Critical Analysis of Angle Modulated Particle Swarm Optimisers

by

Barend Jacobus Leonard

Submitted in partial fulfillment of the requirements for the degree

Master of Science (Computer Science)

in the Faculty of Engineering, Built Environment and Information Technology

University of Pretoria, Pretoria

2017

Publication data:

Barend Jacobus Leonard. Critical Analysis of Angle Modulated Particle Swarm Optimisers. Master's dissertation, University of Pretoria, Department of Computer Science, Pretoria, South Africa, 2017.

Typeset using LaTeX 2ε
Bibliography generated with BibTeX

Critical Analysis of Angle Modulated Particle Swarm Optimisers

by

Barend Jacobus Leonard
E-mail: bleonard@cs.up.ac.za

Abstract

This dissertation presents an analysis of the angle modulated particle swarm optimisation (AMPSO) algorithm. AMPSO is a technique that enables one to solve binary optimisation problems with particle swarm optimisation (PSO), without any modifications to the PSO algorithm. While AMPSO has been successfully applied to a range of optimisation problems, there is little to no understanding of how and why the algorithm might fail. The work presented here includes in-depth theoretical and emprical analyses of the AMPSO algorithm in an attempt to understand it better. Where problems are identified, they are supported by theoretical and/or empirical evidence. Furthermore, suggestions are made as to how the identified issues could be overcome. In particular, the generating function is identified as the main cause for concern. The generating function in AMPSO is responsible for generating binary solutions. However, it is shown that the increasing frequency of the generating function hinders the algorithm's ability to effectively exploit the search space. The problem is addressed by introducing methods to construct different generating functions, and to quantify the quality of arbitrary generating functions. In addition to this, a number of other problems are identified and addressed in various ways. The work concludes with an empirical analysis that aims to identify which of the various suggestions made throughout this dissertatioin hold substantial promise for further research.

Keywords: Angle modulation, Particle swarm optimisation, Binary optimisation.
Supervisors : Prof. A. P. Engelbrecht
Department : Department of Computer Science
Degree : Master of Science

At heart, science is the quest for awesome — the literal awe that you feel when you understand something profound for the first time. It's a feeling we are all born with, although it often gets lost as we grow up and more mundane concerns take over our lives.

Sean M. Carroll

Acknowledgements

I wish to extend my deepest thanks and appreciation to the following people:

- My supervisor, Prof. A.P. Engelbrecht, for his invaluable guidance and support throughout the completion of this research.
- My mother, father, and two sisters, for their unreserved love and encouragement.
- My friends for their constant motivation and understanding.
- My fellow students, friends, and colleagues at the University of Pretoria, in particular Masters' Club, for always listening, for their vital insights, and for all the expensive food.

Contents

List of Figures

List of Graphs

List of Tables

Chapter 1

Introduction

> *Your assumptions are your windows on the world. Scrub them off every once in a while, or the light won't come in.*
>
> *Isaac Asimov*

Particle swarm optimisation (PSO) is a stochastic search method, introduced by Kennedy and Eberhart, which manipulates a population of candidate solutions in order to iteratively find better solutions in a continuous search domain [21]. The algorithm has been successfully applied to a wide range of continuous optimisation problems [13]. Kennedy and Eberhart also proposed a modified version of the PSO algorithm to solve binary optimisation problems, dubbed binary particle swarm optimisation (BPSO) [22]. Even though subsequent studies have made improvements on BPSO, many of the proposed algorithms suffered from all, or a subset of, the same complications, as noted by Pamparà *et al.* [30]:

- **Hamming cliffs**: When continuous values are represented in binary format, it often happens that consecutive continuous values result in relatively dissimilar binary representations. For example, the integers 3_{10} and 4_{10} result in the two binary representations 011_2 and 100_2, respectively. The Hamming distance between the two binary representations is 3. Consequently, many bits may need to change in a binary representation in order to arrive at a continuous solution that is, in fact, adjacent to the current solution.
- **Loss of precision and the curse of dimensionality**: Only a finite number of

bits can be used to represent continuous values. The effect is that most continuous values are merely approximated by the binary representation. Additionally, improving the accuracy of the approximation necessarily results in higher-dimensional binary representations.

- **Search space discretisation**: A further consequence of representing values in a continuous search space with binary strings is that the characteristics of the search landscape are affected. For example, plateaus may be created and would, in turn, affect the performance of the optimisation algorithm [40].

The above considerations motivated the development of a new PSO variant that can solve binary optimisation problems, but without direct interaction with the binary search space. Instead, the PSO algorithm is presented with a continuous optimisation problem whose solution can be used to efficiently generate a binary solution of any desired number of bits. Importantly, once the desired bit count is established, the same binary solution is always generated for any particular solution to the continuous problem. The continuous problem presented to PSO is to find the coefficients of a trigonometric function, such that the function can be used to generate the optimal solution to some binary problem. This approach was first proposed by Franken [18], and later formally introduced by Pamparà *et al.* [34]. Franken [18] presented a trigonometric function with four coefficients that control the function's shape. Although the origin of the function is not entirely clear, Pamparà *et al.* [34] attributed its conception to the field of signal processing in the telecommunications industry, where similar functions are often used to perform *angle modulation*[1] [3, 36]. The resulting technique was therefore named angle modulated particle swarm optimisation (AMPSO). This dissertation presents a critical analysis of AMPSO.

[1]For the remainder of this dissertation, the term "angle modulation" will only be used in the context of optimisation, and will refer solely to the process of searching for trigonometric function coefficients in an attempt to generate optimal binary solutions.

1.1 Motivation

Since the introduction of AMPSO, studies of the algorithm's performance, and comparisons to pre-existing binary optimisation algorithms, have been limited to empirical analyses [30]. In many cases, AMPSO was compared to novel algorithms that employed the same general technique as AMPSO. That is, novel algorithms that solved binary problems by searching for binary solutions via an intermediary continuous problem. Essentially, it was noted that angle modulation was not limited to being combined only with PSO, but could work with any continuous optimisation algorithm. As such, studies were produced to investigate the performance of angle modulation when combined with other continuous optimisation algorithms, such as differential evolution (DE), artificial bee colony optimisation (ABC), and genetic algorithms (GA) [16, 30, 31, 32]. While these studies are interesting in their own right, they shed little light on the theoretical aspects of the AMPSO algorithm. Furthermore, while published works are heavily biased towards positive results, there are cases where the algorithm does not perform well. In such cases, existing studies provide minimal insight as to why the algorithm fails.

As a result of the current state of research on AMPSO, there is no clear understanding of the capabilities and/or limitations of the algorithm. The work presented in this dissertation aims to provide a better understanding of many theoretical aspects of the AMPSO algorithm. This will lay the groundwork for future studies concerning AMPSO specifically, and, in some cases, angle modulation in general, to draw more reliable conclusions from empirical observations.

1.2 Objectives

This section outlines the objectives that this work aims to achieve. The objectives are not necessarily addressed in the order they are presented here. The main reason for this is that this dissertation is the culmination of a series of smaller research projects that were performed and published in isolation. While the various works are presented here as a unified research topic with a single narrative, the overarching contribution of the research only became clear during later investigations (specifically, in chapter 4 of this dissertation). Nonetheless, the earlier studies contributed as well, in the sense that

without them it would not have been obvious that a better theoretical understanding of AMPSO was required. The objectives of this study are as follows:

- To provide evidence of cases where the AMPSO algorithm fails and/or where the behaviour of the algorithm cannot readily be explained based on current knowledge.
- To identify a range of potential underlying causes of failure.
- To investigate the identified aspects theoretically and/or empirically, as deemed appropriate.
- To provide recommendations on how any identified limitations of the AMPSO algorithm could be overcome.
- To provide empirical analyses of the most promising recommendations.

1.3 Contributions

The novel contributions made by the work presented in this dissertation are listed below:

- Three novel variants of the AMPSO algorithm are introduced.
- The AMPSO generating function is studied in detail for the first time in order to establish its effect during the search process.
- It is revealed that there is a non-uniform relationship between step sizes in the continuous search space and step sizes in the binary solution space in AMPSO. The consequences of this relationship are discussed in detail.
- The convergence behaviour of particles in AMPSO is investigated empirically for the first time. The investigation is rooted in established PSO theory.
- A formal definition is provided to quantify the ability of a given generating function to solve arbitrary binary problems of which only the dimensionality is known.
- A very simple generating function that can solve low-dimensional binary problems is constructed.

- The concept of ensemble generating functions is introduced for use with arbitrarily high-dimensional binary problems.
- The frequency distribution of binary solutions in the AMPSO search space is investigated for the first time.
- A new PSO variant for binary optimisation is introduced, which guarantees that binary solutions are uniformly distributed in the search space.
- It is shown that poorly designed binary solution representations can violate the underlying assumptions made by the PSO algorithm, and thus affect the algorithm's performance.

1.4 Dissertation Outline

The rest of this dissertation is structured as follows:

- **Chapter 2** presents an overview of those aspects of optimisation and PSO that are relevant to the research presented in this dissertation.
- **Chapter 3** gives an overview of the AMPSO algorithm, and introduces three variants in an attempt to overcome a number of potential issues in the algorithm's design.
- **Chapter 4** considers a range of properties of the AMPSO algorithm that could potentially have a negative influence on the algorithm's performance. Each identified aspect is discussed in detail, and recommendations are made to overcome identified issues.
- **Chapter 5** provides an empirical analysis to evaluate the merits of the most promising recommendations that were made in chapter 4.
- **Chapter 6** concludes the dissertation.
- **Appendix A** provides a list of important acronyms, and their definitions, that are used or newly defined in this dissertation.

- **Appendix B** lists and defines the mathematical symbols used throughout this work, categorised by the chapter in which they appear.
- **Appendix C** lists the publications that were derived from the work presented in this dissertation.

Chapter 2

Optimisation Problems and Particle Swarm Optimisers

> *The greater the number of collective intellects with which an individual is involved, the more opportunities he has to diversify his knowledge and desire.*
>
> *Pierre Lévy*

This chapter provides a detailed overview of those aspects of optimisation problems and the PSO algorithm that are directly relevant to the research presented in this dissertation. An optimisation algorithm is a search method whose goal is to find the inputs to a given objective function, such that the output of the function is either minimised or maximised. The types of input variables that the objective function is defined for often place limits on the kinds of algorithms that can be used to optimise the function's output. For this reason, adapting an existing optimisation technique to a new kind of problem is not always trivial. This chapter introduces two types of optimisation problems, namely, continuous and discrete optimisation problems. The PSO algorithm was originally developed to solve continuous optimisation problems. However, a discrete version of the algorithm was later introduced to solve binary optimisation problems. The modifications that needed to be made to the PSO algorithm in order to deal with the new problem type are discussed in detail.

The definitions of the different types of optimisation problems are given in Section 2.1.

Section 2.2 gives an overview of the PSO algorithm for continuous optimisation problems. A discrete version of PSO is presented in Section 2.3. The chapter is summarised in Section 2.4.

2.1 Optimisation Problems

Any optimisation problem has an objective function f, which is defined for some domain $\mathcal{S}$. The domain is often referred to as the *search space*. Assuming that the goal is to minimise the output of the objective function, a minimisation problem can be defined as follows: given an objective function $f : \mathcal{S} \rightarrow \mathbb{R}$, find the values for the input variables $\boldsymbol{x} \in \mathcal{S}$ such that $f(\boldsymbol{x})$ is minimised. Maximisation problems can be expressed in terms of minimisation problems by observing that $\max(f) \equiv \min(-f)$. An optimisation problem may also be subject to certain constraints, in which case the valid input variables are constrained to $\boldsymbol{x} \in \mathcal{F}$, where $\mathcal{F} \subseteq \mathcal{S}$ is the *feasible space*.

The types of variables for which an objective function is defined determines the problem type. A *continuous optimisation problem* has real-valued input variables. That is, $\mathcal{S} = \mathbb{R}^{n_x}$, or $x_j \in \mathbb{R}$, for each dimension $j = 1, \ldots, n_x$. If $x_j \in \mathbb{D}$, where $\mathbb{D} \subseteq \mathbb{R}$ is some discrete subset of $\mathbb{R}$, then the problem is referred to as a *discrete optimisation problem*. In this dissertation, a specific kind of discrete optimisation problem will be investigated, namely, *binary optimisation problems*. In the case of a binary optimisation problem, $x_j \in \mathbb{B}$, where $\mathbb{B} = \{0, 1\}$.

For the remainder of this dissertation, the terms "continuous optimisation problem" and "continuous problem" will be used interchangeably. Similarly, no distinction will be made between the terms "binary optimisation problem" and "binary problem".

2.2 Particle Swarm Optimisation

This section provides an overview of the PSO algorithm. The basic workings of the algorithm are presented in Section 2.2.1. The control parameters of PSO are discussed in Section 2.2.2, while some theoretical aspects of PSO are reviewed in Section 2.2.3.

2.2.1 Overview

Particle swarm optimisation is a stochastic, population-based search algorithm, introduced by Kennedy and Eberhart [9, 21]. The PSO algorithm maintains a population, or a *swarm*, of potential solutions, called *particles*. Every particle i in the swarm has a *position* $\boldsymbol{x}_i$ and a *velocity* $\boldsymbol{v}_i$ in the n_x-dimensional search space. In addition, each particle also keeps track of the best position $\boldsymbol{y}_i$ it has found during the search process, known as the particle's *personal best position*. The original PSO algorithm implemented a star topology, such that every particle shares information with every other particle in the swarm. This configuration is referred to as the *global best PSO*, or *gbest PSO*. In the gbest PSO, the best position found by the swarm is called the *global best position* and is denoted by $\hat{\boldsymbol{y}}$.

In the original version of the gbest PSO, the velocity of each particle is updated at every time step $t+1$ during the search process as follows:

$$\boldsymbol{v}_i(t+1) = \boldsymbol{v}_i(t) + c_1\boldsymbol{r}_1(t)[\boldsymbol{y}_i(t) - \boldsymbol{x}_i(t)] + c_2\boldsymbol{r}_2(t)[\hat{\boldsymbol{y}}(t) - \boldsymbol{x}_i(t)], \tag{2.1}$$

where c_1 and c_2 are acceleration coefficients, and $r_{1j}(t)$ and $r_{2j}(t)$ are random values, sampled from $U(0,1)$ in each dimension $j = 1, \ldots, n_x$. The position of each particle is then updated using

$$\boldsymbol{x}_i(t+1) = \boldsymbol{x}_i(t) + \boldsymbol{v}_i(t+1). \tag{2.2}$$

The resulting behaviour is that particles stochastically return to regions of the search space where good solutions have previously been found.

The original PSO algorithm was developed to solve continuous problems. As a result, no constraints are placed on the values that a particle's position may assume. That is, $x_{ij} \in \mathbb{R}$, or $\boldsymbol{x}_i \in \mathbb{R}^{n_x}$. Because particle positions are used as input vectors to the objective function, this satisfies the definition of an unconstrained continuous problem, as discussed in Section 2.1.

2.2.2 Control Parameters

The PSO algorithm has two primary goals. The first goal is to *explore* the search space in order to determine which regions are likely to contain promising solutions. The second goal is to *exploit* the most promising region in an attempt to find the best solution.

The exploration and exploitation capabilities of the PSO algorithm are controlled by the two acceleration coefficients c_1 and c_2 (see Section 2.2.1). The c_1 coefficient is called the *cognitive acceleration coefficient*, because it scales the force that attracts a particle towards its own personal best position. Similarly, the c_2 coefficient is known as the *social acceleration coefficient*, because it scales the attraction force towards the global best position. By adjusting c_1 and c_2, a trade-off between exploration and exploitation is established.

Note from Equation (2.1) that a particle's previous velocity is always added to its new velocity at every time step. The previous velocity term is referred to as the *inertia term* and causes a particle to maintain a bias towards its current direction. However, adding the previous velocity at every time step also has the adverse effect of increasing the magnitude of the particle's velocity at every time step. The effect is that exploitation is hindered, because the step sizes of particles keep increasing. The problem of increasing particle velocities can be alleviated in various ways, including the use of velocity clamping [10], or by using a constriction coefficient [5, 6]. However, the most widely adopted solution is to scale the effect of the particle's previous velocity by including an inertia weight [37].

Shi and Eberhart [37] introduced a modified version of PSO that uses an inertia weight to scale the inertia term in Equation (2.1). To achieve this, Equation (2.1) was modified such that

$$\boldsymbol{v}_i(t+1) = \omega \boldsymbol{v}_i(t) + c_1 \boldsymbol{r}_1(t)[\boldsymbol{y}_i(t) - \boldsymbol{x}_i(t)] + c_2 \boldsymbol{r}_2(t)[\hat{\boldsymbol{y}}(t) - \boldsymbol{x}_i(t)], \tag{2.3}$$

where ω is the inertia weight. Scaling the inertia term by a factor of $\omega < 1$ addresses the problem of increasing particle velocities. Furthermore, the inertia weight also prevents sudden, large changes in direction as particles move through the search space, until enough evidence has been gathered to justify such changes. For specific configurations of ω, c_1, and c_2, swarm convergence is theoretically guaranteed [4, 11, 35, 39].

