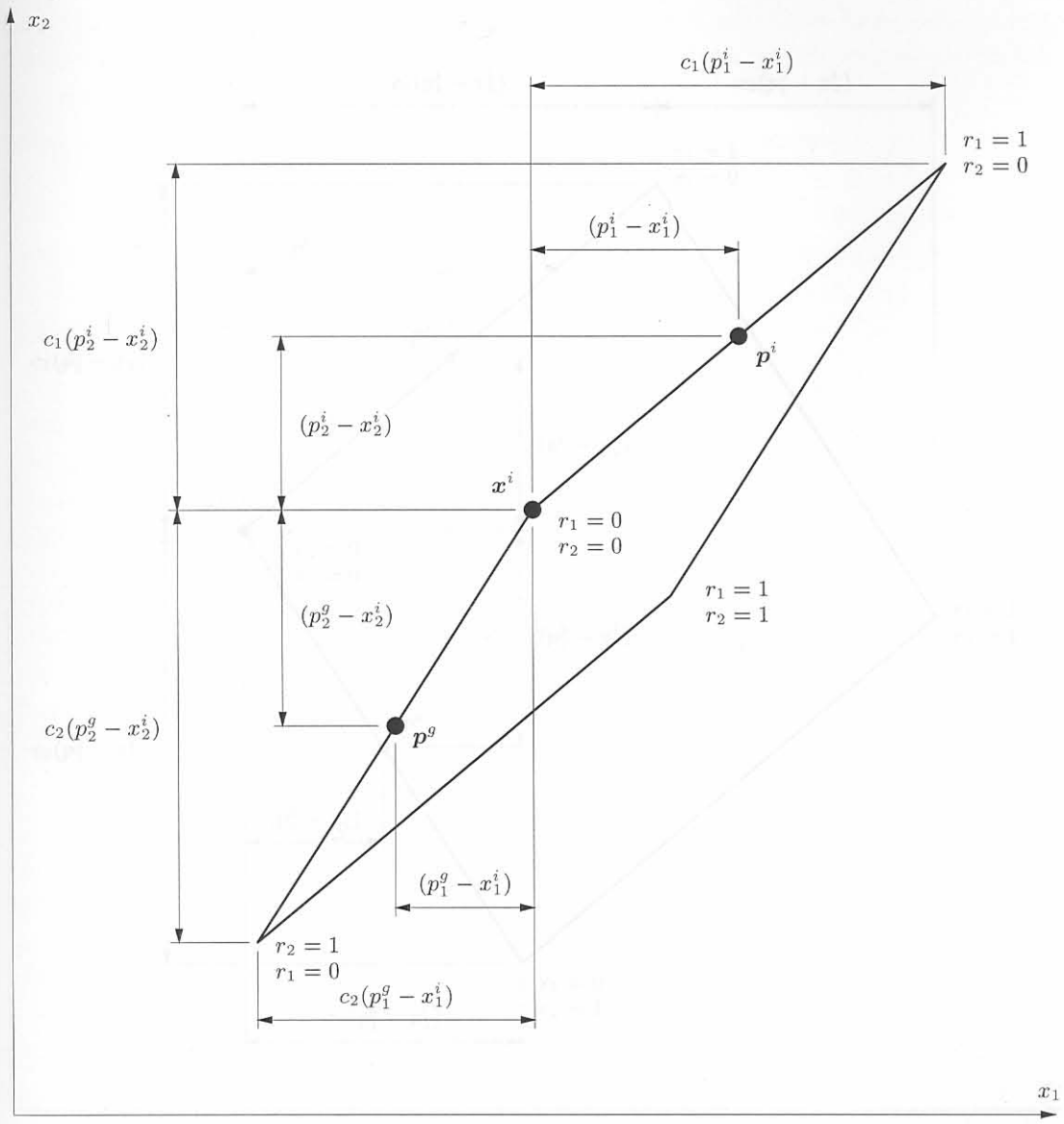


Figure 2.4: Combined cognitive and social search space

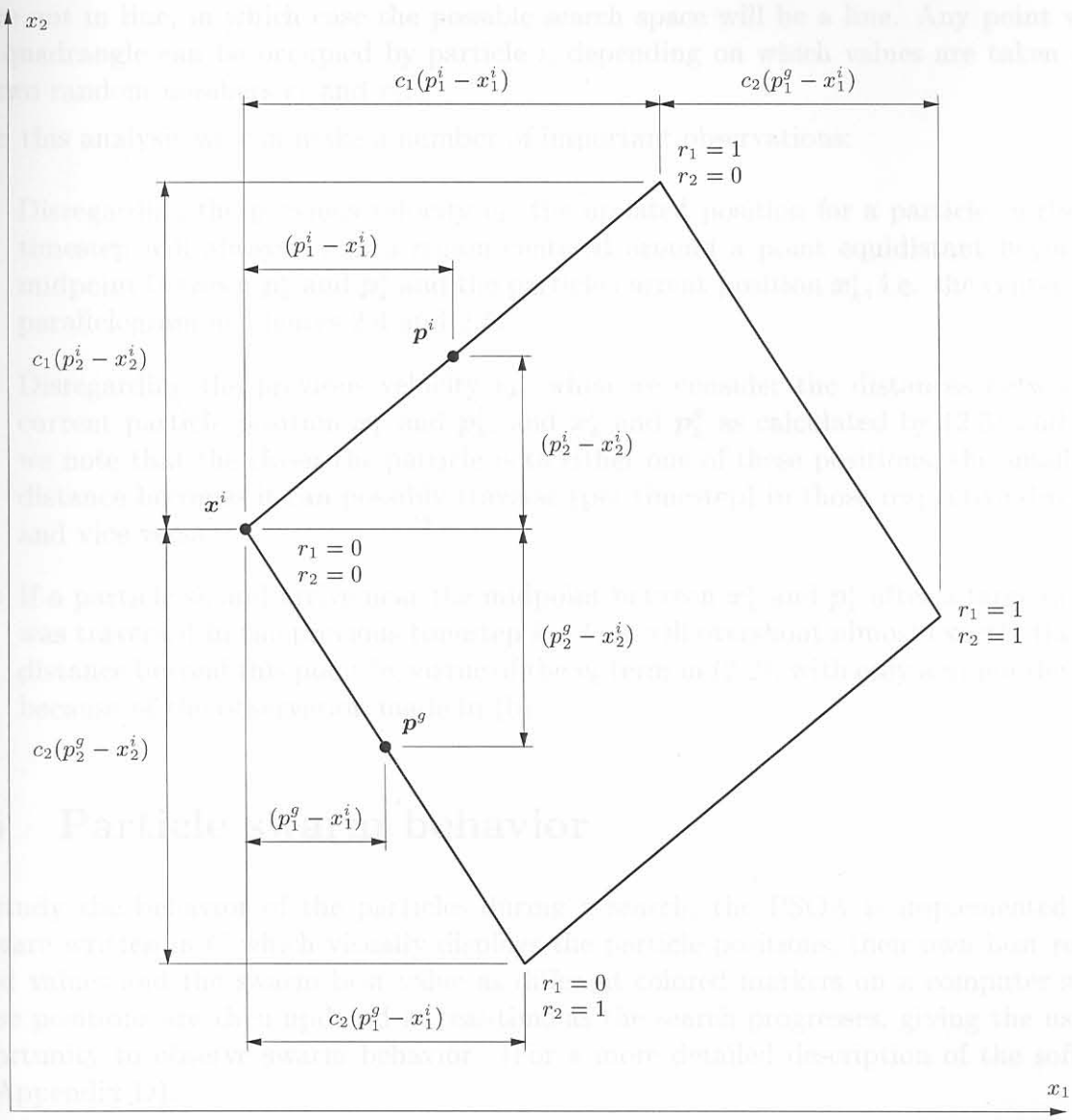


Figure 2.5: Alternative case of combined cognitive and social search space

we see that the maximum distance in this case becomes $c_2(p_1^g - x_1)$ and $c_2(p_2^g - x_2)$, depicted by Figure 2.3. The possible positions the particle i could occupy in the next timestep k is again a line originating from the current position toward $\boldsymbol{p}^g$. This line will also extend beyond $\boldsymbol{p}^g$ if $c_2 > 1$.

If we combine the cognitive and social search contributions we obtain the search area in Figure 2.4. This search area will form a parallelogram in 2-dimensional space if p^g, p^i and x^i are not in line, in which case the possible search space will be a line. Any point within this quadrangle can be occupied by particle i, depending on which values are taken on by the two random numbers r_1 and r_2.

From this analysis we can make a number of important observations:

(a) Disregarding the previous velocity v_k, the updated position for a particle in the next timestep will always be in a region centered around a point equidistant beyond the midpoint between $\boldsymbol{p}_k^i$ and $\boldsymbol{p}_k^g$ and the particle current position $\boldsymbol{x}_k^i$, i.e. the center of the parallelogram in Figures 2.4 and 2.5.

(b) Disregarding the previous velocity v_k, when we consider the distances between the current particle position $\boldsymbol{x}_k^i$ and $\boldsymbol{p}_k^i$, and $\boldsymbol{x}_k^i$ and $\boldsymbol{p}_k^g$ as calculated by (2.3) and (2.4), we note that the closer the particle is to either one of these positions, the smaller the distance becomes it can possibly traverse (per timestep) in those respective directions and vice versa.

(c) If a particle should arrive near the midpoint between $\boldsymbol{x}_k^i$ and $\boldsymbol{p}_k^i$ after a large distance was traversed in the previous timestep $k-1$, it will overshoot almost exactly the same distance beyond this point by virtue of the v_k term in (2.2), with only a minor deviation because of the observation made in (b).

2.5 Particle swarm behavior

To study the behavior of the particles during a search, the PSOA is implemented using software written in C which visually displays the particle positions, their own best remembered values and the swarm best value as different colored markers on a computer screen. These positions are then updated in real-time as the search progresses, giving the user the opportunity to observe swarm behavior. (For a more detailed description of the software, see Appendix D).

For the original PSOA, as formulated in this chapter, it was observed herein that the swarm best position $\boldsymbol{p}_k^g$ usually settles very quickly near the global optimum after jumping around in the problem space D during the initial timesteps. Any outlying particles then quickly distribute themselves evenly around this position. Depending on the nature of the problem either one of the following scenario's take place:

For convex optimization problems, such as the sphere function, the group best position $\boldsymbol{p}_k^g$ will start moving downslope as individual particles in the immediate region around $\boldsymbol{p}_k^g$ find improved fitness values and their best values $\boldsymbol{p}_k^i$ become $\boldsymbol{p}_k^g$. Throughout this movement

of $\boldsymbol{p}_k^g$ the swarm redistributes itself around this centerpoint. Once $\boldsymbol{p}_k^i$ reaches the minima's immediate neighborhood the particle best remembered positions $\boldsymbol{p}_k^i$ rapidly start converging toward $\boldsymbol{p}_k^g$, and the overall swarm diameter contracts. This leads to a progressively smaller area being searched, and the eventual convergence of the swarm toward the minima.

For non-convex functions or functions with excessive numerical noise however, the contraction rate of $\boldsymbol{p}_k^i$ toward $\boldsymbol{p}_k^g$ is either extremely slow or nonexistent. This causes the PSOA either to become very expensive in terms of computational effort (cost), and sometimes prevents convergence.

2.6 Summary

From the swarm behavior discussed in the foregoing, it is clear that some artificial means of contracting the particle swarm diameter needs to be effected by modifying or introducing a new operator into the standard PSOA. This enforcement of a progressively smaller search space for multi-modal or non-convex problems will force a localized search around the best remembered particle position and ultimately lead to convergence. Several methods of forcing progressively smaller search spaces with PSOA will be investigated in the next chapter.

An alternative is to hybridize the particle swarm with an efficient gradient based search algorithm which will perform the local search after the PSOA has found the approximate region of the minima. However, then another difficulty then arises, that of deciding when the particle swarm should be stopped and the local search algorithm started. On one side, a premature transition may lead to the gradient based algorithm converging in a local minima where more extensive search by the PSOA may have found the approximate region of the global minima, and on the other a late transition will lead to wasted function evaluations.

For the purpose of this thesis the PSOA will be used during both the global and local search phases, because it has the ability to perform adequately, and in some cases very well, for both stages. Also, since it is the intent of the author to obtain a comparison of the different variants of the PSOA in the next chapter, it will be desireable to do so without the influence of a gradient based method incorporated in the optimizer.

Chapter 3

Particle swarm variants

3.1 Overview

As with any newly proposed optimization algorithm, the original PSOA displayed several shortcomings, which we hinted at in the previous chapter. We will now elaborate on these shortcomings. Furthermore, a number of variants which have been proposed to improve the performance of Kennedy and Eberhart's original PSOA are also presented.

3.2 On 'local' search capability

The original PSO algorithm displays great initial efficiency at converging to the approximate location of the minima during the global search phase (e.g. see Figure 3.1 for a typical example), but after this the convergence rate decreases rapidly when the refined search phase is entered [11, 15]. This decrease in efficiency is caused by the velocity (2.2) not being reduced adequately, leading to detrimentally large distances between sampling points in (2.1). Large velocities also cause the particles to overshoot the search area, a desireable property for the initial global search, but detrimental in the refined search stage. In order to improve the performance during this final phase it is thus necessary to implement methods for decreasing the distance between successive sampling points, by reducing the velocity during the search. This will result in more finely spaced evaluation points and less overshoot. The reduced overshoot will have the effect of concentrating the swarm in a smaller overall search volume. A number of ways have previously been proposed by which the PSO algorithm could be improved.

3.3 Sequential particle swarm algorithm

The following is an outline of the sequential algorithm structure. The original algorithm is modified and implemented in a sequential or asynchronous manner, implying that particle

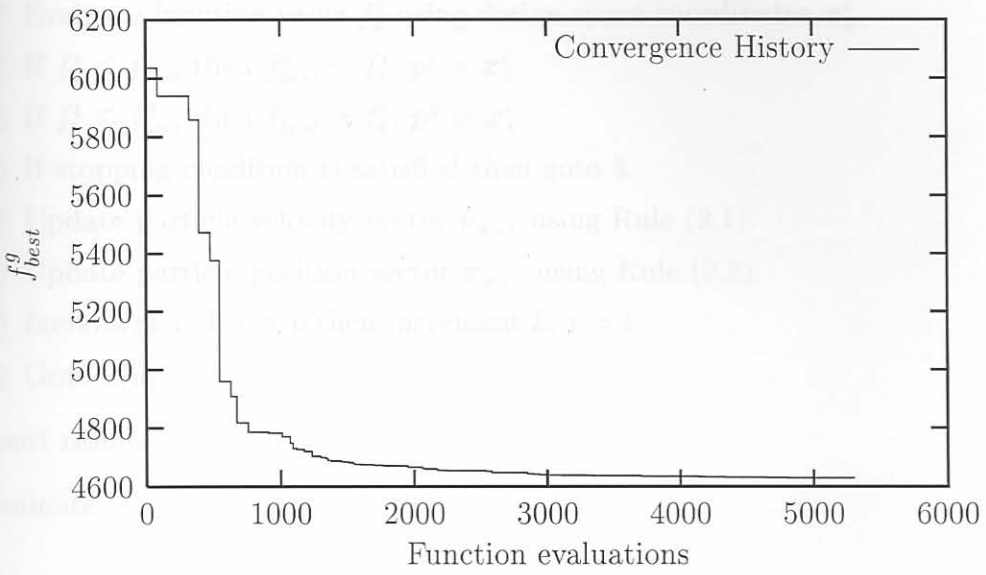


Figure 3.1: Typical history plot

function values, velocities, cognitive and social best remembered positions are updated on a per individual particle basis, rather than a per swarm basis.

Numerical studies by Carlisle and Dozier [20] indicate that the asynchronous method is in general less costly than the synchronous method. The asynchronous method yields improved reaction time to changes in the overall best fitness value and limits unnecessary function evaluations when the stopping condition is satisfied.

This modification was previously proposed by Carlisle and Dozier to limit computational expenses for large swarms [20]. If, for instance, the optimum is found halfway through the swarm's individual particle function value evaluations, the algorithm is stopped without performing the remaining fitness evaluations. Also, if there is an improvement in the swarm best value f^g_{best}, the remainder of the swarm reacts immediately to the swarm best value. (With the classic particle swarm algorithm the improved swarm best value information is only available after the entire swarm's particles have been evaluated in a single pseudo timestep.)

The original position (2.1) and velocity (2.2) rules remain unchanged with this modification.

The asynchronous algorithm becomes:

1. Initialize
   (a) Set constants k_{max}, c_1, c_2
   (b) Randomly initialize particle positions $\boldsymbol{x}^i_0 \in \boldsymbol{D}$ in $\mathbb{R}^n$ for $i = 1, ..., p$
   (c) Randomly initialize particle velocities $0 \leq \boldsymbol{v}^i_0 \leq \boldsymbol{v}^{max}_0$ for $i = 1, ..., p$
   (d) Set $k = 1$
2. Optimize

(a) Evaluate function value f_k^i using design space coordinates $\boldsymbol{x}_k^i$

(b) If $f_k^i \leq f_{best}^i$ then $f_{best}^i = f_k^i$, $\boldsymbol{p}^i = \boldsymbol{x}_k^i$

(c) If $f_k^i \leq f_{best}^g$ then $f_{best}^g = f_k^i$, $\boldsymbol{p}^g = \boldsymbol{x}_k^i$

(d) If stopping condition is satisfied then goto 3.

(e) Update particle velocity vector $\boldsymbol{v}_{k+1}^i$ using Rule (2.1).

(f) Update particle position vector $\boldsymbol{x}_{k+1}^i$ using Rule (2.2).

(g) Increment i. If $i > p$ then increment k, $i = 1$.

(h) Goto 2(a).

3. Report results

4. Terminate

The asynchronous (sequential) algorithm is also depicted in Figure 3.2.

3.4 Variants on Kennedy and Eberhart's PSOA

The variants on the original PSOA of Kennedy and Eberhart are detailed in this section. By no means are the modifications listed in the following exhaustive. However, they probably represent the most significant and most commonly used variants. In the implementations of the modifications that follow, an asynchronous method for updating the swarm best value $\boldsymbol{p}_k^g$ and particle best value $\boldsymbol{p}_k^i$ is used. A global neighborhood [14, 31] is used throughout when exchanging information about the swarm best values and positions.

3.4.1 Introduction of constant inertia weight

This variant, due to Shi and Eberhart [11], constitutes the first significant variation on the original particle swarm algorithm. An inertia term w is introduced into the original velocity rule (2.2) as follows:

$$\boldsymbol{v}_{k+1}^i = w\boldsymbol{v}_k^i + c_1 r_1 \left(\boldsymbol{p}_k^i - \boldsymbol{x}_k^i\right) + c_2 r_2 \left(\boldsymbol{p}_k^g - \boldsymbol{x}_k^i\right). \tag{3.1}$$

The scalar w performs a scaling operation on the velocity $\boldsymbol{v}_k$, analogous to introducing 'momentum' to the particle. Higher values for w results in relatively straight particle trajectories, with significant 'overshooting' or 'overflying' at the target, resulting in a good global search characteristic. Lower values for w result in erratic particle trajectories with a reduction in overshoot, both desireable properties for a refined localized search.

The most serious drawback of the introduction of constant inertia is the problem dependency of w. In a typical implementation, an intermediate value for w is selected, resulting in a search that is unoptimal during both the 'global' and 'local' phases of the search.

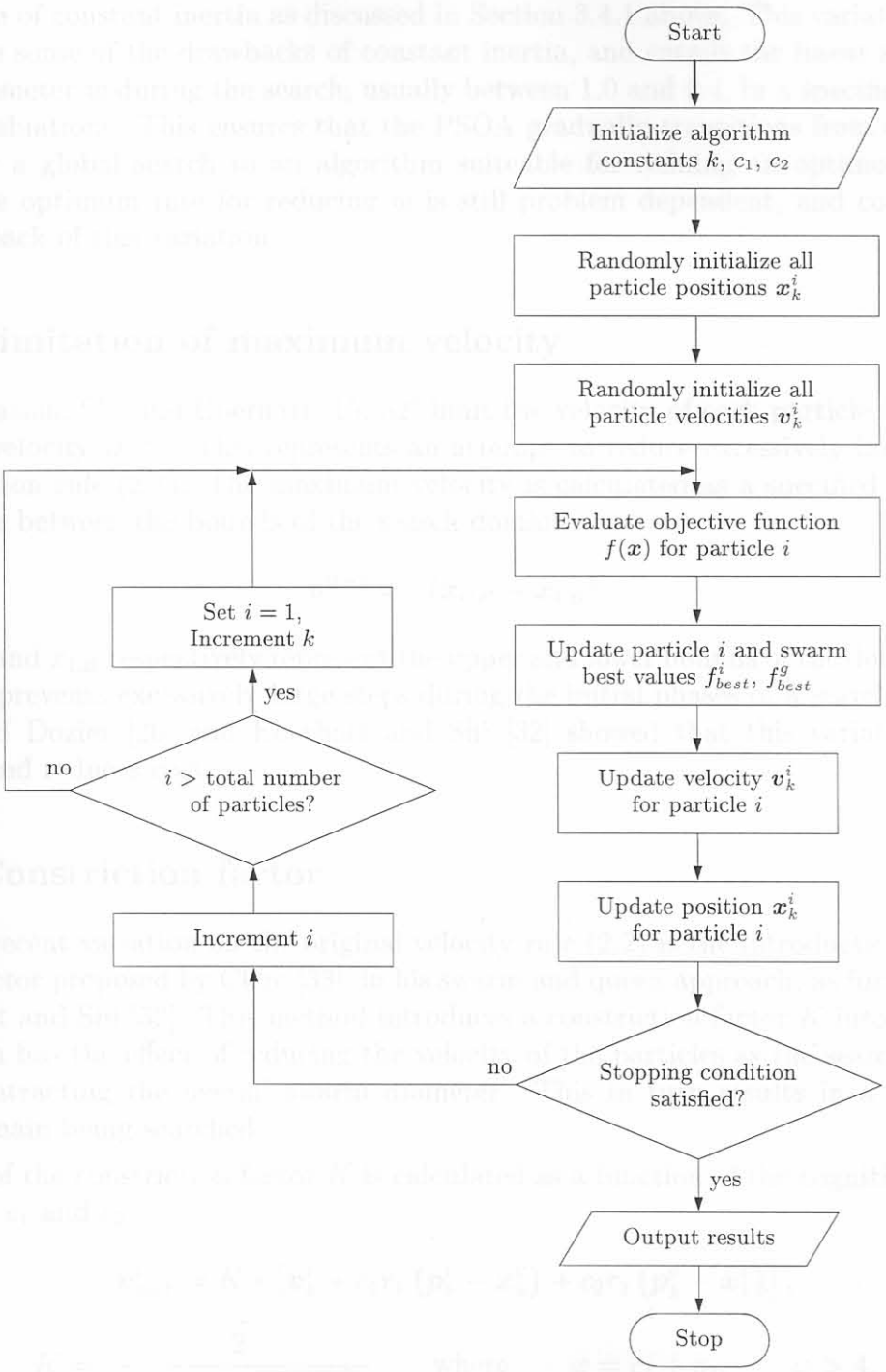


Figure 3.2: Flow diagram for the sequential particle swarm optimization algorithm

3.4.2 Linear inertia reduction

Linear inertia reduction, also proposed by Shi and Eberhart [11, 15], is a variation on the introduction of constant inertia as discussed in Section 3.4.1 above. This variation attempts to eliminate some of the drawbacks of constant inertia, and entails the linear scaling of the inertia parameter w during the search, usually between 1.0 and 0.4, in a specified number of function evaluations. This ensures that the PSOA gradually transitions from an algorithm suitable for a global search to an algorithm suiteable for refining an optimum in a local search. The optimum rate for reducing w is still problem dependent, and constitutes the main drawback of this variation.

3.4.3 Limitation of maximum velocity

In this variation, Shi and Eberhart [15, 32] limit the velocity of each particle to a specified maximum velocity $\boldsymbol{v}^{max}$. This represents an attempt to reduce excessively large step sizes in the position rule (2.1). The maximum velocity is calculated as a specified fraction γ of the distance between the bounds of the search domain:

$$\boldsymbol{v}^{max} = \gamma(\boldsymbol{x}_{UB} - \boldsymbol{x}_{LB}) \tag{3.2}$$

where x_{UB} and x_{LB} respectively represent the upper and lower bounds of the domain D. This once again prevents excessively large steps during the initial phases of a search. Previously, Carlisle and Dozier [20] and Eberhart and Shi [32] showed that this variation increases reliability and reduces cost.

3.4.4 Constriction factor

A notable recent variation on the original velocity rule (2.2) is the introduction of the constriction factor proposed by Clerc [33], in his swarm and queen approach, as further explored by Eberhart and Shi [32]. This method introduces a constriction factor K into velocity rule (2.2), which has the effect of reducing the velocity of the particles as the search progresses, thereby contracting the overall swarm diameter. This in turn results in a progressively smaller domain being searched.

The value of the constriction factor K is calculated as a function of the cognitive and social parameters c_1 and c_2:

$$\boldsymbol{v}_{k+1}^{i} = K * \left[\boldsymbol{v}_{k}^{i} + c_1 r_1 \left(\boldsymbol{p}_{k}^{i} - \boldsymbol{x}_{k}^{i}\right) + c_2 r_2 \left(\boldsymbol{p}_{k}^{g} - \boldsymbol{x}_{k}^{i}\right)\right], \tag{3.3}$$

$$K = \frac{2}{|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}|} \quad \text{where} \quad \varphi = c_1 + c_2, \quad \varphi > 4. \tag{3.4}$$

In their search for an 'off-the-shelf' PSOA, Carlisle and Dozier [20] show that cognitive and social values of $c_1 = 2.8$ and $c_2 = 1.3$ yield good results for their test set.

3.4.5 Dynamic inertia and maximum velocity reduction

This variation, proposed by Fourie and Groenwold [25], aims to reduce the sensitivity to problem dependent parameters associated with previous implementations of inertia [15, 32]. In this approach, a simultaneous dynamic reduction in inertia and maximum velocity is implemented to decrease the swarm domain in a controlled fashion. The approach is outlined as follows: Firstly, the initial inertia w_0 is prescribed, while the initial maximum velocity vector $\boldsymbol{v}^{max}$ is again calculated as a fraction of the domain using (3.2). The swarm domain is then effectively reduced by decreasing the inertia and maximum velocity by fractions α and β respectively, if no improvement in the swarm fitness values $\boldsymbol{p}_k^g$ and $\boldsymbol{p}_k^i$ occur after a to be specified number of iterations h:

$$\text{if } f(\boldsymbol{p}_k^g) \geq f(\boldsymbol{p}_{k-h}^g), \text{ then } w_{k+1} = \alpha w_k,\ \boldsymbol{v}_k^{max} = \beta \boldsymbol{v}_k^{max}, \tag{3.5}$$

with $0 < \alpha, \beta < 1$, prescribed. Rather than reducing the inertia and maximum velocity in a linear fashion, dynamic inertia reduction allows for the adjustment of the algorithm parameters according to the success history of the swarm. For reasons of clarity, we will denote h the 'dynamic delay period' in the remainder of this work.

3.5 Other variants

Several other modifications to the PSOA have been proposed which will be briefly mentioned but not analysed, since they fall outside the scope of this thesis.

3.5.1 Discrete binary particle swarm

A discrete binary particle swarm optimizer was proposed by Kennedy and Eberhart [34], which was benchmarked with a multi modal problem generator against genetic algorithms [19] by Kennedy and Spears. This work was subsequently generalized and applied to the well known traveling salesman problem by Clerc.

3.5.2 Tracking moving extrema

Recently, a modification was proposed by Carlisle and Dozier [35, 36] and Eberhart and Shi [37], whereby a moving extrema in a dynamic problem environment could be tracked.

3.5.3 Hybridizing with other types of algorithms

A number of workers have attempted to improve the PSOA's performance by hybridizing it with other well known methods such as clustering [38], and evolutionary methods [39, 40]. In addition, the PSOA can of course be hybridized with efficient gradient based methods.

Chapter 4

Global optimization

4.1 Introduction

In this chapter the PSOA variants formulated in Chapter3 are applied to an extended Dixon-Szegö test. Numerical results are presented in Appendix A.

4.2 Problem formulation

Firstly, we formally define the global optimization problem: Consider the unconstrained (or bounds constrained) mathematical programming problem represented by the following: Given a real valued objective function $f(\boldsymbol{x})$ defined on the set $\boldsymbol{x} \in D$ in $\mathbb{R}^n$, find the point $\boldsymbol{x}^*$ and the corresponding function value f^* such that

$$f^* = f(\boldsymbol{x}^*) = \min\{f(\boldsymbol{x}) | \boldsymbol{x} \in D\} \ , \tag{4.1}$$

if $\boldsymbol{x}^*$ exists and is unique. Alternatively, find a low approximation $\tilde{f}$ to f^*.

If the objective function and/or the feasible domain D are non-convex, then there may be many local minima which are not optimal. Hence, from a *mathematical* point of view, problem (4.1) is essentially insolvable, due to a lack of mathematical conditions characterizing the global optimum, as opposed to a strictly convex continuous function, which is characterized by the Karush-Kuhn-Tucker conditions at the minimum.

The problem of globally optimizing a real valued function is inherently intractable (unless hard restrictions are imposed on the objective function) in that no practically useful characterization of the global optimum is available. Indeed the problem of determining an accurate estimate of the global optimum is mathematically ill-posed in the sense that very similar objective functions may have global optima very distant from each other [41]. Nevertheless, the need in practice to find a relative low local minimum has resulted in considerable research over the last decades to develop algorithms that attempt to find such a low minimum. A comprehensive survey of global optimization is presented by Törn and Zilinskas [1].

Acronym	Name	n	ϵ_a
G1	Griewank G1	2	0.001
G2	Griewank G2	10	0.1
GP	Goldstein-Price	2	0.001
C6	Six-hump camelback	2	0.001
SH	Shubert, Levi No. 4	2	0.001
RA	Rastrigin	2	0.001
BR	Branin	2	0.001
H3	Hartman 3	3	0.001
H6	Hartman 6	6	0.001
S5	Shekel 5	5	0.001
H7	Shekel 7	7	0.001
S10	Shekel 10	10	0.001

Table 4.1: The extended Dixon-Szegö test set.

Acronym	Name
PSO-CI	PSOA with constant inertia
PSO-CIV	PSOA with constant inertia and maximum velocity limiting
PSO-LI	PSOA with linear inertia
PSO-LIV	PSOA with linear inertia and maximum velocity limiting
PSO-C	PSOA with constriction factor
PSO-DIV	PSOA with dynamic inertia and maximum velocity limiting

Table 4.2: Acronyms used to denote algorithm variants.

4.3 The extended Dixon-Szegö test set

The set of well known problems (e.g. see [42]) presented in Table 4.1 will be used to obtain comparative numerical results for the variants of the PSOA under consideration. The ϵ_a values given in the table are the allowable errors used as convergence criteria in the *a priori* stopping condition (discussed later). A mathematical description of the test set is detailed in Appendix A.

4.4 Numerical results for the different PSOA variants

In this section numerical results are presented for the extended Dixon-Szegö test set presented in Table 4.1. Acronyms used to denote the different PSOA variations are tabulated in Table 4.2. Each problem is analyzed $n = 50$ times and the group best fitness value f^g_{best} at termination is reported. The best run n is classified in terms of the best fitness value found between all of the runs. However, similar results can be obtained using the least amount of function evaluations N_{fe} as criteria, since the standard deviation $\overline{\sigma}$ compare closely for all of the problems, as defined by:

$$\overline{\sigma} = \sqrt{\frac{\sum_{i=1}^{n} \left(f^g_{best\ i} - \bar{f}^g_{best}\right)^2}{n}} \tag{4.2}$$

with

$$\bar{f}^g_{best} = \frac{\sum_{i=1}^{n} f^g_{best\ i}}{n} \tag{4.3}$$

The reader is cautioned however, when interpreting the best f^g_{best} and average values $\bar{f}^g_{best}$ reported in the tables because they are highly volatile even for a high number of repeated searches due to the stochastic nature of the algorithm.

Since it is not the objective to test the performance of different stopping methods, but rather the algorithm itself, a simple *a priori* stopping criteria is used. This method compares the known solution value with the swarm best value and stops the algorithm when the swarm fitness converges to within a certain prescribed absolute error ϵ_a (see Table 4.1).

The maximum allowable function evaluations is set to 30000. If the *a priori* stopping condition is not satisfied within this period the search is deemed not to have converged. Unless otherwise stated, a swarm of 20 particles is used, with the cognitive and social constants c_1 and c_2 both set to 2 for all but the constriction factor variant. For the constriction factor variant the c_1 and c_2 parameters are set to 2.8 and 1.3 respectively, as recommended by Carlisle and Dozier [20]. For the dynamic inertia and maximum velocity modification parameter values $\alpha = \beta = 0.99$ and $h = 10$ are used. These are arbitrarily chosen as numerical experimentation has indicated that the algorithm is relatively insensitive to these parameter values, as will be shown later. $\gamma = 1.0$ is used for all problems where the maximum velocity limitation is applied.

In the interest of obtaining a robust and versatile algorithm no exhaustive attempt is made to optimize the algorithm parameters for individual problems. *For the sake of brevity, all of the results in the following sections are summarized in Figure A.1.*

4.4.1 Standard PSOA

The numerical results obtained with the original PSOA indicates that this search method converged for only 3 out of the 12 problems, and then only with very poor reliability. Hence, no numerical results will be presented in tabulated form.

4.4.2 Constant inertia weight variant

Numerical results are obtained for this method with the inertia weight w varied between 0.1 and 1.0 at 0.1 increments (Figure A.2). Reliability is defined as the amount of times out of the total number of searches (n) the algorithm converges to the optimum value within an allowable error value ϵ_a. From Figures A.2(a) and A.2(b) it can be seen that the optimum value for w, when considering the average cost (Figure A.2(a)) and the optimum reliability (Figure A.2(b)) for the test set lies in the region of $w = 0.5$ to 0.7. Reliability decreases outside this region, more pronounced for the higher values of w than for the lower values. The average cost also increases rapidly above $w = 0.7$. It is interesting to note that for $w = 1$, which reduces the velocity rule (2.2) to the original implementation of the PSOA by Kennedy and Eberhart, none of the runs on any of the problems in the set converged to the required error values.

With the above in mind tabulated results are presented with $w = 0.6$. Very good results are obtained for most of the problems in the set.

The enforcement of a maximum velocity limitation marginally improves the average required function evaluations for most of the problems with the exception of the Shekel 7 problem (Table A.11). This enforcement also worsens the reliability for the Shekel 5 and Shekel 10 problems (Tables A.10, A.12).

4.4.3 Linearly decreasing inertia weight variant

For these results the the inertia weight w is scaled linearly between 0.8 and 0.4 during the first 4000 function evaluations of the search. This variation yields improvements in terms of average cost over the PSO-CIV variation for the Shekel group of problems (Tables A.10, A.11, A.12), with the reliability remaining more or less the same. For most of the other problems however, the average cost increases, indicating that the optimum rate of inertia reduction could be problem dependent.

With a maximum velocity limitation both the average cost and the reliability are improved for all of the problems, with the exception of Shekel 5 (Table A.10), where only the reliability is improved.

4.4.4 Constriction factor variant

Numerical experimentation and work done by others [20] indicate that the maximum allowable velocity modification does not contribute to an increased efficiency for this variant if

bounds constraints are enforced as in our case. Numerical results are very impressive notwithstanding, with a vast increase in convergence rates over the fixed and linearly decreasing inertia variants. For some of the more difficult higher dimensional problems however, the algorithm reliability decreases noticeably (Tables A.10, A.11, A.12).

4.4.5 Dynamically decreasing inertia weight and maximum velocity variant

This method is marginally slower in converging than the constriction factor method for most of the lower dimensional problems (Tables A.1, A.3, A.4, A.6, A.7, A.8, A.9) but for the more difficult higher dimensional problems (Tables A.2, A.10, A.11, A.12) a reduced cost, and in some cases improved reliability compared to constriction is obtained. By using this method of adjusting the inertia parameter w, the inertia modification regains equal footing with the constriction factor method.

4.4.6 Synchronous vs. asynchronous particle swarm algorithm

Finally, the synchronous and asynchronous particle swarm algorithm variants are compared for all of the problems in the test set with all the PSOA variants. From the numerical results presented in Tables A.13, A.14, A.15, (summarized in Figures A.3-A.8), the synchronous method is shown to be less costly and more reliable for all of the problems in the set, supporting the findings of Carlisle and Dozier [20].

4.5 Fitness history

For the difficult G2 problem, a typical history plot for all of the variants is presented in Figure 4.1. From this figure it can clearly be seen that the constriction and dynamic variants are superior to the other variants when considering convergence rates. In terms of reliability however, the linear inertia reduction variant outperforms the constriction and dynamic methods by a small margin.

If we use a points system with equal marks awarded for both the best average number of function evaluations and reliability, and pick the best out of the six variants for each problem the PSOA-CI, PSOA-CIV and PSOA-DIV variants rank highest.

4.6 Summary

From the numerical results presented in this chapter is obvious that the constant-inertia, constriction, and the dynamic inertia/velocity reduction variants are the main contenders when both reliability and cost are considered. Although the linear inertia reduction variation

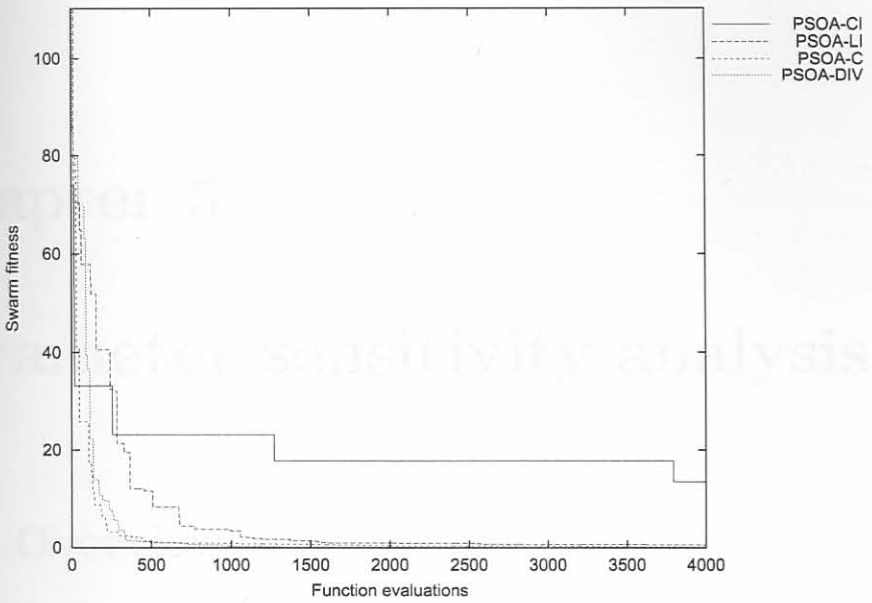


Figure 4.1: PSOA variants comparison: Swarm fitness history

on the PSOA delivers high reliability, this advantage is offset by the high cost for the more difficult problems.

Chapter 5

Parameter sensitivity analysis

5.1 Overview

In this chapter the two most promising variants of the PSOA, namely the constriction method of Clerc and the dynamic inertia and maximum velocity reduction method of Fourie and Groenwold, are subjected to a parameter sensitivity analysis. The main objective of this study is to ascertain what parameter values will result in a general purpose algorithm which will perform well for the entire test set.

5.1.1 Cognitive/social ratio

While it is noted that linear inertia reduction yields good reliability, the results presented in Section 4.4 indicate that constriction and the dynamic inertia/velocity reduction variants are the main contenders when both reliability and cost are considered. Choosing between these two contenders seems difficult, and should probably be judged in future on problems with higher dimensionality than considered herein.

Previously, Kennedy asserted that the sum of the cognitive and social values c_1 and c_2 should approximately equal 4.0, [13], if the cognitive and social ratio is adjusted in the constriction factor method. Carlisle and Dozier [20] have shown that it is advantageous to adjust the cognitive/social ratio to favor cognitive learning (an individualistic swarm). They report that values of 2.8 and 1.3 respectively for the cognitive and social components yield the best performance for the test set they consider.

In the following subsections, it is investigated whether this is true for the extended Dixon-Szegö test set under consideration. Numerical results presented in Figures A.10(a), A.10(b), A.11(a), A.11(b) are obtained by varying the cognitive value c_1 between 0 and 4.1, with the social value calculated in each case as $c_2 = 4.1 - c_1$, as suggested by Carlisle and Dozier [20].

Constriction

Figure A.10(a) indicates that the optimum cognitive value for the extended Dixon-Szegö test set tends to be in the region between 1.5 and 3, indicating that the recommended setting of 2.8 by Carlisle and Dozier is also appropriate for the problems in the extended Dixon-Szegö test set when considering cost. Figure A.10(b) shows the optimum value for reliability to reveal a greater problem dependency, with graphs of the Hartman and Shekel family of problems peaking at c_1 values in the region of 3.5. The Griewank G2 problem however shows a sharp decrease in reliability for values of c_1 above 3. Again, a value of 2.8 would probably be a realistic compromise to ensure reasonable reliability.

Dynamic inertia reduction and maximum velocity limitation

Results with dynamic inertia reduction and maximum velocity limitation (Figure A.11) indicates that this variant of the PSOA is relatively insensitive to the cognitive/social ratio. The cost remains low throughout the range of variation of the c_1 parameter, with the sharp increase to 30000 function evaluations at values above 3.8, indicating that none of the iterations converged. The reliability is also relatively insensitive to the cognitive parameter c_1 for the majority of the problems, with a drop in reliability at values above 3. The insensitivity to low values of cognitive learning indicates the successfullness of the purely 'social' swarm (e.g. see [12]). A reasonable value for this variant is 2.0, which was initially suggested by Kennedy and Eberhart [9] for the 'standard' PSOA.

To further investigate the sensitivity to cognitive/social ratio, the study is repeated, but with $c_1 = c_2$ (Figure A.12). The results again indicate a relatively low sensitivity, with only the Griewank G2 and Shekel problems revealing a slight increase in cost for $c_1 = c_2 > 2$.

5.1.2 Swarm population size

The effect of swarm population size on constriction has been extensively studied by Carlisle and Dozier [20], Eberhart and Shi [32], and Shi and Eberhart [16].

For constriction, our findings closely supports the findings of Carlisle and Dozier, who maintain that, while an increase in population tends to lessen the required swarm *iterations*, the accompanying *cost* (N_{fe}) increases. This is reflected in Figure A.13. Although populations of as little as 5 particles find the optimum at low cost, the sharp decrease in reliability with small population sizes dictate a lower bound when reliability is considered. A swarm population of 20-30 seems a reasonable compromise between cost and reliability.

For dynamic inertia reduction, very similar results to those of constriction are obtained (Figure A.14). A swarm size of 20 seems sufficient as a threshold value to prevent reduced reliability at the low end of the graph, while retaining reasonable cost.

5.1.3 Dynamic delay period and reduction parameters

The effect of the dynamic delay period h on the cost and reliability is depicted in Figure A.15. Both cost and reliability are quite insensitive to the value of h. The only exception to this is the Griewank G2 problem, which reveals a reduction in reliability for values of h above 10.

The effect of the reduction parameters α and β in (3.5) are studied in Figure A.16. For the sake of simplicity, $0.95 \leq \alpha = \beta \leq 1$ is selected. The study reveals a rapid increase in cost for $\alpha = \beta > 0.99$ (Figure A.16(a)), since the algorithm approximates the constant inertia variant as α, β approach 1. For values of $\alpha = \beta < 0.99$, the reliability decreases sharply (Figure A.16(b)), suggesting an optimal value of 0.99 for the extended Dixon-Szegö test set.

5.1.4 Initial velocity fraction

Dynamic inertia reduction is rather insensitive to the value of the initial velocity fraction γ (Figure A.17), although the reliability decreases sharply below $\gamma = 0.3$. A practical setting would probably be $\gamma = 0.5$.

5.2 Recommendations

It is proposed that either the constriction or the dynamic inertia reduction variants of the PSOA are used in global optimization. For constriction, the previously proposed values of $c_1 = 2.8$ and $c_2 = 1.3$ are supported. For dynamic inertia reduction, it is proposed that $c_1 = c_2 = 2.0$, $h = 10$, and $\alpha = \beta = 0.99$. As far as swarm population size is concerned, both variations scale well, with a population size of 20 particles being optimal.

5.3 Summary

The PSOA and some of its variants have been applied to an extended Dixon-Szegö test set in global optimization. It is shown that constriction and dynamic inertia reduction are the main contenders when considering both reliability and cost.

For problems of low dimensionality, dynamic inertia reduction is marginally outperformed by constriction. For problems of higher dimensionality, dynamic inertia reduction seems slightly superior.

Dynamic inertia reduction is shown to be less sensitive to parameter variations than constriction, for which the optimum choice of cognitive c_1 and social c_2 scaling parameters tends to be problem dependent.

Chapter 6

Sizing Design of Truss Structures

6.1 Overview

In this chapter the application of the constriction and dynamic inertia and maximum velocity variants to the optimal sizing design of truss structures are studied. A simple methodology is proposed to accommodate the stress and displacement constraints during initial iterations, when a large number of particles may be infeasible. In this approach, increased social or peer pressure is exerted on infeasible particles to increase their rate of migration to feasible regions.

The development of this chapter is as follows: Firstly, the optimal size and shape design problem is formulated, whereafter the method for accommodating constraints into the PSOA is outlined. This is followed by the application of the PSOA to several well known problems in size optimization with dimensionality of up to 21.

6.2 Problem formulation

In the optimal sizing design of truss structures, the cross-sections of structural members are selected as the design variables $\boldsymbol{x}$. The minimum attainable structural weight is selected as the objective function, subject to allowable stress, displacement and linear buckling constraints. The optimal design problem detailed in Section 4.2 is then reformulated to include constraints as follows: Find the minimum weight f^* such that

$$f^* = f(\boldsymbol{x}^*) = \min f(\boldsymbol{x}) = \boldsymbol{a}^T\boldsymbol{x} \,, \tag{6.1}$$

subject to the general inequality constraints

$$g_j(\boldsymbol{x}^*) \leq 0, j = 1, 2, \ldots, m \,, \tag{6.2}$$

where $\boldsymbol{a}$ and $\boldsymbol{x}$ are column vectors in $\mathbb{R}^n$ and f and g_j are scalar functions of the design variables $\boldsymbol{x}$. The inequality constraints g_j represent the stress, strain, displacement or linear buckling constraints. The finite element method (FEM) may be used to approximate the objective function f and the constaint functions g_j.

6.3 Accommodation of constraints

To facilitate the inclusion of the constraints (6.2) in the PSOA, (6.1) is modified to become

$$\tilde{f} = f(\boldsymbol{x}) + \sum_{j=1}^{m} \lambda_j [g_j(\boldsymbol{x})]^2 \mu_j(g_j) \; , \tag{6.3}$$

with

$$\mu_j(g_j) = \begin{cases} 0 & \text{if} \quad g_j(\boldsymbol{x}) \leq 0 \\ 1 & \text{if} \quad g_j(\boldsymbol{x}) > 0 \end{cases} \; , \tag{6.4}$$

and penalty parameters $\lambda_j > 0$, prescribed. In a typical search, the λ_j parameters are increased linearly with the number of function evaluations. This prevents undue enforcement of the constraints in early stages of the search, while ensuring that the final constraint violations are sufficiently small.

6.3.1 Social pressure

To increase the likelihood of particles migrating to feasible regions in the initial stages of the search, increased social or peer pressure is exerted, at the cost of cognitive learning. In this approach, the best particle value p^i_{best} and best swarm value g_{best} are only updated if the normalized infeasibility (NI^i) is smaller than a to be specified tolerance NI_{allow}. Hence cognitive learning of an initially infeasible bird, represented by c_1 and $\boldsymbol{p}^i_k$, is sacrificed. Only social pressure, represented by c_2 and $\boldsymbol{p}^g_k$, is retained.

This simple idea is implemented as follows: Steps 2(b) and 2(c) in the formal algorithm presented in Section 3.3 are replaced by:

2. (b) If $f^i_k < p^i_{best}$ and $NI^i < NI_{allow}$ then $p^i_{best} = f^i_k$, $\boldsymbol{p}^i_k = \boldsymbol{x}^i_k$, else $c_1 = 0$
   (c) If $f^i_k < g_{best}$ and $NI^i < NI_{allow}$ then $g_{best} = f^i_k$, $\boldsymbol{p}^g_k = \boldsymbol{x}^i_k$

Selecting $c_1 = 0$ is not necessarily optimal, and superior approaches will doubtless be suggested in future.

6.4 Stopping criteria

A number of logical stopping criteria may be specified for the PSOA. Amongst others, the algorithm can be terminated when the average swarm velocity or momentum reaches a prescribed fraction of the initial velocity and momentum. The algorithm can also be terminated when a specific number of iterations or function evaluations S occur without improvement in the best position $\boldsymbol{p}^g_k \leftrightarrow g_{best}$, within a prescribed tolerance ϵ_s.

In constructing stopping criteria, it is important to develop criteria which protect against over sampling of the objective function. Simultaneous, the competing objective of premature convergence should be prevented. While not necessarily optimal, two criteria here are

considered here, namely the *a priory* condition, and the logical condition. They are selected here solely for their simplicity and illustrative powers.

A number of additional stopping criteria may be constructed. For instance, the algorithm can be terminated when the average momentum of the swarm reaches a prescribed fraction of the initial average momentum. A similar argument may off course be used for the average velocity of the swarm.

6.4.1 *A priori* stopping condition

The algorithm is terminated once it obtains the *a priori* known optimum within a prescribed tolerance. This stopping condition is commonly used in the training of neural networks [23], where the output error is minimized. In general, this is not a sensible stopping criterion for structural optimization. However, numerical results are included herein, since the criterion gives a good indication of how fast the algorithm converges to the region of the optimum, and presents a useful guide in estimating the required overhead in terms of number of function evaluations required by other stopping methods.

6.4.2 Logical stopping condition

In this condition, the swarm best value f_{best}^{g} is monitored as the search progresses. If there is no improvement for S specified function evaluations within a specified threshold tolerance ϵ_s, the search is stopped.

6.5 On swarm parameters

As shown by [15], the PSOA is sensitive to, in particular, the parameters w, c_1 and c_2, although dynamic inertia reduction reduces this sensitivity as opposed to constriction, as shown in Chapter 5.

In this study, unless otherwise stated, all results are generated using swarms consisting of 20 agents, with the cognitive and social scaling factors c_1 and c_2 both set to 2. In each case, the allowable normalized infeasibility NI_{allow} is set to 0.02, and the penalty parameters λ_j, $j = 1, 2, \ldots, m$, are linearly scaled from 10^3 to 10^6 in 4000 function evaluations, whereafter λ_j is constant.

6.6 Numerical results

The test set under consideration is tabulated in Table 6.1. The table gives the dimension n, the number of constraints m and the nature of the problems under consideration.

For the numerical results presented in Appendix B.2 each problem is analyzed 10 times, with the normalized infeasibility NI, the average fitness value f_{ave}, standard deviation $\bar{\sigma}$ and cost

Problem Name	Problem Nature	n	m
10-Bar	Convex	10	32
10-Bar	Non-Convex	10	34
25-Bar	Non-Convex	8	84
36-Bar	Convex	21	95

Table 6.1: Structural test problems

N_{fe} being reported for each problem. For the sake of completeness, the best fitness value with its position and the associated number of function evaluations is also given.

In the following discussions, ϵ_a represents the tolerance in the *a priori* stopping condition, and '*Reliability*' the number of times the algorithm converged within 10000 function evaluations. S represents the number of function evaluations elapsed without improvement in best function value, within a tolerance ϵ_s, before termination.

6.6.1 Convex 10-bar truss

The structure is depicted in Figure B.1, and is described in, amongst others, [43]. For the *a priori* stopping condition, tabulated results are presented in Table B.1. (In the table, social pressure is not exerted on initially infeasible birds.) The table reveals that the function evaluations N_{fe} required for convergence is relatively low. In addition, the cost does not increase dramatically as ϵ_a is decreased from 0.05 to 0.01. However, the reliability decreases as ϵ_a becomes stricter, while the normalized infeasibility also decreases. Even so, the normalized infeasibility is some 6% for $\epsilon_a = 1\%$.

Upon the introduction of social pressure, (Table B.9), the reliability becomes 100%, while the decrease in constraint violations (from 16% to zero) is significant. Simultaneously, the computational effort N_{fe} decreases. (In Table B.1, the cost of unconverged searches is not reflected in the average cost.)

Finally, the high overhead of the logical stopping criterion is reflected by a comparison between Tables B.10 and B.6. In both cases, social pressure is exerted, while the latter table uses the logical stopping criterion, with $S = 2000$, 1000 and 500. Apparently, $S = 1000$ is adequate. Nevertheless, the the results indicate that the development of improved stopping criteria in future is of interest.

6.6.2 Non-convex 10-bar truss

This problem is also depicted in Figure B.1, and is amongst others, described in [43, 44]. The physical geometry for this problem is identical to the convex 10-bar truss, the only difference being a modified loading condition which induces multiple local minima in the fitness function [45, 46]

For the sake of brevity, extensive results regarding the influence of the *a priori* stopping condition and social pressure are not given in tabulated form for this (and the following) problems. It suffices to state that the same trend as for the convex 10-bar truss is observed.

Numerical results for the logical stopping criterion and social pressure are presented in Table B.14. Once again, requiring $S = 1000$ is adequate. The normalized infeasibility upon convergence is acceptably small (approximately 0.1%).

6.6.3 Non-convex 25-bar truss

This structure is depicted in Figure B.2 [43, 47], and numerical results for the logical stopping criterion combined with social pressure are presented in Table B.7. For this problem, there is a more pronounced sensitivity to the value of S, and $S = 500$ would suffice.

6.6.4 Convex 36-bar truss

The final test problem is depicted in Figure B.3 [43, 45]. This convex problem is relatively difficult, with 21 design variables, and 95 constraints present. Numerical results using the *a priori* stopping criterion are presented in Table B.4, and using the logical stopping criterion in Table B.8.

The solutions obtained by using the *a priori* stopping condition (Table B.4) are all slightly infeasible, indicating that a higher value for λ_j would be beneficial. Nevertheless, in all cases the constraint violations are less than 1%.

In comparison with the *a priori* stopping criteria results, the logical stopping method performs very poorly, both in terms of cost and constraint violations. This poor performance was the main motivation for the introduction of the social pressure operator, for which numerical results are presented in the following section.

6.6.5 Effect of social pressure

The effect of social pressure is investigated for all of the structural test problems, and results with different *a priori* stopping values are summarized in Appendix B.2.3. Dramatic improvements in cost and normalized infeasability can be observed for all of the problems in this summary. When considering the logical stopping condition a similar trend is observed (compare Tables B.8 and B.16).

6.6.6 Comparison between PSOA-C and PSOA-DIV variants

Table B.17 illustrates the difference in performance between the constriction factor and dynamic inertia and velocity reduction. The average (f_{ave}) and best (f_{best}) fitness values obtained compare closely for the two methods, with the standard deviation $\bar{\sigma}$ for dynamic inertia and velocity reduction roughly 1/3 of the value for the constriction factor. The cost

(N_{fe}) associated with the average and best fitness values for dynamic inertia and velocity reduction are roughly half the respective costs with constriction. (The same trend is observed for the other problems studied.)

6.6.7 Fitness history

Typical fitness histories, comparing the constriction and dynamic inertia and maximum velocity variants are presented in Figures B.4, B.5, B.6, and B.7. In terms of convergence rate, it is clear that the PSOA-DIV variant outperforms the PSOA-C variant.

6.7 Summary

The derivative free particle swarm optimization algorithm can effectively be used for the optimal sizing design of truss structures. While few results for constrained functions using the PSOA have previously been presented, social pressure is used herein to increase the likelihood of migration to feasible regions during the initial phases of the swarm search, thereby sacrificing the cognitive learning ability of initially infeasible particles. Use of the social operator leads to reduced cost with the logical stopping criteria, as well as limiting the constraint violations within a specified tolerance in all of the structural test problem cases.

Chapter 7

Closure

7.1 Conclusions

The studies performed herein, as well as work previously performed by other workers, reveals the potential of the gradient free PSOA for a wide variety of problems. This includes neural network training, discrete optimization, and the tracking of an optimum of which the position changes with time. In addition, this study demonstrates the suitability of the PSOA for the global programming problem, and constrained sizing design of truss structures.

7.2 Recommendations

It is proposed that either the constriction variant or the dynamic inertia and maximum velocity reduction variants are used in global or structural optimization. For the constriction variant, it is proposed that cognitive and social parameters of 2.8 and 1.3 respectively, should be used. While the dynamic inertia and maximum velocity variant is shown to be relatively insensitive to the cognitive and social parameters, it is recommended that c_1 and c_2 both be set to 2.0, as originally proposed by Eberhart and Kennedy. The dynamic delay period h should be approximately 10, with the velocity and inertia reduction parameters (α and β) both between values of 0.990 and 0.999, preferably 0.99. A swarm population of 20 particles delivers satisfactory results for both the constriction and dynamic variants in terms of cost and reliability.

7.3 Directions for future studies

Although the PSOA is now an accepted algorithm receiving wide acknowledgment, the algorithm is still in it's infancy. As such there is still scope for improvement. A number of topics that deserve attention in the near future include the following:

1. The stopping criteria used herein are very basic. In order to minimize undue expense in function evaluations (without invoking premature convergence in global searches), a detailed investigation of suitable stopping criteria is required.

2. Hybridizing with other methods has recently been proposed, in an effort to combine the good global search capabilities of the PSOA with the refined search capability of gradient based methods. This promises to be a fruitful line of research, in particular for structural applications, where cost efficiency is extremely important.

3. Since the structure of the PSOA allows for easy parallelization, it is proposed that the algorithm be implemented on the existing Beowulf cluster at the University of Pretoria.

Bibliography

[1] A. Törn and A. Zilinskas. *Global optimization*, volume 350 of *Lecture notes in computer science*. Springer-Verlag, Berlin, Heidelberg, 1989.

[2] J.A. Snyman and L.P. Fatti. A multi-start global minimization algorithm with dynamic search trajectories. *J. Optim. Theory Appl.*, 54:121–141, 1987.

[3] J.R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. MIT Press, 1992.

[4] L.J. Fogel. Evolutionary programming in perspective: The top down view. In J.M. Zurada, R.J. Marks II, and C. Robinson, editors, *Computational Intelligence: Imitating life*, Piscataway, NJ, 1994.

[5] I. Rechenberg. Evolutionary strategy. In J.M. Zurada, R.J. Marks II, and C. Robinson, editors, *Computational Intelligence: Imitating life*, Piscataway, NJ, 1994. IEEE Press.

[6] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading, MA, 1989.

[7] C.W. Reynolds. Flocks, herds and schools: a distributed behavioral model. In *Computer Graphics*, volume 21, pages 25–34, 1987.

[8] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In S. Krasner, editor, *The Ubiquity Adaptation in Natural and Artificial Systems*, Pretoria, South Africa, 1990. Submitted.

[9] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Perth, Australia, IEEE Service Center, Piscataway, NJ, 1995.

[10] R.C. Eberhart and J. Kennedy. New optimizer using particle swarm theory. In *Proceedings of the 1995 6th International Symposium on Micro Machine and Human Science*, volume 6, pages 39–43, 1995.

[11] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary computation*, pages 69–73. IEEE Press, Piscataway, USA, 1998.

[12] J. Kennedy. The particle swarm: social adaptation of knowledge. In *Proceedings of the International Conference on Evolutionary Computation*, pages 303–308, Indianapolis, IN, 1997. IEEE Service Center, Piscataway, USA.