2.2.3 Convergence in Mean and Standard Deviation

The term "convergence" is often used in literature to refer to different concepts. Throughout this dissertation, "convergence" will be used to refer to the stabilisation of particles

around known positions in the search space:

A particle i is said to be order-1 stable when the expected position $E[\boldsymbol{x}_i]$ of the particle is equal to some point $\boldsymbol{\mu}$. Under the correct conditions, particles in PSO will eventually become order-1 stable [11, 35, 39]. That is,

$$\lim_{t\to\infty} E[\boldsymbol{x}_i(t)] = \boldsymbol{\mu}. \tag{2.4}$$

Poli [35] also showed that PSO is order-2 stable. That is, the standard deviation σ_i of the particle's positions around $\boldsymbol{\mu}$ converges, such that

$$\lim_{t\to\infty} \sigma_i(t) = \frac{1}{2}\sqrt{\frac{c(\omega+1)}{c(5\omega-7)-12\omega^2+12}} \cdot |\hat{\boldsymbol{y}}(t) - \boldsymbol{y}_i(t)|, \tag{2.5}$$

where $c = c_1 = c_2$. Consequently, a particle i will continue searching within a region bounded by σ_i, unless $\boldsymbol{y}_i(t) = \hat{\boldsymbol{y}}(t)$. When σ_i remains stationary, the particle is said to be order-2 stable. Order-2 stability is equivalent to convergence. The parameter values given in Table 2.1 have been shown to lead to convergent particle trajectories [4, 35, 39].

If a particle i is order-1 stable, the standard deviation $\sigma_i(t)$ can be interpreted as the expected magnitude of the velocity of the particle at time $t+1$. The expected average magnitude of particle velocities in the swarm at time $t+1$ is then given by

$$E[v_{avg}(t+1)] = \frac{\sum_{i=1}^{n_s} \sigma_i(t)}{n_s}, \tag{2.6}$$

where n_s is the number of particles in the swarm, and the particles in the swarm are order-1 stable. When using the parameter values listed in Table 2.1, the deviation is given by

$$\sigma_i(t) = 1.0432 \cdot |\hat{\boldsymbol{y}}(t) - \boldsymbol{y}_i(t)|. \tag{2.7}$$

2.3 Binary Particle Swarm Optimisation

A discrete version of PSO that can optimise binary problems was introduced by Kennedy and Eberhart [22, 23]. This algorithm is known as BPSO. An overview of BPSO is given in Section 2.3.1. The algorithm is then critiqued in Section 2.3.2.

Table 2.1: Convergent PSO parameter values.

Parameter	Value
ω	0.729844
c_1	1.496180
c_2	1.496180

2.3.1 Overview

In order to solve binary problems with PSO, the first required modification is that particle positions must be binary vectors. That is, $\boldsymbol{x}_i \in \mathbb{B}^{n_x}$. Particle movements then imply a bit flip in one or more dimensions of the position vector. The problem with binary position vectors is that the velocity update equation in PSO (Equation (2.3)) is no longer applicable, since it entails performing real-valued operations on the individual dimensions of a particle's position.

The solution that Kennedy and Eberhart proposed was to change the interpretation velocity vector of a particle. Instead of a step size, the velocity v_{ij} of a particle in BPSO is used to compute the probability that the position of particle i in dimension j is 1. This interpretation immediately implies that particle velocities must be normalised such that $v'_{ij} \in [0, 1]$. In order to obtain normalised velocities, the velocity of each particle is scaled using the sigmoid function, such that

$$v'_{ij}(t) = sig(v_{ij}(t)) = \frac{1}{1 + e^{-v_{ij}(t)}}. \tag{2.8}$$

The position of a particle is then updated probabilistically, such that

$$x_{ij}(t+1) = \begin{cases} 1 & \text{if } r_{3j} < v'_{ij}(t) \\ 0 & \text{otherwise,} \end{cases} \tag{2.9}$$

where $r_{3j} \sim U(0, 1)$.

2.3.2 Critique

The BPSO algorithm described in Section 2.3.1 has been criticised in literature for fundamentally changing the behaviour of the PSO algorithm [12, 24, 30]. In particular, it has been noted that the BPSO algorithm changes the meaning of the control parameters in PSO, and that, as a result, PSO theory does not apply to BPSO. This section highlights and discusses the implications that the new interpretation has on for the various components of the PSO algorithm.

2.3.2.1 Acceleration Coefficients

In BPSO the two acceleration coefficients, c_1 and c_2, no longer control the trade-off between exploration and exploitation. Instead, maximum exploration is achieved when $v'_{ij} = 0.5$. That is, the probability that each bit equals 1 is exactly 0.5. Thus, because of the sigmoid normalisation in Equation (2.8), maximum exploration is achieved in BPSO when $v_{ij} = 0$, for all $j = 1, \ldots, n_x$.

2.3.2.2 Inertia Weight and Velocity Clamping

The meaning of the inertia weight is also different in the context of BPSO. Instead of smoothing particle trajectories, the purpose of ω in BPSO is only to prevent the magnitudes of particle velocities to grow too rapidly during the beginning of the search. When particle velocities in BPSO become much larger or much smaller than 0, the algorithm can no longer explore the search space. Therefore, ω should be close to zero during the beginning of the search process. In addition to the inertia weight, velocity clamping is also required in BPSO, to prevent saturation of the sigmoid normalisation function. If particle velocities become too large or too small, the respective probability of a bit being 1 or 0 approaches 1, effectively bringing an end to the search process. To prevent this, particle velocities should be clamped such that their magnitudes remain within the active range of the sigmoid function.

2.3.2.3 Particle Positions and Fitness Values

Another important insight is that the concept of particle positions in the search space no longer makes sense in the context of BPSO. The reason for this is that a particle does not even have a position until its velocity is evaluated using Equation (2.9). Furthermore, the random variable r_{3j} in Equation (2.9) means that consecutive evaluations of the same particle would likely yield different positions. As a direct consequence, the fitness values associated with particles no longer convey any useful information about the search space. The result is that the cognitive and social attraction forces in Equation (2.1) — which are supposed to guide particles to good solutions in the search space — become meaningless in the context of BPSO.

2.3.2.4 PSO Theory

In addition to the concerns discussed above, it should be noted that PSO theory, as explained in Section 2.2.3, is no longer applicable in the case of BPSO. Essentially, this means that a new theoretical framework is required in order to fully understand the BPSO algorithm, and to perform meaningful studies regarding its behaviour. Thus, a reasonable conclusion is that, while BPSO borrows terminology from the PSO paradigm, it really belongs in a different category of optimisation algorithms.

2.4 Summary

This chapter gave a formal definition of continuous- and discrete optimisation problems. Two versions of the particle swarm optimisation (PSO) algorithm were then introduced. The first version of PSO that was discussed is used to solve continuous problems and includes an inertia weight to smooth particle trajectories as they move through the search space. The second version is called binary PSO (BPSO), and is used to solve binary optimisation problems.

BPSO was criticised for changing the interpretation of the velocity vectors in PSO. This new interpretation implies that none of the control parameters in BPSO retain their original meaning. Furthermore, PSO theory is not applicable to the BPSO algorithm.

Consequently, it was concluded that BPSO should rather not be classified as a particle swarm optimiser, even though it borrows the terminology.

Chapter 3

Angle Modulated Particle Swarm Optimisation

Man's mind, once stretched by a new idea, never regains its original dimensions.

Oliver Wendell Holmes Sr.

AMPSO is a technique that enables the standard PSO algorithm to optimise binary problems without requiring any modifications to the PSO algorithm. The technique was originally suggested by Franken [18], but was formally introduced by Pamparà *et al* [34]. This chapter gives an overview of the AMPSO algorithm. Additionally, a number of potential drawbacks of the technique are identified. Three variants are then introduced in an attempt to overcome these drawbacks. The new variants are compared with the normal AMPSO algorithm to confirm whether they are superior.

Section 3.1 gives an overview of the basic AMPSO algorithm. Section 3.2 discusses a number of shortcomings in the AMPSO algorithm. In Section 3.3, three new variants of the AMPSO algorithm are introduced. The experimental setup is explained in Section 3.4, while the results are discussed in Section 3.5. The chapter is summarised in Section 3.6.

3.1 Overview

AMPSO makes use of the PSO algorithm (see Section 2.2) to optimise the coefficients (a, b, c, and d) of the following trigonometric function:

$$g(x) = sin[2\pi(x-a)b\ cos(2\pi(x-a)c)] + d. \tag{3.1}$$

This function is often referred to as the *generating function*. The generating function has four coefficients that control various aspects of the function as follows:

- a: controls the horizontal shift,
- b: controls the amplitude of the *cos* wave, which, in turn, controls the frequency of the *sin* wave,
- c: controls the frequency of the *cos* wave, and
- d: controls the vertical shift.

When using PSO to search for good coefficients for g, the position of a particle i has the form $\boldsymbol{x}_i = (a, b, c, d)$. In order to generate a binary solution from a given particle's position, the values a, b, c, and d are substituted into g. Then the generating function is sampled at regular intervals $x = 0, 2, 3, \ldots, n_b - 1$, where n_b is the required number of binary digits. A binary solution $\mathcal{B} \in \mathbb{B}^{n_b}$ is constructed by noting the value of $g(x)$ at every sampling position:

$$\mathcal{B}_j = \begin{cases} 0 & \text{if } g(x) \leq 0, \\ 1 & \text{otherwise.} \end{cases} \tag{3.2}$$

Consequently, an n_b-dimensional binary problem may potentially be solved using a four-dimensional PSO. The process of generating a binary solution in AMPSO is illustrated in Figure 3.1.

Note that AMPSO makes no modifications whatsoever to the original PSO algorithm. That is, particle positions are still real-valued vectors, and the interpretation of particle velocities remains unchanged. As a result, the meanings of the control parameters in PSO also remain unchanged, and PSO theory readily applies to AMPSO.

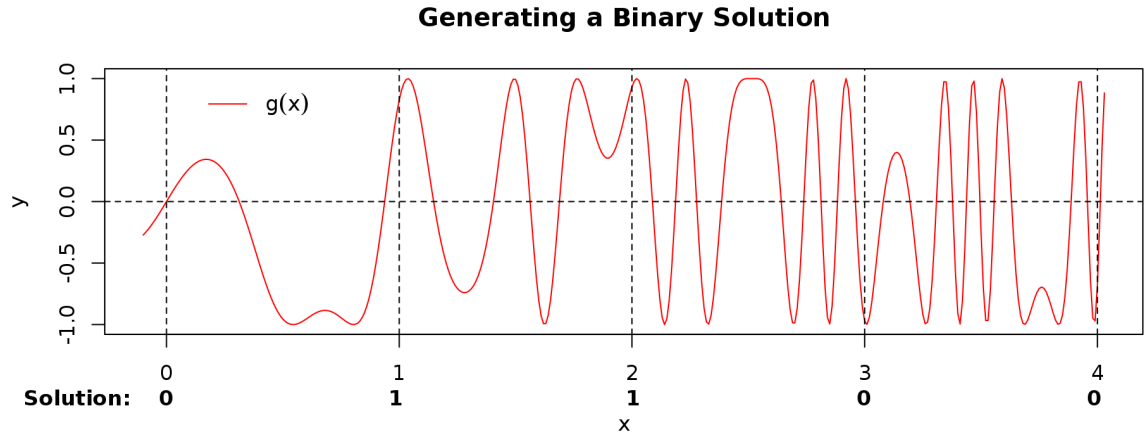


Figure 3.1: A five-dimensional binary solution is generated by sampling g at regular intervals. In this example, $\boldsymbol{x}_i = (0.0, 0.5, 0.8, 0.0)$, or $a = 0$, $b = 0.5$, $c = 0.8$, and $d = 0$.

3.2 Potential AMPSO Pitfalls

This section notes a number of potential flaws in the AMPSO algorithm. The work presented here is the first and only study, as of this writing, to identify and attempt to rectify these weaknesses in AMPSO.

The generating function enables AMPSO to produce different binary solutions. However, some solutions may be easier to generate than other solutions, because of the following shortcomings in the AMPSO technique:

1. **The generating function has a fixed amplitude:** The generating function g is a *sin* wave, which means that it has an amplitude of 1, and AMPSO has no means of adjusting the amplitude. Modifying the amplitude of the generating function would effectively scale the effect of the vertical shift coefficient, d, potentially allowing AMPSO to find some solutions faster.

2. **The sampling interval is always the same:** AMPSO samples the generating function at integer intervals. If AMPSO could adjust the sampling interval, some solutions may be easier to produce.

3. **Sampling always starts at 0:** Allowing AMPSO to determine the sampling starting position automatically may enhance the search capabilities of the algo-

rithm.

4. **The initialisation domain limits the diversity of solutions:** As noted above, the generating function in AMPSO has an amplitude of 1. During initialisation, particles are usually distributed in the range $[-1, 1]^4$ [30, 34]. The implication is that all solutions are initialised with $-1 \leq d \leq 1$, which means that the vertical shift of the generating function can never exceed its amplitude. Thus, the range of the generating function will always include $g(x) = 0$. From Equation (3.2), this implies that 0-bits can never be entirely eliminated from any candidate solution during initialisation with absolute certainty.

3.3 AMPSO Variants

This section introduces three variants of the AMPSO algorithm that address the above-mentioned limitations. The amplitude AMPSO (A-AMPSO) algorithm is discussed in Section 3.3.1. Section 3.3.2 gives an overview of the min-max AMPSO (MM-AMPSO) algorithm, while the increased-domain AMPSO (ID-AMPSO) variant is discussed in Section 3.3.3.

3.3.1 Amplitude AMPSO

The A-AMPSO algorithm augments particle positions with an additional dimension to control the amplitude of the generating function. This gives rise to five-dimensional particles of the form $\boldsymbol{x}_i = (a, b, c, d, e)$, while the generating function changes to

$$g(x) = e\ sin[2\pi(x - a)b\ cos(2\pi(x - a)c)] + d, \tag{3.3}$$

where e is the amplitude of g. Equation (3.3) is then substituted into Equation (3.2) when generating binary solutions. This variant addresses shortcomings 1 and 4 from Section 3.2, as is illustrated in Figure 3.2.

3.3.2 Min-Max AMPSO

The MM-AMPSO variant adds two additional dimensions to particles' positions. The two new dimensions control the bounds of the sampling range of the generating function. The position of a particle becomes a six-dimensional vector of the form $\boldsymbol{x}_i = (a, b, c, d, \alpha_1, \alpha_2)$. Let $\alpha_\ell = min\{\alpha_1, \alpha_2\}$ be the lower bound, and $\alpha_u = max\{\alpha_1, \alpha_2\}$ be the upper bound, then the generating function is sampled at every δ^{th} position in the range $[\alpha_\ell, \alpha_u)$, where

$$\delta = \frac{\alpha_u - \alpha_\ell}{n_b}. \tag{3.4}$$

This variant addresses shortcomings 2 and 3 from Section 3.2. This is illustrated in Figure 3.3.

3.3.3 Increased-Domain AMPSO

The final variant is ID-AMPSO. In ID-AMPSO the initialisation range is simply increased from $[-1, 1]^4$ to $[-1.5, 1.5]^4$. This variant addresses shortcoming 4 from Section 3.2, but without the increase in particle dimensionality that was required for A-AMPSO (Section 3.3.1). An illustration is provided in Figure 3.4.

3.4 Experiments

To determine whether the proposed variants in Section 3.3 provide improvements over the standard AMPSO algorithm, an empirical study was performed. This section outlines the experimental procedure that was followed.

Section 3.4.1 describes the benchmark problems that were used, while the algorithmic setup is explained in Section 3.4.2.

3.4.1 Benchmark Problems

Five binary benchmark problems were used in this study. The implementations of these problems were provided by the computational intelligence library (CIlib) [7, 33]. Each problem is discussed below.

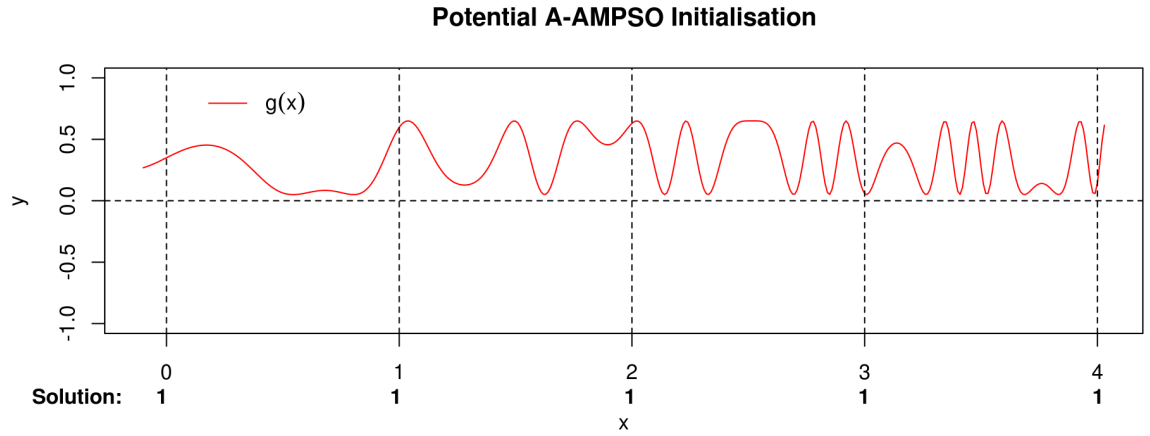


Figure 3.2: An illustration of the solution generated by a newly initialised A-AMPSO particle $\boldsymbol{x}_i = (0.3, 0.0, 0.5, 0.8, 0.35)$, or $e = 0.3$, $a = 0$, $b = 0.5$, $c = 0.8$, and $d = 0.35$. The amplitude coefficient allows the algorithm to initialise solutions that contain no 0 bits. By adjusting the d coefficient, solutions with no 1 bits can also be initialised.

3.4.1.1 N-Queens

The n-queens problem is a chess board problem, where the objective is to place n queens on an $n \times n$ chess board in such a way that no queen is able to capture any other queen, by the standard rules of chess. Various solutions to the problem can be found in [8], [29], and [38]. The n-queens problem is a minimization problem.

For the purpose of this study, a solution to the n-queens problem was represented with an n^2-bit string, where a 1-bit indicates that the corresponding square on the chess board contains a queen, and a 0-bit indicates an empty square.

The problem was investigated for board sizes of 8×8, 9×9, 10×10, 11×11, 12×12, 20×20, and 25×25. These board sizes led to solution representations of 64, 81, 100, 121, 144, 400, and 625 bits, respectively.

3.4.1.2 Knights' Coverage

The knights' coverage problem [17] is also a chess-board problem and is defined as follows: for any $n \times n$ chess board, use the minimum number of knights to *cover* the maximum number of squares on the chess board. A square on the chess board is considered to be

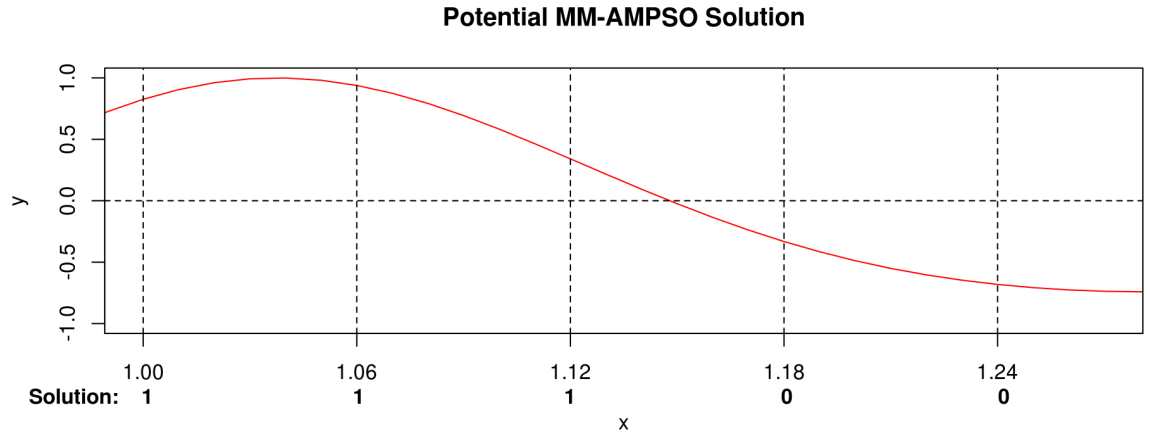


Figure 3.3: An illustration of the solution generated by an MM-AMPSO particle with position $\boldsymbol{x}_i = (1.0, 1.3, 0.0, 0.5, 0.8, 0.0)$, or $\alpha_1 = 1.0$, $\alpha_2 = 1.3$, $a = 0$, $b = 0.5$, $c = 0.8$, and $d = 0.35$. The sampling range is controlled by α_1 and α_2. This allows the algorithm generate a broader range of solutions more easily. Note that MM-AMPSO can easily generate solutions with no 0's or no 1's as well.

covered if and only if a knight is occupying the square, or a knight may move to the square in a single move from its current location. The allowed movements of knights are defined by the standard rules of chess. This problem is a minimization problem.

A solution to the knights' coverage problem was represented by an n^2-bit string. A 1-bit indicates that the corresponding square on the chess board contains a knight, and a 0-bit indicates an empty square.

For this study, the problem was optimized for chess board sizes of 8×8, 9×9, 10×10, 11×11, 12×12, and 20×20. The resulting solution representations had 64, 81, 100, 121, 144, 225, and 400 dimensions, respectively.

3.4.1.3 Knight's Tour

The knight's tour problem [20] is yet another chess board problem. Given an $n \times n$ chess board, the aim is to find a sequence of moves for a single knight, such that every square on the board is visited exactly once. The knight may start on any square, but its movement is restricted by the normal rules of chess. The knight's tour is a maximization problem.

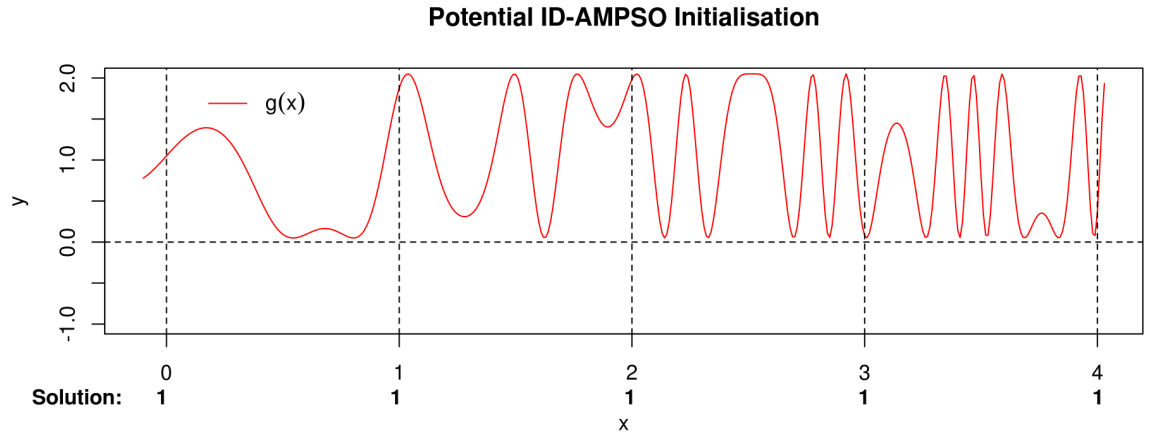


Figure 3.4: An illustration of the solution generated by a newly initialised ID-AMPSO particle $\boldsymbol{x}_i = (0.0, 0.5, 0.8, 1.05)$, or $a = 0$, $b = 0.5$, $c = 0.8$, and $d = 1.05$. The increased initialisation domain allows the algorithm to initialise solutions that are guaranteed to contain no 0 bits, because the amplitude of g is 1. By adjusting the d coefficient, solutions with no 1 bits can also be initialised.

From any position on the chess board, a knight has a maximum of eight valid moves. Each move can therefore be encoded as a 3-bit binary value. The complete solution to this problem is then a bit string with $3n^2$ bits.

This problem was investigated for the following board sizes: 4×4, 5×5, 6×6, 7×7, 8×8, 10×10, and 12×12. The respective solution representations were 48, 75, 108, 147, 192, 300, and 432 bits in length.

3.4.1.4 Deceptive Problems

Finally, the following two deceptive problems were used in this study: order-3 deceptive, and order-5 deceptive. The concept of deception was introduced by Goldberg and aims to deliberately mislead the evolutionary process modelled in genetic algorithms [19]. Deceptive problems are designed in such a way that a *deceptive attractor* leads the search away from the global optimum. For example, for an order-3 deceptive problem f, the bits in a candidate solution are grouped into $\frac{n_b}{3}$ three-bit groups. Assuming that f is a maximization problem, with a global maximum at $(1, 1, 1, \ldots)$ and a global minimum at $(0, 0, 0, \ldots)$, the relationships in Table 3.1 must hold for every group of three bits in

Table 3.1: Order-3 Deceptive Relationships

$f(\ldots 0^{**} \ldots) > f(\ldots 1^{**} \ldots)$	$f(\ldots 00^{*} \ldots) > f(\ldots 11^{*} \ldots), f(\ldots 01^{*} \ldots), f(\ldots 10^{*} \ldots)$
$f(\ldots {}^{*}0^{*} \ldots) > f(\ldots {}^{*}1^{*} \ldots)$	$f(\ldots 0^{*}0 \ldots) > f(\ldots 1^{*}1 \ldots), f(\ldots 0^{*}1 \ldots), f(\ldots 1^{*}0 \ldots)$
$f(\ldots {}^{**}0 \ldots) > f(\ldots {}^{**}1 \ldots)$	$f(\ldots {}^{*}00 \ldots) > f(\ldots {}^{*}11 \ldots), f(\ldots {}^{*}01 \ldots), f(\ldots {}^{*}10 \ldots)$

a candidate solution. Both the order-3 and order-5 deceptive problems are defined as maximization problems.

For this study, the order-3 and order-5 deceptive problems were optimized in the following dimensions: 30, 45, 60, 75, and 90.

3.4.2 Algorithmic Setup

The three AMPSO variants proposed in Section 3.3 were compared to the original AMPSO on the five benchmark functions described in Section 3.4.1. Parameters settings were obtained using iterated F-race [1], [2]. The parameters were optimised simultaneously for all four algorithms across all problems and dimensions, and were then used for every test case. The parameters are shown in Table 3.2.

Each algorithm executed for 1000 iterations, and average results over 30 runs are reported in Section 3.5. Finally, pair-wise Mann-Whitney U tests were performed to determine significant wins and losses for all algorithms across all problems at a 95% confidence interval.

3.5 Results and Discussion

Graphs 3.1 to 3.8 show the fitness profiles of the different algorithms for the lowest and highest dimensions on the various benchmark problems.

It is observed from Graph 3.1 that AMPSO, ID-AMPSO and A-AMPSO performed comparably in 64 dimensions on the n-queens problem, with AMPSO reaching a lower average fitness at times. MM-AMPSO lagged notably behind. This indicates that the two additional dimensions added to the position vector in MM-AMPSO imposed additional

Table 3.2: Best performing parameter values.

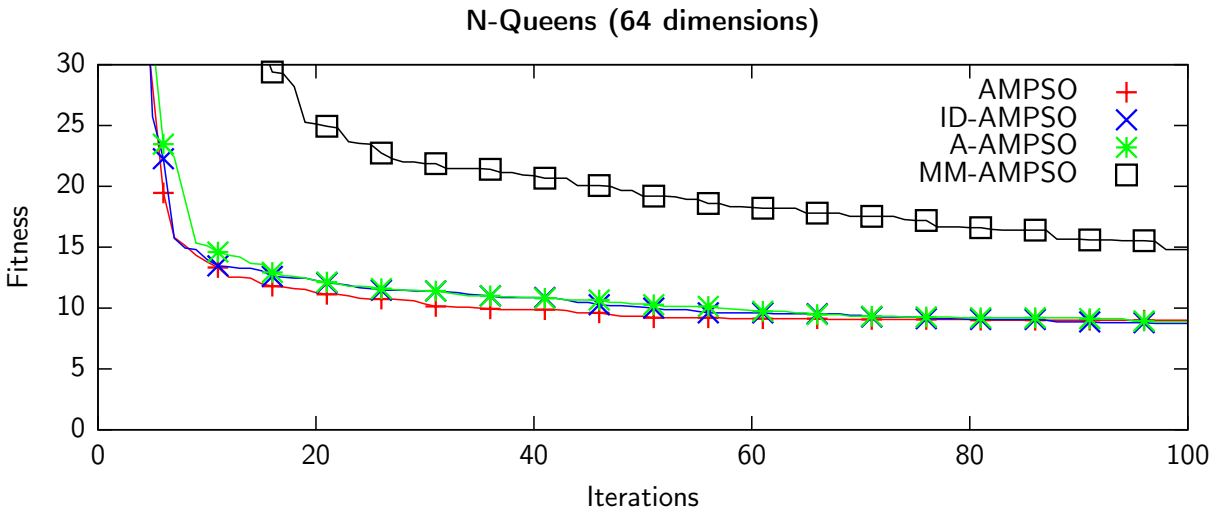
Parameter	Value
ω	0.4237
c_1	1.6369
c_2	1.5943

complexity that the algorithm was not able to overcome for this particular problem. Graph 3.2 shows that, for 652 dimensions, AMPSO has the steepest initial gradient, but again ID-AMPSO and A-AMPSO eventually reached fitness values close to that of AMPSO.

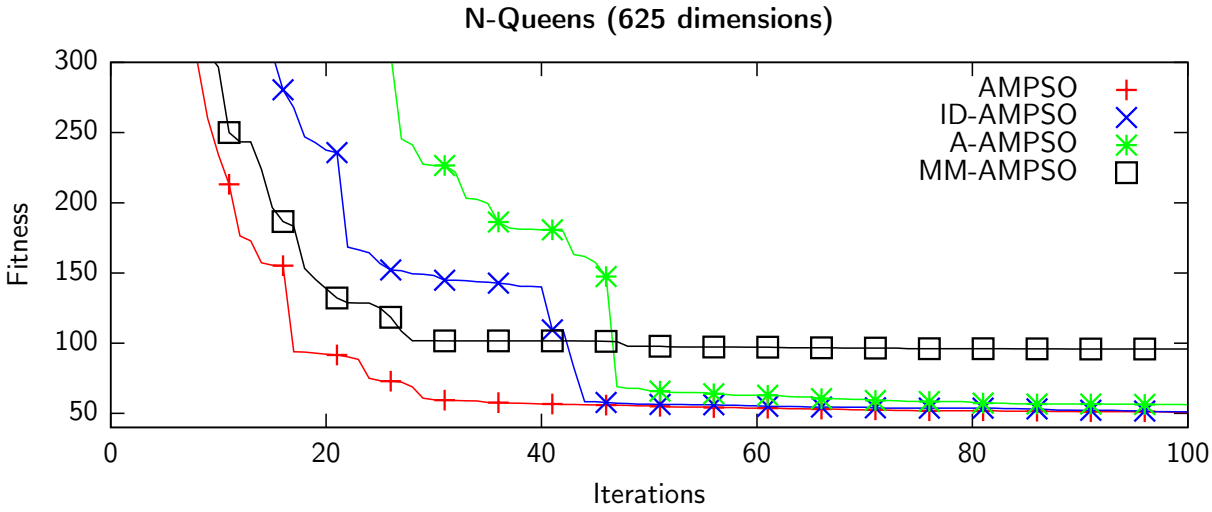
Graphs 3.3 and 3.4 show the fitness profiles on the knights' coverage problem. It is observed that all algorithms performed comparably in 64 dimensions, with MM-AMPSO obtaining a slightly lower fitness value on average. In 400 dimensions, MM-AMPSO had a lower initial gradient compared to the other algorithms. However, MM-AMPSO overtook all the competitors in the last 100 iterations. The results for the knight's coverage problem are in contrast with what was observed for the n-queens problem. These results indicate that the ability to adjust the sampling domain of the generating function during the search process can be beneficial in some problem cases.

For the knight's tour problem, Graph 3.5 shows that AMPSO obtained the highest average fitness in 48 dimensions. All algorithms stagnated at inferior solutions on this problem, with MM-AMPSO having the worst average fitness. In Graph 3.6, it is observed that MM-AMPSO still had the worst average fitness in 432 dimensions, but that ID-AMPSO obtained the best average fitness after 1000 iterations. Again this result indicates that there are problem cases for which AMPSO suffers some loss in performance due to the limitations of the generating function. In the case of the knight's tour problem, the larger initialisation domain of the ID-AMPSO variant is only beneficial in high dimensions, and only towards the end of the 1000-iteration search period.