[13] J Kennedy. The behavior of particles. In V.W. Porto, N Saravan, D Waagen, and A.E. Eiben, editors, *Evolutionary Programming*, number 7 in Evolutionary Programming VII, pages 581–589, San Diego, CA, 1998. Berlin: Springer-Verlag.

[14] P.N. Suganthan. Particle swarm optimiser with neighbourhood operator. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1958–1962, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[15] Yuhui Shi and Russel C. Eberhart. Parameter selection in particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, pages 591–600, Berlin, 1998. Springer. Lecture Notes in Computer Science 1447.

[16] Yuhui Shi and Russell C. Eberhart. Empirical study of particle swarm optimization. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1945–1950, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[17] Peter J. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, pages 601–610, Berlin, 1998. Springer. Lecture Notes in Computer Science 1447.

[18] Russell C. Eberhart and Yuhui Shi. Comparison between genetic algorithms and particle swarm optimization. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, pages 611–616, Berlin, 1998. Springer. Lecture Notes in Computer Science 1447.

[19] J. Kennedy and W. M. Spears. Matching algorithms to problems: an experimental test of the particle swarm and some genetics algorithms on the multimodal problem generator. In *Proceedings of the 1998 IEEE International Conference on Evolutionary Computation*, pages 78–83, 1998.

[20] A. Carlisle and G. Dozier. An off-the-shelf pso. In *Proceedings of the Workshop on Particle Swarm Optimization*, Purdue School of Engineering and Technology, Indianapolis, USA, 2001.

[21] F. van den Bergh and A.P. Engelbrecht. Cooperative learning in neural networks using particle swarm optimizers. In *SAICSIT 2000*, 2000.

[22] F. van den Berg. Particle swarm weight initialization in multi-layer perceptron artificial neural networks. In *Proceedings of the International Conference on Artificial Intelligence*, Durban, South Africa, 1999.

[23] Russell C. Eberhart and Xiaohui Hu. Human tremor analysis using particle swarm optimization. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1927–1930, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[24] Z He, C Wei, L Yang, X Gao, S Yao, R Eberhart, and Y Shi. Extracting rules from fuzzy neural network by particle swarm optimization. In *Proceedings of the IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska, USA, 1998.

[25] P.C. Fourie and A.A. Groenwold. Particle swarms in size and shape optimization. In J.A. Snyman and K. Craig, editors, *Proc. Workshop on Multidisciplinary Design Optimization*, pages 97–106, Pretoria, South Africa, August 2000.

[26] J.F. Schutte and A.A. Groenwold. Sizing design of truss structures using particle swarms. 2001. Submitted.

[27] P.C. Fourie and A.A. Groenwold. The particle swarm algorithm in topology optimization. In *Proc. Fourth World Congress of Structural and Multidisciplinary Optimization*, Dalian, China, May 2001. In Press.

[28] Yoshikazu Fukuyama, Shinichi Takayama, Yosuke Nakanishi, and Hirotaka Yoshida. A particle swarm optimization for reactive power and voltage control in electric power systems. In Wolfgang Banzhaf, Jason Daida, Agoston E. Eiben, Max H. Garzon, Vasant Honavar, Mark Jakiela, and Robert E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, volume 2, pages 1523–1528, Orlando, Florida, USA, 13-17 July 1999. Morgan Kaufmann.

[29] Shigenori Naka, Takamu Genji, Toshiki Yura, and Yoshikazu Fukuyama. Practical distribution state estimation using hybrid particle swarm optimization. In *Proceeding of IEEE Power Engineering Society Winter Meeting*, Columbus, Ohio, USA, 2001.

[30] A. R. Cockshott and B. E. Hartman. Improving the fermentation medium for echinocandin b production. part ii: Particle swarm optimization. In *Process Biochemistry*, volume 36, pages 661–669, 2001.

[31] James Kennedy. Small worlds and mega-minds: Effects of neighborhood topology on particle swarm performance. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin Yao, and Ali Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1931–1938, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[32] R. C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 84–88, Piscataway, NJ, 2000. IEEE Service Center.

[33] Maurice Clerc. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In Peter J. Angeline, Zbyszek Michalewicz, Marc Schoenauer, Xin

Yao, and Ali Zalzala, editors, *Proceedings of the Congress of Evolutionary Computation*, volume 3, pages 1951–1957, Mayflower Hotel, Washington D.C., USA, 6-9 July 1999. IEEE Press.

[34] J. Kennedy and R.C. Eberhart. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 Conference on Systems, Man and Cybernetics*, pages 4104–4109. IEEE Service Center, Piscataway, NJ, 1997.

[35] A. Carlisle and G. Dozier. Adapting particle swarm optimization to dynamic environments. In *International Conference on Artificial Intelligence*, volume I, pages 429–434, Las Vegas, NV, 2000.

[36] A. Carlisle and G. Dozier. Tracking changing extrema with particle swarm optimizer. Technical report, Auburn University, 2001.

[37] Russell C. Eberhart and Yuhui Shi. Tracking and optimizing dynamic systems with particle swarms. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, pages 94–100. IEEE Press, 2001.

[38] James Kennedy. Stereotyping: Improving particle swarm performance with cluster analysis. In *Proc. of the 2000 Congress on Evolutionary Computation*, pages 1507–1512, Piscataway, NJ, 2000. IEEE Service Center.

[39] Angeline P. Using selection to improve particle swarm optimization. In *IEEE International Conference on Evolutionary Computation*, Anchorage, Alaska, USA, 1998.

[40] Morten Løvbjerg, Thomas Kiel Rasmussen, and Thiemo Krink. Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the third Genetic and Evolutionary Computation Conference (GECCO-2001)*, 2001.

[41] F. Schoen. Stochastic techniques for global optimization: A survey of recent advances. *J. Global Optim.*, 1:207–228, 1991.

[42] L.C.W. Dixon and G.P. Szegö. *Towards global optimization.* N-Holland Publ. Co., 1975.

[43] U.T. Ringertz. On methods for discrete structural optimization. *Engineering Optimization*, 13:47–64, 1988.

[44] M. Sunar and A.D. Belegundu. Trust region methods for structural optimization using exact second order sensitivity. *International Journal of Numerical methods in engineering*, 32:275–293, 1991.

[45] A.A. Groenwold, N. Stander, and J.A. Snyman. A pseudo-discrete rounding method for structural optimization. *Struct. Opt.*, 11:218–227, 1996.

[46] K. Svanberg. On local and global minima in structural optimization. *New Directions in Optimum Structural Design*, 1984.

[47] L. Schmit and C. Fleury. Discrete-continuous variable structural synthesis using dual methods. *AIAA Journal*, 18:1515–1524, 1980.

[48] A.O. Griewank. Generalized descent for global optimization. *J. Optim. Theory Appl.*, 34:11–39, 1981.

[49] S. Lucidi and M. Piccioni. Random tunneling by means of acceptance-rejection sampling for global optimization. *J. Optim. Theory Appl.*, 62:255–277, 1989.

[50] L.A. Rastrigin. *Systems of Extremal Control.* Nauka, Moscow, 1974.

[51] F.H. Branin and S.K. Hoo. *A Method for Finding Multiple Extrema of a function of n Variables*, pages 231–237. Academic Press, London, 1972.

[52] T Bäck, U. Hammel, and H.-P. Schwefel. Evolutionary computation: Comments on the history and current state. *IEEE transactions on Evolutionary Computation*, 1(1):3–16, 1997.

[53] J.H. Holland. Outline for a logical theory of adaptive systems. *J. Assoc. Conput. Mach.*, 3:297–314, 1962.

[54] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MI:University of Michigan Press, 1975.

[55] J.H. Holland and J.S. Reitman. Cognitive systems based on adaptive algorithms. In D.A. Waterman and F. Hayes-Roth, editors, *Pattern-Directed Interference systems.* New York: Academic, 1978.

[56] K. De Jong. *An Analysis of the Behavior of a class of Genetic Adaptive Systems.* PhD dissertation, University of Michigan, Ann Arbor, Department of Computer and Communication Sciences,, 1975.

[57] K.A. De Jong. On using genetic algorithms to search program spaces. In *2nd Int. Conf. on Genetic Algorithms and Their Applications*, pages 210–216, Hillsdale, NJ, 1987.

[58] K.A. De Jong. Are genetic algorithm function optimizers? *Parallel Problem Solving from Nature*, 2:3–13, 1992.

[59] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison Wesley, Reading, MA, 1989.

[60] D.E. Goldberg. A note on boltzmann tournament selection for genetic algorithms and population-oriented simulated annealing. *Complex systems*, 4:445–460, 1990.

[61] D.E. Goldberg and K. Deb. *A comparative analysis of selection schemes used in genetic algorithms.* Morgan Kaufmann, San Mateo, CA, 1991.

[62] D.E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis and first results. *Complex systems*, 3:493–530, 1989.

[63] L.J. Fogel. Autonomous automata. *Ind. Res.*, 4:14–19, 1962.

[64] L.J. Fogel. *On the organization of intellect.* PhD thesis, University of California, Los Angeles, 1964.

[65] J.W. Atmar. *Speculation on the evolution of intelligence and possible realization in machine form.* PhD thesis, New Mexico Sate Univ., Las Cruces, 1976.

[66] G.H. Burgin. On playing two-person zero-sum games against nonminmax players. *IEEE Trans. Syst. Cybern.*, SSC-5(4):369–370, 1969.

[67] G.H. Burgin. Systems identification by quasilinearization and evolutionary programming. *J. Cybern.*, 3(2):56–75, 1973.

[68] I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution.* Frommann-Holzboog, Stuttgart, Germany, 1973.

[69] I. Rechenberg. Evolutionsstrategie '94. In *Werkstatt Boinik und Evolutionstechnik*, volume 1. Frommann-Holzboog, 1994.

[70] H.-P. Schwefel. Evolutionsstrategie un numerische optimierung. Master's thesis, Technische Universität Berlin, Bermany, 1975.

[71] H.-P. Schwefel. *Evolution and Optimum Seeking.* Sixth-Generation Computer Technology Series. Wiley, New York, 1995.

[72] M. Herdy. Reproductive isolation as strategy parameter in hierarchically organized evolution strategies. In *Parallel Problem Solving from Nature*, number 2, pages 207–217, Amsterdam, the Netherlands, 1992. Elsevier.

[73] F. Kursawe. A variant of evolution strategies for vector optimization. In *Parallel Problem Solving from Nature*, number 2, pages 193–197, Berlin, Germany, 1991. Springer.

[74] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. of Chem. Phys.*, 21(6):1087–1092, 1953.

[75] E.H.L. van Laarhoven, P.J.M.and Aarts. *Simulated Annealing: Theory and Applications.* Kluwer Ac. Publ, Dordrecht, 1987.

[76] S. Webb. Spect reconstruction by simulated annealing. *Phys. Med. Biol.*, 34(3):259–281, 1989.

[77] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 26(1):29–41, 1996.

[78] T. Stützle and H. Hoos. The max-min ant system and local search for the traveling salesman problem. In *Proceedings of ICEC'97 - 1997 IEEE 4th International Conference on Evolutionary Computation*, pages 308–313. IEEE Press, 1997.

[79] T. Stützle and H. Hoos. Improvements on the ant system: Introducing the max-min ant system. In *ICANNGA97 - Third International Conference on Artificial Neural Networks and Genetic Algorithms*, University of East Anglia, Norwich, UK, 1997.

[80] F. Glover and M. Laguna. *Modern Heuristic Techniques for Combinatorial Problems*. Blackwell Scientific Publications, Oxford, 1993.

[81] M. Laguna, J.P. Kelly, J.L. González Velarde, and F. Glover. Tabu search for the multilevel generalized assignment problem. *European Journal of Operational Research*, 82:176–189, 1995.

[82] R.W. Becker and G.V. Lago. A global optimization algorithm. In *In Proceedings of the 8th Allerton Conference on Circuits and Systems Theory*, pages 3–12, 1970.

[83] A.A Törn. Cluster analysis as a tool in a global optimization model. In *In Proceedings of Third International Congress of Cybernetic and Systems*, pages 249–260, Bucharest, 1977. Springer Verlag.

[84] A.A Törn. Cluster analysis using seed points and density-determined hyperspheres with an application to global optimization. In *IEEE trans. on Systems, Man and Cybernetics*, volume 7, pages 610–616, 1977.

Appendix A

The extended Dixon-Szegö test set

Problems G1 and G2 (Griewank G1 and G2 functions, respectively) [48].

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} x_i^2/d - \prod_{i=1}^{n} \cos\left(x_i/\sqrt{i}\right) + 1.$$

For G1, $n = 2$ and $d = 200$; for G2, $n = 10$ and $d = 4000$.

SEARCH DOMAIN FOR G1:

$$D = \{(x_1, x_2) \in R^2 : -100.0 \leq x_i \leq 100.0, \ \ i = 1, 2\}.$$

SEARCH DOMAIN FOR G2:

$$D = \{(x_1, x_2, \cdots, x_{10}) \in R^{10} : -600.0 \leq x_i \leq 600.0, \ \ i = 1, 2, \cdots, 10\}.$$

SOLUTION:

$$\boldsymbol{x}^* = (0.0, \cdots, 0.0) \quad f^* = 0.0.$$

Problem GP (Goldstein-Price) [42]. ODL.B6-28

OBJECTIVE FUNCTION:

$$\begin{aligned} f(\boldsymbol{x}) = \ & [1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)] \times \\ & [30 + (2x_1 - 3x_2)^2 (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2)]. \end{aligned}$$

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -2.0 \leq x_i \leq 2.0, \ \ i = 1, 2\}.$$

SOLUTION:

$$\boldsymbol{x}^* = (0.0, -1.0) \quad f^* = 3.0.$$

Problem C6 (Six-hump Camelback) [49]. ODL.B2-7

OBJECTIVE FUNCTION:

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

SEARCH DOMAIN:

$$D = \{x_1 \in R^1 : -3.0 \leq x_1 \leq 3.0\}$$
$$D = \{x_2 \in R^1 : -2.0 \leq x_2 \leq 2.0\}$$

SOLUTION:

$$\boldsymbol{x}_1^* = (0.0898, -0.7126) \quad \boldsymbol{x}_2^* = (-0.0898, 0.7126) \quad f^* = -1.0316285$$

Problem SH (Shubert function, Levi no. 4) [49].

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = \{\sum_{i=1}^{5} i\cos[(i+1)x_1 + i]\}\{\sum_{i=1}^{5} i\cos[(i+1)x_2 + i]\}$$

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -10.0 \leq x_i \leq 10.0, \ \ i = 1, 2\}$$

SOLUTION:

$$\boldsymbol{x}_1^* = (5.48289, -1.426531) \quad f^* = -186.73091$$

Problem RA (Rastrigin) [50]

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = x_1^2 + x_2^2 - \cos(18x_1) - \cos(18x_2)$$

SEARCH DOMAIN:

$$D = \{(x_1, x_2) \in R^2 : -1.0 \leq x_i \leq 1.0, \ \ i = 1, 2\}.$$

SOLUTION:

$$\boldsymbol{x}^* = (0.0, 0.0) \quad f^* = -2.0$$

Problem BR (Branin) [51]

OBJECTIVE FUNCTION:

$$f(\boldsymbol{x}) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1 - \frac{1}{8\pi}\right)\cos(x_1) + 10$$

SEARCH DOMAIN:

$$D = \{x_1 \in R^1 : -5.0 \leq x_1 \leq 10.0\}$$
$$D = \{x_2 \in R^1 : 0.0 \leq x_2 \leq 15.0\}$$

SOLUTION:

$$\boldsymbol{x}_1^* \approx (3.142, 2.275) \quad f^* \approx 0.398$$

Problem H3, H6 (Hartman 3, 6) [42]

OBJECTIVE FUNCTION:

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

where $x = (x_1, \ldots, x_n)$, and

H3 : $m = 4,\ n = 3$

i	a_{ij}			c_i	p_{ij}		
1	3.0	10.0	30.0	1.0	0.3689	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.4699	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.1091	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.03815	0.5743	0.8828

H6 : $m = 4,\ n = 6$

i	a_{ij}						c_i
1	10.0	3.0	17.0	3.5	1.7	8.0	1.0
2	0.05	10.0	17.0	0.1	8.0	14.0	1.2
3	3.0	3.5	1.7	10.0	17.0	8.0	3.0
4	17.0	8.0	0.05	10.0	0.1	14.0	3.2

i	p_{ij}					
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381

SEARCH DOMAIN:

$$D = \{(x_1, \ldots, x_n) \in R^n : 0.0 \leq x_i \leq 1.0,\ \ i = 1, \ldots, n\}$$

SOLUTIONS:

H3:

$$x^* = (0.11461478, 0.55564892, 0.85254688),\ \ f^* = -3.8627821.$$

H6:

$$x^* = (0.20168955, 0.15000963, 0.47687211, 0.27533377, 0.31165102, 0.65730111),$$

$$f^* = -3.322368.$$

Problem S5, S7, S10 (Shekel 5, 7, 10) (SQRIN) [42]

OBJECTIVE FUNCTION:

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

where:

i	a_i				c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.3
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

SEARCH DOMAIN:

$$D = \{(x_1, \ldots, x_4) \in R^4 : 0.0 \leq x_i \leq 10.0,\ i = 1, \ldots, 4\}.$$

SOLUTIONS:

S5:

$$x^* = (4.00003727, 4.00013375, 4.00003730, 4.00013346)\ \ f^* = -10.153200.$$

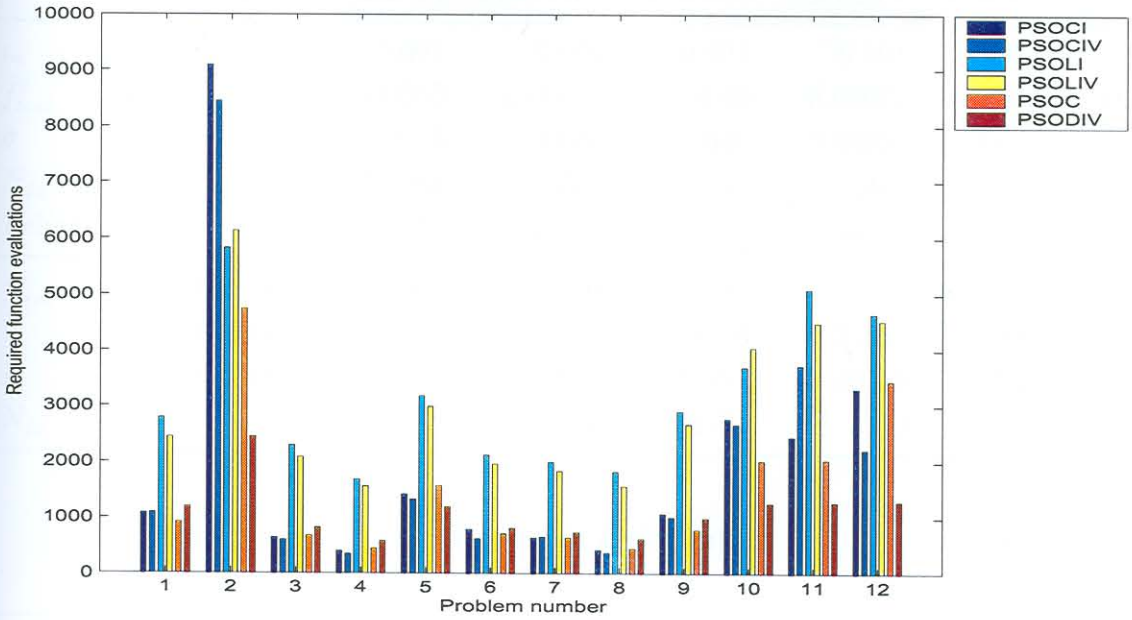
S7:

$$x^* = (4.00057280, 4.00069020, 3.99948997, 3.99960620)\ \ f^* = -10.402941.$$

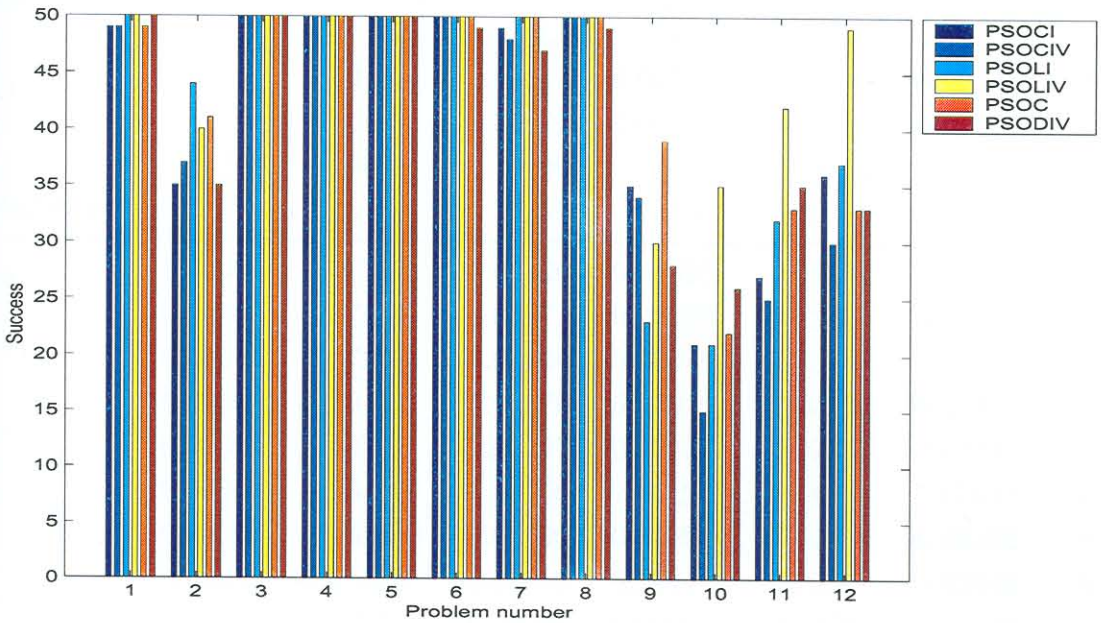
S10:

$$x^* = (4.00074671, 4.00059326, 3.99966290, 3.99950981)\ \ f^* = -10.536410.$$

A.1 Numerical results for the extended Dixon-Szegö test set



(a) Cost for the different variants



(b) Reliability for the different variants

Figure A.1: Cost and reliability comparison for the different variants

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		0.00053	0.00054	0.00049	0.00053	0.00054	0.00050
$\bar{\sigma}$		0.00025	0.00031	0.00030	0.00027	0.00032	0.00032
N_{fe} (ave.)		1081	1089	2789	2443	918	1197
Reliability		49/50	49/50	50/50	50/50	49/50	50/50
f^g_{best}	0.00000	0.00007	0.00001	0.00002	0.00000	0.00003	0.00000
x_1	0.00000	-0.00223	0.00099	0.00626	0.00091	0.00387	0.00023
x_2	0.00000	0.01617	-0.00643	-0.00361	0.00119	0.00944	-0.00066
N_{fe}		842	2093	4171	2512	831	932

Table A.1: Griewank 1

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.1	0.1	0.1	0.1	0.1	0.1
$\bar{f}^g_{best}$		0.09439	0.09514	0.09550	0.09493	0.09375	0.09501
$\bar{\sigma}$		0.00497	0.00622	0.00726	0.00581	0.00725	0.00943
N_{fe} (ave.)		9084	8450	5826	6131	4743	2451
Reliability		35/50	37/50	44/50	40/50	41/50	35/50
f^g_{best}	0.00000	0.07962	0.07326	0.06626	0.07801	0.06853	0.05242
x_1	0.00000	-3.07656	0.06871	-0.07063	-3.12687	-6.25902	-0.08723
x_2	0.00000	0.07963	-0.00104	0.00018	4.75444	-4.43530	4.45253
x_3	0.00000	-10.9494	5.42670	5.43029	0.03736	5.45368	0.00068
x_4	0.00000	0.08763	-12.5407	-0.02418	6.07357	-0.17799	0.00950
x_5	0.00000	-0.25870	0.10766	-7.00124	-0.14234	7.17090	0.12157
x_6	0.00000	0.19254	-0.04966	0.02699	-7.69208	0.00288	0.00710
x_7	0.00000	-8.25078	-0.05076	8.20641	0.05870	0.03646	-0.07244
x_8	0.00000	-0.02052	0.00009	0.00001	-0.12609	-0.11542	-0.08604
x_9	0.00000	-0.35799	9.39031	0.08981	0.20924	-0.17104	0.01753
x_{10}	0.00000	0.3992	-0.0937	-10.046	0.4659	-9.8733	9.30478
N_{fe}		4867	11574	11806	4646	8519	2417

Table A.2: Griewank 2

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		3.00049	3.00049	3.00050	3.00050	3.00053	3.00047
$\bar{\sigma}$		0.00032	0.00027	0.00030	0.00030	0.00028	0.00029
N_{fe} (ave.)		641	602	2295	2086	679	824
Reliability		50/50	50/50	50/50	50/50	50/50	50/50
f^g_{best}	3.0000	3.00000	3.00006	3.00004	3.00001	3.00001	3.00001
x_1	0.0000	-0.00000	-0.00051	-0.00014	-0.00013	0.00020	0.00023
x_2	-1.0000	-0.99994	-1.00020	-0.99974	-0.99999	-0.99999	-0.99993
N_{fe}		537	715	2280	2105	747	904

Table A.3: Goldstein-Price

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		-1.03112	-1.03113	-1.03123	-1.03123	-1.03117	-1.03113
$\bar{\sigma}$		0.00031	0.00028	0.00027	0.00027	0.00029	0.00028
N_{fe} (ave.)		402	348	1687	1566	452	584
Reliability		50/50	50/50	50/50	50/50	50/50	50/50
f^g_{best}	-1.03163	-1.03162	-1.03160	-1.03162	-1.03162	-1.03161	-1.03162
x_1	$\pm$0.0898	-0.08906	-0.09086	-0.08921	0.09167	-0.09179	0.09126
x_2	$\mp$0.71260	0.71339	0.71428	0.71347	-0.71272	0.71221	-0.71294
N_{fe}		682	218	1796	1621	627	609

Table A.4: Six-hump Camelback

	Optimum solution	PSO variant PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		-186.730	-186.730	-186.730	-186.730	-186.730	-186.730
$\bar{\sigma}$		0.00030	0.00029	0.00029	0.00027	0.00032	0.00032
N_{fe} (ave.)		1422	1332	3184	2990	1572	1197
Reliability		50/50	50/50	50/50	50/50	50/50	50/50
f^g_{best}	-186.731	-186.730	-186.730	-186.730	-186.730	-186.730	-186.730
x_1	5.48289	-7.70838	-7.70830	-0.80040	-7.70836	-7.08355	4.85802
x_2	-1.42653	-0.80027	-0.80037	4.85812	-0.80023	-1.42511	-0.80027
N_{fe}		1372	685	3167	3295	1144	1255

Table A.5: Schubert, Levi no. 4

	Optimum solution	PSO variant PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		-1.99952	-1.99947	-1.99951	-1.99942	-1.99949	-1.99947
$\bar{\sigma}$		0.00031	0.00028	0.00031	0.00030	0.00030	0.00029
N_{fe} (ave.)		790	624	2122	1965	713	814
Reliability		50/50	50/50	50/50	50/50	50/50	49/50
f^g_{best}	-2.00000	-1.99998	-1.99998	-1.99998	-1.99999	-1.99997	-1.99999
x_1	0.00000	0.00034	-0.00024	-0.00036	0.00005	0.00044	-0.00013
x_2	0.00000	0.00004	0.00022	-0.00009	-0.00028	-0.00006	0.00027
N_{fe}		867	697	1844	1439	948	836

Table A.6: Rastrigin

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		0.39845	0.39840	0.39850	0.39848	0.39848	0.39847
$\bar{\sigma}$		0.00033	0.00034	0.00034	0.00032	0.00033	0.00030
N_{fe} (ave.)		643	658	1997	1838	645	743
Reliability		49/50	48/50	50/50	50/50	50/50	47/50
f^g_{best}	0.398	0.39790	0.39791	0.39793	0.39791	0.39791	0.39789
x_1	3.142	-1.86969	2.17809	2.18510	2.17763	2.18814	2.18117
x_2	2.275	9.42601	3.14254	3.13868	3.14319	3.14061	3.14232
N_{fe}		381	458	1986	1848	487	708

Table A.7: Branin

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		-3.86218	-3.86218	-3.86220	-3.86216	-3.86218	-3.86221
$\bar{\sigma}$		0.00028	0.00022	0.00028	0.00027	0.00025	0.00026
N_{fe} (ave.)		425	374	1827	1567	451	625
Reliability		50/50	50/50	50/50	50/50	50/50	49/50
f^g_{best}	-3.86278	-3.86270	-3.86267	-3.86275	-3.86277	-3.86272	-3.86274
x_1	0.11461	0.11882	0.10160	0.11875	0.11606	0.11021	0.12257
x_2	0.55565	0.55419	0.55604	0.55493	0.55549	0.55605	0.55531
x_3	0.85255	0.85263	0.85279	0.85240	0.85284	0.85185	0.85262
N_{fe}		343	354	1441	852	219	743

Table A.8: Hartman 3

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		-3.32159	-3.32158	-3.32161	-3.32162	-3.32161	-3.32160
$\bar{\sigma}$		0.00016	0.00016	0.00019	0.00020	0.00019	0.00017
N_{fe} (ave.)		1075	1013	2908	2675	787	997
Reliability		35/50	34/50	23/50	30/50	39/50	28/50
f^g_{best}	-3.32237	-3.32202	-3.32198	-3.32207	-3.32202	-3.32217	-3.32196
x_1	0.20169	0.20192	0.20187	0.19887	0.19938	0.20075	0.20118
x_2	0.15001	0.14500	0.15094	0.14956	0.14827	0.15239	0.15542
x_3	0.47687	0.47733	0.47345	0.48056	0.47493	0.47399	0.47839
x_4	0.27533	0.27666	0.27517	0.27563	0.27617	0.27548	0.27594
x_5	0.31165	0.31124	0.31114	0.31242	0.31345	0.31160	0.31159
x_6	0.65730	0.65719	0.65420	0.65875	0.65640	0.65618	0.65863
N_{fe}		1250	1141	3057	2425	853	1051