Graphs 3.7 and 3.8 show the fitness profiles for the order-5 deceptive problems. Fitness graphs for the order-3 deceptive problems are omitted, but are similar to those in

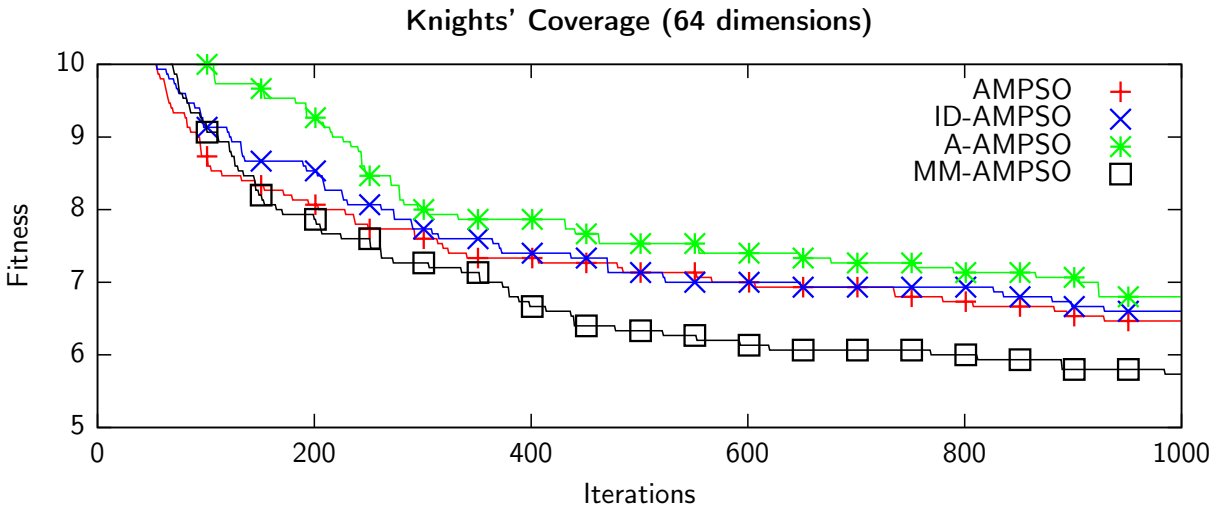


Graph 3.1: Fitness profiles for the 64-dimensional n-queens problem.

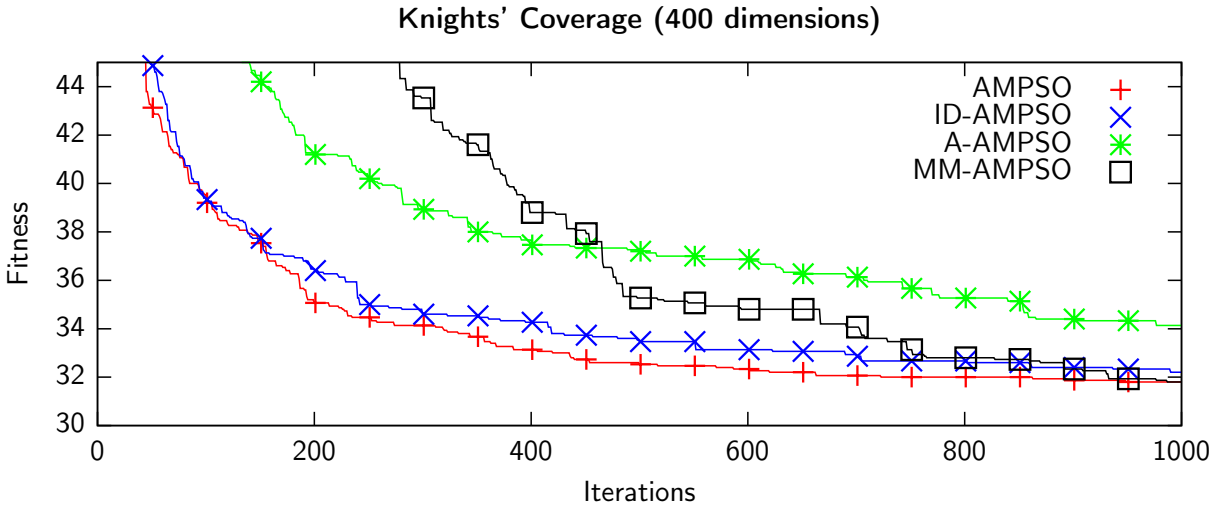


Graph 3.2: Fitness profiles for the 625-dimensional n-queens problem.

Graphs 3.7 and 3.8. It is immediately obvious that all three proposed variants obtained much better average results than AMPSO in the initial phases of the search process. In fact, the variants consistently found the optimal solutions to these problems during initialization. The reason for this is that the solution to all the deceptive problems is a bit string consisting only of 1-bits. Recall from Section 3.3 that all three the variants are able to create such solutions during initialization. Graphs 3.7 and 3.8 also indicate that the problem's dimensionality affects only ID-AMPSO's ability to consistently initialize



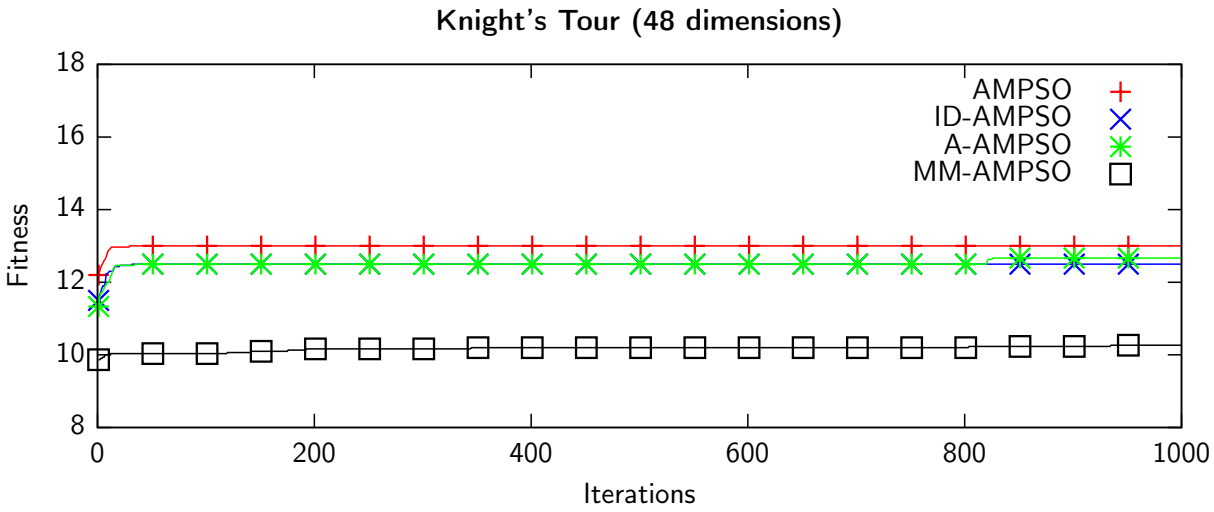
Graph 3.3: Fitness profiles for the 64-dimensional knights' coverage problem.



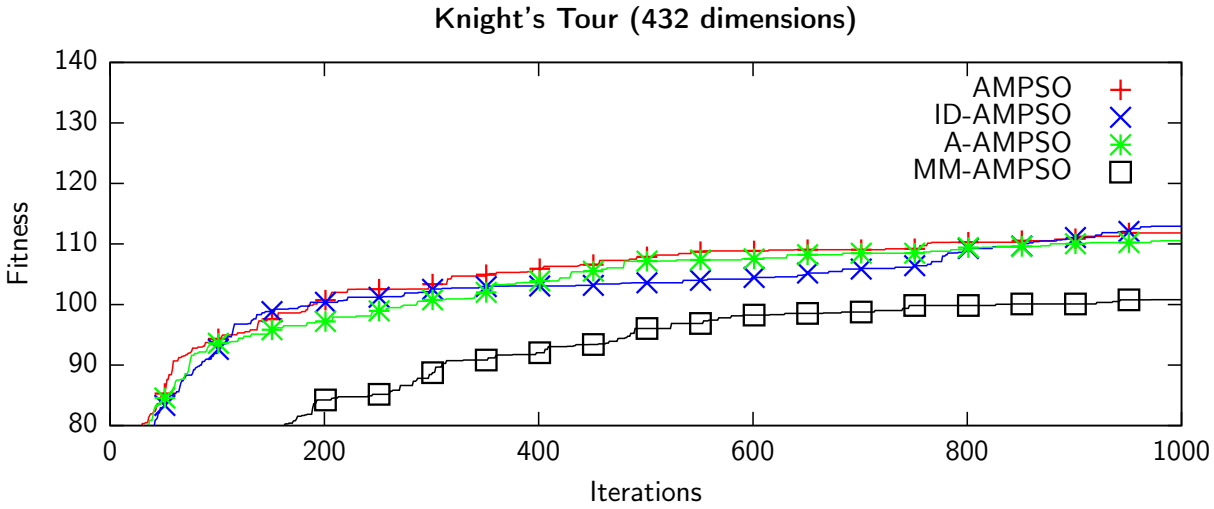
Graph 3.4: Fitness profiles for the 400-dimensional knights' coverage problem.

1-bit solutions, and the effect only persists through the first few iterations. AMPSO is noticeably affected by an increase in dimensionality for the order-5 deceptive problem. However, due to a phenomenon known as *roaming particles* [14, 15] — where particles move outside the initialisation domain — AMPSO is also able to find the optimal solutions to all the deceptive problems within the first 10 to 70 iterations, regardless of dimensionality.

Tables 3.3 to 3.5 report the statistical wins and losses for all algorithms on the



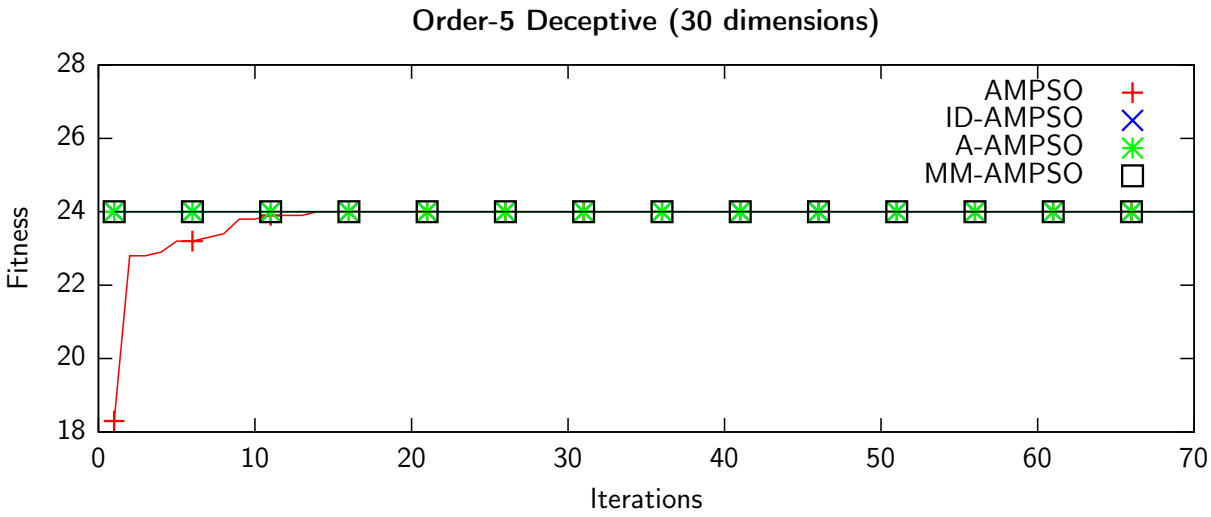
Graph 3.5: Fitness profiles for the 48-dimensional knight's tour problem.



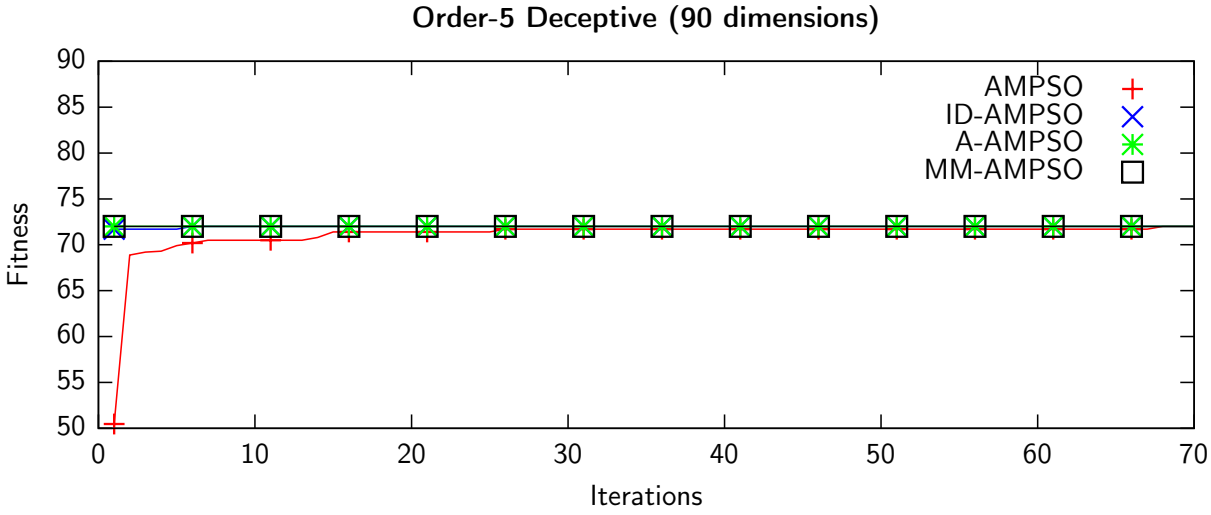
Graph 3.6: Fitness profiles for the 432-dimensional knight's tour problem.

n-queens-, knights' coverage-, and knight's tour problems after 1000 iterations. A statistical win or loss necessarily implies a statistically significant difference in performance. Statistical significance was measured at a 95% confidence interval. The statistical results for the deceptive problems are not reported in tables, because, as discussed above, all the algorithms solved all the deceptive problems, implying no statistically significant difference in performance after 1000 iterations.

Each table row indicates the wins and losses for all four algorithms on a specific



Graph 3.7: Fitness profiles for the 30-dimensional order-5 deceptive problem.



Graph 3.8: Fitness profiles for the 90-dimensional order-5 deceptive problem.

problem. Because a pair-wise comparison was performed, an algorithm can have at most three wins or three losses. In addition, a win for one algorithm necessarily implies a loss for some other algorithm. In the cases where one algorithm obtained more wins than all the other algorithms, the number of wins for the winning algorithm is printed in bold.

Table 3.3 shows that AMPSO, A-AMPSO, and ID-AMPSO all obtained statistically superior fitness values to the other three algorithms on some of the n-queens problems. AMPSO obtained the most wins overall, while MM-AMPSO lost in all cases. The results

Table 3.3: Statistical Wins and Losses for the n-queens Problem

Dimensions	AMPSO		A-AMPSO		ID-AMPSO		MM-AMPSO	
	wins	losses	wins	losses	wins	losses	wins	losses
64	**3**	0	1	1	1	1	0	3
81	1	0	1	1	**2**	0	0	3
100	**2**	0	1	1	1	0	0	3
121	1	0	1	0	1	0	0	3
144	1	1	**2**	0	1	0	0	3
400	**2**	0	1	0	1	1	0	3
625	2	0	1	2	2	0	0	3

corroborate the observations made in Graphs 3.1 and 3.2.

Table 3.4 shows that AMPSO, A-AMPSO, and ID-AMPSO all performed well on the knight's coverage problem. However, A-AMPSO was superior in most low-dimensional problems, while MM-AMPSO showed very good performance in high dimensions. This confirms that the gain in performance, due to a little complexity to the AMPSO algorithm, can be significant.

Finally, Table 3.5 shows that AMPSO was superior when considering low-dimensional knight's tour problems, while A-AMPSO and ID-AMPSO performed comparably to the AMPSO algorithm in 108 to 192 dimensions. However, MM-AMPSO is superior to all other algorithms in high dimensions. This demonstrates how an increase in dimensionality of the knight's tour problem affect the performance of the various algorithms. Furthermore, Table 3.5 illustrates that the enhancements made to the AMPSO algorithm by MM-AMPSO, while having a negative effect on performance in low dimensions, allow the algorithm to overcome the increased complexity of the problem in higher dimensions (300 or more dimensions).

While it is clear that the AMPSO variants proposed in this chapter provided statistical improvements in some specific problem cases, the average results obtained by the variants are generally not much better than AMPSO when taken on face value. Fur-

Table 3.4: Statistical Wins and Losses for the Knight's Coverage Problem

Dimensions	AMPSO		A-AMPSO		ID-AMPSO		MM-AMPSO	
	wins	losses	wins	losses	wins	losses	wins	losses
64	1	0	1	0	1	0	0	3
81	0	1	**3**	0	0	1	0	1
100	1	1	**3**	0	1	1	0	3
121	0	1	**3**	0	1	1	0	2
144	1	1	**2**	0	1	0	0	3
225	1	1	0	3	1	1	**3**	0
400	1	0	0	3	1	0	1	0

Table 3.5: Statistical Wins and Losses for the Knight's Tour Problem

Dimensions	AMPSO		A-AMPSO		ID-AMPSO		MM-AMPSO	
	wins	losses	wins	losses	wins	losses	wins	losses
48	**2**	0	1	0	1	1	0	3
75	**3**	0	1	1	1	1	0	3
108	1	0	1	0	1	0	0	3
147	1	0	1	0	1	0	0	3
192	1	0	**2**	0	1	1	0	3
300	0	1	0	1	0	1	**3**	0
432	0	1	0	1	0	1	**3**	0

thermore, the limitations that were identified in Section 3.3 were presented without any empirical evidence that they truly affect the search capabilities of the algorithm. Might a deeper understanding of the AMPSO algorithm shed light on other — perhaps more severe — flaws in the algorithm? This notion is pursued in the next chapter.