Table A.9: Hartman 6

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		-10.1524	-10.1525	-10.1524	-10.1524	-10.1524	-10.1525
$\bar{\sigma}$		0.00020	0.00023	0.00020	0.00019	0.00018	0.00023
N_{fe} (ave.)		2772	2672	3700	4046	2020	1262
Reliability		21/50	15/50	21/50	35/50	22/50	26/50
f^g_{best}	-10.1532	-10.1529	-10.1529	-10.1529	-10.1528	-10.1527	-10.1530
x_1	4.00004	3.99934	4.00039	3.99977	4.00137	3.99991	4.00035
x_2	4.00013	4.00098	3.99873	4.00033	4.00113	4.00111	4.00097
x_3	4.00004	3.99882	4.00037	4.00150	3.99950	3.99819	4.00060
x_4	4.00013	3.99990	4.00075	3.99942	3.99970	4.00014	4.00084
N_{fe}		1216	1293	3295	3319	972	1267

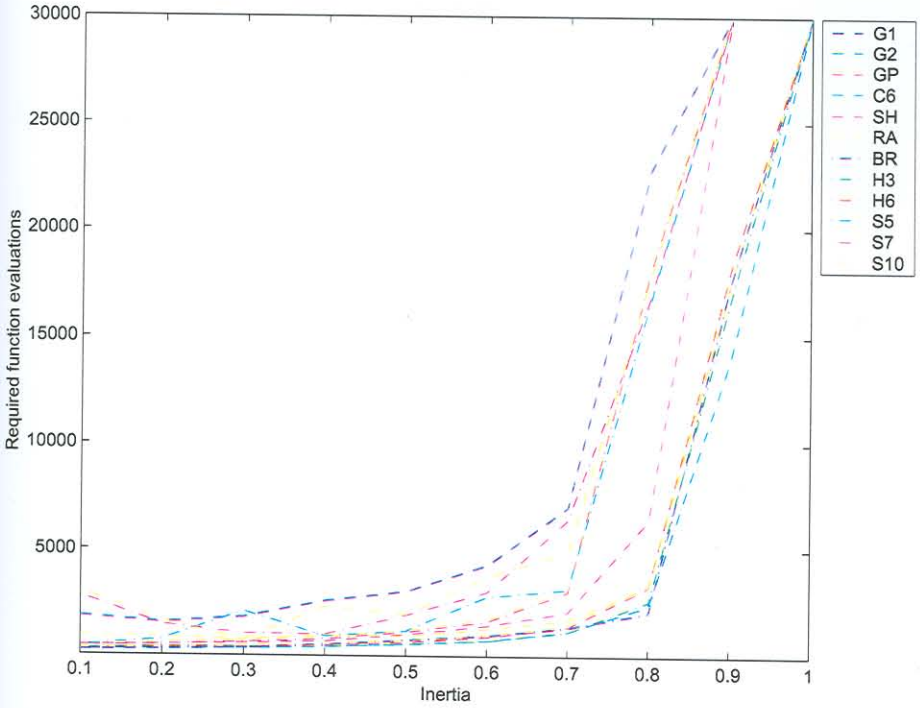
Table A.10: Shekel 5

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		-10.4022	-10.4021	-10.4022	-10.4022	-10.4021	-10.4022
$\bar{\sigma}$		0.00024	0.00018	0.00027	0.00024	0.00021	0.00023
N_{fe} (ave.)		2449	3731	5094	4497	2036	1280
Reliability		27/50	25/50	32/50	42/50	33/50	35/50
f^g_{best}	-10.4029	-10.4027	-10.4026	-10.4027	-10.4028	-10.4028	-10.4028
x_1	4.00057	4.00017	4.00175	4.00076	4.00070	4.00035	4.00033
x_2	4.00069	4.00188	4.00107	4.00160	4.00040	4.00045	4.00124
x_3	3.99949	4.00023	4.00038	3.99888	3.99945	4.00039	3.99930
x_4	3.99961	3.99952	3.99909	3.99910	3.99876	3.99915	4.00010
N_{fe}		1808	1432	3279	3156	1297	1849

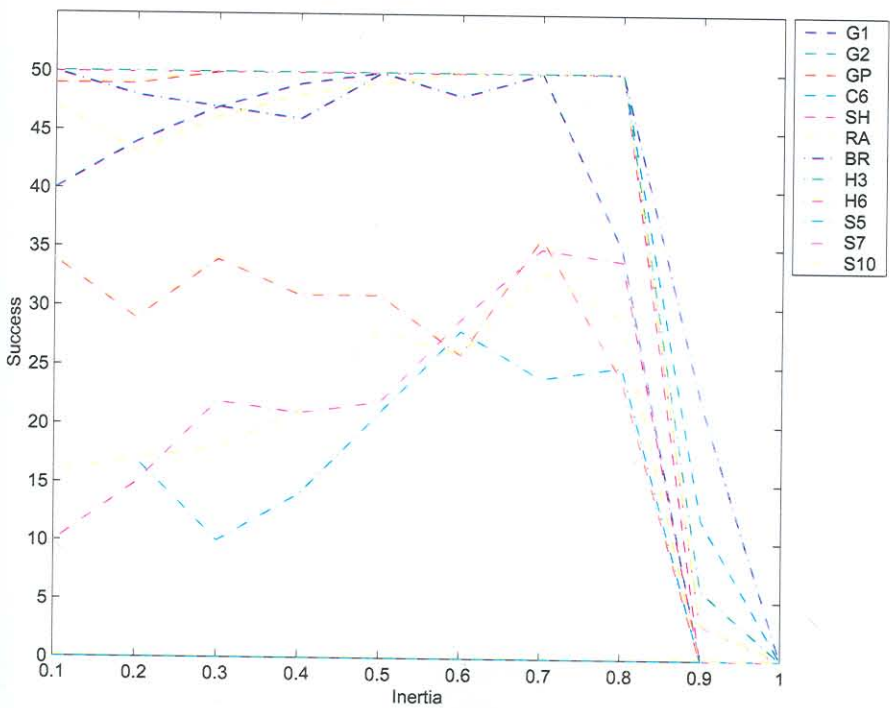
Table A.11: Shekel 7

	Optimum	PSO variant					
	solution	PSO-CI	PSO-CIV	PSO-LI	PSO-LIV	PSO-C	PSO-DIV
ϵ_a		0.001	0.001	0.001	0.001	0.001	0.001
$\bar{f}^g_{best}$		-10.5357	-10.5356	-10.5357	-10.5357	-10.5356	-10.5357
$\bar{\sigma}$		0.00025	0.00018	0.00022	0.00023	0.00023	0.00024
N_{fe} (ave.)		3317	2221	4654	4532	3451	1296
Reliability		36/50	30/50	37/50	49/50	33/50	33/50
f^g_{best}	-10.5364	-10.5363	-10.5360	-10.5362	-10.5363	-10.5361	-10.5362
x_1	4.00075	4.00120	4.00137	3.99971	4.00052	4.00173	4.00160
x_2	4.00059	4.00097	4.00119	4.00149	4.00022	4.00018	4.00021
x_3	3.99966	3.99907	3.99814	3.99958	3.99987	3.99875	4.00019
x_4	3.99951	3.99984	3.99915	3.99921	3.99886	3.99896	3.99961
N_{fe}		1392	1730	3373	2857	1212	1155

Table A.12: Shekel 10



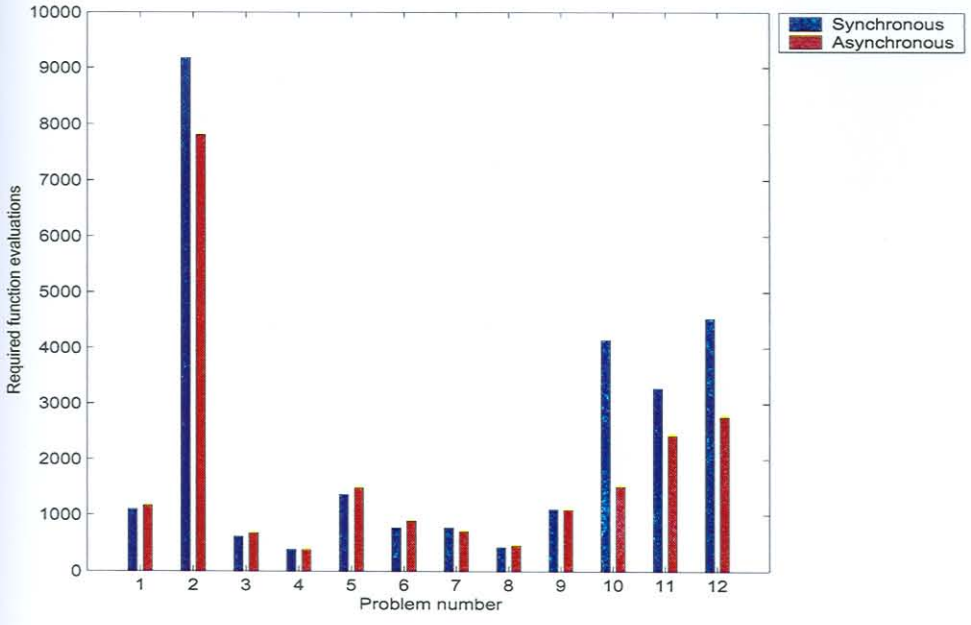
(a) Cost as a function of w



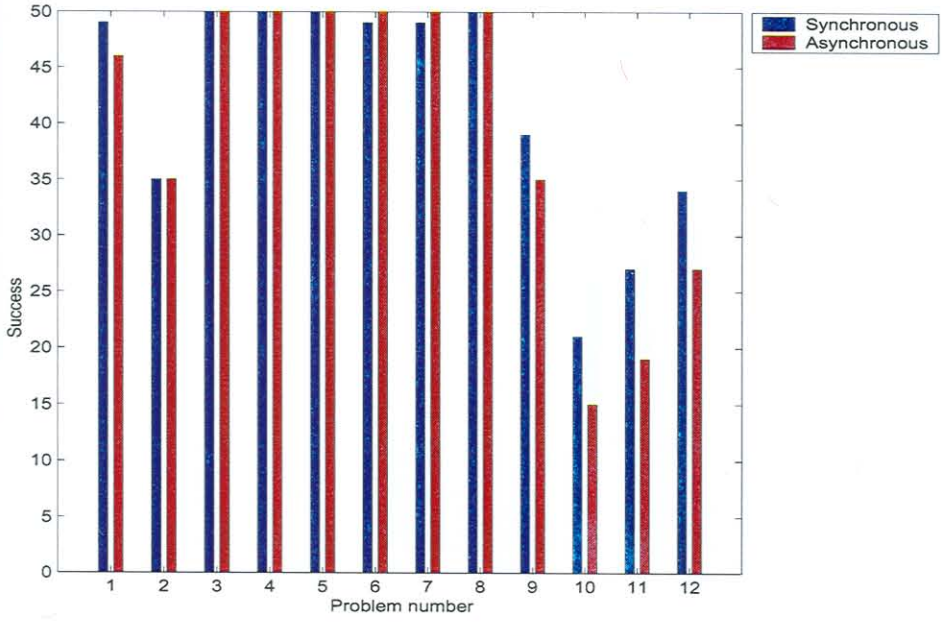
(b) Reliability as a function of w

Figure A.2: PSOA-CI variant: Cost and reliability as a function of the inertia weight w

A.2 Asyncronous vs. syncronous PSOA

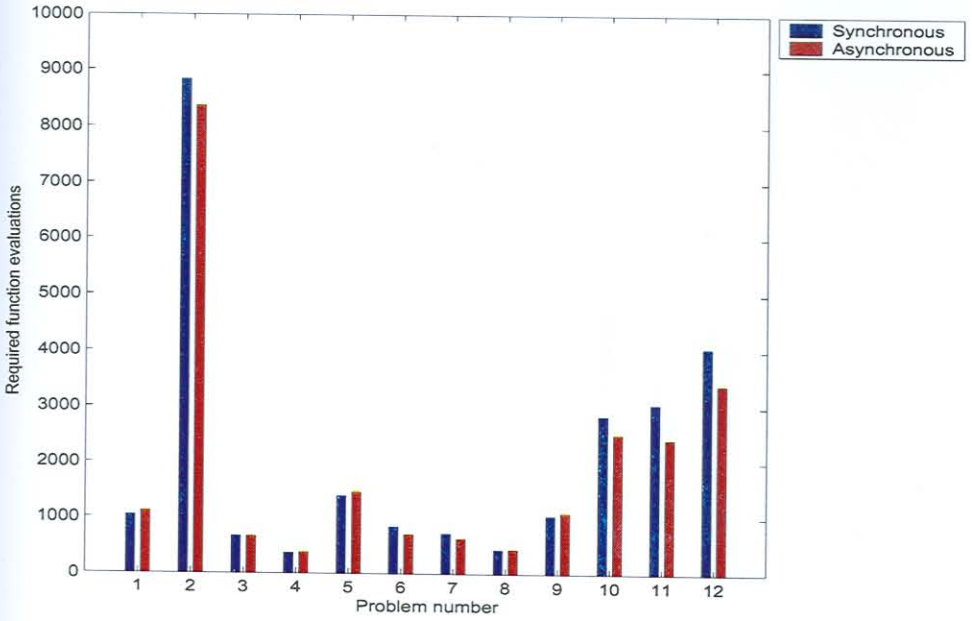


(a) Cost

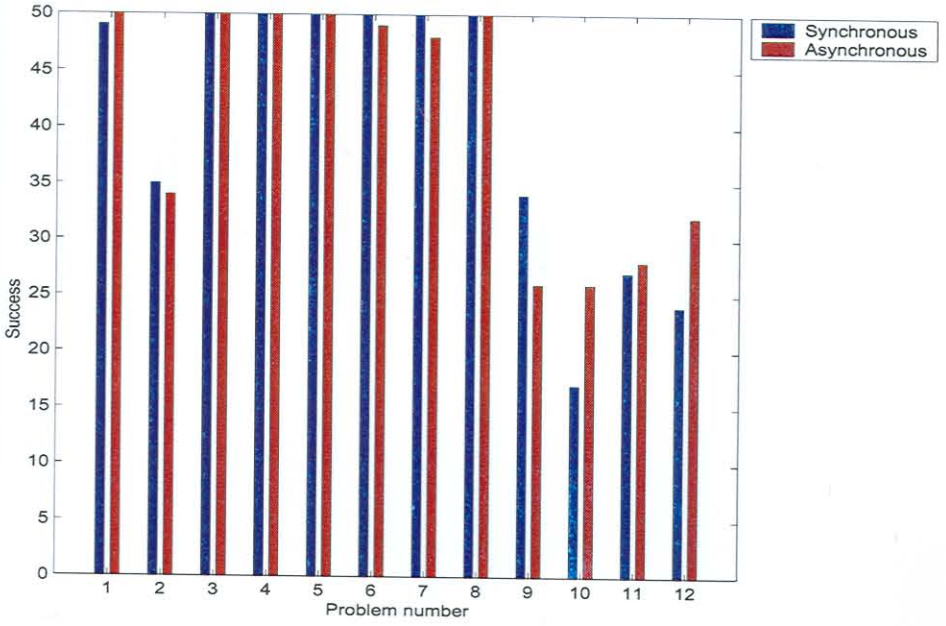


(b) Reliability

Figure A.3: PSOA-CI variant: Cost and reliability comparison bestween asynchronous and synchronous variants

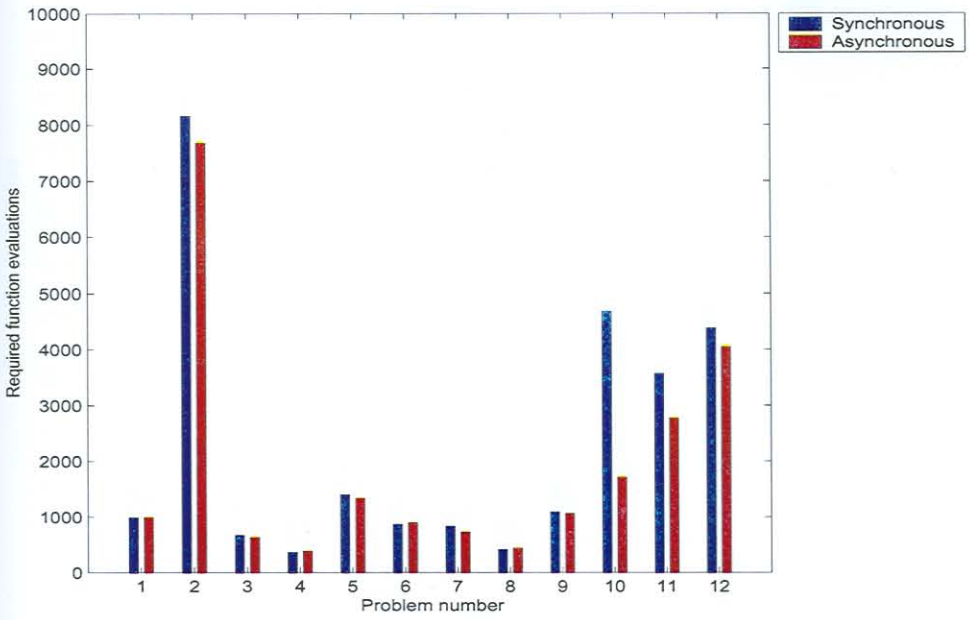


(a) Cost

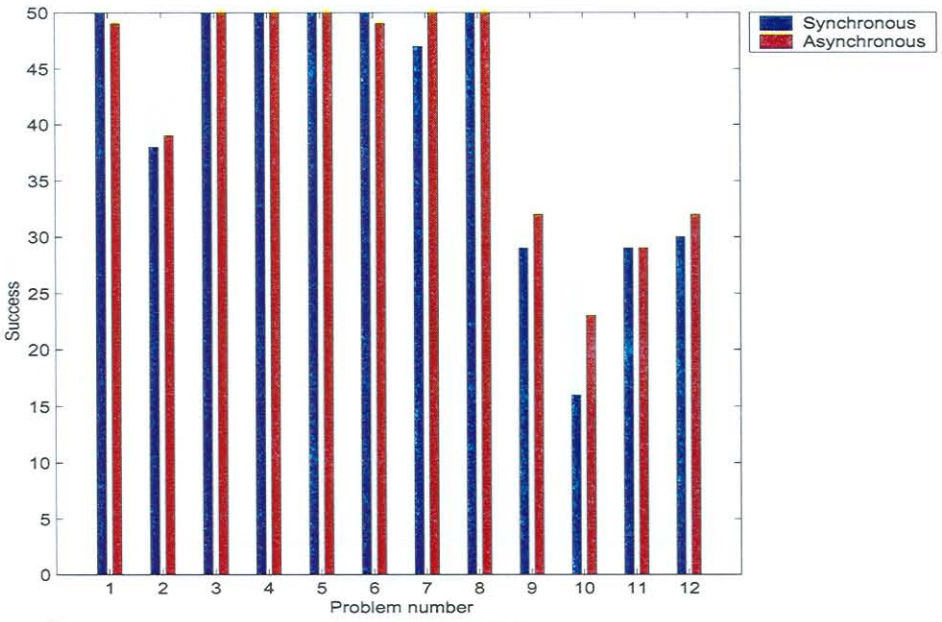


(b) Reliability

Figure A.4: PSOA-CIV variant: Cost and reliability comparison bestween asynchronous and synchronous variants

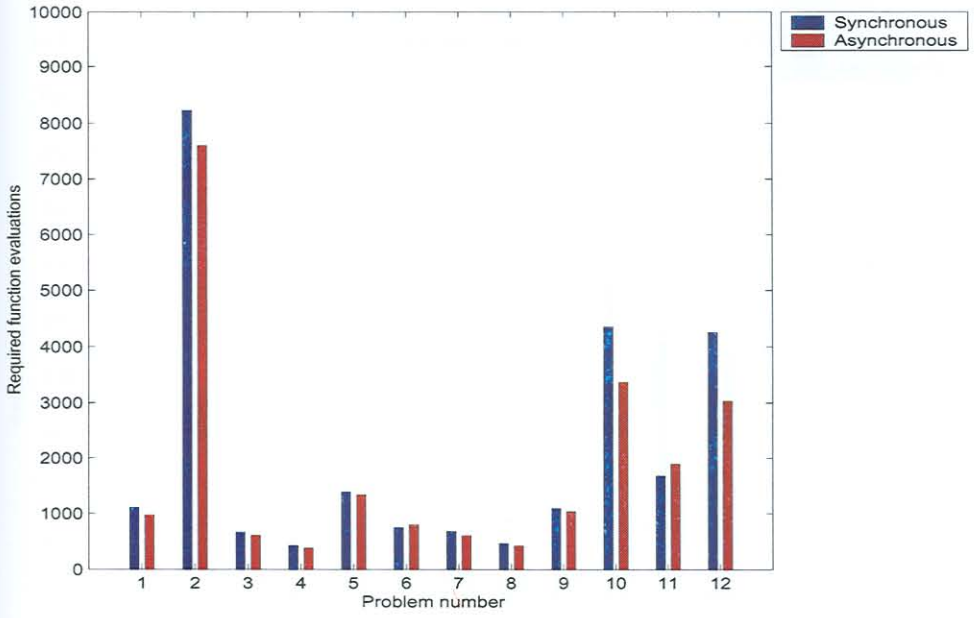


(a) Cost

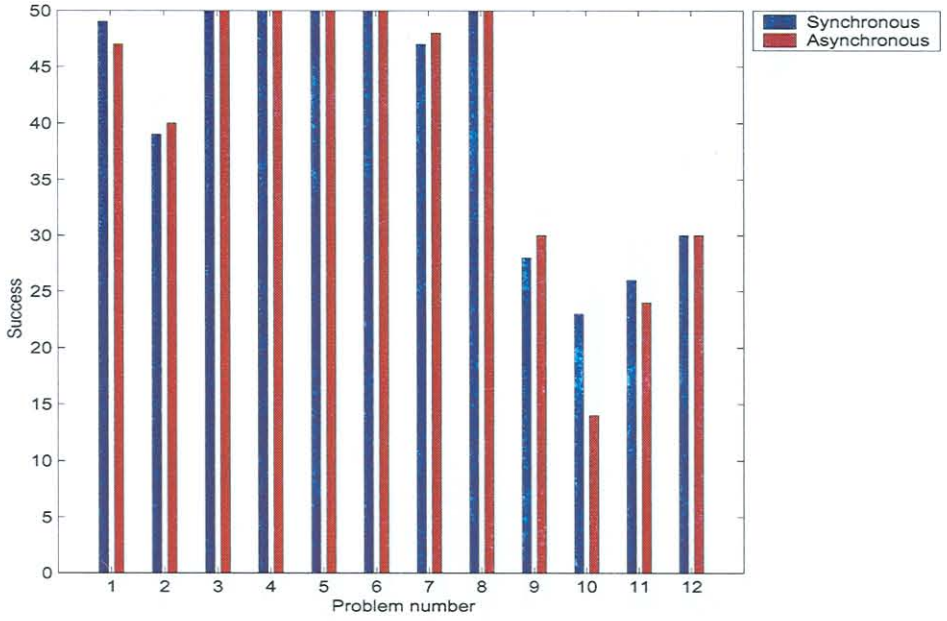


(b) Reliability

Figure A.5: PSOA-LI variant: Cost and reliability comparison bestween asynchronous and synchronous variants

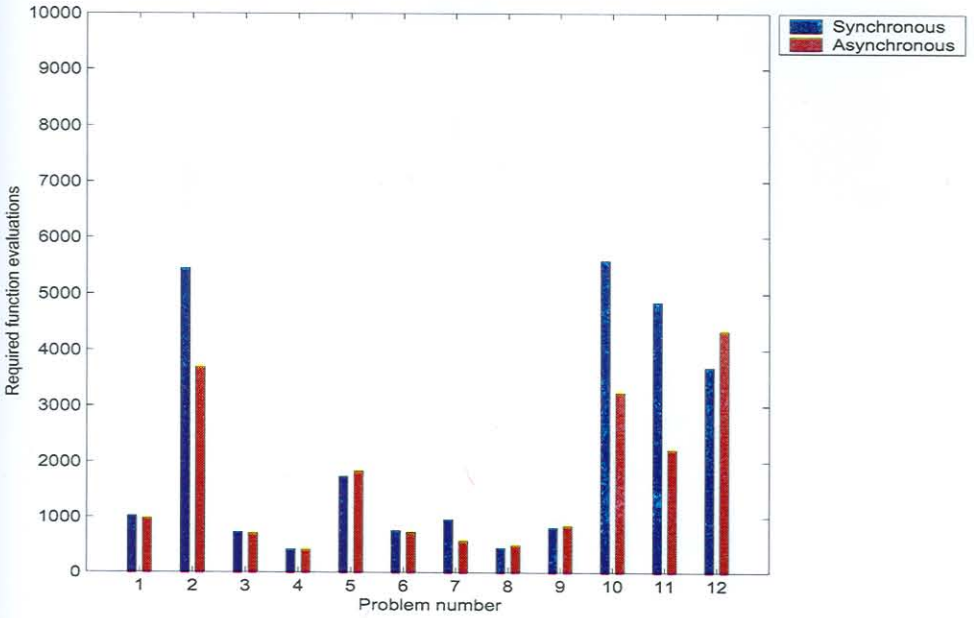


(a) Cost

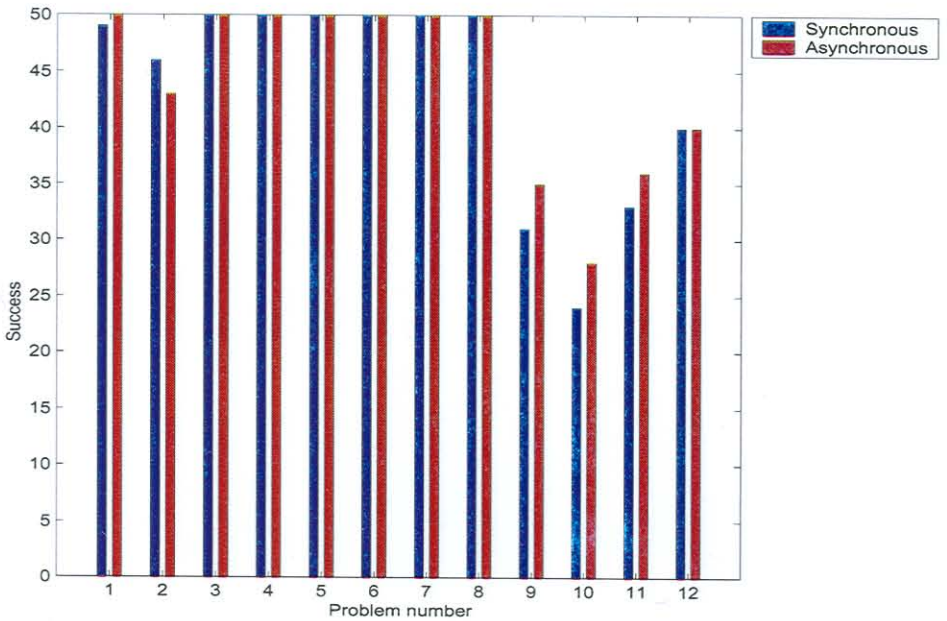


(b) Reliability

Figure A.6: PSOA-LIV variant: Cost and reliability comparison bestween asynchronous and synchronous variants

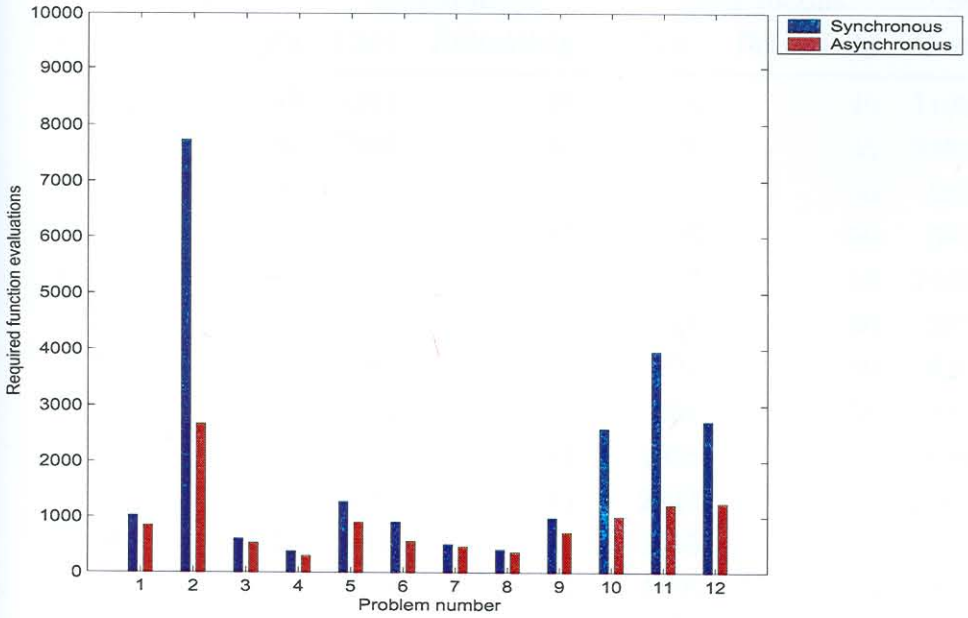


(a) Cost

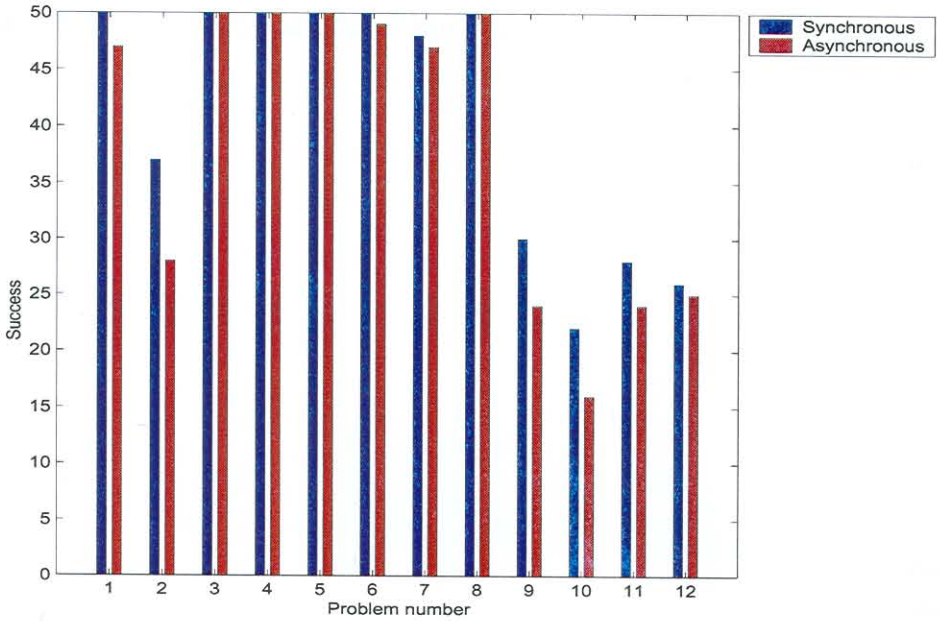


(b) Reliability

Figure A.7: PSOA-C variant: Cost and reliability comparison bestween asynchronous and synchronous variants



(a) Cost



(b) Reliability

Figure A.8: PSOA-DIV variant: Cost and reliability comparison bestween asynchronous and synchronous variants

PSO-CI				PSO-CIV			
Asyncronous		Synchronous		Asyncronous		Synchronous	
Cost	*Reliability*	Cost	*Reliability*	Cost	*Reliability*	Cost	*Reliability*
1097	49	1163	46	1036	49	1108	50
9173	35	7809	35	8831	35	8365	34
611	50	675	50	670	50	666	50
384	50	383	50	366	50	381	50
1366	50	1478	50	1389	50	1460	50
772	49	891	50	844	50	707	49
772	49	705	50	721	50	630	48
423	50	454	50	431	50	436	50
1099	39	1088	35	1035	34	1086	26
4143	21	1503	15	2844	17	2505	26
3278	27	2419	19	3055	27	2417	28
4523	34	2753	27	4056	24	3393	32

Table A.13: PSOA-CI and PSOA-CIV variants: Cost and reliability comparison for asynchronous and synchronous algorithm variants

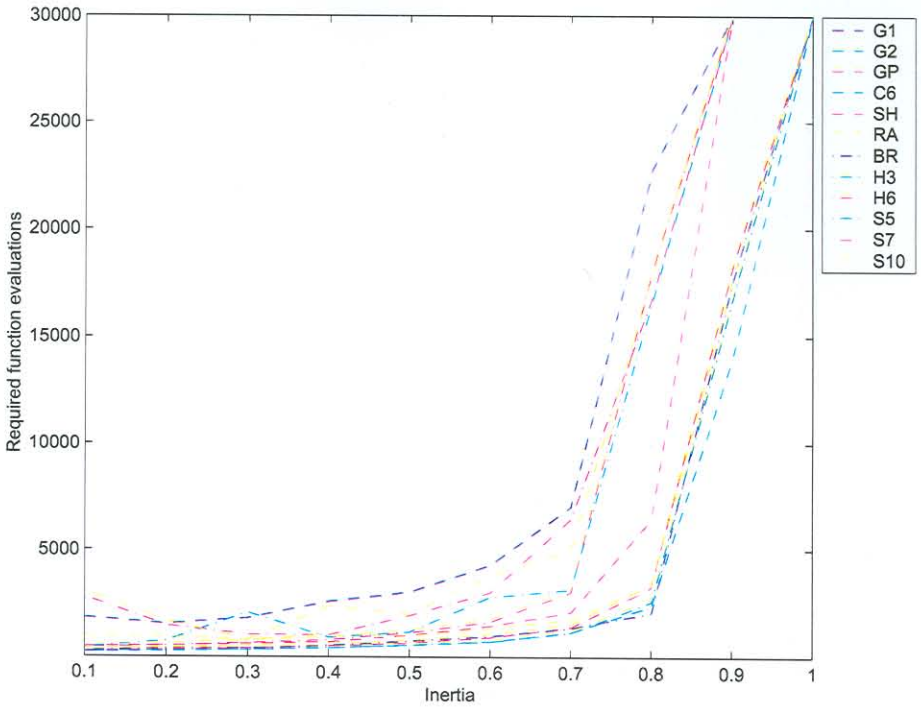
PSO-LI				PSO-LIV			
Asyncronous		Synchronous		Asyncronous		Synchronous	
Cost	*Reliability*	Cost	*Reliability*	Cost	*Reliability*	Cost	*Reliability*
989	50	995	49	1112	49	973	47
8158	38	7682	39	8229	39	7601	40
678	50	635	50	667	50	606	50
367	50	390	50	422	50	377	50
1402	50	1339	50	1389	50	1337	50
870	50	896	49	742	50	796	50
836	47	729	50	674	47	603	48
420	50	443	50	464	50	420	50
1093	29	1061	32	1093	28	1032	30
4684	16	1709	23	4351	23	3373	14
3566	29	2775	29	1678	26	1893	24
4380	30	4047	32	4256	30	3030	30

Table A.14: PSOA-LI and PSOA-LIV variants: Cost and reliability comparison for asynchronous and synchronous algorithm variants

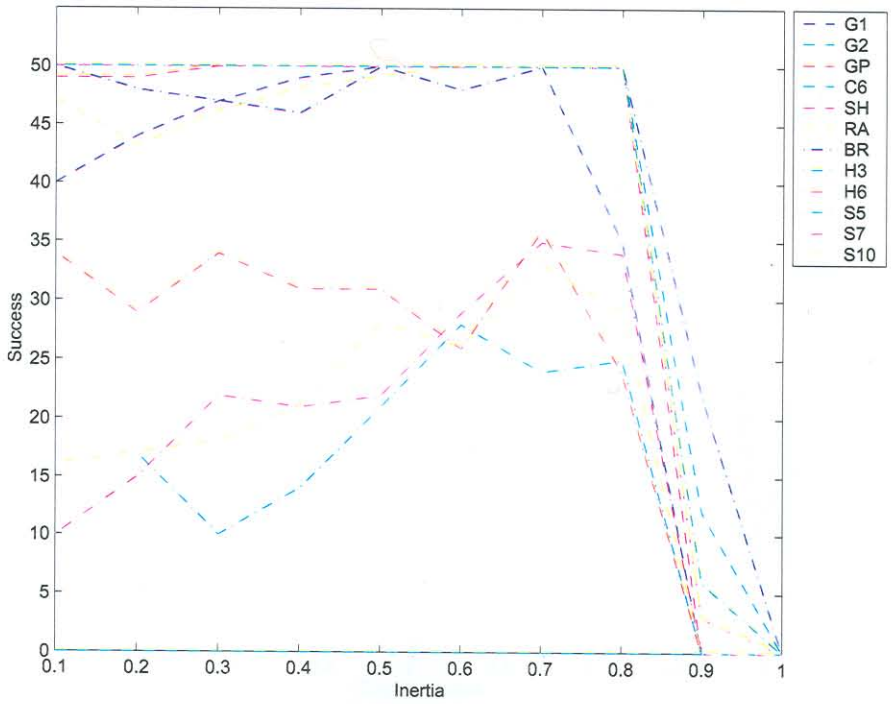
PSO-C				PSO-DIV			
Asyncronous		Synchronous		Asyncronous		Synchronous	
Cost	*Reliability*	Cost	*Reliability*	Cost	*Reliability*	Cost	*Reliability*
1021	49	978	50	1026	50	847	47
5452	46	3687	43	7728	37	2674	28
734	50	712	50	607	50	532	50
419	50	415	50	377	50	299	50
1733	50	1835	50	1276	50	905	50
758	50	731	50	909	50	559	49
959	50	574	50	506	48	468	47
449	50	493	50	409	50	362	50
821	31	845	35	986	30	722	24
5587	24	3236	28	2587	22	1008	16
4855	33	2217	36	3955	28	1215	24
3687	40	4335	40	2715	26	1246	25

Table A.15: PSOA-C and PSOA-DIV variants: Cost and reliability comparison for asynchronous and synchronous algorithm variants

A.3 Numerical results for the parameter sensitivity study

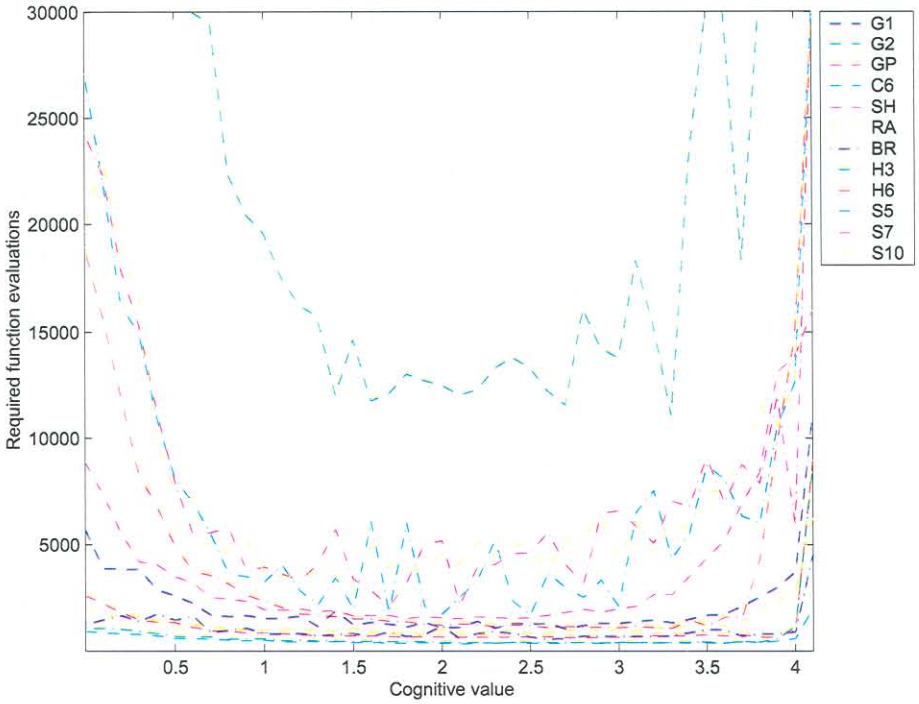


(a) Cost as a function of the inertia weight w

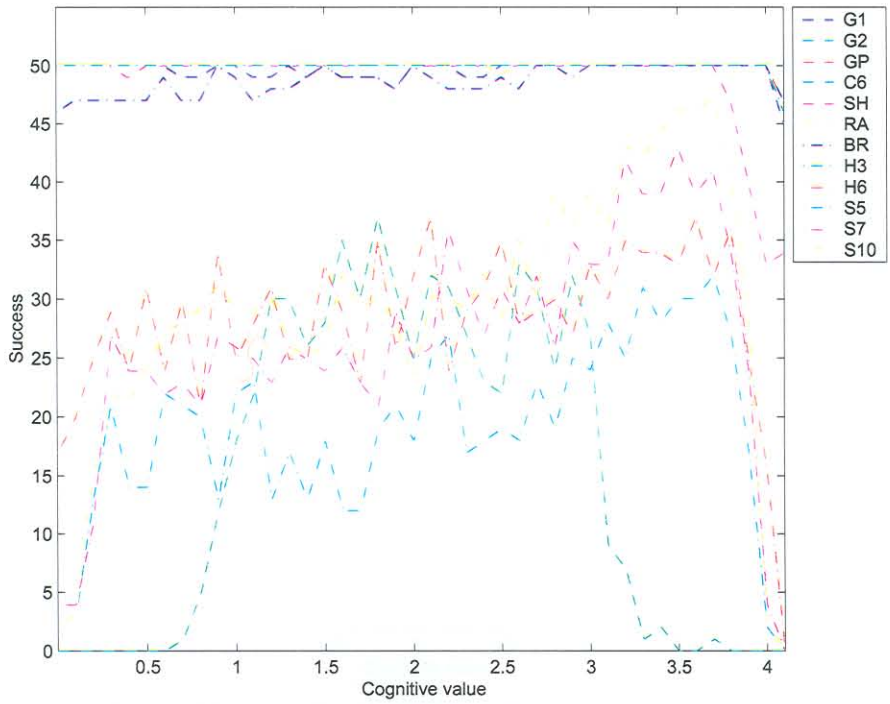


(b) Reliability as a function of the inertia weight w

Figure A.9: Constant inertia: Cost and reliability as a function of the inertia weight w

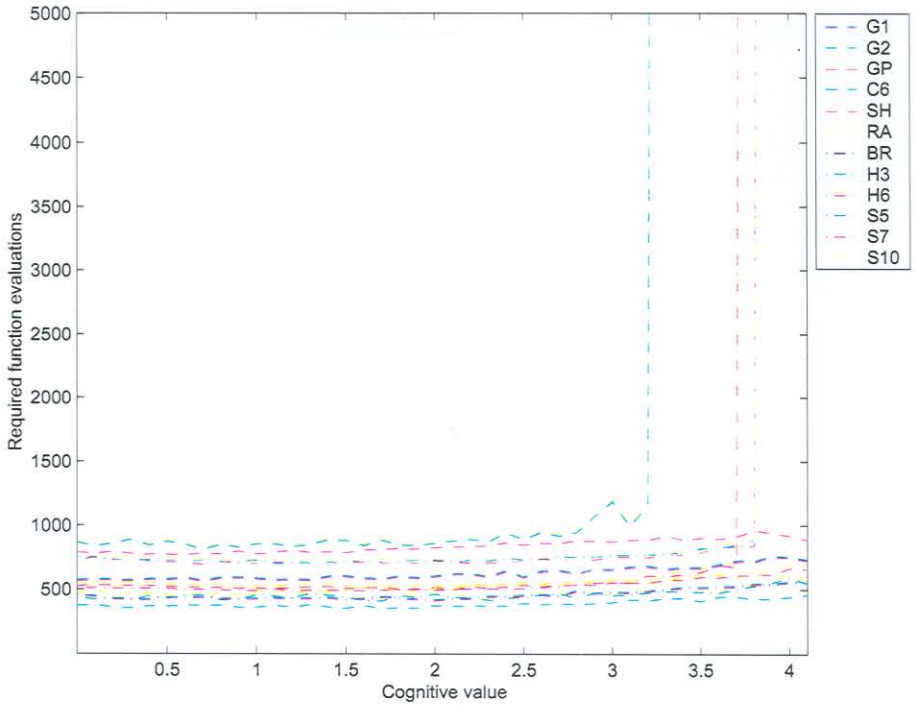


(a) Cost as a function of the cognitive parameter c_1

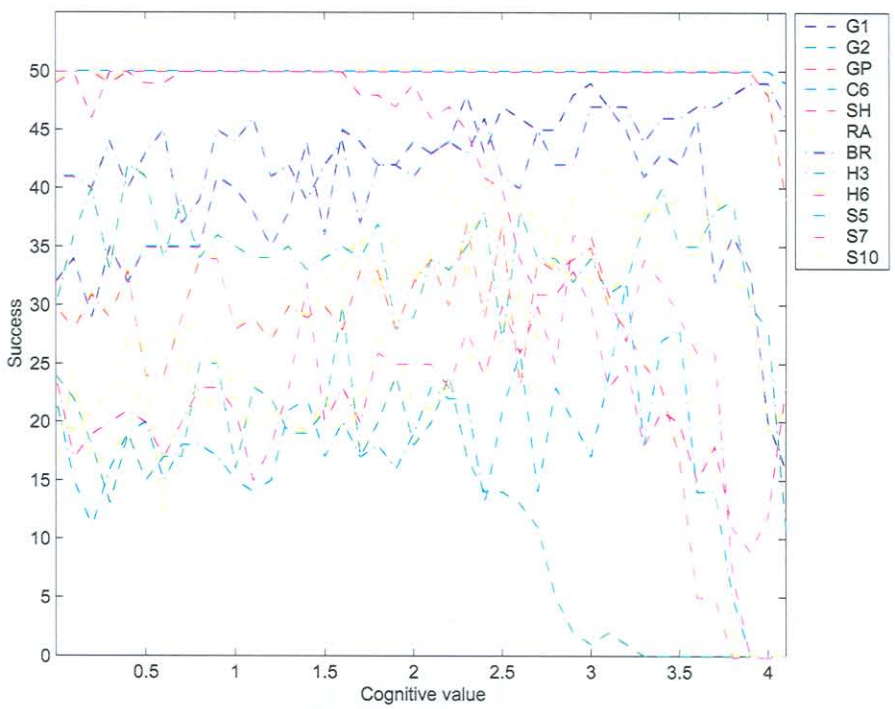


(b) Reliability as a function of the cognitive parameter c_1

Figure A.10: Constriction: Cost and reliability as a function of the cognitive parameter c_1

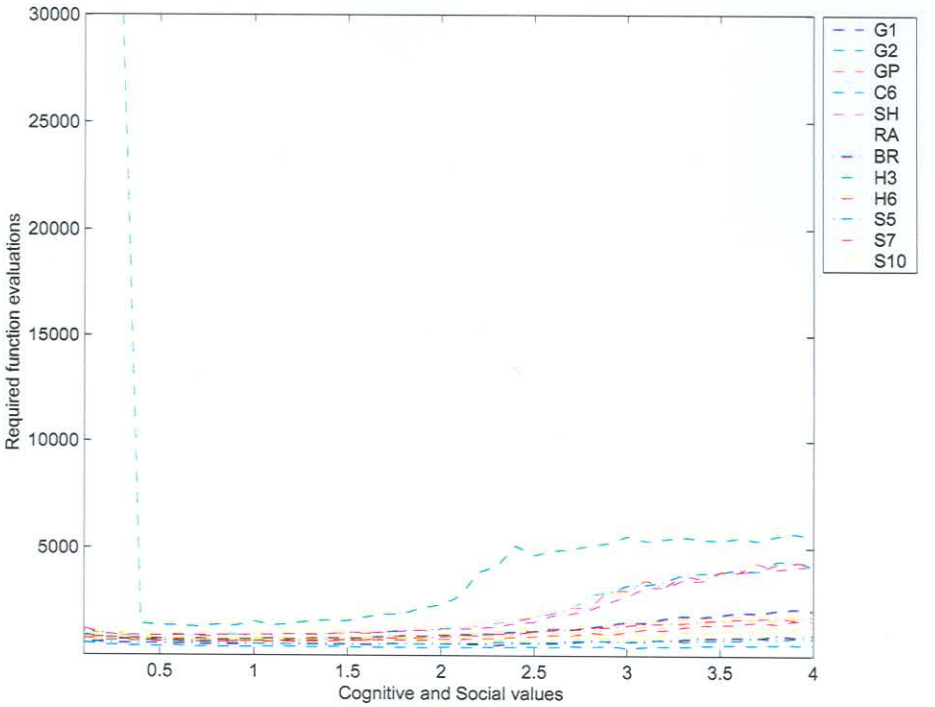


(a) Cost as a function of the cognitive parameter c_1

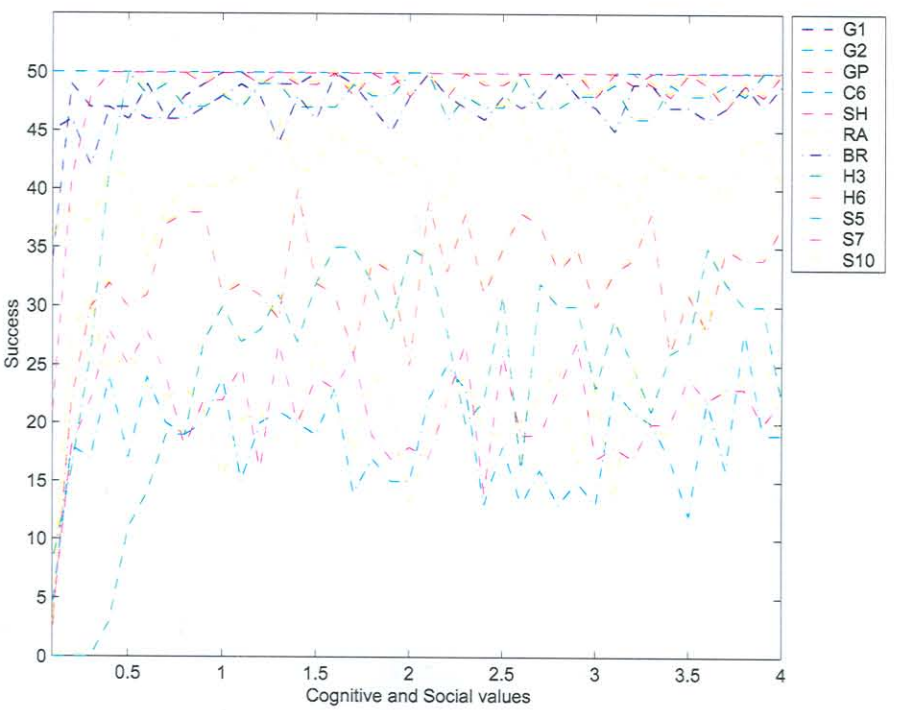


(b) Reliability as a function of the cognitive parameter c_1

Figure A.11: Dynamic inertia reduction: Cost and reliability as a function of the cognitive parameter c_1

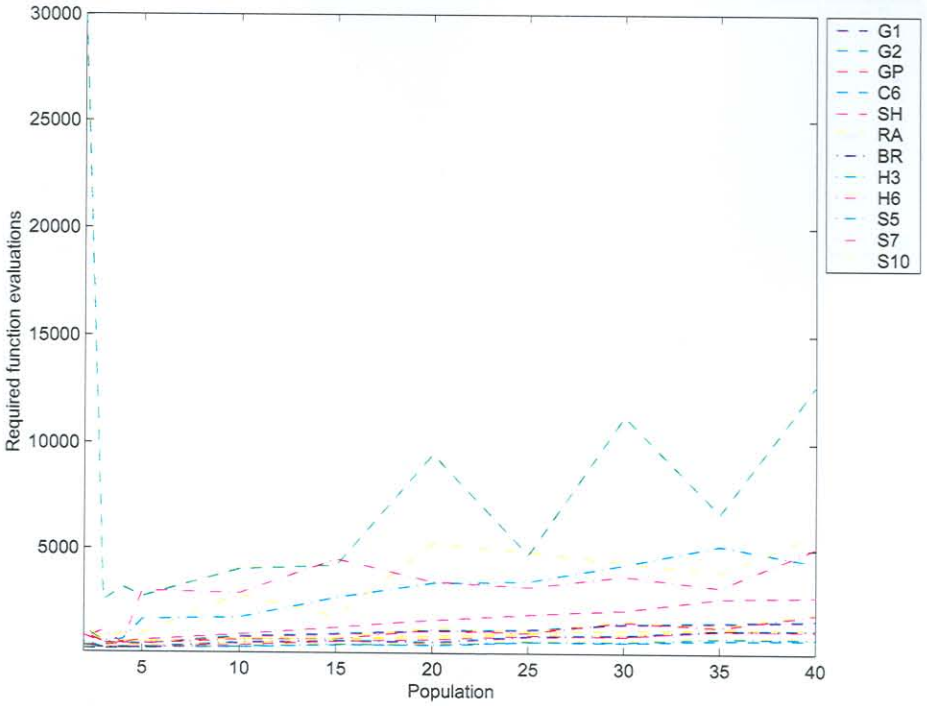


(a) Cost with $c_1 = c_2$

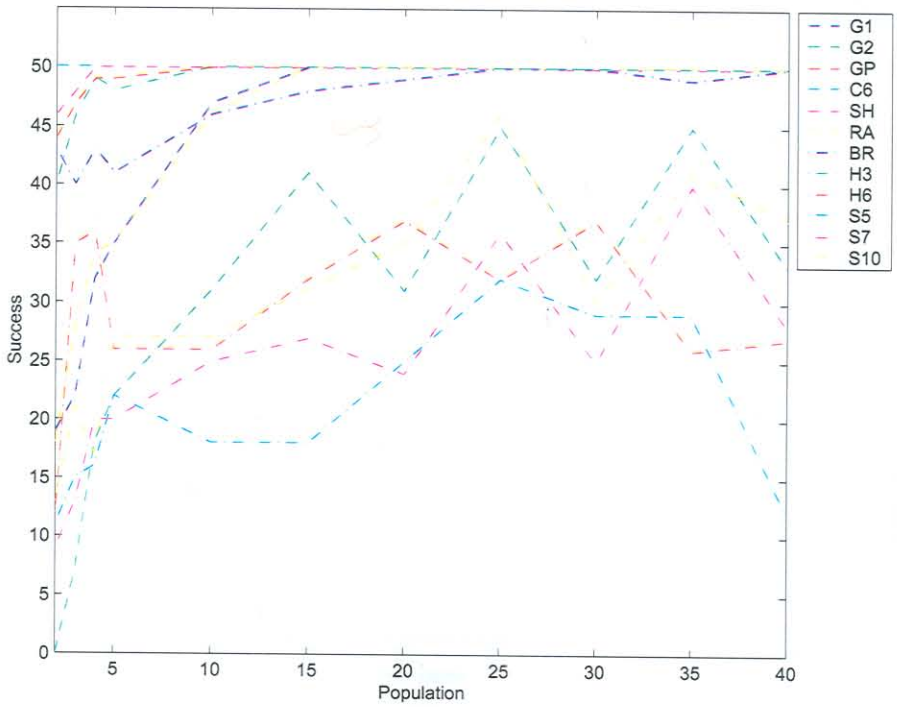


(b) Reliability with $c_1 = c_2$

Figure A.12: Dynamic inertia reduction: Cost and reliability as a function of cognitive and social parameters with $c_1 = c_2$

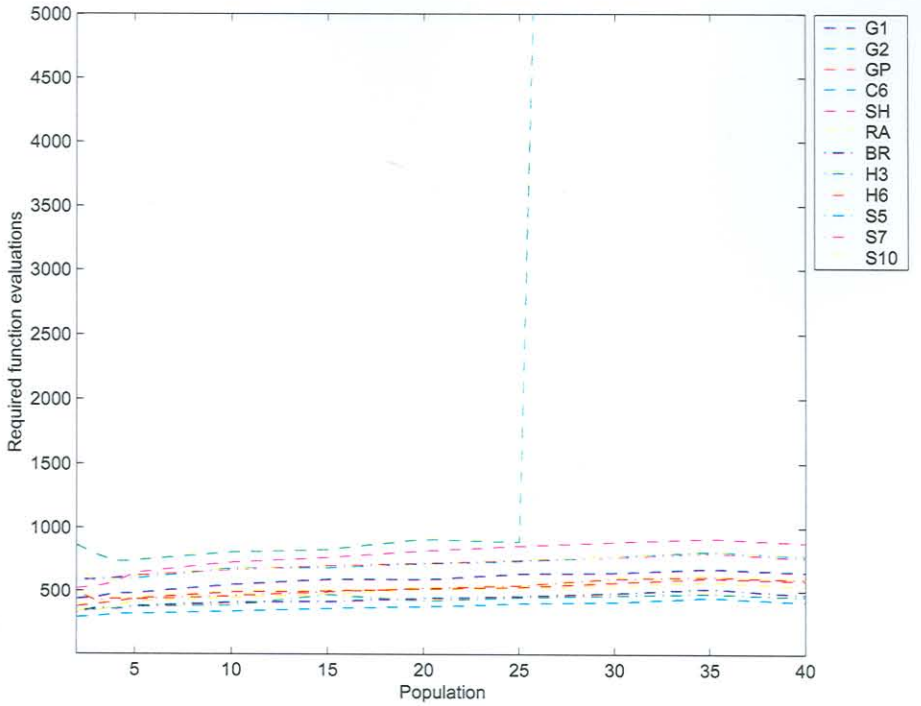


(a) Cost as a function of population size

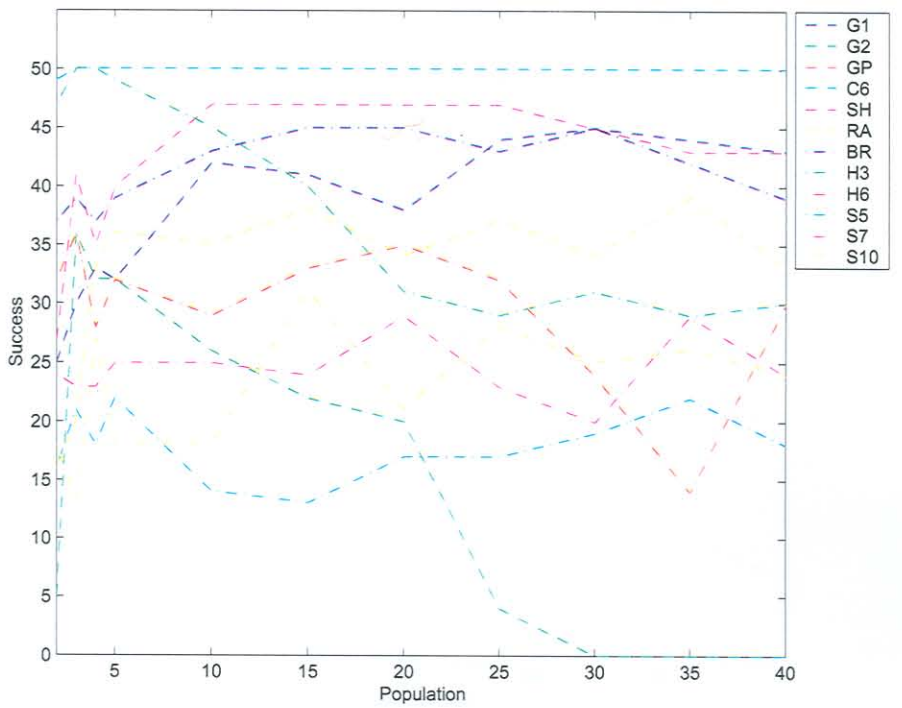


(b) Reliability as a function of population size

Figure A.13: Constriction: Cost and reliability as a function of swarm population size p