3.6 Summary

This chapter presented the angle modulated particle swarm optimization (AMPSO) algorithm. A number of limitations due to the algorithm's bit-generating function were discussed, and three variants of the algorithm were proposed to address these limitations. The three variants that were proposed are: amplitude AMPSO (A-AMPSO), min-max AMPSO (MM-AMPSO), and increased-domain AMPSO (ID-AMPSO). The AMPSO algorithm was then compared to the three variants on a number of binary benchmark problems in various dimensions.

It was observed that, in some problem cases, the limitations imposed on AMPSO affect the performance of the algorithm. Furthermore, it was observed that the additional complexity in the proposed variants was not favourable in low dimensions, but allowed the algorithms to obtain statistically equal or superior performance to AMPSO in higher dimensions.

In addition, it was shown that all three variants allowed for a greater variety of initial solutions to be generated. This ability allowed the variants to solve deceptive problems during initialization, while AMPSO required between 10 to 70 iterations to find optimal solutions.

It was concluded that a deeper understanding of exactly how the AMPSO algorithm works might reveal other flaws in the algorithm. Thus, a better understanding of AMPSO could lead to a new variant or technique that performs better accross a wider range of problems and dimensions than the variants that were introduced in this chapter. The next chapter presents a detailed study of various aspects of the AMPSO algorithm.

Derived Publications

The empirical study presented in this chapter was published in [26].

Chapter 4

Critical Considerations on Angle Modulated Particle Swarm Optimisers

If you try and take a cat apart to see how it works, the first thing you have on your hands is a non-working cat.

Douglas Adams

Chapter 3 identified a number of potential problems with AMPSO. Three variants of AMPSO were introduced to attempt to overcome these problems. The A-AMPSO and MM-AMPSO variants introduced additional complexity in terms of dimensionality to the PSO algorithm, but showed performance improvements in a handful of problem cases. The remaining variant, ID-AMPSO, showed no statistically significant improvement over AMPSO.

Most notably, the approach followed to identify potential flaws in the algorithm has, thus far, not been backed by empirical or theoretical evidence. As a result, there is no clear understanding of when or why the AMPSO algorithm (or, indeed, any of the variants introduced in Section 3.3) might fail to produce good results. Therefore, the aim of this chapter is to study various aspects of the AMPSO algorithm in order to identify and provide evidence for any effects or characteristics that may have a negative influence on the performance of AMPSO. Where applicable, this chapter also presents potential solutions to many of the problems found during the course of the analyses. However, empirical testing of the proposed solutions will only be done in the following chapter.

Section 4.1 investigates the periodicity of the generating function. The relationship between the continuous search space and the binary solution space is discussed in Section 4.2. Section 4.3 studies the convergence behaviour of AMPSO. The problem of finding good generating functions is addressed in Section 4.4, while the frequency distribution of binary solutions in the AMPSO search space is discussed in Section 4.5. Section 4.6 investigates the importance of good binary solution representations. The chapter is summarised in Section 4.7.

4.1 Periodicity of the Generating Function

To begin, consider the periodicity of the generating function. A periodic function is a function whose values repeat at some fixed interval T. Formally, if $f(x)$ is periodic, then there exists a $T \neq 0$, such that

$$f(x+T) = f(x), \ \forall x \in \mathbb{R}. \tag{4.1}$$

The use of a periodic generating function in AMPSO raises the concern that it could cause repetition in the generated bit string. The interval of repetition in the resulting binary solution is not necessarily the same as the period T of the generating function, but rather depends on T, as well as the sampling domain. Nevertheless, as the dimensionality n_b of the binary problem increases, so does the probability of producing solutions that contain repetition, if the generating function is periodic.

Of course, in order to know whether this problem is a valid concern in AMPSO, it is necessary to know if g (Equation (3.1)) is periodic. One might easily make the assumption that g is periodic, because it is a *sin* wave. However, closer inspection of the function reveals that it is, in fact, not periodic, as long as $bc \neq 0$. A proof is provided below. For ease of reference, Equation (3.1) is redefined here.

The AMPSO generating function is given by

$$g(x) = sin[2\pi(x-a)b\ cos(2\pi(x-a)c)] + d. \tag{4.2}$$

In the case where $b = 0$, Equation (4.2) reduces to a constant value:

$$g(x) = sin(0) + d = d. \tag{4.3}$$

When $c = 0$,

$$g(x) = sin[2\pi(x - a)b] + d, \tag{4.4}$$

which clearly is periodic. Now, consider the case where $bc \neq 0$. Without loss of generality, let

$$g(x) = sin(f(x)), \tag{4.5}$$

where

$$f(x) = bx\ cos(cx). \tag{4.6}$$

Now, $sin(x) = 0 \Leftrightarrow x = \pi n,$ with $n \in \mathbb{Z}$. Therefore,

$$g(x) = 0 \Leftrightarrow f(x) = \pi n. \tag{4.7}$$

Consider the function f on any interval,

$$I_k = \begin{cases} \left[\frac{2\pi k}{c} - \frac{\pi}{2c}, \frac{2\pi k}{c}\right] & \text{if } c > 0 \\ \left[\frac{2\pi k}{c}, \frac{2\pi k}{c} - \frac{\pi}{2c}\right] & \text{if } c < 0, \end{cases} \tag{4.8}$$

where $k \in \mathbb{N}^+$. Observe that $cos(x)$ has roots at $x = \pi n - \frac{\pi}{2}$. Therefore,

$$\begin{aligned} f\left(\frac{2\pi k}{c} - \frac{\pi}{2c}\right) &= b\left(\frac{2\pi k}{c} - \frac{\pi}{2c}\right)\ cos\left(2\pi k - \frac{\pi}{2}\right) \\ &= 0. \end{aligned} \tag{4.9}$$

Furthermore, observe that $cos(2\pi n) = 1$. Therefore,

$$\begin{aligned} f\left(\frac{2\pi k}{c}\right) &= \frac{2\pi bk}{c} \cdot\ cos(2\pi k) \\ &= \frac{2\pi bk}{c}. \end{aligned} \tag{4.10}$$

Figure 4.1 illustrates the relevant intervals of f for $k = 1, \ldots, 6$, $b = 1$, and $c = 1$. From the observations above, it is evident that the range of f on I_k is $\left[0, \frac{2\pi bk}{c}\right]$ if $bc > 0$, or $\left[\frac{2\pi bk}{c}, 0\right]$ if $bc < 0$. In either case, the length of this range is $\left|\frac{2\pi bk}{c} - 0\right|$). Now, because f is continuous, there exists a value $x^* \in I_k$ such that $f(x^*) = \pi n$ for every πn in the range of f on I_k. Furthermore, f is monotonic on any interval I_k, because $cos(x)$ is monotonic

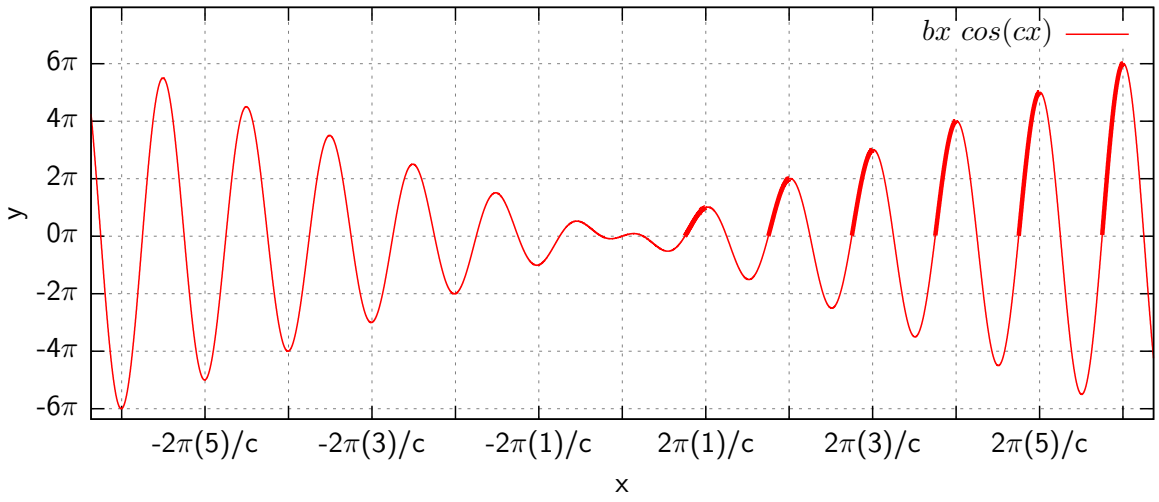


Figure 4.1: The range of f on I_k is $\left[0, \frac{2\pi bk}{c}\right]$ when $bc > 0$. Every interval I_k is highlighted in bold for $k = 1, \ldots, 6$, $b = 1$, and $c = 2$. Note that the range of f on every highlighted interval contains $\left\lfloor \left| \frac{2bk}{c} \right| \right\rfloor + 1$ multiples of π, including 0π.

on any interval $\left[2\pi k - \frac{\pi}{2}, 2\pi k\right]$. Therefore, the number of multiples of π contained in the range of f on I_k is given by

$$\begin{aligned} \Pi(k) &= \left\lfloor \frac{\left| \frac{2\pi bk}{c} - 0 \right|}{\pi} \right\rfloor + 1 \\ &= \left\lfloor \left| \frac{2bk}{c} \right| \right\rfloor + 1. \end{aligned} \tag{4.11}$$

Note that $\Pi(k)$ is equal to the number of roots of g in the interval I_k. Now,

$$\lim_{k \to \infty} \Pi(k) = \infty. \tag{4.12}$$

The same argument can be used for the sequence of intervals (J_j), where

$$\bigcup_{j \in \mathbb{N}} J_j = (\mathbb{R}^+ \cup \{0\}) - \bigcup_{k \in \mathbb{N}} I_k, \tag{4.13}$$

Then, by simple inductive argument, it is clear that $\Pi(k)$ also diverges over larger, consecutive intervals. Therefore, the roots of g become more frequent as $x \to \infty$. The same is true for $x \to -\infty$, because g is a *sin* wave, which is symmetric.

Because the number of roots of g increases indefinitely over consecutive intervals as $|x| \to \infty$, there can be no sequence of intervals, (K_k), such that the length of any interval K_k is equal to the period T, and the number of roots of g in every interval K_k is the same, and

$$\bigcup_{k \in \mathbb{N}} K_k = \mathbb{R}. \tag{4.14}$$

That is,

$$g(x + T) \neq g(x) \; \forall x \in \mathbb{R}. \tag{4.15}$$

Therefore, g is aperiodic if $bc \neq 0$.

4.2 Spatial Disconnect

While the problem with periodic generating functions has now been ruled out for the specific case where g is used as the generating function, it is clear from Section 4.1 that the roots of g increase in frequency as $|x| \to \infty$. This effect is illustrated in Figure 4.2. The rate at which the frequency of roots increases is dependent on the c coefficient of g. The perpetual increase in the frequency of roots of g may seem harmless, but it has severe consequences for the AMPSO algorithm, as explained next.

The AMPSO algorithm is concerned with two spaces: the continuous *coefficient space*, where the PSO algorithm is defined, and the binary *solution space*, wherein the solution to the arbitrary binary problem exists. Recall from Section 2.2.2 that the PSO algorithm works in two primary phases, known as the exploration and exploitation phases. During the exploration phase, particle step sizes are relatively large so that many parts of the search space are sampled sparsely in order to determine which regions in the search space are likely to contain good solutions. During the exploitation phase, particle step sizes become smaller so that the good regions that were found during the exploration phase can be thoroughly searched in an attempt to find the best solution. When solving continuous problems (which is what PSO was designed for), the magnitude of the velocity of a particle is the distance between its current and previous position. This distance is generally also an indication of how similar two solutions are. The assumption is that if some arbitrary solution in the continuous search space is good, then other close-by

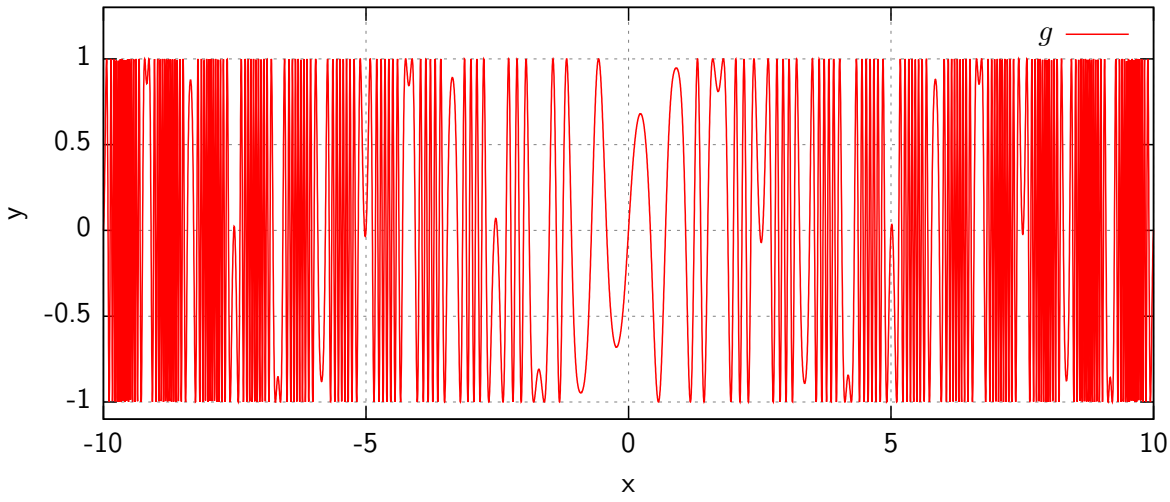


Figure 4.2: The roots of g increase in frequency as $|x| \to \infty$.

solutions are probably also good. However, when particle positions are mapped to binary solutions using angle modulation, it may happen that two solutions which are close to each other in the coefficient space are actually far apart in the solution space. When this happens, the ability of PSO to exploit the solution space is hindered. This problem is henceforth referred to as *spatial disconnect*.

Note that the assumption that good solutions are grouped together is an assumption made by PSO, but not necessarily by all optimisation algorithms. That is, other optimisation algorithms may not be influenced by the spatial disconnect problem. One example of such an algorithm is simulated annealing (SA) [25], where new solutions are not necessarily generated close to the current best solution. The use of SA and other similar algorithms with angle modulation is beyond the scope of this dissertation, but might be worthwhile considering as alternatives to PSO.

One way of measuring the similarity between two binary solutions is to calculate the *Hamming distance* between them. The Hamming distance between two binary strings of equal length is the number of corresponding bits in the two binary strings that differ in value. Figure 4.3 illustrates the concept of Hamming distance.

The problem of spatial disconnect can be demonstrated by the following process:

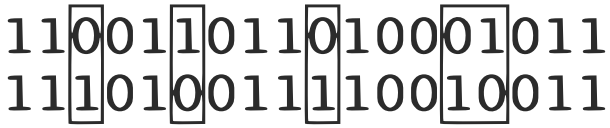


Figure 4.3: There are five corresponding bits with different binary values in these two binary strings, so the Hamming distance between them is 5.

1. Determine a step size s.
2. Uniformly initialise a random vector $\mathbf{p}_1$ in the range $[-1, 1]^4$.
3. Substitute the coefficients of g with the elements of $\mathbf{p}_1$.
4. Generate an n_b-dimensional bit string, using g. Call the bit string b_1.
5. Create a uniform random vector $\mathbf{s}$ with 4 dimensions in the range $[-1, 1]^4$, and length s.[1]
6. Let $\mathbf{p}_2 = \mathbf{p}_1 + \mathbf{s}$.
7. Substitute the coefficients of g with the elements of $\mathbf{p}_2$.
8. Generate an n_b-dimensional bit string, using g. Call the bit string b_2.
9. Divide each bit string (b_1 and b_2) into equally-sized groups of bits.
10. Determine the Hamming distance between each of the corresponding groups of bits in b_1 and b_2.
11. Repeat steps 2 to 10 n times, and calculate the average Hamming distance between the corresponding groups of bits in b_1 and b_2.

The process outlined above simulates the movement of a particle from a random point $\mathbf{p}_1$ in the coefficient space to another point $\mathbf{p}_2$, with velocity $\mathbf{s}$. Note that the length s of $\mathbf{s}$ is the amount of change in the coefficient space. By repeating the entire

[1]The length s in this case refers to the square-root of the sum of all the vector elements, as opposed to the cardinality of $\mathbf{s}$, which is 4.

process for decreasing values of s, the average amount of change in the binary solution can be measured and compared to s. By dividing the binary strings b_1 and b_2 into blocks (step 9 above), the amount of change in lower- and higher dimensions of the binary solution can also be compared. For PSO to be able to successfully exploit the binary solution space, there must be a direct correlation between the amount of change in the coefficient space and the amount of change in the solution space. Furthermore, because $\mathbf{s}$ is chosen uniformly, and s is constant while repeating steps 2 to 10, the amount of change measured should, on average, be uniform across all dimensions of the solution space.