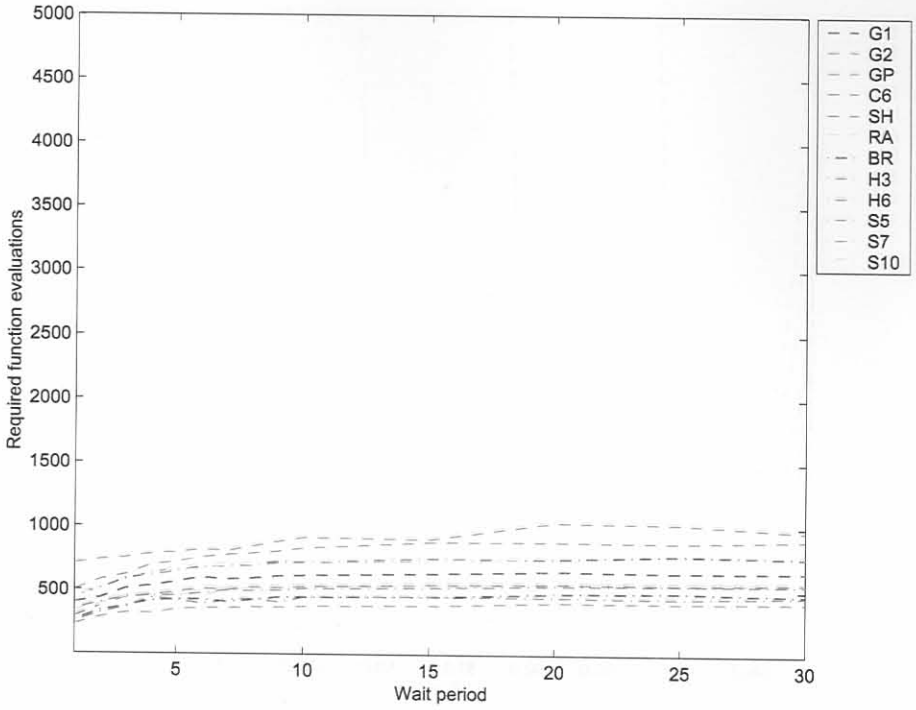


(a) Cost as a function of population size

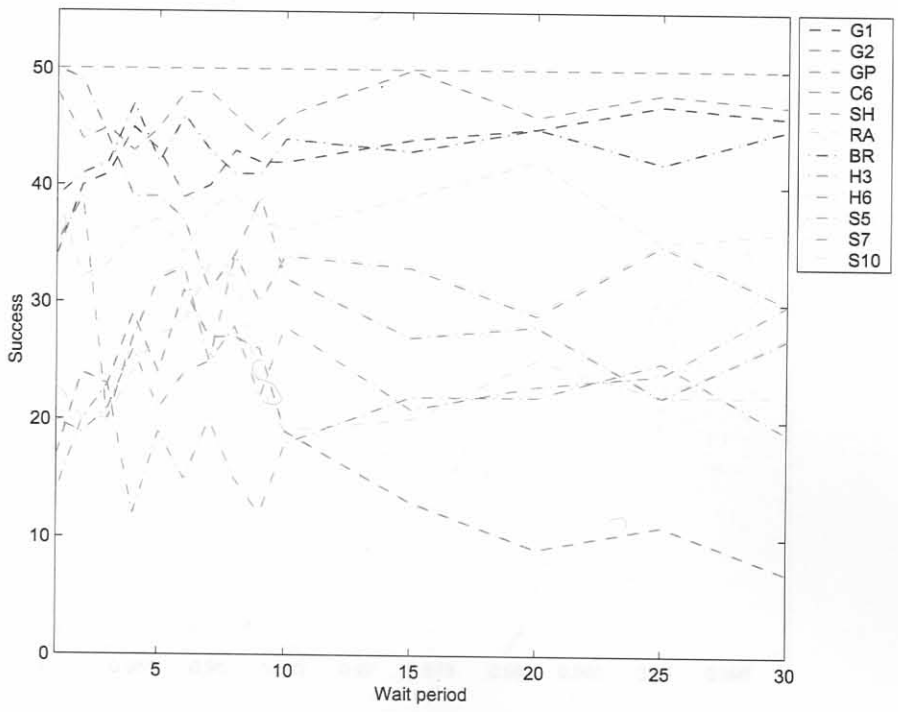


(b) Reliability as a function of population size

Figure A.14: Dynamic inertia reduction: Cost and reliability ratio as a function of swarm population p

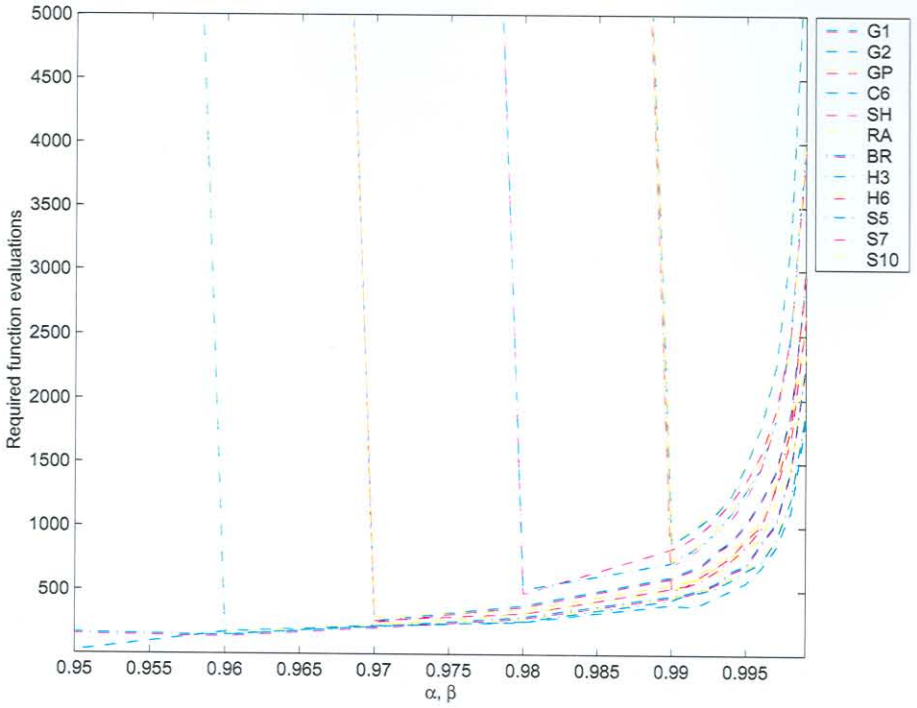


(a) Cost as a function of the dynamic delay period

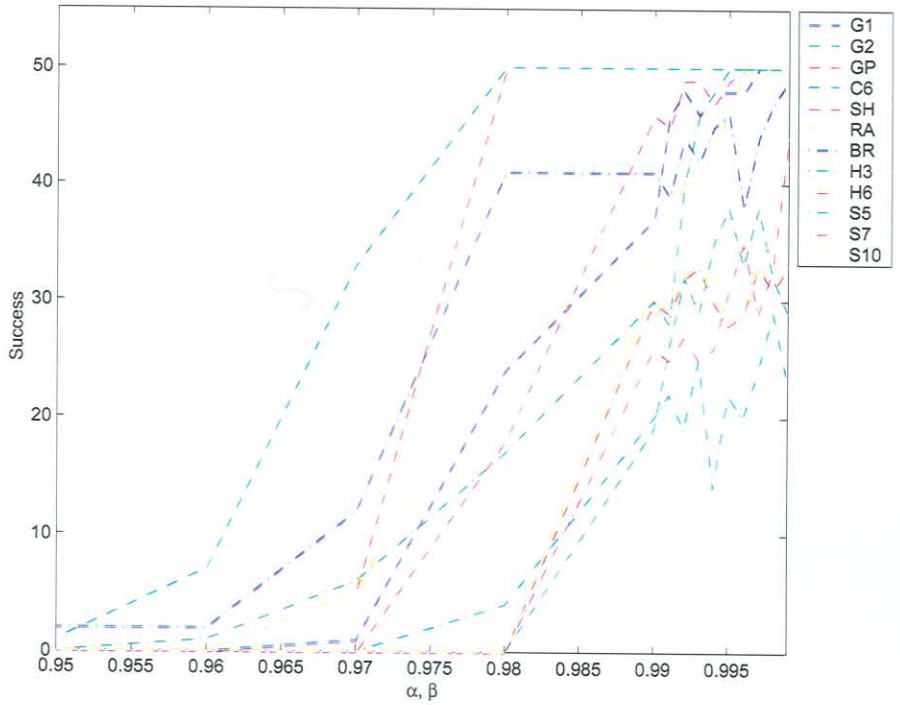


(b) Reliability as a function of the dynamic delay period

Figure A.15: Dynamic inertia reduction: Cost and reliability as a function of the dynamic delay period h

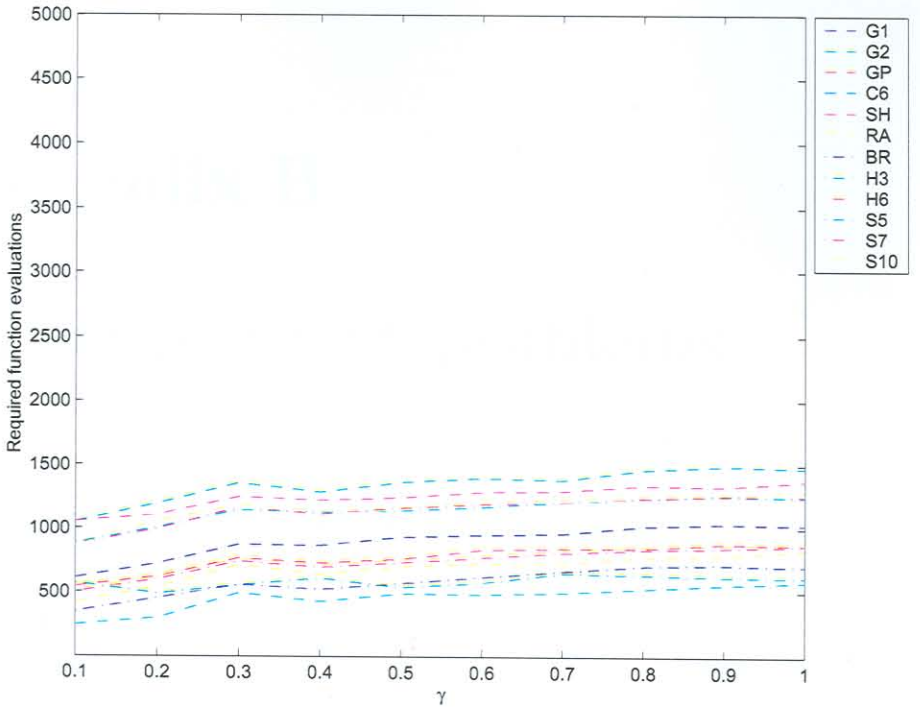


(a) Cost as a function of the reduction parameters α and β (with $\alpha = \beta$)

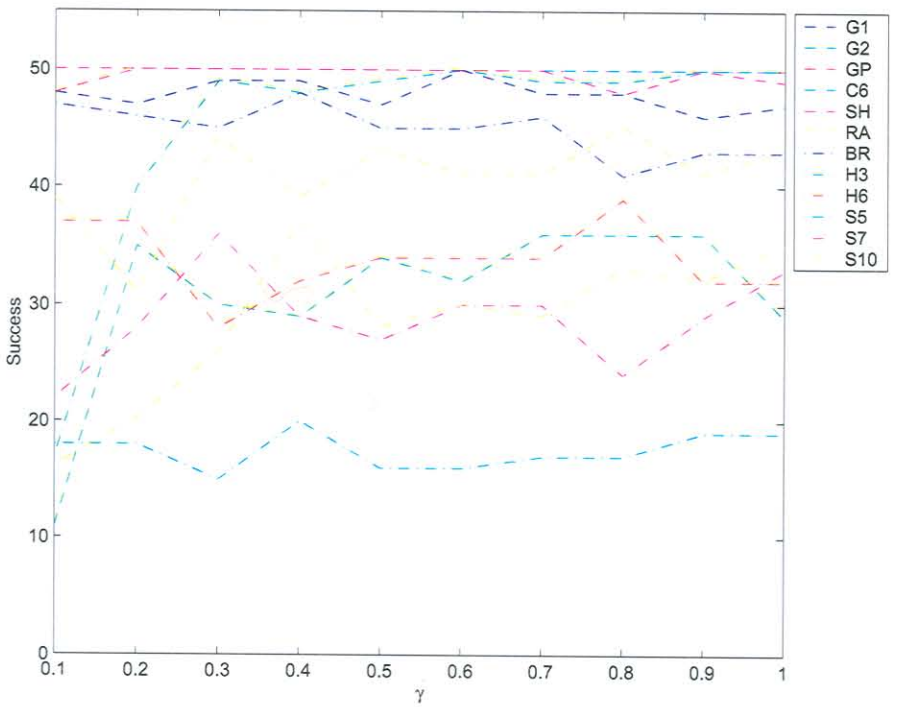


(b) Reliability as a function of the reduction parameters α and β (with $\alpha = \beta$)

Figure A.16: Dynamic inertia reduction: Cost and reliability as a function of the reduction parameters α and β



(a) Cost as a function of the initial velocity fraction γ



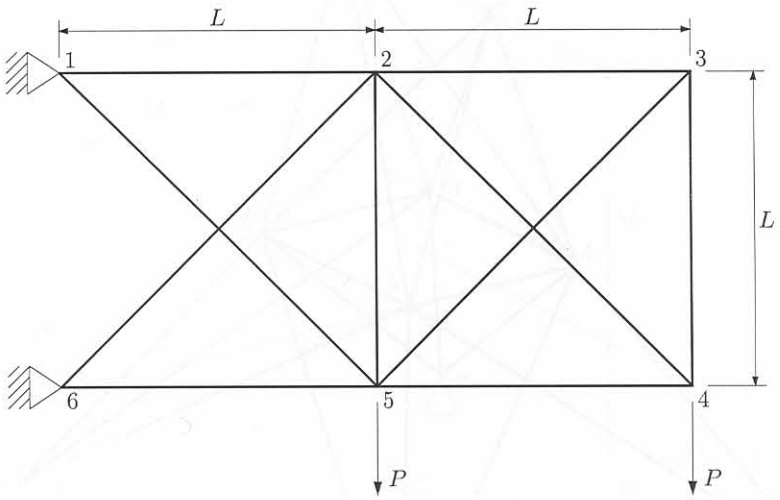
(b) Reliability as a function of the initial velocity fraction γ

Figure A.17: Dynamic inertia reduction: Cost and reliability as a function of the initial velocity fraction γ

Appendix B

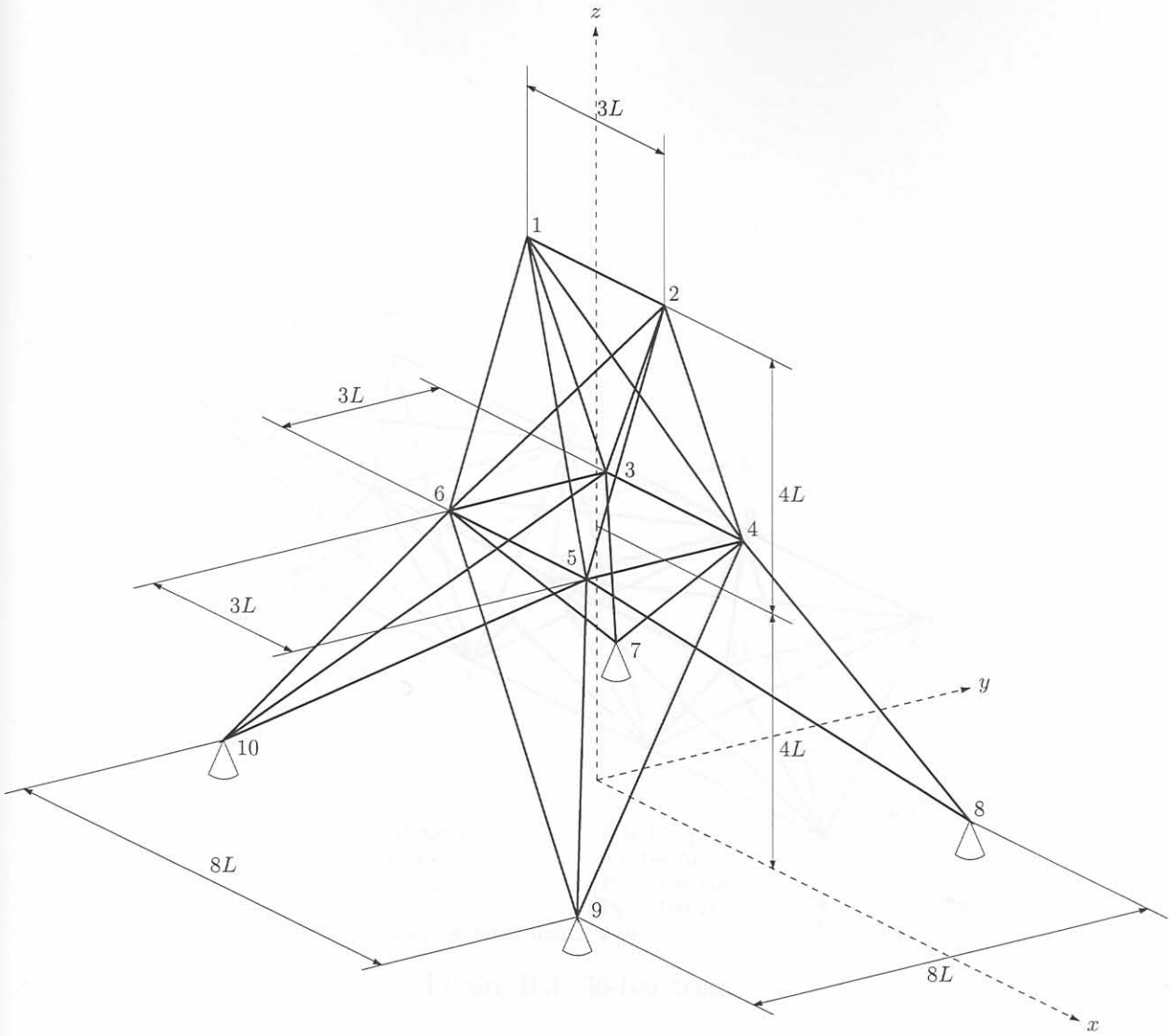
Structural test problems

B.1 Test problem geometries



Material:	$E = 10^7$ psi
Density:	0.1 lbm/in^3
Yield stress:	25000 psi
Displacement limit:	2 in
Length:	$L = 360$ in
Load:	$P = 100$ kip

Figure B.1: 10-bar truss



Material:	$E = 10^7$ psi
Density:	0.1 lbm/in^3
Stress limit:	40000 psi
Displacement limit:	0.35 in
Length:	$L = 25$ in

Figure B.2: 25-bar truss

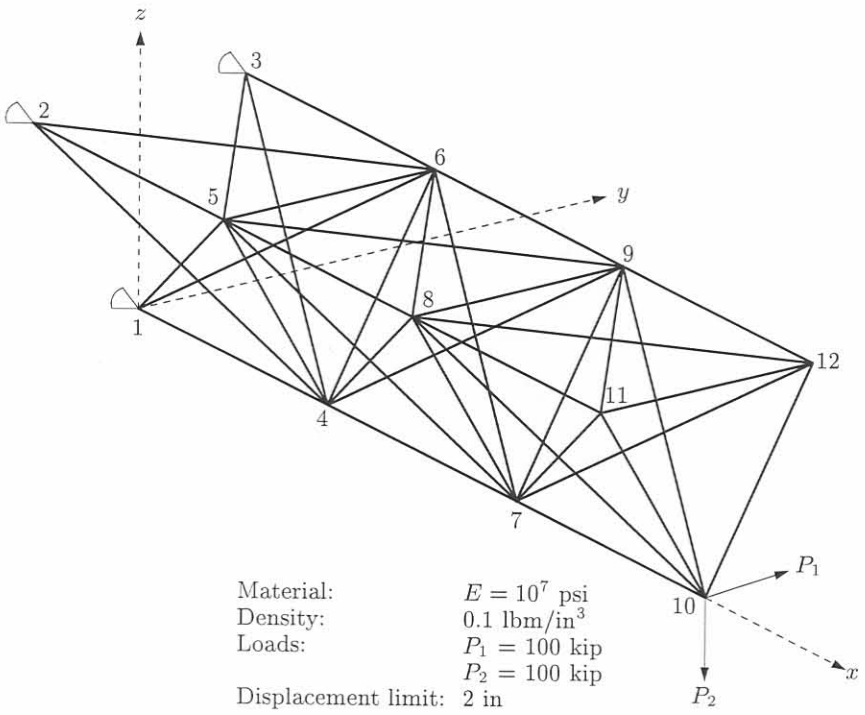


Figure B.3: 36-bar truss

B.2 Numerical results

B.2.1 Penalty constraint method without social pressure

	Optimum solution	PSO-DIV variant solution		
ϵ_s (%)		5	2	1
$\bar{f}^g_{best}$ (lbs)		4781.55	4682.07	4643.86
$\bar{\sigma}$		34.75	10.32	6.66
N_{fe} (ave.)		781	949	1185
Reliability		10/10	9/10	8/10
Best found f^g_{best} (lbs)	4607.1	4713.56	4665.98	4632.90
NI	0	0.1628	0.0426	0.0604
x_1	25.358	19.674	21.142	24.528
x_2	5.000	5.706	5.039	5.326
x_3	17.839	18.897	19.904	15.263
x_4	11.238	7.655	10.988	14.938
x_5	5.000	5.094	6.901	5.171
x_6	5.000	6.442	6.125	5.524
x_7	5.000	10.601	5.469	5.493
x_8	15.501	14.787	14.801	14.552
x_9	15.893	9.400	13.410	13.297
x_{10}	5.000	9.837	6.249	6.104
N_{fe}		492	1177	888

Table B.1: Convex 10-bar truss results with *a priori* stopping criterion

	Optimum solution	PSO-DIV variant solution		
ϵ_s (%)		5	2	1
$\bar{f}^g_{best}$ (lbs)		5292.40	5153.62	5105.78
$\bar{\sigma}$		15.120	5.776	4.818
N_{fe} (ave.)		1304	1609	2019
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	5060.85	5260.94	5144.22	5094.34
NI	0	0.0784	0.129	0.0886
x_1	30.522	29.208	27.461	29.148
x_2	0.100	0.462	0.787	0.355
x_3	23.200	22.853	24.337	23.354
x_4	15.223	11.717	12.868	13.999
x_5	0.100	0.344	0.336	0.130
x_6	0.551	0.833	0.971	0.327
x_7	7.457	10.230	7.614	9.086
x_8	21.036	26.118	23.568	19.466
x_9	21.528	18.619	20.730	21.847
x_{10}	0.100	1.135	0.105	0.650
N_{fe}		1034	1002	1541

Table B.2: Non-convex 10-bar truss results with *a priori* stopping criterion

	Optimum solution	PSO-DIV variant solution		
ϵ_s (%)		5	2	1
$\bar{f}^g_{best}$ (lbs)		569.22	555.08	550.29
$\bar{\sigma}$		3.639	1.144	3.639
N_{fe} (ave.)		1324	1941	2528
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	545.04	559.55	552.32	549.30
NI	0	0.000	0.000	0.00996
x_1	0.010	0.268	0.525	0.085
x_2	2.042	1.741	2.134	1.960
x_3	3.002	3.341	2.705	3.087
x_4	0.010	0.226	0.019	0.063
x_5	0.010	0.154	0.021	0.073
x_6	0.683	0.679	0.672	0.756
x_7	1.623	1.881	1.737	1.675
x_8	2.671	2.486	2.740	2.533
N_{fe}		1322	2787	2083

Table B.3: 25-bar truss results with *a priori* stopping criterion

	Optimum solution	PSO-DIV variant solution		
ϵ_s (%)		5	2	1
$\bar{f}^g_{best}$ (lbs)		37457.34	36415.98	36078.53
$\bar{\sigma}$		44.012	37.836	4.740
N_{fe} (ave.)		2970	5833	7438
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	35726	37375.39	36325.53	35602.03
NI	0	0.00836	0.000	0.0192
x_1	38.715	29.886	34.809	48.914
x_2	24.111	39.694	25.539	16.455
x_3	7.138	6.318	10.703	8.539
x_4	95.047	93.355	94.658	97.451
x_5	59.794	47.074	54.997	64.444
x_6	14.435	13.139	15.291	14.104
x_7	5.000	7.213	5.240	5.056
x_8	5.000	7.962	6.816	5.380
x_9	14.564	17.831	16.800	15.543
x_{10}	5.000	5.263	6.162	5.075
x_{11}	5.000	5.492	5.966	5.058
x_{12}	5.000	7.800	5.264	5.043
x_{13}	28.042	38.046	36.440	23.952
x_{14}	28.042	27.796	27.225	28.301
x_{15}	27.684	36.206	25.934	27.852
x_{16}	27.684	18.230	29.674	28.017
x_{17}	28.653	31.476	29.977	30.275
x_{18}	28.653	33.133	23.198	24.458
x_{19}	5.000	5.940	5.159	5.072
x_{20}	5.000	5.154	5.183	5.033
x_{21}	5.000	5.409	5.653	5.012
N_{fe}		2421	6546	10374

Table B.4: Convex 36-bar truss results with *a priori* stopping criterion

	Optimum solution	PSO-DIV variant solution		
Stopping N_{fe}		2000	1000	500
ϵ_a		0.01	0.01	0.01
$\bar{f}^g_{best}$ (lbs)		4606.26	4611.13	4606.56
$\bar{\sigma}$		0.2540	5.590	0.794
N_{fe} (ave.)		26193	16298	7048
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	4607.1	4606.06	4607.19	4606.04
NI	0	0.001040	0.00433	0.00102
x_1	25.358	25.269	24.956	25.311
x_2	5.000	5.000	5.001	5.000
x_3	17.839	17.898	18.256	17.919
x_4	11.238	11.312	11.204	11.200
x_5	5.000	5.000	5.000	5.000
x_6	5.000	5.000	5.000	5.000
x_7	5.000	5.000	5.209	5.000
x_8	15.501	15.478	15.132	15.495
x_9	15.893	15.842	16.046	15.859
x_{10}	5.000	5.000	5.000	5.000
N_{fe}		18984	19131	8167

Table B.5: Convex 10-bar truss results with logical stopping criterion

	Optimum solution	PSO-DIV variant solution		
Stopping N_{fe}		2000	1000	500
ϵ_a		0.01	0.01	0.01
$\bar{f}^g_{best}$ (lbs)		5064.05	5064.36	5064.70
$\bar{\sigma}$		7.306	5.119	7.986
N_{fe} (ave.)		23242	13219	6688
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	5060.85	5059.89	5060.10	5060.06
NI	0	0.0175	0.000	0.00131
x_1	30.522	29.992	29.880	29.997
x_2	0.100	0.100	0.100	0.100
x_3	23.200	23.088	23.505	23.629
x_4	15.223	15.329	15.281	15.232
x_5	0.100	0.100	0.100	0.100
x_6	0.551	0.570	0.564	0.552
x_7	7.457	7.459	7.426	7.419
x_8	21.036	21.169	21.193	21.166
x_9	21.528	21.708	21.546	21.450
x_{10}	0.100	0.100	0.100	0.100
N_{fe}		21824	16991	8537

Table B.6: Non-convex 10-bar truss results with logical stopping criterion

	Optimum solution	PSO-DIV variant solution		
Stopping N_{fe}		2000	1000	500
ϵ_a		0.01	0.01	0.01
$\bar{f}^g_{best}$ (lbs)		546.56	547.67	547.11
$\bar{\sigma}$		1.092	1.501	1.402
N_{fe} (ave.)		9914	6520	3003
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	545.04	545.33	545.77	545.89
NI	0	0.000136	0.00007	0.0100
x_1	0.010	0.010	0.010	0.010
x_2	2.042	2.025	2.093	2.143
x_3	3.002	2.995	3.140	2.962
x_4	0.010	0.010	0.010	0.010
x_5	0.010	0.010	0.010	0.010
x_6	0.683	0.641	0.672	0.614
x_7	1.623	1.645	1.525	1.564
x_8	2.671	2.729	2.675	2.793
N_{fe}		6436	5659	4122

Table B.7: 25-bar truss results with logical stopping criterion

	Optimum solution	PSO-DIV variant solution		
Stopping N_{fe}		2000	1000	500
ϵ_a		0.01	0.01	0.01
$\bar{f}^g_{best}$ (lbs)		35743.58	35726.16	35673.95
$\bar{\sigma}$		65.370	116.551	0.445
N_{fe} (ave.)		39886	23577	14666
Reliability		9/10	10/10	10/10
Best found f^g_{best} (lbs)	35726	35672.98	35673.23	35673.12
NI	0	0.167	0.028	0.008
x_1	38.715	38.577	38.691	39.184
x_2	24.111	24.273	24.310	23.735
x_3	7.138	7.226	7.048	7.154
x_4	95.047	94.594	94.604	94.536
x_5	59.794	59.825	59.597	59.624
x_6	14.435	14.318	14.478	14.410
x_7	5.000	5.000	5.000	5.000
x_8	5.000	5.000	5.000	5.000
x_9	14.564	14.453	14.559	14.501
x_{10}	5.000	5.000	5.000	5.000
x_{11}	5.000	5.000	5.000	5.000
x_{12}	5.000	5.000	5.001	5.000
x_{13}	28.042	28.039	28.118	27.880
x_{14}	28.042	27.969	27.691	28.055
x_{15}	27.684	27.525	27.460	27.667
x_{16}	27.684	27.550	27.790	27.497
x_{17}	28.653	28.533	28.515	28.501
x_{18}	28.653	28.522	28.555	28.618
x_{19}	5.000	5.000	5.000	5.000
x_{20}	5.000	5.000	5.000	5.000
x_{21}	5.000	5.000	5.001	5.000
N_{fe}		37506	22427	12329

Table B.8: Convex 36-bar truss results with logical stopping criterion

B.2.2 Social pressure modification method

	Optimum solution	PSO-DIV variant solution		
ϵ_s (%)		5	2	1
$\bar{f}^g_{best}$ (lbs)		4819.40	4694.14	4650.74
$\bar{\sigma}$		14.042	5.206	1.882
N_{fe} (ave.)		425	936	1302
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	4607.1	4790.98	4682.30	4647.62
NI	0	0.0000	0.0000	0.0000
x_1	25.358	24.761	25.665	24.181
x_2	5.000	5.701	5.122	5.032
x_3	17.839	19.879	21.799	21.096
x_4	11.238	11.817	8.908	11.449
x_5	5.000	6.231	5.286	5.071
x_6	5.000	6.904	5.242	5.063
x_7	5.000	7.290	7.832	6.155
x_8	15.501	10.476	13.130	12.456
x_9	15.893	16.221	13.696	15.765
x_{10}	5.000	6.876	6.365	5.913
N_{fe}		610	954	1180

Table B.9: Social pressure modified PSOA: Convex 10-bar truss results with *a priori* stopping criterion

	Optimum solution	PSO-DIV variant solution		
ϵ_s (%)		5	2	1
$\bar{f}^g_{best}$ (lbs)		5293.93	5153.06	5108.54
$\bar{\sigma}$		11.401	8.142	3.451
N_{fe} (ave.)		803	1083	1366
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	5060.85	5277.19	5134.86	5100.52
NI	0	0.0066	0.0000	0.0032
x_1	30.522	26.401	28.174	29.812
x_2	0.100	0.184	0.480	0.186
x_3	23.200	22.034	23.646	25.610
x_4	15.223	17.376	14.359	16.452
x_5	0.100	0.497	0.139	0.128
x_6	0.551	1.355	0.245	0.496
x_7	7.457	6.992	8.497	7.422
x_8	21.036	25.164	23.827	20.411
x_9	21.528	22.834	21.015	20.741
x_{10}	0.100	0.491	0.113	0.153
N_{fe}		897	969	1206

Table B.10: Social pressure modified PSOA: Non-convex 10-bar truss results with *a priori* stopping criterion

	Optimum solution	PSO-DIV variant solution		
ϵ_s (%)		5	2	1
$\bar{f}^g_{best}$ (lbs)		569.24	554.56	550.26
$\bar{\sigma}$		3.034	1.893	0.204
N_{fe} (ave.)		1029	1329	2616
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	545.04	561.42	550.38	549.82
NI	0	0.000	0.000	0.000
x_1	0.010	0.292	0.092	0.055
x_2	2.042	1.964	2.208	2.012
x_3	3.002	3.014	3.165	2.744
x_4	0.010	0.142	0.051	0.025
x_5	0.010	0.433	0.068	0.039
x_6	0.683	0.621	0.726	0.735
x_7	1.623	1.809	1.452	1.823
x_8	2.671	2.683	2.616	2.638
N_{fe}		1062	765	3069

Table B.11: Social pressure modified PSOA: 25-bar truss results with *a priori* stopping criterion

	Optimum solution	PSO-DIV variant solution		
ϵ_s (%)		5	2	1
$\bar{f}^g_{best}$ (lbs)		37450.93	36489.65	36074.39
$\bar{\sigma}$		59.918	67.874	11.952
N_{fe} (ave.)		1232	1973	2431
Reliability		10/10	9/10	10/10
Best found f^g_{best} (lbs)	35726	37340.09	36385.04	36044.45
NI	0	0.0144	0.012	0.011
x_1	38.715	31.449	41.804	35.538
x_2	24.111	27.103	21.082	22.024
x_3	7.138	16.533	6.752	18.632
x_4	95.047	86.386	97.298	90.770
x_5	59.794	51.480	64.502	60.527
x_6	14.435	18.869	15.385	14.742
x_7	5.000	5.674	5.510	5.214
x_8	5.000	6.386	7.810	5.518
x_9	14.564	19.458	15.522	17.323
x_{10}	5.000	16.851	5.298	5.191
x_{11}	5.000	16.223	5.944	5.450
x_{12}	5.000	5.160	5.316	5.646
x_{13}	28.042	33.090	35.448	27.734
x_{14}	28.042	25.307	15.944	30.645
x_{15}	27.684	27.160	26.140	27.238
x_{16}	27.684	30.415	29.242	27.214
x_{17}	28.653	29.223	29.592	30.169
x_{18}	28.653	31.422	27.427	25.223
x_{19}	5.000	5.342	5.249	5.147
x_{20}	5.000	5.778	5.803	5.065
x_{21}	5.000	5.505	5.110	5.136
N_{fe}		1091	1778	1490

Table B.12: Social pressure modified PSOA: Convex 36-bar truss results with *a priori* stopping criterion

	Optimum solution	PSO-DIV variant solution		
Stopping N_{fe}		2000	1000	500
ϵ_a		0.01	0.01	0.01
$\bar{f}^g_{best}$ (lbs)		4606.11	4606.12	4612.72
$\bar{\sigma}$		0.163	0.077	15.733
N_{fe} (ave.)		9505	8552	6269
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	4607.1	4606.03	4606.04	4606.03
NI	0	0.000	0.000	0.000
x_1	25.358	25.342	25.337	25.313
x_2	5.000	5.000	5.000	5.000
x_3	17.839	17.843	17.817	17.854
x_4	11.238	11.244	11.234	11.245
x_5	5.000	5.000	5.000	5.000
x_6	5.000	5.000	5.000	5.000
x_7	5.000	5.000	5.000	5.000
x_8	15.501	15.467	15.541	15.470
x_9	15.893	15.886	15.842	15.893
x_{10}	5.000	5.000	5.000	5.000
N_{fe}		7392	6379	5866

Table B.13: Social pressure modified PSOA: Convex 10-bar truss results with logical stopping criterion

	Optimum solution	PSO-DIV variant solution		
Stopping N_{fe}		2000	1000	500
ϵ_a		0.01	0.01	0.01
$\bar{f}^g_{best}$ (lbs)		5067.51	5062.33	5066.69
$\bar{\sigma}$		17.509	5.121	7.309
N_{fe} (ave.)		10194	8011	4885
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	5060.85	5059.85	5059.89	5060.16
NI	0	0.00128	0.001	0.001
x_1	30.522	29.999	29.998	29.901
x_2	0.100	0.100	0.100	0.100
x_3	23.200	23.268	23.323	23.484
x_4	15.223	15.129	15.320	15.171
x_5	0.100	0.100	0.100	0.100
x_6	0.551	0.554	0.553	0.545
x_7	7.457	7.454	7.456	7.445
x_8	21.036	21.232	21.280	21.320
x_9	21.528	21.670	21.448	21.494
x_{10}	0.100	0.100	0.100	0.100
N_{fe}		10260	8212	6569

Table B.14: Social pressure modified PSOA: Non-convex 10-bar truss results with logical stopping criterion

	Optimum solution	PSO-DIV variant solution		
Stopping N_{fe}		2000	1000	500
ϵ_a		0.01	0.01	0.01
$\bar{f}^g_{best}$ (lbs)		546.84	547.95	550.34
$\bar{\sigma}$		1.478	2.371	3.223
N_{fe} (ave.)		9596	6301	3277
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	545.04	545.21	545.55	545.58
NI	0	0.000	0.000	0.000
x_1	0.010	0.010	0.010	0.018
x_2	2.042	2.121	2.138	2.139
x_3	3.002	2.893	3.052	2.878
x_4	0.010	0.010	0.010	0.018
x_5	0.010	0.010	0.010	0.014
x_6	0.683	0.671	0.663	0.667
x_7	1.623	1.611	1.527	1.604
x_8	2.671	2.717	2.704	2.727
N_{fe}		7126	6774	3492

Table B.15: Social pressure modified PSOA: 25-bar truss results with logical stopping criterion

	Optimum solution	PSO-DIV variant solution		
Stopping N_{fe}		2000	1000	500
ϵ_a		0.01	0.01	0.01
$\bar{f}^g_{best}$ (lbs)		38983.18	35742.00	35762.24
$\bar{\sigma}$		70.389	60.567	56.896
N_{fe} (ave.)		9287	7844	5102
Reliability		10/10	10/10	10/10
Best found f^g_{best} (lbs)	35726	35673.91	35673.90	35674.71
NI	0	0.00829	0.00826	0.00837
x_1	38.715	38.607	38.109	38.178
x_2	24.111	24.665	24.890	25.031
x_3	7.138	6.811	7.077	6.885
x_4	95.047	94.767	94.926	94.320
x_5	59.794	59.637	59.424	59.633
x_6	14.435	14.391	14.432	14.520
x_7	5.000	5.000	5.000	5.000
x_8	5.000	5.000	5.000	5.000
x_9	14.564	14.514	14.549	14.539
x_{10}	5.000	5.000	5.000	5.000
x_{11}	5.000	5.000	5.000	5.000
x_{12}	5.000	5.000	5.000	5.000
x_{13}	28.042	28.087	27.902	27.588
x_{14}	28.042	27.806	28.011	28.233
x_{15}	27.684	27.220	27.611	27.737
x_{16}	27.684	27.721	27.413	27.571
x_{17}	28.653	28.756	28.612	28.563
x_{18}	28.653	28.455	28.496	28.571
x_{19}	5.000	5.000	5.000	5.000
x_{20}	5.000	5.000	5.000	5.000
x_{21}	5.000	5.000	5.000	5.000
N_{fe}		18846	12802	10250

Table B.16: Social pressure modified PSOA: Convex 36-bar truss results with logical stopping criterion

	Optimum solution	Constriction factor	Dynamic inertia and velocity red.
ϵ_s (%)		1	1
$\bar{f}^g_{best}$		36054.13	36074.39
$\bar{\sigma}$		29.991	11.952
N_{fe} (Ave.)		4442	2431
Reliability		10/10	10/10
f_{best}	35726	35975.82	36044.45
NI	0	0.00291	0.011
x_1	38.715	34.040	35.538
x_2	24.111	32.391	22.024
x_3	7.138	6.479	18.632
x_4	95.047	100.794	90.770
x_5	59.794	61.501	60.527
x_6	14.435	13.427	14.742
x_7	5.000	5.227	5.214
x_8	5.000	5.119	5.518
x_9	14.564	12.137	17.323
x_{10}	5.000	5.291	5.191
x_{11}	5.000	5.571	5.450
x_{12}	5.000	5.070	5.646
x_{13}	28.042	27.526	27.734
x_{14}	28.042	27.858	30.645
x_{15}	27.684	29.411	27.238
x_{16}	27.684	24.788	27.214
x_{17}	28.653	26.359	30.169
x_{18}	28.653	30.316	25.223
x_{19}	5.000	5.119	5.147
x_{20}	5.000	5.265	5.065
x_{21}	5.000	5.020	5.136
N_{fe}		4436	1490

Table B.17: 36-bar truss: Comparison between constriction factor and dynamic inertia / velocity reduction variants

B.2.3 Summary of standard penalty method vs. penalty method with social pressure

	Standard penalty method	Penalty with social pressure
	$\epsilon_s = 5\%$	
$\bar{f}^g_{best}$ (lbs)	4781.55	4819.40
$\bar{\sigma}$	34.746	14.04
N_{fe} (ave.)	780	425
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	4713.56	4790.98
Normalized infeasability	0.1628	0.0000
N_{fe}	492	610
	$\epsilon_s = 2\%$	
$\bar{f}^g_{best}$ (lbs)	4682.07	4694.14
$\bar{\sigma}$	10.322	5.206
N_{fe} (ave.)	949	936
Convergence ratio	9/10	10/10
Best found f^g_{best} (lbs)	4665.98	4682.30
Normalized infeasability	0.0426	0.0000
N_{fe}	1177	954
	$\epsilon_s = 1\%$	
$\bar{f}^g_{best}$ (lbs)	4643.86	4650.74
$\bar{\sigma}$	6.659	1.882
N_{fe} (ave.)	1185	1302
Convergence ratio	8/10	10/10
Best found f^g_{best} (lbs)	4632.90	4647.62
Normalized infeasability	0.0604	0.0000
N_{fe}	888	1180

Table B.18: Convex 10-bar problem: Comparison between constraint implementations

	Standard penalty method	Penalty with social pressure
	$\epsilon_s = 5\%$	
$\bar{f}^g_{best}$ (lbs)	5292.40	5293.94
$\bar{\sigma}$	15.120	11.401
N_{fe} (ave.)	1304	803
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	5260.94	5277.12
Normalized infeasability	0.0784	0.0066
N_{fe}	1034	897
	$\epsilon_s = 2\%$	
$\bar{f}^g_{best}$ (lbs)	5153.62	5153.06
$\bar{\sigma}$	5.776	8.142
N_{fe} (ave.)	1609	1083
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	5144.22	5134.86
Normalized infeasability	0.129	0.000
N_{fe}	1002	969
	$\epsilon_s = 1\%$	
$\bar{f}^g_{best}$ (lbs)	5105.78	5108.54
$\bar{\sigma}$	4.818	3.451
N_{fe} (ave.)	2019	1366
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	5094.34	5100.52
Normalized infeasability	0.0886	0.0032
N_{fe}	1541	1206

Table B.19: Non-convex 10-bar problem: Comparison between constraint implementations

	Standard penalty method	Penalty with social pressure
	$\epsilon_s = 5\%$	
$\bar{f}^g_{best}$ (lbs)	569.22	569.24
$\bar{\sigma}$	3.639	3.034
N_{fe} (ave.)	1324	1029
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	559.55	561.42
Normalized infeasability	0.000	0.000
N_{fe}	1322	1062
	$\epsilon_s = 2\%$	
$\bar{f}^g_{best}$ (lbs)	555.08	554.56
$\bar{\sigma}$	1.144	1.893
N_{fe} (ave.)	1941	1329
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	552.32	550.38
Normalized infeasability	0.000	0.000
N_{fe}	2787	765
	$\epsilon_s = 1\%$	
$\bar{f}^g_{best}$ (lbs)	550.29	550.26
$\bar{\sigma}$	3.639	0.204
N_{fe} (ave.)	2528	2616
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	549.30	549.82
Normalized infeasability	0.0099	0.000
N_{fe}	2083	3069

Table B.20: Non-convex 25-bar problem: Comparison between constraint implementations

	Standard penalty method	Penalty with social pressure
	$\epsilon_s = 5\%$	
$\bar{f}^g_{best}$ (lbs)	37457.34	37450.93
$\bar{\sigma}$	44.012	59.918
N_{fe} (ave.)	2970	1232
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	37375.39	37340.09
Normalized infeasability	0.0083	0.0144
N_{fe}	2421	1091
	$\epsilon_s = 2\%$	
$\bar{f}^g_{best}$ (lbs)	36415.98	36489.65
$\bar{\sigma}$	37.836	67.874
N_{fe} (ave.)	5833	1973
Convergence ratio	10/10	9/10
Best found f^g_{best} (lbs)	36325.53	36385.04
Normalized infeasability	0.000	0.0120
N_{fe}	6546	1778
	$\epsilon_s = 1\%$	
$\bar{f}^g_{best}$ (lbs)	36078.53	36074.39
$\bar{\sigma}$	4.740	11.952
N_{fe} (ave.)	7438	2431
Convergence ratio	10/10	10/10
Best found f^g_{best} (lbs)	35602.03	36044.45
Normalized infeasability	0.0192	0.0114
N_{fe}	10374	1490

Table B.21: Convex 36-bar problem: Comparison between constraint implementations

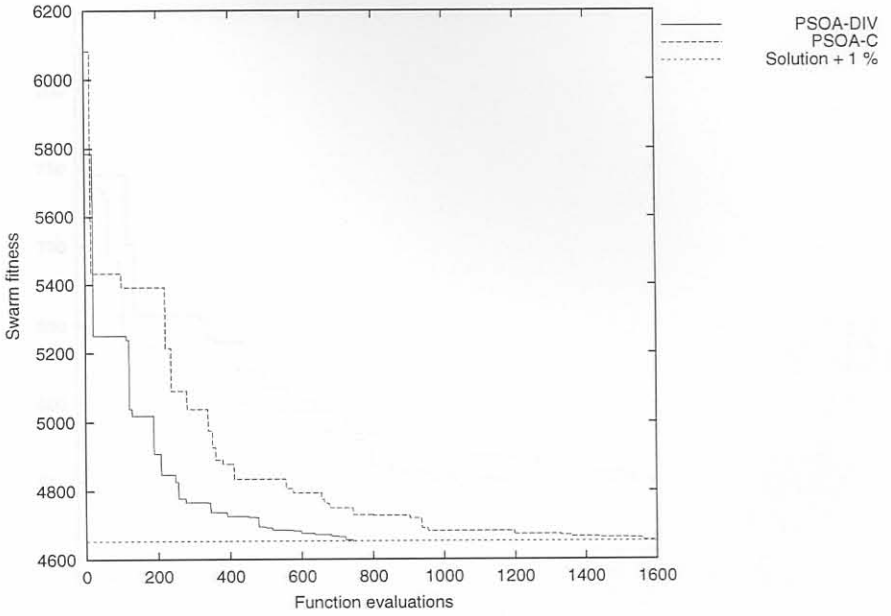


Figure B.4: Convex 10-bar truss: Typical history plot for contriction and dynamic inetia and maximum velocity variants

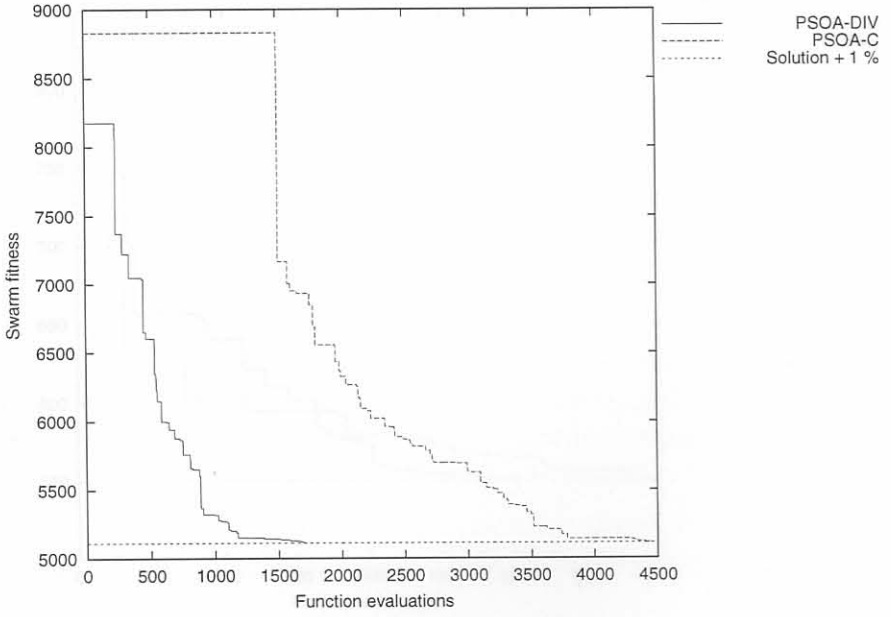


Figure B.5: Non-convex 10-bar truss: Typical history plot for contriction and dynamic inetia and maximum velocity variants

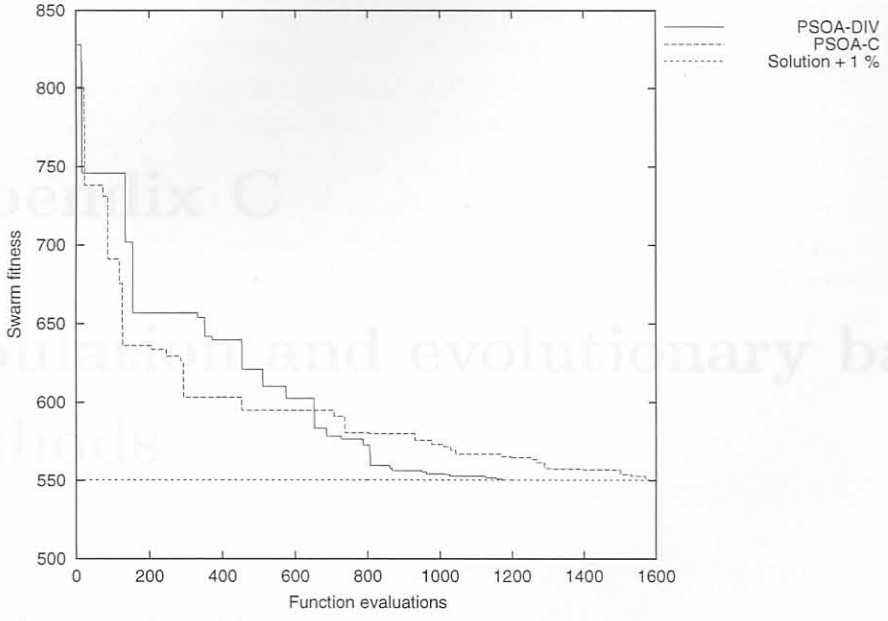


Figure B.6: Non-convex 25-bar truss: Typical history plot for contriction and dynamic inetia and maximum velocity variants

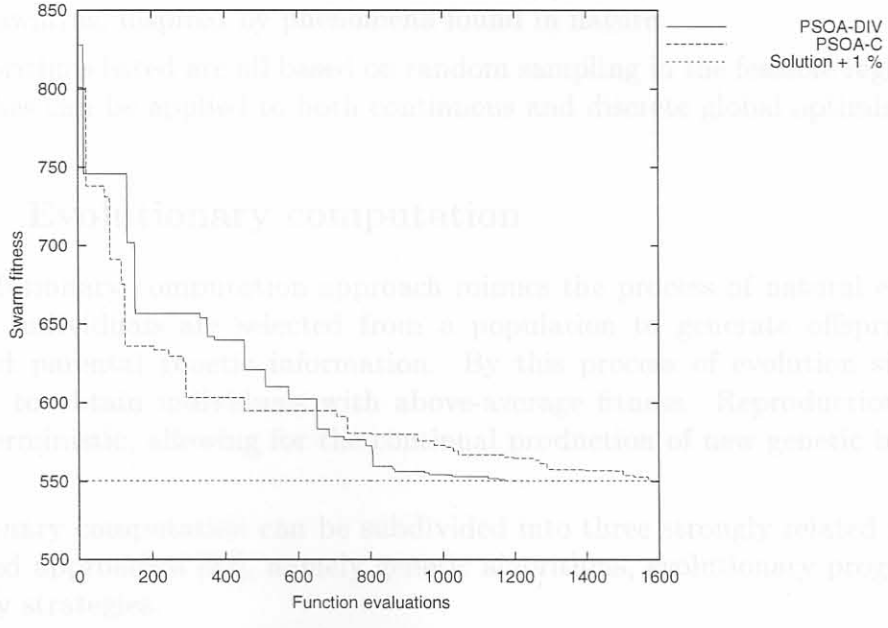


Figure B.7: Non-convex 36-bar truss: Typical history plot for contriction and dynamic inetia and maximum velocity variants

Appendix C

Population and evolutionary based methods

C.1 Introduction

In this appendix a brief overview of various adaptive stochastic approaches to global optimization are detailed. This presentation is by no means exhaustive and is meant only as an introduction to other related approaches to global optimization, some of which are, like particle swarms, inspired by phenomena found in nature.

The algorithms listed are all based on random sampling in the feasible region. Some of these approaches can be applied to both continuous and discrete global optimization problems.

C.1.1 Evolutionary computation

The evolutionary computation approach mimics the process of natural evolution by which superior individuals are selected from a population to generate offspring, which inherit disturbed parental genetic information. By this process of evolution simulation it is attempted to obtain individuals with above-average fitness. Reproduction of individuals is non-deterministic, allowing for the continual production of new genetic information (mutation).

Evolutionary computation can be subdivided into three strongly related but independently developed approaches [52], namely genetic algorithms, evolutionary programming and evolutionary strategies.

Genetic algorithms

Genetic algorithms (GA's) were first introduced by Holland [53, 54, 55] and subsequently studied by De Jong [56, 57, 58], Goldberg [59, 60, 61, 62], and others.

The genetic algorithm approach entails a 'population' of candidate solution points which is sequentially 'evolved' in a heuristic process which mimics biological evolution as found in nature. The adaptive search consists of a competitive selection process where the least fit candidates in the population have a low probability of survival. The remainder are then 'recombined' or 'paired' with other candidates by exchanging components or 'genes'. There is also a 'mutation' operator which may (randomly) adjust one or more set of genes in a candidate. Since this process of recombination and mutation occurs sequentially, each generation of candidate solution points will be biased toward regions in the problems space of increased fitness. Genetic 'drift', caused by mutation, prevents the search from stagnating and provides a means for refined 'local' search during the terminal phase of the search.

Evolutionary programming

Evolutionary programming (EP) was first introduced by Fogel [63, 64], and further studied by Atmar [65], Burgin [66, 67] and others.

Evolutionary programming was initially offered as an ambitious means of creating artificial intelligence. It involved the evolution of finite state machines to predict events on the basis of former observations. A finite state machine is an abstract machine which takes a sequence of symbols as an input and transforms them by means of a finite set of transition rules with finite states to a sequence of output symbols. The performance of this machine is usually gauged by its ability to predict events correctly.

Evolutionary strategies

Evolutionary strategies (ES) were first developed by Rechenberg [68, 69] and Schwefel [70, 71], and extended by Herdy [72], Kursawe [73] and others.

This approach was originally designed with the objective of solving difficult discrete and continuous parameter optimization problems. The main difference between ES and GA's lie in the calculation of fitness of a specific genotype, and the manner in which the operators (mutation, recombination and selection) manipulate this genotype. More specifically, while mutation is only used in GA's to avoid stagnation, this operator becomes the primary means of 'evolving' toward a solution in ES. A further difference is the manner in which selection is applied. Selection in the case of ES is absolutely deterministic, whereas this is not the case in the context of GA's. Therefore, arbitrary small differences in fitness can play a large role in deciding on the survival of a individual in ES.

C.1.2 Simulated annealing

This Monte Carlo based approach to global optimization is inspired by a physical analogy of the atomic structure in a crystalline material which strives to arrive at a stable configuration (minimum potential energy, globally or locally). This approach was first applied by Metropolis *et al.* [74] and subsequently developed further by van Laarhoven and Aarts [75] and Webb [76], amongst others.

In the analogy the current energy state of the thermodynamic system is equivalent to the current solution to the combinatorial problem; the energy equation for the thermodynamic system is analogous to the objective function, and the ground state is analogous to the global minimum. The major difficulty in implementation of the algorithm is that there is no obvious analogy for the temperature T with respect to a free parameter in the combinatorial problem. Furthermore, avoidance of entrapment in local minima (quenching) is dependent on the "annealing schedule", i.e. the choice of initial temperature, how many iterations are performed at each temperature, and how much the temperature is decremented at each step as cooling proceeds.

The simplified working of the algorithm is as follows: A random step in the problem space is taken and the energy state (objective function) evaluated at this position. Any step which yields a overall decrease in the overall energy state is accepted. The step size is reduced as the search progresses to facilitate a more refined search as the minima is approached.

C.1.3 Ant colony optimization

The ant colony optimization approach was first introduced by Dorigo *et al.* [77] and extended by Stutzle and Hoos [78, 79] among others.

The ant colony optimization (ACO) meta-heuristic takes it's inspiration from the foraging behavior of ants, in particular their ability to find the shortest routes between their nests and food sources. While traveling to and from these food sources ants deposit a pheromone trail which serves as markers, to aid themselves and other ants to return to the nest or to find the route to the food source. The ants are able to find the shortest route to their food supply when presented with a set of alternate routes by information inherent in the pheromone trail. Shorter routes get a higher density of pheromone deposit, because ants which choose this route, per chance, will more rapidly reconstitute the interrupted pheromone trail than those who choose the longer path. Because ants tend to choose, by probability, to follow the strongest pheromone trail, the pheromone will accumulate more rapidly on the shorter routes. This reinforces the use of shorter paths by process of positive feedback. This method of optimization has been successfully applied to the well known traveling salesman problem, among others.

C.1.4 Tabu Search

This method was first introduced by Glover [80] and studied further by Laguna [81]. The motivation behind this type of approach is to 'forbid' search moves to previously explored search points in the (usually discrete) solution space. With the tabu search philosophy it is sometimes allowable to temporarily accept new inferior solutions to avoid paths already investigated. This approach can lead to exploring new regions of D, with the goal of avoiding local minima and ultimately finding a solution by 'globalized' search.

Tabu search has traditionally been applied to combinatorial optimization (e.g., scheduling, routing, and traveling salesman) problems.The technique can be adapted to include continuous problems by discrete approximation (encoding) of the continuous problem (in a similar

fashion as done in genetic algorithms).

C.2 Clustering methods

The clustering method among others, was first presented by Becker and Lago [82] and remained unexploited until the work of Törn [83, 84]. It was further developed by a variety of authors (Dixon and Szegö [42]). Clustering global optimization methods can be viewed as a modified form of the standard multi-start approach, which performs a local search from several points distributed over the entire search domain. A drawback of multi-start is that when many starting points are used, the same local minimum may be identified several times, thereby leading to an inefficient global search. Clustering methods attempt to avoid this inefficiency by carefully selecting points at which the local search is initiated. The three main steps of clustering methods are:

(1) Sample points in the search domain D,

(2) Transform the sampled point to group them around the local minima, and

(3) Apply a clustering technique to identify groups that (hopefully) represent neighborhoods of local minima.