Figures 4.4 to 4.8 show the average Hamming distances across all dimensions of the solution space for various values of s. In this case, $n_b = 100$, and the resulting bit strings were divided into twenty-five 4-bit groups for the purpose of measuring Hamming differences. All averages were measured by repeating steps 2 to 10 thirty times. Figure 4.4 shows that, when $s = 0.1$, the average Hamming distance is more or less uniform across all dimensions of the solution space. However, in Figures 4.5 to 4.8, it is observed that as the value of s is decreased, the distribution of change in the solution space becomes increasingly non-uniform. In particular, while there is a correlation between the amount of change in the coefficient space and the amount of change in the solution space, the correlation is stronger for lower dimensions of the solution space. Practically, this implies that the lower-dimensional bits will stabilise at higher particle velocities than the higher-dimensional bits in the binary solution. Hence, a spatial disconnect exists.

The spatial disconnect is caused by a combination of the increasing frequency of roots of g (see Section 4.2) and the manner in which g is sampled to generate binary solutions. The original AMPSO algorithm samples g on integer intervals $x = 0, 1, 2, \ldots, n_b - 1$. This relatively fast increase in the value of x means that the frequency of roots of g at the sample positions also increases quickly (albeit, depending on the value of c), as can be seen in Figure 4.2. The higher the frequency at a given sampling position x, the higher the probability that the sign at x will change when the coefficients of g change by a small amount. Recall from Section 3.1 that a change in the sign at x is equivalent to a bit flip in the binary solution. It is important to realise that the results reported here are independent of any specific binary problem, and that the spatial disconnect is

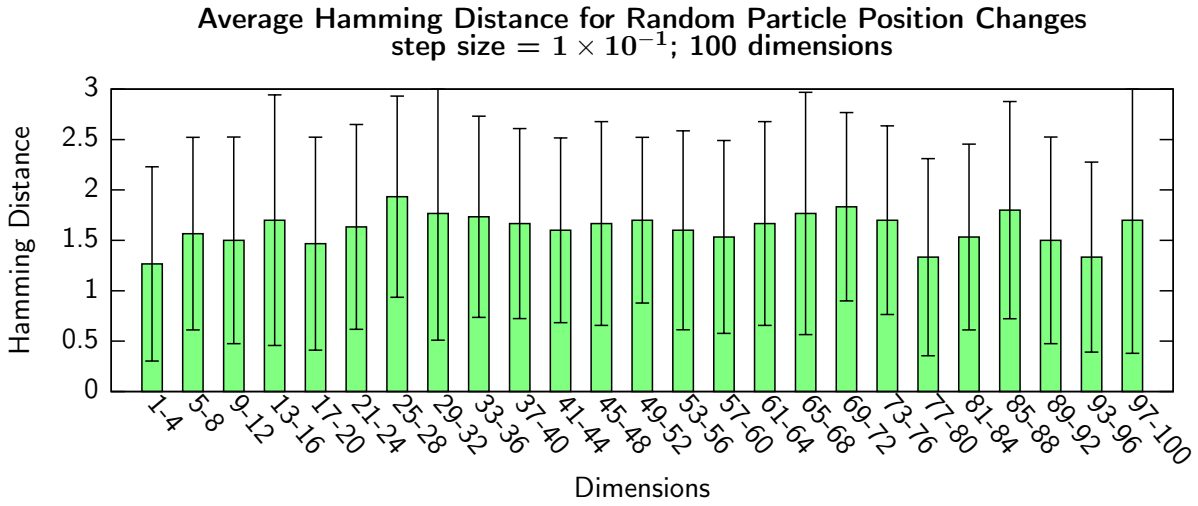


Figure 4.4: For particle step sizes of $s = 0.1$, the amount of change is uniform across all dimensions of the binary solution.

caused by the generating function. Note the absence of a binary problem in steps 1 to 11 above.

The problem of spatial disconnect can be partially overcome by MM-AMPSO, which has the ability to decrease the sampling domain, and thereby the sampling interval, so that x does not increase as fast. This provides an explanation for MM-AMPSO's good performance in some high-dimensional problem cases that were studied in Section 3.3. However, this problem persists in both the ID-AMPSO and A-AMPSO variants.

Another question that arises from the results presented in this section is: do particles in AMPSO eventually slow down sufficiently to exploit the higher dimensions of the solution space? Note that even if particles do slow down sufficiently, the non-uniform distribution of change is still detrimental to performance if the optimal values of lower dimensional bits are in any way dependent on the values of higher dimensional bits. Nonetheless, the question remains interesting. Section 4.3 analyses the velocities of particles in AMPSO throughout the duration of the search process.

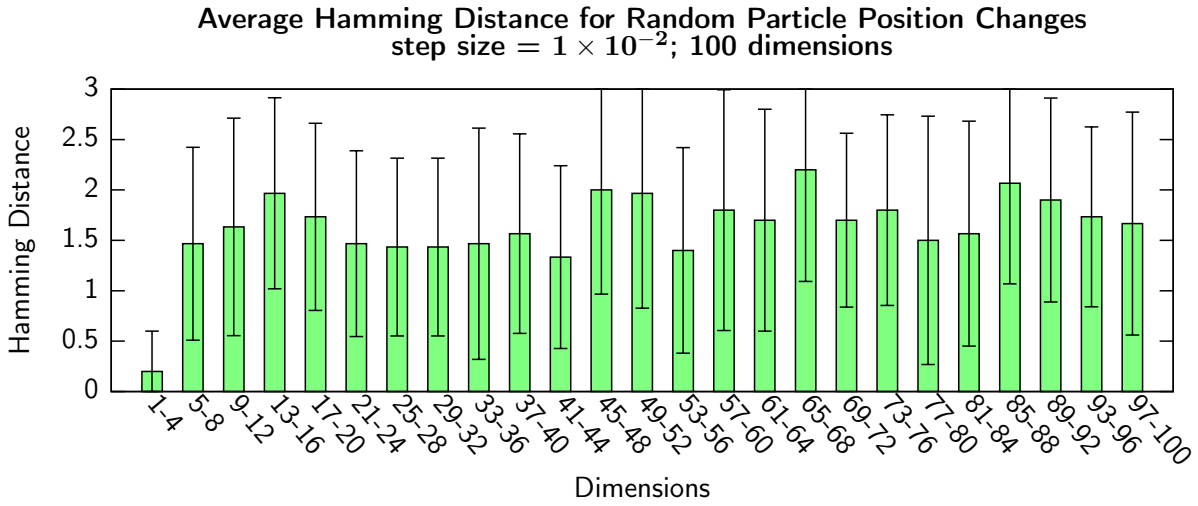


Figure 4.5: For particle step sizes of $s = 0.01$, the amount of change is low in the first four dimensions of the binary solution.

4.3 Velocities in Angle Modulated Particle Swarms

This section presents the first empirical investigation into the behaviour of angle modulated particle swarms, with respect to particle velocities.

The work presented in this section is based on the theoretical aspects of PSO that were discussed in Section 2.2.3. The experimental setup is outlined in Section 4.3.1, while the results are presented in Section 4.3.2. Section 4.3.3 discusses the implications of the results in terms of the spatial disconnect that was discovered in Section 4.2.

4.3.1 Experimental Setup

In order to analyse particle velocities in angle modulated particle swarms, experiments were constructed using AMPSO, A-AMPSO, and MM-AMPSO. ID-AMPSO was not considered further in this study, because there is no statistical difference in performance between ID-AMPSO and AMPSO (see Section 3.3.3).

All algorithms were executed on the same set of problems used in Section 3.4, with the exception of the deceptive problems, which were showed to be easily solvable by AMPSO and all three variants. The following problems were evaluated:

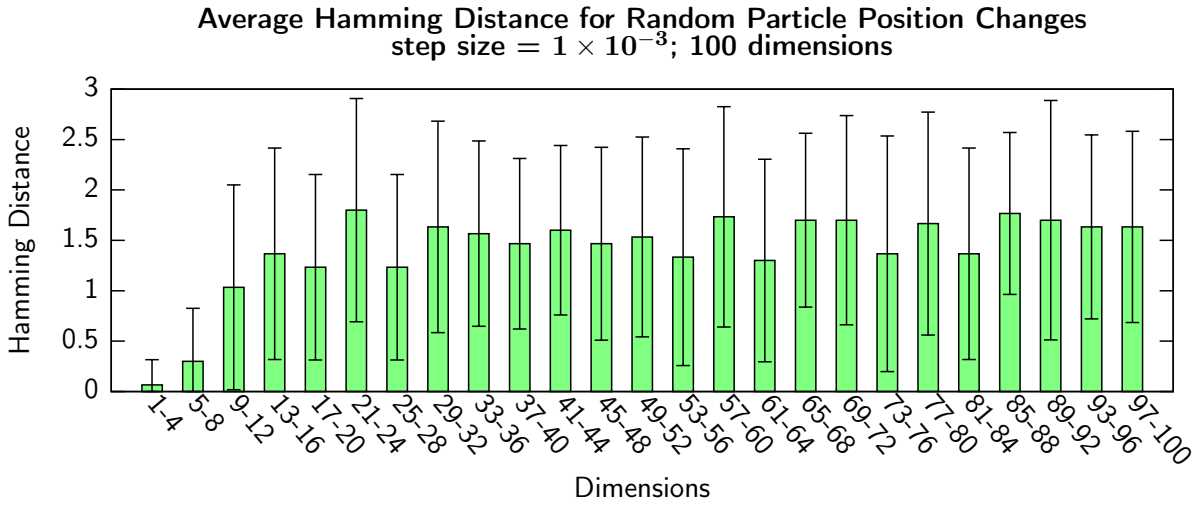


Figure 4.6: For particle step sizes of $s = 0.001$, the amount of change is low in the first four dimensions of the binary solution, slightly higher in dimensions four to eight, and high in the remaining dimensions.

- N-Queens (64-, 100-, 400-, and 625 dimensions),
- Knight's tour (48-, 108-, 300-, and 432 dimensions), and
- Knight's coverage (64-, 100-, 400-, and 625 dimensions).

In every case, the parameter values in Table 2.1 were used. These parameters are used because they have been shown to produce convergent behaviour, which is what this experiment aims to investigate. In each case, the algorithm was allowed to execute for 1000 iterations. The swarm consisted of 20 particles, and the following measurements were recorded at every iteration:

- expected average magnitude of particle velocities (Equation (2.6)),
- actual average magnitude of particle velocities, and
- whether the global best fitness had changed since the previous iteration.

Because the purpose of these experiments is to gain insight into the behaviour of angle modulated swarms by analysing particle velocities during the search process, it does not

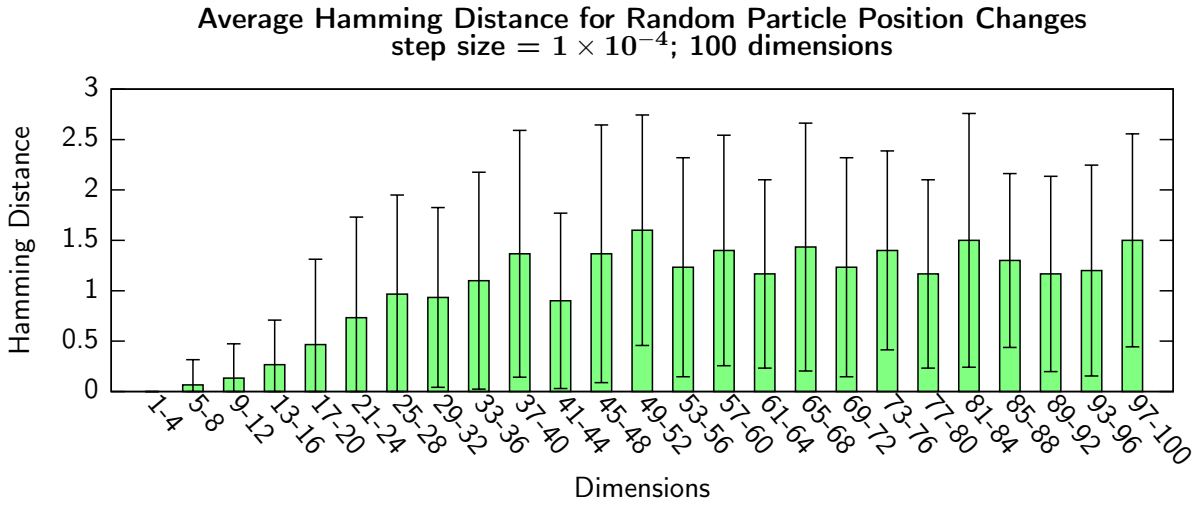


Figure 4.7: For particle step sizes of $s = 1 \times 10^{-4}$, there is no change in the first four dimensions of the binary solution. The amount of change increases gradually from dimensions five to 40, and remains high in the remaining dimensions.

make sense to average the measurements over a number of independent runs. Thus, although the experiments were performed for 30 independent runs, each of the graphs that are reported in Section 4.3.2 shows the behaviour of a single execution. However, unless stated otherwise, the results are representative of the kind of behaviour that was observed across independent executions of the experiments.

4.3.2 Results

Graphs 4.1 to 4.3 show the three most typical cases of the behaviour of angle modulated swarms with respect to average particle velocities. With a few exceptions, these three cases were observed throughout all experiments, across all problems and dimensions. The graphs show the actual- and expected average particle velocities in the swarm at each iteration. The intermittent vertical lines indicate when changes in the global best fitness occurred.

In Graph 4.1, particle velocities initially decrease. Changes in the global best fitness are also initially frequent, as expected. However, the average velocity quickly stagnates (after about 100 iterations) at a value between 0.1 and 1. The stagnation is evident

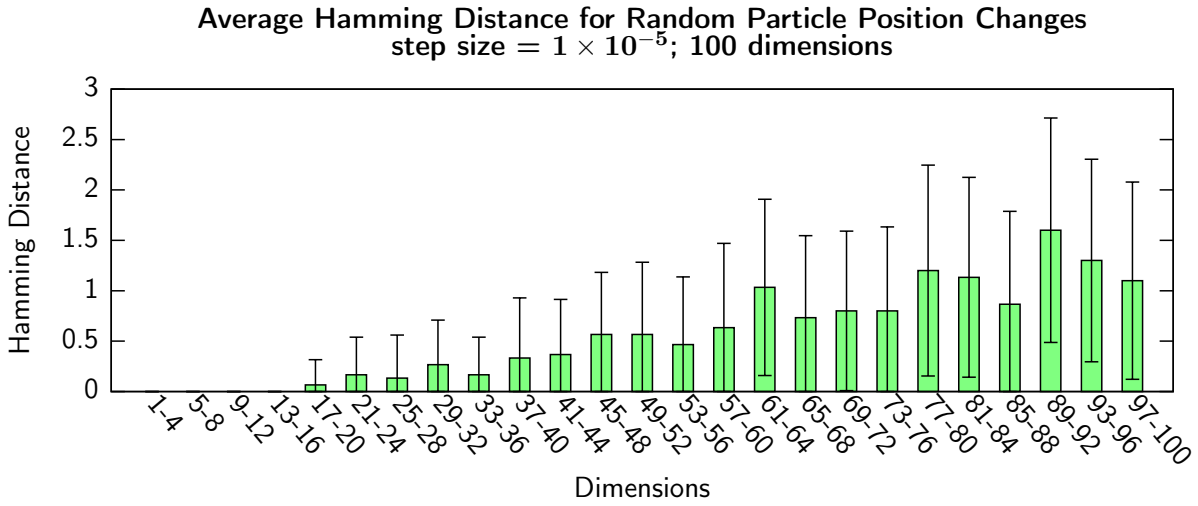
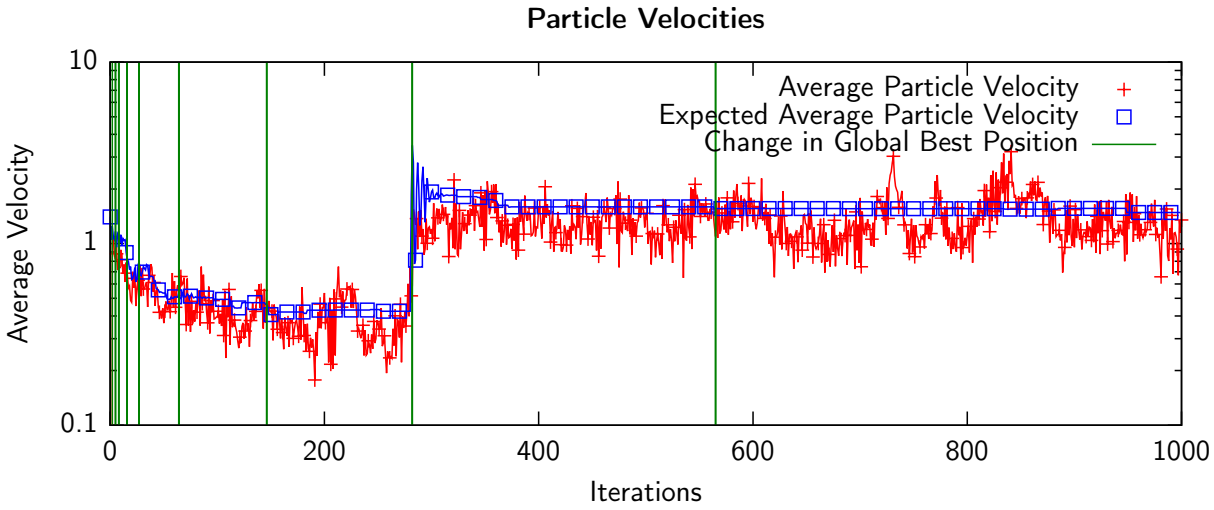


Figure 4.8: For particle step sizes of $s = 1 \times 10^{-5}$, there is no change in the first 16 dimensions of the binary solution. The amount of change increases gradually from dimensions 17 to 100.

from the fact that the expected average velocity $E[v_{avg}]$ converges to a constant value, indicating that the swarm is order-2 stable at this point. Minor fluctuations in $E[v_{avg}]$ are still caused by changes in the particles' personal best positions, but it is not until a change in the global best fitness occurs (around iteration 280 in this case) that the swarm destabilises momentarily. Unfortunately, the swarm stabilises at a higher velocity after this event occurs. From Equations (2.7) and (2.6), it is evident that the global best position has, on average, shifted further away from the particles' personal best positions, causing an increase in average particle velocity.

Another common case is shown in Graph 4.2, where particle velocities initially decrease until the swarm stabilises, as in the previous case. However, in this case, subsequent changes in the global best position do not destabilise the swarm. The fact that the swarm remains stable sometimes, even though the global best fitness has changed, indicates that the global best position in those cases did not change by much. That is, in those cases, the new global best position is relatively close to the old global best position in the coefficient space.

Another, less common, scenario is shown in Graph 4.3. In this case, the initial decrease in average velocity is especially short-lived, while a number of consecutive desta-



Graph 4.1: Particle velocities initially decrease, but stabilise after about 100 iterations. After around 280 iterations, a change in the global best position destabilises the swarm. The swarm then becomes stable at a higher average velocity.

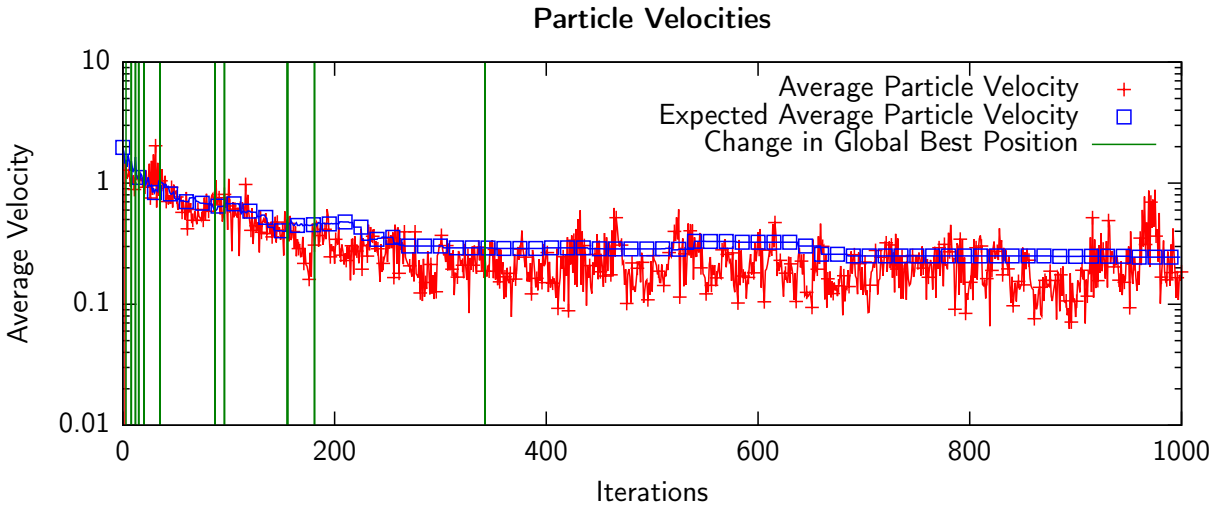
bilising changes in the global best position cause the average particle velocity to grow progressively. This indicates that the global best position gradually shifts further away from the particles' personal best positions as the search continues. In these cases, the average velocity of the swarm generally eventually stabilises at a value between 1 and 10.

Finally, Graph 4.4 depicts an uncommon scenario that was only observed for MM-AMPSO in a handful of cases. This graph shows that the average velocities of particles can grow to values as high as 100 if destabilising changes in the global best position keep occurring.

In a few cases the average velocities of particles were observed to drop below 0.1. However, there were no cases were the average particle velocities ever dropped below 0.01. Henceforth, the phenomenon where particle swarms become order-2 stable at high average velocities will be referred to as *inadequate convergence*.

4.3.3 Discussion

The results shown in Section 4.3.2 indicate that particle velocities in AMPSO tend to stabilise at relatively high values. In the most common scenarios, particle velocities

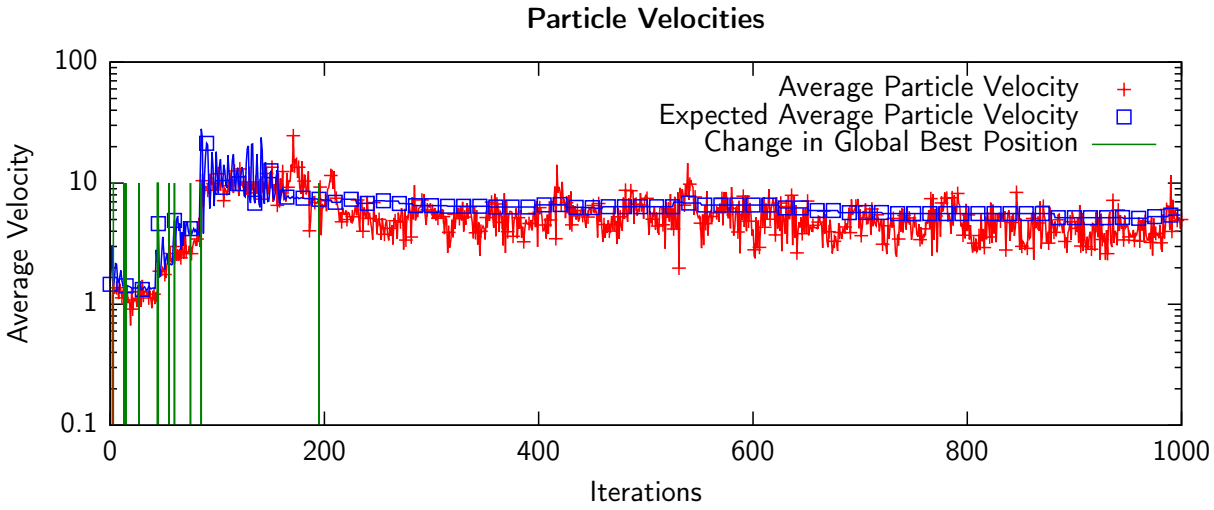


Graph 4.2: Particle velocities initially decrease, but the swarm stabilises after a few hundred iterations, with an average particle velocity of between 0.1 and 1.

stabilised at values between 0.1 and 1.

When these results are compared to those reported in Section 4.2, it becomes clear that neither AMPSO, nor any of the variants ever progress to the exploitation phase of the search process. Recall from Figure 4.4 that a step size as small as 0.1 in the coefficient space causes, on average, 1.5 out of four bits to flip across all dimensions of the binary solution.

While spatial disconnect (see Section 4.2) highlights the severity of the problem, it also offers an explanation for inadequate convergence in the case of angle modulated swarms. When a particle i in PSO is attracted towards $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}_i$, the assumption is that good solutions are likely to be found in the vicinity of other previously-found good solutions. In moving towards those previously-found locations, the particle progressively discovers better solutions, and $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}_i$ move closer together. The effect is that the particle progressively stabilises at lower velocities, until $|\boldsymbol{y}_i - \hat{\boldsymbol{y}}|$ becomes sufficiently small, so that σ_i also becomes very small, relative to the search domain. However, in the case of angle modulated swarms, the generating function may cause spatial disconnect, nullifying the assumption that good solutions are grouped together. In this case, while a particle is still attracted towards previously-found good solutions, it is unlikely to find better solutions in those regions, so the distance between $\hat{\boldsymbol{y}}$ and $\boldsymbol{y}_i$ does not decrease



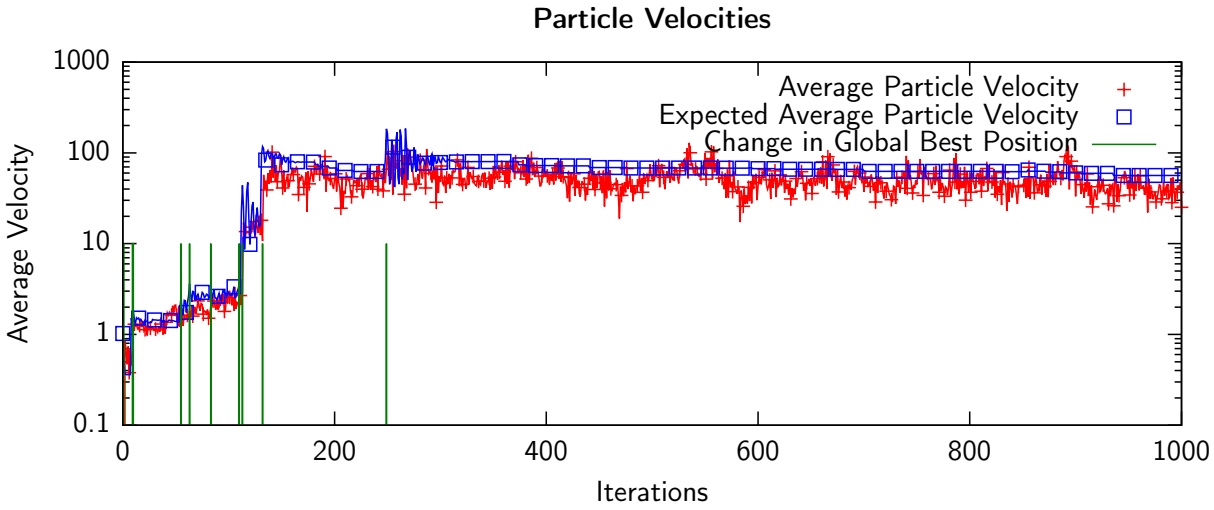
Graph 4.3: Particle velocities do not get a chance to decrease, because a number of consecutive changes in the global best position causes the two attractors to become more separated over time.

enough, relative to size of the coefficient space. As a result, the particle becomes order-2 stable at a high velocity. Essentially, a particle in this state indefinitely explores the coefficient space in a region of size σ_i around $\boldsymbol{\mu}$. If, by chance, any particle in the swarm does find a better global best solution, the other particles may become destabilised if the global best position moves by a large enough distance. However, the destabilised particles will yet again stabilise as they fail to find better solutions near the new point of attraction, *ad infinitum*.

A potential solution to the problems that have been discovered so far is to replace the generating function. The problem of finding a new generating function is addressed in Section 4.4.

4.4 Generating Function Potential

The generating function presented by [34] was chosen seemingly arbitrarily and without justification. Indeed, the only justification to be found in literature is from Franken, when he postulated the possibility of a "... mathematical function that will allow for more intricate arrangements of 0's and 1's, and adapting this function through the use



Graph 4.4: Particle velocities escalate to extremely high values as the global best position moves very far from the particles' personal best positions, on average.

of PSO" [18].

While all the problems that were discussed in Sections 4.1, 4.2, and 4.3 have to be considered when choosing a new generating function, another very important question has eluded researchers thus far: can one produce every possible bit string of length n_b by varying the coefficients of the generating function? Or, stated differently, does the generating function have the *potential* to produce any arbitrary bit string of length n_b? This question is of paramount importance, because if the generating function cannot produce any arbitrary bit string, AMPSO might, in some cases, never be able to produce optimal solutions.

A formal definition of generating function potential is given is Section 4.4.1. Section 4.4.2 demonstrates the use of generating function potential by constructing a novel generating function that can solve simple binary problems. Section 4.4.3 discusses the implication of generating function potential for lower-than-n_b-dimensional problems once a suitable generating function has been found. Section 4.4.4 addresses the problem of finding a generating function that has the potential to solve arbitrarily high-dimensional binary problems. Section 4.4.5 discusses the applicability of generating function potential to AMPSO variants.

4.4.1 Definition of Generating Function Potential

The following definition assumes that sample values in AMPSO are mapped to binary digits using Equation (3.2).

Given an arbitrary generating function Υ, the potential of Υ to generate any arbitrary bit string of length n_b by varying the coefficients of Υ and sampling the function at regular, fixed intervals is given by

$$P_{\Upsilon}^{n_b} = \frac{|\mathbb{B}_{\Upsilon}^{n_b}|}{2^{n_b}}, \tag{4.16}$$

where $\mathbb{B}_{\Upsilon}^{n_b}$ is the set of binary strings of length n_b that Υ can generate. Equation (4.16) will henceforth simply be referred to as the n_b-potential of Υ. An n_b-potential of 1 indicates that it is possible to generate every possible bit string of length n_b by varying the coefficients of Υ. Lower values of $P_{\Upsilon}^{n_b}$ indicate that there are bit strings of length n_b that can never be generated by the function. Note that the n_b-potential of Υ says nothing about the likelihood of generating any specific bit string of length n_b.

To demonstrate the usefulness of generating function potential, an example is given below.

4.4.2 Example of Finding an Appropriate Generating Function

One of the motivations for introducing angle modulation, was the fact that the method reduces the dimensionality of the problem [34]. However, for very low-dimensional binary problems (three dimensions or lower), the dimensionality of the problem actually increases when angle modulation is applied (assuming that g is used as the generating function), because g has four coefficients. This example will show that it is possible to replace the generating function with one that has fewer coefficients to prevent the increase in dimensionality for simple binary problems.

Consider a binary problem with a 3-bit solution, and a simple generating function:

$$h_1(x) = sin(x). \tag{4.17}$$

This generating function has no coefficients and will therefore always produce the same bit string, assuming that h_1 is sampled at $x = 0, 1, 2$. Table 4.1 lists all the possible

solutions to a 3-dimensional binary problem. Column 2 indicates that h_1 always produces the bit string '011':

$$\begin{aligned} &\text{bit 1: } sin(0) = 0.0 \rightarrow\ 0 \\ &\text{bit 2: } sin(1) \approx 0.8 \rightarrow\ 1 \\ &\text{bit 3: } sin(2) \approx 0.9 \rightarrow\ 1, \end{aligned} \tag{4.18}$$

which means that $|\mathbb{B}^3_{h_1}| = |\{\texttt{'011'}\}| = 1$. Therefore, the 3-potential of $h_1(x)$ is

$$\begin{aligned} P^3_{h_1} &= \frac{|\mathbb{B}^3_{h_1}|}{2^3} \\ &= \frac{1}{8} \\ &= 0.125. \end{aligned} \tag{4.19}$$

The low 3-potential of h_1 indicates that this function would be a very poor choice of generating function to use with angle modulation. Of course, h_1 is clearly a toy example, because there are no coefficients to optimise.

Consider a slightly more complex function:

$$h_2(x) = sin(vx). \tag{4.20}$$

Adding the v coefficient means that the function can now produce different bit strings by varying v. In the case of AMPSO, it also implies 1-dimensional particles of the form $\boldsymbol{x}_i = (v)$.

Table 4.1 shows that there exist four values for v that produce the first four possible solutions, respectively. These values were found empirically, assuming the same sampling positions as for h_1. However, no values for v could be found to produce the remaining four solutions. The fact that no values for v could be found to produce the last four solutions does not necessarily imply that it is impossible to produce those solutions. However, in this case it is easy to show that h_2 cannot generate solutions whose first bit is '1'. Indeed, consider the calculation of the first bit:

$$h_2(0) = sin(v(0)) = 0.0 \rightarrow 0, \tag{4.21}$$

regardless of the value of v. Therefore, $|\mathbb{B}^3_{h_2}| = 4$, and $P^3_{h_2} = 0.5$. The 3-potential of h_2 is a significant improvement over the 3-potential of h_1, but still not good enough.

Table 4.1: Different binary solutions are obtained by varying the value of v. Assuming that $x \in \{0, 1, 2\}$, a '-' indicates that it is not possible to generate a particular solution. A '*' indicates that a particular solution is always generated.