If this procedure successfully identifies groups that represent neighborhoods of local minima, then redundant local searches can be avoided by simply starting a local search at some point within each cluster.

Appendix D

Particle swarm optimization software

D.1 Users guide

D.2 OEPSA

In this appendix the OEPSA (Optimization Environment for Particle Swarm Algorithms) software is presented.

D.2.1 Overview

During the investigation into the particle swarm paradigm, an environment was developed which had to comply with the following requirements:

(a) Portability - The software had to be developed with multiple platforms and operating systems as target environments in mind. ANSI C was the language of choice because of it's common usage and ease of portability.

(b) Visualization - In order to be able to study particle behavior during a search, some means of realtime visualization of the swarm was required. The standard Linux graphics library, svgalib, was used for this purpose.

(c) Batch processing - For the benchmarking of multiple problems with varying algorithm parameters an efficient batch processor was needed.

(d) Postprocessing of results - In order for the results of test runs to be reported (with minimal extra effort) in tables and graphs, the results had to be processed and stored in a structured manner.

(e) Ability to interface with external programs - The software also had to be able to interact with extenal programs such as the finite element solver used in optimizing

the truss structures. This interaction typically entails the passing of design variables to the solver and receiving a fitness value and constraint values from the solver after analysis.

(f) Script file - In order to change the algorithm parameters without recompiling code, and to ease batch processing of large number of problems with varying parameters, a script file reader which accepts values specified in an external human readable text file was implemented.

All of the numerous algorithm modifications to the particle swarm detailed in this thesis have been incorporated into the program, and can be selected by changing the appropriate setting in the script file (see Section D.3.1).

D.3 Visual interface

The visual interface is one of the most important aspects of the software since it allows the user to observe particle behavior. During parameter sensitivity studies it is also useful to observe any behavioral changes brought about by parameter variations.

The visual interface only allows two user defined dimensions to be displayed at any time.

The screen is divided into two sections, the left section displays the particle swarm in real-time moving through the problem space. The solution point is indicated by a light blue circle and particles are indicated as white dots. Individual particle's best remembered positions $\boldsymbol{p}_k^i$ are represented by yellow dots and the overall swarm best coordinates $\boldsymbol{p}_g^i$ by a red circle with the corresponding fitness value displayed next to it.

The right section of the screen display continually updates information in the best fitness coodinates and varying parameter values. The top right section displays the current coordinates of the swarm best value $\boldsymbol{p}_g^i$ together with the swarm best fitness value g_{best}. The center right section displays the solution coordinates and optimim fitness value, and the inertia weight w and maximum allowable velocity v^{max}. The bottom section is taken up by a graph which displays the fitness history as the search progresses.

D.3.1 Input script file

An example script file with typical values for parameters is presented below, followed by explanations of the various applicable settings. On/off implies that either a 1 or 0 should be used respectively.

`first_problem = 1`	First problem in batch set.
`last_problem = 16`	Last problem in batch set.
`problem_repeats = 50`	Number of problem repetitions.
`use_display = 0`	Realtime visualization (on/off).

`display_refresh = 20`	Number of function evaluations between display updates.
`history_output = 0`	Write fitness history to file (on/off).
`output_filename = result`	Batch result output filename.
`respawn_in_feasable = 0`	Re-initialize infeasable particle until feasable (on/off).
`modification = 3`	Type of PSOA to use: 0 = Standard pso (w = 1), 1 = Linearly decreasing momentum (w), 2 = Constriction factor (K), 3 = Dynamic inertia + velocity.
`bounds_method = 3`	Type of bounds to enforce: 1 = No bounds, 2 = Respawn, 3 = Bounce.
`stopping_method = 1`	Type of stopping criterion: 1 = Error minimization, 2 = Inertia stopping method, 3 = No-improve stopping method, 4 = Maximum velocity.
`tolerance_method = 1`	If `stopping_method = 1`, specify tolerance method to be: 1 = stop when value comes within absolute tolerance, 2 = stop when value comes within percentage value of solution,
`tolerance = 0.001`	Specify either an absolute or percentile tolerance (see above).
`inertia_stop = 0.01`	If `stopping_method = 2`, the value of inertia weight to be stopped upon.
`no_improvement_stop = 1000` `no_improvement_tolerance = 0.01`	If `stopping_method = 3`, the number of Nfe with no improvement in $f(\boldsymbol{p}_k^g)$, within `no_improvement_tolerance` before algorithm is stopped.
`velocity_stop = 0.01`	Absolute value to be used as stopping criteria if `stopping_method = 4`.
`limit_max_velocity = 1`	Maximum velocity enforcement on/off.
`amount_of_particles = 20`	Swarm population.
`max_function_evaluations = 30000`	Maximum allowed function evaluations.
`allowable_infeasability = 0.02`	Normalized infeasability value which is acceptable to user.
`velocity_fraction = 1`	Initial fraction of bounds to be taken

	as maximum step size or "velocity".
`initial_inertia = 1.0`	Linear inertia reduction initial value.
`final_inertia = 0.5`	Linear inertia reduction final value.
`inertia_change_end = 4000`	N_{fe} at which to stop linear inertia reduction.
`wait_for_improvement = 10`	Number of function evaluations to wait before reducing velocity and inertia.
`inertia_reduction_fraction = 0.01`	Absolute increment by which to reduce inertia.
`velocity_reduction_fraction = 0.01`	Absolute increment by which to reduce velocity.
`initial_lambda = 1000`	λ penalty initial value
`final_lambda = 1000000`	λ penalty final value
`lambda_change_end = 4000`	Number of function evaluations N_{fe} where `final_lambda` is reached
`personal_scaler = 2.0`	Cognitive value c_1
`group_scaler = 2.0`	Social value c_2

D.4 Source code

In this section selected fragments of the software developed are detailed and discussed. For the sake of brevity only sections directly responsible for the PSOA's workings are presented, and the remainder (visual interface, batch processor, output postprocessor etc.) are ignored.

D.4.1 Particle swarm initialization

The following functions are responsible for initializing swarm positions $\boldsymbol{x}_0^i$, initial velocities $\boldsymbol{v}_0^i$, and inertia values w_0. Function values are calculated for particle positions $\boldsymbol{x}_0^i$. The best swarm value f_{best}^g is selected from these, with $\boldsymbol{p}_0^i$ set equal to the appropriate coordinate. The maximum allowable velocity $\boldsymbol{v}^{max}$ is also calculated.

```
void initialize_particle_positions(void)
    {

    /* this function initializes all the particle positions */
    int i,j,test;
    int randominteger;
    float randomfloat;
```

```
    float range[MAXDIMENSIONS];

    for (i = 0; i < problem_data.problem_dimensions; ++i)
        range[i] = problem_data.position_upperbound[i] -
problem_data.position_lowerbound[i];

    if (problem_data.respawn_in_feasable == 0)
        {
        for (i = 0; i < problem_data.amount_of_particles; ++i)
            {
            for (j = 0; j < problem_data.problem_dimensions; ++j)
                {
                randominteger = rand();
                randomfloat = (float) randominteger/RAND_MAX;
                particle[i].coordinates[j] = randomfloat*range[j] +
problem_data.position_lowerbound[j];
                } /* for j */
            } /* for i */
        }
    else
        {
        printf(''\nRespawning particles into feasable region'');
        for (i = 0; i < problem_data.amount_of_particles; ++i)
            {
            do
                {
                test = 0;

                for (j = 0; j < problem_data.problem_dimensions; ++j)
                    {
                    randominteger = rand();
                    randomfloat = (float) randominteger/RAND_MAX;
                    particle[i].coordinates[j] = randomfloat*range[j] +
problem_data.position_lowerbound[j];
                    }

                /* test if initialized position is within constraints */
                function_result =
evaluate_problem(problem_data.problem_number,problem_data.problem_dimensions,
particle[i].coordinates);

                if (function_result.normalized_infeasability >
problem_data.allowable_infeasability)
                    {
                    test = 1;
                    printf(''\nRe-initializing particle %i which violates constraints by %f'', i,
function_result.normalized_infeasability);
```

```
                }

            } while (test == 1);
            printf(''\nParticle %i normalized infeasability = %f'', i,
function_result.normalized_infeasability);

            particle[i].constraints_violated = 0;
            } /* for i */
        } /* if else */
    } /* initialize_particle_positions */


void initialize_particle_velocities(void)
    {

    /* this function initializes all the particle velocities */
    int i,j;
    int randominteger;
    float randomfloat;
    float velocity_range[MAXDIMENSIONS];

    /* Init velocity bounds */
    for (j = 0; j < problem_data.problem_dimensions; ++j)
        {
        velocity_range[j] = velocity_fraction *
(problem_data.position_upperbound[j]-problem_data.position_lowerbound[j]);
        }

    for (i = 0; i < problem_data.amount_of_particles; i++)
        {
        for (j = 0; j < problem_data.problem_dimensions; ++j)
            {
            randominteger = rand();
            randomfloat = (float)randominteger / RAND_MAX;
            particle[i].velocity[j] = randomfloat * velocity_range[j] -
(velocity_range[j] * 0.5);
            } /* for j */
        } /* for i */
    } /* initialize_particle velocities */


void initialize_particle_inertia_values(void)
    {
    /* this function sets up all the particle inertia values */
    int i,j;
    int randominteger;
    float randomfloat;
```

```
    float range;

    /* Init all at specified initial_inertia value */
    for (i = 0; i < problem_data.amount_of_particles;i++)
        particle[i].inertia = problem_data.initial_inertia;
    } /* initialize_particle_inertia_values */


void initialize_particle_function_values(void)
    {

    /* This function initializes all the particle function values */
    int i,j;
    int infeasable_particles = 0;

    for (i = 0; i < problem_data.amount_of_particles; ++i)
        {
        function_result =
evaluate_problem(problem_data.problem_number,problem_data.problem_dimensions,
particle[i].coordinates);

        particle[i].current_value = function_result.value;
        particle[i].best_value = function_result.value;
        particle[i].normalized_infeasability = function_result.normalized_infeasability;

        for (j = 0; j < problem_data.problem_dimensions; ++j)
            {
            particle[i].best_value_coordinates[j] = particle[i].coordinates[j];
            }

        /* feasability check */
        if (particle[i].normalized_infeasability >= problem_data.allowable_infeasability)
            {
            particle[i].constraints_violated = 1;
            printf(''\nConstraints violated =
%f'',function_result.normalized_infeasability);
            ++infeasable_particles;
            }
        else
            {
            particle[i].constraints_violated = 0;
            }

        /* Initialize best swarm value (get lowest value of the particle[].best.value
        array and store it and its coordinates in the comm_data struct)       */

        /* initialize swarm best if NI < NI_tol, and search for better one */
```

```
        if (particle[i].constraints_violated == 0)
           {
           comm_data.swarm_best_value = particle[i].best_value;  /* for temporary
comparison */
           comm_data.swarm_best_normalized_infeasability =
particle[i].normalized_infeasability;
           }

        } /* for particle i */

        printf(''\nParticles outside feasable region =
%i/%i'',infeasable_particles,problem_data.amount_of_particles);

    /* search for best particle fitness value */
    for (i = 0; i < problem_data.amount_of_particles; ++i)
       {
       if ( (particle[i].best_value < comm_data.swarm_best_value) &&
(particle[i].constraints_violated == 0) )
          {
          comm_data.swarm_best_value = particle[i].best_value;
          comm_data.swarm_best_normalized_infeasability =
particle[i].normalized_infeasability;

          for (j = 0; j < problem_data.problem_dimensions; ++j)
             {
             comm_data.swarm_best_coordinates[j] = particle[i].best_value_coordinates[j];
             } /* for j */
          } /* for i */
       } /* if */
    } /* initialize_particle_function_values */


void initialize_swarm_max_velocity(void)
   {

   /* This function initializes the swarm maximum velocity */
   double range;
   int j;

   for (j = 0; j < problem_data.problem_dimensions; ++j)
      {
      comm_data.swarm_max_velocity[j] = velocity_fraction *
(problem_data.position_upperbound[j]-problem_data.position_lowerbound[j]);
      }
   } /* initialize_swarm
```

D.4.2 Search

The following code implements the velocity (2.2) and position (2.1) rules. Allowance for the limitation of maximum velocity in the velocity update function and bounds implementation in the position update function are also implemented. Additional functions required during the search are detailed, which update particle inertia values, maximum allowed velocities and fitness values.

```
void update_particle_velocity(int i)
    {

    /* This function is used to update a single particle velocity during the search */
    int j;
    int randominteger1;
    double randomdouble1;
    int randominteger2;
    double randomdouble2;

    double range;
    double personal_scaler   = problem_data.personal_scaler;
    double group_scaler      = problem_data.group_scaler;
    double old;
    double personal;
    double group;
    double varphi;

    for (j = 0; j < problem_data.problem_dimensions; ++j)
        {
        randominteger1 = rand();
        randomdouble1 = (double) randominteger1/RAND_MAX;
        randominteger2 = rand();
        randomdouble2 = (double) randominteger2/RAND_MAX;

        old = particle[i].velocity[j];

        /* social pressure */
        if ( (particle[i].constraints_violated == 1) &&
(problem_data.use_social_pressure == 1) )
            {
            personal = 0;
            }
```

```
        else
            {
            personal =
randomdouble1*(particle[i].best_value_coordinates[j]-particle[i].coordinates[j]);
            }
        group =
randomdouble2*(comm_data.swarm_best_coordinates[j]-particle[i].coordinates[j]);
        if (problem_data.modification != 2)
            {
            particle[i].velocity[j] = particle[i].inertia*old + personal_scaler*personal
+ group_scaler*group;
            }
        else /* constriction factor */
            {
            varphi = personal_scaler + group_scaler;
            constriction_factor = 2/(fabs(2 - varphi - sqrt(SQUARE(varphi) - 4*varphi)
));
            particle[i].velocity[j] = constriction_factor*(old + personal_scaler*personal
+ group_scaler*group);
            }

        /* max speed check */
        if (problem_data.limit_max_velocity == 1)
            {
            if (particle[i].velocity[j] > comm_data.swarm_max_velocity[j])
                particle[i].velocity[j] = comm_data.swarm_max_velocity[j];
            if (particle[i].velocity[j] < -comm_data.swarm_max_velocity[j])
                particle[i].velocity[j] = -comm_data.swarm_max_velocity[j];
            } /* if */
        } /* for j */
    } /* update_particle_velocity */


void update_particle_position(int i)
   {

   /* This function is used to update a particle position during the search */
   int j,k;
   int randominteger;
   float randomfloat;
   double range[MAXDIMENSIONS];
   double temp;
   double temp2;
   double upperbound, lowerbound;
```

```
   switch(problem_data.bounds_method)
      {
      case 1:
      /* no bounds method */

      for (j = 0; j < MAXDIMENSIONS; ++j)
         {
         particle[i].coordinates[j] = particle[i].coordinates[j] +
particle[i].velocity[j];
         }

      break;


      case 2:
      /* random respawn position if out of bounds method */
      for (j = 0; j < problem_data.problem_dimensions; ++j)
         {
         particle[i].coordinates[j] = particle[i].coordinates[j] +
particle[i].velocity[j];

         if ((particle[i].coordinates[j] < problem_data.position_lowerbound[j]) ||
(particle[i].coordinates[j]>problem_data.position_upperbound[j]))
            {
            randominteger = rand();
            randomfloat = (float) randominteger/RAND_MAX;
            particle[i].coordinates[j] = randomfloat*range[j] +
problem_data.position_lowerbound[j];
            } /* if */
         } /*for j */
      break;

      case 3:
      /* bounce off boundaries method (velocity reversal/sign changover)*/

      for (j = 0; j < problem_data.problem_dimensions; ++j)
         {
         particle[i].coordinates[j] = particle[i].coordinates[j] +
particle[i].velocity[j];
         if (particle[i].coordinates[j] < problem_data.position_lowerbound[j])
            {
            particle[i].velocity[j] = fabs(particle[i].velocity[j]);
            particle[i].coordinates[j] = particle[i].coordinates[j] +
particle[i].velocity[j];
            } /* if */
```

```
        else if (particle[i].coordinates[j] > problem_data.position_upperbound[j])
            {
            particle[i].velocity[j] = -fabs(particle[i].velocity[j]);
            particle[i].coordinates[j] = particle[i].coordinates[j] +
particle[i].velocity[j];
            } /* else */
        } /* for j */
      break;
      } /* case */
   } /* update_particle_position */


void update_particle_function_value(int i)
   {

    /* This function updates a particle function value during search */
    function_result =
evaluate_problem(problem_data.problem_number,problem_data.problem_dimensions,
particle[i].coordinates);
    particle[i].current_value = function_result.value;
    particle[i].normalized_infeasability = function_result.normalized_infeasability;


    /* Infeasability check */
    if (particle[i].normalized_infeasability > problem_data.allowable_infeasability)
        {
        particle[i].constraints_violated = 1;
        printf(''\nconstraints violated = %f'',function_result.normalized_infeasability);
        } /* if */
    else
        {
        particle[i].constraints_violated = 0;
        } /* else */
    } /* update_particle_function_value */


void update_inertia_max_velocity(void)
    {

    /* Function to alter modification parameters, eg. article inertia, velocity etc */
    double inertia_range;
    double velocity_range;
    double temp;
    double fraction;
    int i,j;

    inertia_range = problem_data.initial_inertia - problem_data.final_inertia;
```

```
    velocity_range = problem_data.initial_velocity - problem_data.final_velocity;

    switch(problem_data.modification)
        {
        case 0:
        /*standard pso */

        for (i = 0; i < problem_data.amount_of_particles; ++i)
            {
            particle[i].inertia = 1;
            }
        break;

        case 1:
        /* linearly decreasing inertia */

        if (comm_data.function_evaluations < problem_data.inertia_change_end)
            {
            temp = (double)(problem_data.inertia_change_end -
comm_data.function_evaluations);
            temp = (double)(temp / problem_data.inertia_change_end);
            for (i = 0; i < problem_data.amount_of_particles; ++i)
                {
                particle[i].inertia = problem_data.final_inertia + inertia_range*temp;
                } /* for i */
            } /* if */
         else
            {
            for (i = 0; i < problem_data.amount_of_particles; ++i)
                {
                particle[i].inertia = problem_data.final_inertia;
                } /* for i */
            } /* else */

        break;

        case 2:
        /* constriction factor */

        /* nothing done to either inertia or velocity */

        break;

        case 3:
        /* Dynamic inertia and velocity reduction modification */

        if (no_improvement_iteration >= problem_data.wait_for_improvement)
```

```
            {
            no_improvement_iteration = 0;

            for (i = 0; i < problem_data.amount_of_particles; ++i)
                {
                particle[i].inertia = particle[i].inertia * (1 -
problem_data.inertia_reduction_fraction);
                } /* for */

            for (j = 0; j < problem_data.problem_dimensions; ++j)
                {
                comm_data.swarm_max_velocity[j] = comm_data.swarm_max_velocity[j] * (1 -
problem_data.velocity_reduction_fraction);
                } /* for */
            } /* if */

        break;

        } /* end switch */
    } /* end update_inertia_max_velocity */


void update_best_function_value(int i)
    {

    /* Updates particle_best_value and swarm_best_value and their coordinates */
    int j;

    if ((particle[i].current_value < particle[i].best_value) &&
(particle[i].constraints_violated != 1))
        {

        particle[i].best_value = particle[i].current_value;

        for (j = 0; j < problem_data.problem_dimensions; ++j)
            {
            particle[i].best_value_coordinates[j] = particle[i].coordinates[j];
            } /* for j */
        } /* if */


    if ((particle[i].current_value < comm_data.swarm_best_value) &&
(particle[i].constraints_violated != 1))
        {

        if ( (particle[i].current_value + problem_data.no_improvement_tolerance) <
comm_data.swarm_best_value)
```

```
            {
            no_improvement = 0;
            }

        comm_data.swarm_best_value = particle[i].current_value;
        comm_data.swarm_best_normalized_infeasability =
particle[i].normalized_infeasability;
        no_improvement_iteration = 0;

        for (j = 0; j < problem_data.problem_dimensions; ++j)
            {
            comm_data.swarm_best_coordinates[j] = particle[i].coordinates[j];
            } /* for j */
        } /* if */
    } /* update_best_function_value */
```

D.4.3 Termination

The following functions implement several stopping methods, among them the logical and *a priori* stopping criteria.

```
void check_for_stop(FILE *outputfilepointer, int stopping_method)
    {

    /* Function to check if stopping condition is satisfied */
    int j;
    double temp_max_velocity;

    switch(stopping_method)
        {

        case 1:
        /* Absolute (error minimization) */

        if ((comm_data.swarm_best_value <= (problem_data.solution_func_value +
comm_data.tolerance))&&(found_solution_flag == 0))
            {

            found_solution_flag = 1;

            result[problem_iteration].converged_flag = 1;
            result[problem_iteration].function_evaluations =
comm_data.function_evaluations;
```

```
            result[problem_iteration].normalized_infeasability =
comm_data.swarm_best_normalized_infeasability;
            result[problem_iteration].swarm_best_value = comm_data.swarm_best_value;

            for (j = 0; j < problem_data.problem_dimensions; ++j )
                {
                result[problem_iteration].swarm_best_coordinates[j] =
comm_data.swarm_best_coordinates[j];
                }

            ++comm_data.times_converged;

            display_results();

            write_outputfile(outputfilepointer,problem_iteration);

            } /* end if */

        break;

        case 2:
        /* Inertia */
        if (particle[0].inertia <= problem_data.inertia_stop)
            {

            found_solution_flag = 1;

            result[problem_iteration].converged_flag = 1;
            result[problem_iteration].function_evaluations =
comm_data.function_evaluations;
            result[problem_iteration].normalized_infeasability =
comm_data.swarm_best_normalized_infeasability;
            result[problem_iteration].swarm_best_value = comm_data.swarm_best_value;

            for (j = 0; j < problem_data.problem_dimensions; ++j )
                {
                result[problem_iteration].swarm_best_coordinates[j] =
comm_data.swarm_best_coordinates[j];
                }

            ++comm_data.times_converged;

            display_results();

            write_outputfile(outputfilepointer,problem_iteration);
```

```
            } /* end if */

        break;

        case 3:
        /* No improvement */

        if (no_improvement >= problem_data.no_improvement_stop)
            {

            found_solution_flag = 1;

            result[problem_iteration].converged_flag = 1;
            result[problem_iteration].function_evaluations =
comm_data.function_evaluations;
            result[problem_iteration].normalized_infeasability =
comm_data.swarm_best_normalized_infeasability;
            result[problem_iteration].swarm_best_value = comm_data.swarm_best_value;

            for (j = 0; j < problem_data.problem_dimensions; ++j )
                {
                result[problem_iteration].swarm_best_coordinates[j] =
comm_data.swarm_best_coordinates[j];
                }

            ++comm_data.times_converged;

            display_results();

            write_outputfile(outputfilepointer,problem_iteration);

            } /* end if */

        break;

        case 4:
        /* Velocity */

        temp_max_velocity = 0;

        /* check if velocity vector magnitude is smaller than velocity_stop */
        for (j = 0; j < problem_data.problem_dimensions; ++j)
            {
            temp_max_velocity = SQUARE(comm_data.swarm_max_velocity[j]);
            } /* for j */

            if ( (sqrt(temp_max_velocity)) <= problem_data.velocity_stop)
```

```
                {
                found_solution_flag = 1;

                result[problem_iteration].converged_flag = 1;
                result[problem_iteration].function_evaluations =
comm_data.function_evaluations;
                result[problem_iteration].normalized_infeasability =
comm_data.swarm_best_normalized_infeasability;
                result[problem_iteration].swarm_best_value = comm_data.swarm_best_value;

                for (j = 0; j < problem_data.problem_dimensions; ++j )
                    {
                    result[problem_iteration].swarm_best_coordinates[j] =
comm_data.swarm_best_coordinates[j];
                    } /* j */

                ++comm_data.times_converged;

                display_results();

                write_outputfile(outputfilepointer,problem_iteration);

                } /* if */

            break;
        } /* switch */
    } /* check_for_stop */
```

D.4.4 Main program

The initial part of the main program is presented below (with the postprocessing section removed). The sequential algorithm is implemented herein, but this can easily be changed to an asynchronous implementation by updating on a per swarm basis.

```
int main(void)
 {
    FILE *inputfilepointer, *outputfilepointer, *historyfilepointer, *bestfilepointer,
*parameterfilepointer;
    int stopping_condition_flag = 0;
    int counter = 0;
    int iteration = 0;
    int pcle;
    int repeat;
```

```
    int k,z,t,j;
    int updatecount;
    int random_init;
    int temp_converged;


/* Seed random number generator using current time */
    time_t curr_time;
    srand( (unsigned) time(&curr_time));

comm_data.times_converged = 0;

if ( (inputfilepointer = fopen(''script'',''r'') ) == NULL)
    {
        printf(''\a\nCould not open input script file \n'');
        exit(1);
        }

    read_inputfile(inputfilepointer);


/*  Open file for problem data and solutions found output */

  if ( (outputfilepointer = fopen(output_filename,''w'') ) == NULL)
    {
        printf(''\a\nCould not open problem data output file \n'');
        exit(1);
        }

/*  Open file for problem data parameter variation output */

  if ( (parameterfilepointer = fopen(''parameter'',''w'') ) == NULL)
    {
        printf(''\a\nCould not open parameter results output file \n'');
        exit(1);
        }


/*  Open file for function value history output */

  if (history_output == 1)
    {
        if ( (historyfilepointer = fopen(''history'',''w+'') ) == NULL)
         {
           printf(''\a\nCould not open history file for reading/writing\n'');
           exit(1);
           }
```

```
        if ( (bestfilepointer = fopen(''best_history'',''w'') ) == NULL)
         {
           printf(''\a\nCould not open best_history file for writing\n'');
           exit(1);
           }
        }

    for (test_problem = first_problem; test_problem <= last_problem; ++test_problem)
      {
        problem_data.problem_number = test_problem;
        initialize_problem_set_variables(problem_data.problem_number);
        set_tolerance();

    write_outputfile_header(outputfilepointer);


       for (t = 0; t < problem_data.problem_dimensions; ++t)
        {
          total_length[t] = (problem_data.position_upperbound[t]-
problem_data.position_lowerbound[t]);
         }

    print_problemdata_screen();

    /* start problem iterations */

       for (problem_iteration = 0; problem_iteration < problem_data.problem_repeats;
++problem_iteration)
         {
      found_solution_flag = 0;
         comm_data.function_evaluations = 0;
          iteration = 0;
         random_init = 0;
         no_improvement_iteration = 0;
         no_improvement = 0;


      /* print history start indicator*/
         if (history_output == 1)
           {
            fprintf(historyfilepointer,''\nstart#%i&%i'',problem_data.problem_number,
problem_iteration+1);
           } /* end if */

      /* initialize display */
```

```
if (use_display == 1)
   {
   vga_init();   /* initialize graphics mode */
   cleardisplay();

   vga_setmode(VGAMODE);  /* set to appropriate mode */
   gl_setcontextvga(VGAMODE);
   physicalscreen = gl_allocatecontext();
   gl_getcontext(physicalscreen);

setcustompalette();    /* initialize custom palette */
   /* initfont() here caused trouble with planar 256 color modes. */

   gl_setcontextvgavirtual(VGAMODE);
   backscreen = gl_allocatecontext();
   gl_getcontext(backscreen);

initfont();

   gl_setcontextvgavirtual(VGAMODE);
   background = gl_allocatecontext();
   gl_getcontext(background);

   drawbackground();

   framerate = 0;
   framecount = 0;
   frameclock = clock();
   } /* if usedisplay */


 /* swarm initialization */

  initialize_particle_positions();
  initialize_particle_velocities();
  initialize_particle_inertia_values();
  initialize_particle_function_values();
  initialize_swarm_max_velocity();

 /* start sequential swarm iteration loop */

 pcle = 0;

 do
    {

    /* run through all the particles in sequence */
```

```
        if (use_display == 1)
           {
           if (updatecount  > display_refresh)
              {
              drawscreen();
              drawgraph();
             updatecount = 0;
              }
           }

    if (use_craziness == 1)
           {
           if (random_init  > craziness_period)
              {
              initialize_particle_velocities();
              random_init = 0;
              }
           }

        ++random_init;     /* craziness counter */
        ++no_improvement_iteration;
        ++no_improvement;

        update_particle_velocity(pcle);
        update_particle_position(pcle);
        update_particle_function_value(pcle);
          update_best_function_value(pcle);
        update_inertia_max_velocity();

        if (history_output == 1)
           {
           fprintf(historyfilepointer,''\n%f'',comm_data.swarm_best_value);
           }

    /* Stopping condition */
    check_for_stop(outputfilepointer,problem_data.stopping_method);


    if (use_display == 1)
             {
             /* Update frame rate every 3 seconds. */
             framecount++;
             if (clock() - frameclock >= CLOCKS_PER_SEC)
                {
                framerate = framecount * CLOCKS_PER_SEC / (clock() - frameclock);
                framecount = 0;
```

```
                frameclock = clock();
                } /* end if */
              }

          ++updatecount;

          if (pcle == (problem_data.amount_of_particles - 1))
            {
            pcle = 0;
            }
            else
            {
            ++pcle;
        }
          } /* while pcle < p - 1 */
       while ((comm_data.function_evaluations <=
problem_data.max_function_evaluations)&&(found_solution_flag == 0));


       if (comm_data.function_evaluations == problem_data.max_function_evaluations)
          {
          result[problem_iteration].swarm_best_value = 0;
          result[problem_iteration].function_evaluations = 0;
          result[problem_iteration].converged_flag = 0;
          }

    /* print history end indicator*/
        if (history_output == 1)
          {
           fprintf(historyfilepointer,''\nend#%i&%i'',problem_data.problem_number,
problem_iteration+1);
           } /* end if */


    found_solution_flag = 0;
       comm_data.function_evaluations = 0;

    } /* end problem_iteration */
```