Solution	$h_1(x) = sin(x)$	$h_2(x) = sin(vx)$	$h_3(x) = sin(v(x+1))$
		v	v
000	–	5	6
001	–	4	5
010	–	2	4
011	*	1	17
100	–	–	2
101	–	–	3
110	–	–	14
111	–	–	1

The problem with h_2 can easily be corrected by adding a constant horizontal shift term to prevent the first bit from always being sampled at $sin(0)$:

$$h_3(x) = sin(v(x + 1)). \tag{4.22}$$

Table 4.1 shows that, for h_3, there exist values for v to generate every possible 3-bit binary solution. Hence, $P^3_{h_3} = 1$, meaning that h_3 is a generating function, with a single coefficient, that has the potential to solve any 3-dimensional binary problem. Table 4.1 also shows that all the possible solutions exist for $1 \leqslant v \leqslant 17$. Therefore, a suitable initialisation range for particles in the PSO algorithm has also been found (although a smaller range that is equally suitable might exist).

A point of concern regarding h_3 is that the function is clearly periodic and may therefore give rise to repetition in the binary solution. However, as indicated in Section 4.1, periodic generating functions are of greater concern for higher values of n_b. In the case where $n_b = 3$, a periodic generating function is not problematic.

4.4.3 Implication for Lower-than-n_b-Dimensional Binary Problems

Another useful observation to make from Table 4.1 is that the solutions to every 2-bit binary problem are contained in the 3-bit solutions. That is, if one ignores the final bit of every 3-bit solution in Table 4.1, then every 2-bit binary string is present in the list. The implication is that if a generating function is able to generate all 3-bit binary strings, then it is also able to generate all 2-bit binary strings, simply by not sampling the final bit. By the same reasoning, it then follows that all 1-bit solutions can also be generated. This implication necessarily holds for any value of n_b and can be formally stated as follows:

$$P_\Upsilon^{n_b} = 1 \quad \Rightarrow \quad P_\Upsilon^{n_b-1} = P_\Upsilon^{n_b-2} = \ldots = P_\Upsilon^1 = 1. \tag{4.23}$$

Therefore, if $P_\Upsilon^{n_b} = 1$, then Υ is a generating function with the potential to solve any binary problem with dimensionality n_b or lower.

4.4.4 Generating Functions to Solve High-Dimensional Binary Problems

It has now been shown that generating function potential can be used to quantify the usefulness of any arbitrary generating function. However, up to this point, only very simple functions were considered. This section investigates the use of generating function potential to find appropriate generating functions to solve high-dimensional binary problems.

4.4.4.1 Estimating Generating Function Potential

The most challenging aspect of determining the n_b-potential of a generating function Υ is finding the set $\mathbb{B}_\Upsilon^{n_b}$ of bit strings that Υ can generate. One conceivable way of generating $\mathbb{B}_\Upsilon^{n_b}$ is to sample Υ at $x = 0, 1, 2, \ldots, n_b - 1$ for every possible permutation of the coefficients of Υ and recording which bit strings are generated. Of course, this is impossible, because the coefficients of Υ are continuous. However, the values of the coefficients can be limited to some finite subset $\mathbb{C} \subseteq \mathbb{R}$. Let C_Υ denote the number of

coefficients of Υ. By limiting the values that the coefficients may assume, a set $\mathbb{P}_{C_\Upsilon}$ of all allowed permutations of the coefficients of Υ can be generated. The number of permutations $|\mathbb{P}_{C_\Upsilon}|$ is given by

$$|\mathbb{P}_{C_\Upsilon}| = |\mathbb{C}|^{C_\Upsilon}. \tag{4.24}$$

For example, if Υ has two coefficients whose values are limited to $0.0, 0.1$, and 0.2, then the permutations listed in Table 4.2 are possible, and

$$\begin{aligned} |\mathbb{P}_{C_\Upsilon}| &= |\mathbb{C}|^{C_\Upsilon} \\ &= 3^2 \\ &= 9. \end{aligned} \tag{4.25}$$

The complexity of generating a bit string of length n_b from Υ, using every possible permutation in $\mathbb{P}_{C_\Upsilon}$, is

$$\mathcal{O}(n_b \cdot |\mathbb{P}_{C_\Upsilon}|) = \mathcal{O}(n_b \cdot |\mathbb{C}|^{C_\Upsilon}), \tag{4.26}$$

which increases exponentially with the number of coefficients C_Υ.

If the method described above is used to calculate the n_b-potential of an arbitrary generating function, the following should be noted:

1. Because $\mathbb{C}$ is a finite subset of $\mathbb{R}$, the calculated n_b-potential is only an estimate of the function's true n_b-potential.

2. The true n_b-potential of the function is necessarily greater than or equal to the estimated value. Therefore, if the estimated n_b-potential of Υ equals 1, then $P_\Upsilon^{n_b} = 1$.

3. To obtain the best estimate of the true n_b-potential of the function, $|\mathbb{C}|$ should be as large as possible.

4. Because any given permutation will always generate the same binary solution, $\mathbb{C}$ must be chosen such that

$$|\mathbb{P}_{C_\Upsilon}| \geqslant 2^{n_b}. \tag{4.27}$$

Without this constraint, an estimated n_b-potential of 1 can never be obtained.

The n_b-potential of g (Equation (3.1)) was estimated using the approach outlined above. In this case, $\mathbb{C} = \{-1.000, -0.975, -0.950, \ldots, 0.00, 0.025, 0.050, \ldots, 1.000\}$, and

$n_b = 16$. The range $[-1, 1]$ was chosen, because that is the range within which AMPSO is generally initialised [26, 30, 34]. From Equation (4.25),

$$\begin{aligned} |\mathbb{P}_{C_g}| &= |\mathbb{C}|^{C_g} \\ &= 81^4 \\ &= 43\ 046\ 721, \end{aligned} \tag{4.28}$$

which satisfies the constraint in Equation (4.27):

$$\begin{aligned} |\mathbb{P}_{C_g}| &\geqslant 2^{n_b} \\ 43\ 046\ 721 &\geqslant 2^{16} \\ &\geqslant 65\ 536. \end{aligned} \tag{4.29}$$

Generating a bit string from g for each of the 81^4 permutations of the coefficients of g, yielded an estimated 16-potential of 1. That is, every possible bit string of length 16 could be generated. Therefore,

$$P_g^{16} = 1. \tag{4.30}$$

Increasing n_b to 17 still satisfied the constraint in Equation (4.27), but yielded an estimate of $P_g^{17} \geqslant 0.99932$. In order to prove that $P_g^{17} = 1$, $|\mathbb{C}|$ needs to be further increased. Unfortunately, this approach quickly becomes infeasible, because of the increasing computational complexity to estimate $|\mathbb{B}_\Upsilon^{n_b}|$.

4.4.4.2 Ensemble Generating Functions

An alternative method to solve arbitrarily high-dimensional binary problems is to make use of multiple generating functions to solve parts of the binary solution independently. A separate generating function would then be used for each $\frac{n_b}{\varphi}$-bit group of bits in the binary solution, where φ is the total number of generating functions. Together, the φ generating functions are referred to as an *ensemble generating function*, Θ. Assuming that the φ generating functions are all the same function Υ, and that $P_\Upsilon^{\frac{n_b}{\varphi}} = 1$, it follows that $P_\Theta^{n_b} = 1$. The PSO that results from using Θ as the generating function has dimensionality

$$\begin{aligned} n_x &= \frac{n_b}{\frac{n_b}{\varphi}} \cdot C_\Upsilon \\ &= \varphi C_\Upsilon. \end{aligned} \tag{4.31}$$

Table 4.2: Each line in this table represents one possible permutation of the coefficients of Υ. Each permutation can potentially generate a different binary solution.

Coefficent 1	Coefficient 2
0.0	0.0
0.0	0.1
0.0	0.2
0.1	0.0
0.1	0.1
0.1	0.2
0.2	0.0
0.2	0.1
0.2	0.2

The position of a particle i then has the form

$$\begin{aligned}\boldsymbol{x}_i = (&\psi_{\Upsilon_1 1}, \psi_{\Upsilon_1 2}, \ldots, \psi_{\Upsilon_1 C_\Upsilon}, \psi_{\Upsilon_2 1}, \psi_{\Upsilon_2 2}, \ldots, \psi_{\Upsilon_2 C_\Upsilon}, \\ &\ldots, \psi_{\Upsilon_\varphi 1}, \psi_{\Upsilon_\varphi 2}, \ldots, \psi_{\Upsilon_\varphi C_\Upsilon}),\end{aligned} \tag{4.32}$$

where $\psi_{\Upsilon_i j}$ is the j^{th} coefficient of the i^{th} Υ function.

In the specific case of h_3, this implies that an n_b-dimensional binary problem can be solved using an $\frac{n_b}{3}$-dimensional PSO, with particle positions taking the following form: $\boldsymbol{x}_i = (v_1, v_2, v_3, \ldots, v_\varphi)$. However, if n_b is too big, the periodicity of h_3 might cause repeating patterns to start appearing in the binary solution. Of course, g is also a potential candidate to replace Υ. In the case of g, the resulting PSO has dimensionality $\frac{n_b}{4}$, from Equations (4.30) and (4.31). However, recall from Figure 4.6 that the problem of spatial disconnect already manifests when g is used to solve 16-dimensional binary problems.

The use of AMPSO with ensemble generating functions will henceforth be referred to as ensemble AMPSO (E-AMPSO).

4.4.5 Using Generating Function Potential with AMPSO Variants

One point of concern regarding generating function potential is whether it is applicable to the AMPSO variants described in Section 3.3. In the case of ID-AMPSO, no modifications were made to the algorithm, so generating function potential automatically applies. In the case of A-AMPSO, an additional coefficient was introduced, but the definition of generating function potential is not dependent on the number of coefficients, so generating function potential also applies to A-AMPSO. The only point of possible confusion is MM-AMPSO (refer to Section 3.3.2). The MM-AMPSO algorithm has the ability to vary its sampling domain. More precisely, the way in which MM-AMPSO calculates the sample positions might cast some doubt regarding the applicability of the implication for lower-than-n_b-dimensional problems, discussed in Section 4.4.3.

Consider the case where $\alpha_\ell = 0$, $\alpha_u = 5$, and $n_b = 5$. From Equation (3.4), $\delta = 1$ for this particular configuration. Further assume that $a = 0$, $b = 0.5$, $c = 0.8$, $d = 0$. When a bit string is generated, the scenario depicted in Figure 3.1 is obtained. Thus, the generated bit string is '`01100`'. Now, according to Section 4.4.3, the bit string '`0110`' should be obtainable by neglecting to sample the final bit. To do so, the value of n_b is reduced to 4. But reducing the value of n_b changes the value of δ, and therefore also the sampling positions. This means that some bits in the newly generated 4-dimensional bit string might change. The concern in this case is that it is no longer guaranteed that the generating function will be able to produce all lower-dimensional bit strings. Fortunately, this effect is easily corrected by adjusting the value of α_u. In this particular case, the value of α_u can be set to 4 to obtain the correct value for δ to produce the desired bit string.

From the example above, it is clear that by making the necessary adjustment to α_u, MM-AMPSO can also generate any lower-than-n_b-dimensional bit string, assuming that the generating function has an n_b-potential of 1. Thus, the concept of generating function potential, including the implication for lower-than-n_b-dimensional problems, is equally applicable to AMPSO and all the AMPSO variants contained in this thesis.

4.5 Frequency Distribution of Candidate Solutions

It should be noted that $P_{\Upsilon}^{n_b}$, as described in Section 4.4, says nothing about the distribution of solutions in the search space. This section investigates how candidate binary solutions are distributed througout the AMPSO search space. Section 4.5.1 discusses the importance of uniform candidate solution frequency distributions. Section 4.5.2 introduces an alternative method to solve binary problems with PSO, which guarantees a uniform frequency distribution of candidate solutions.

4.5.1 Frequency Distribution of Binary Solutions in AMPSO

Consider again the illustration given in Figure 3.1. In that illustration, the values $a = 0$, $b = 0.5$, $c = 0.8$, and $d = 0$ produced the binary solution '`01100`'. However, it is trivial to see that choosing a sufficiently small, but non-zero positive value for a would produce the exact same binary solution. The fact that different permutations of the coefficients can produce the same binary solution has two obvious effects. Firstly, the search space contains plateaus, and, secondly, all solutions are not guaranteed to exist in the search space *with the same frequency*. That is, even if $P_{\Upsilon}^{n_b} = 1$, meaning that every possible binary solution exists in the search space, some solutions may be much more common (and therefore easier to find) than other solutions.

In order for PSO to find good solutions, it is desirable that every potential candidate solution should exist in the search space with the same frequency. Without a uniform solution frequency distribution, PSO may not be able to find good solutions if they happen to be infrequent compared to bad solutions. When solving continuous problems, the solution frequency distribution in the PSO search space is uniform, because every real number exists in $\mathbb{R}$ exactly once. However, when angle modulation is applied to PSO, the relationship between the continuous generating function and the binary solution space causes a non-uniform frequency distribution of binary solutions in the AMPSO search space. This is because there are more possible permutations of the coefficients of g than there are binary solutions.

To illustrate, consider an arbitrary 8-bit binary problem that is being optimised with AMPSO. Any given permutation of the coefficients of g will result in a specific

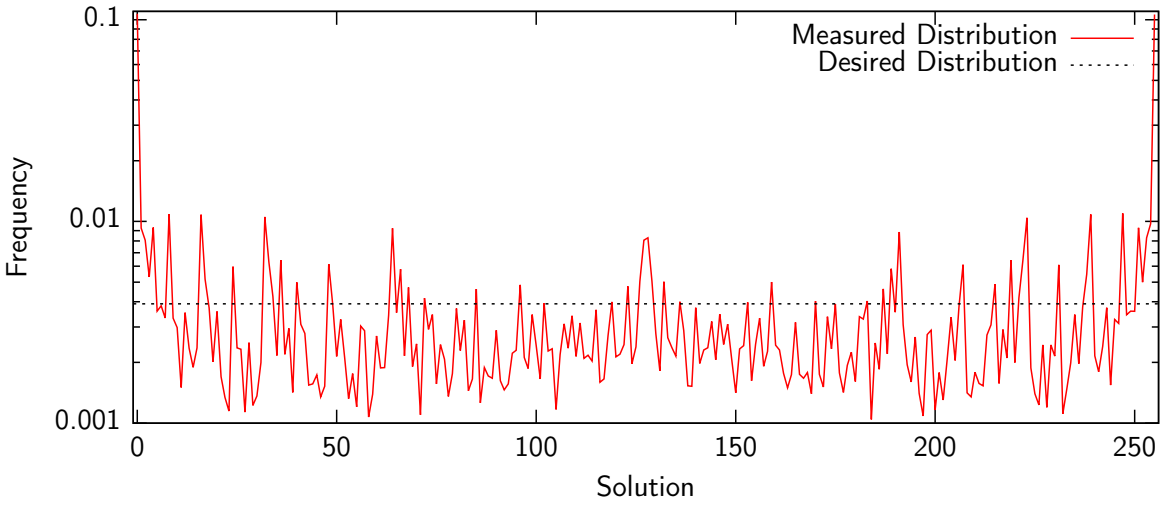


Figure 4.9: An estimation of the frequency distribution of binary solutions in the AMPSO search space for 8-dimensional binary problems, with g as the generating function.

binary solution. The frequency distribution of binary solutions can then be seen by generating a solution for every possible permutation of the coefficients of g and counting the number of times that each unique solution is generated. This is obviously not possible, because the coefficients of g are real values, which means that there is an infinite number of permutations. However, as was the case in Section 4.4.4.1, the frequency distribution of solutions in the AMPSO search space can be estimated by limiting the values that any coefficient may assume to some subset $\mathbb{C} \subseteq \mathbb{R}$. Figure 4.9 shows the frequency distribution of 8-bit binary solutions in the AMPSO search space when $\mathbb{C} = \{-1, -0.9, 0.8, \ldots, 0.0, 0.1, 0.2, \ldots, 1.0\}$. The total number of permutations of the coefficients of g in this estimation is $|\mathbb{C}|^4 = 21^4 = 194,481$. The total number of 8-bit binary solutions is $2^8 = 256$. In Figure 4.9, every value on the x-axis is an integer which represents the corresponding binary solution. For example, the value 0 corresponds to the binary string '00000000', the value 1 corresponds to '00000001', etc.

It is clear from Figure 4.9 that — although every 8-bit binary solution exists — the frequency distribution of binary solutions in the AMPSO search space is not close to uniform. In fact, 20% of the search space is dominated by the '00000000' and '11111111' solutions. This provides an explanation for the ease at which the deceptive problems were

solved by AMPSO and all of its variants in Chapter 3. A number of smaller spikes in the frequency distribution plot indicate that some solutions are much more common than others. The twelve most common solutions (not counting the two solutions mentioned above) are listed below:

$$4_{10} = 00000100_2, \qquad 128_{10} = 10000000_2,$$
$$8_{10} = 00001000_2, \qquad 191_{10} = 10111111_2,$$
$$16_{10} = 00010000_2, \qquad 223_{10} = 11011111_2,$$
$$32_{10} = 00100000_2, \qquad 239_{10} = 11101111_2,$$
$$64_{10} = 01000000_2, \qquad 247_{10} = 11110111_2,$$
$$127_{10} = 01111111_2, \qquad 251_{10} = 11111011_2.$$

The two least common solutions are $71_{10} = 01000111_2$ and $184_{10} = 10111000_2$.

It is obvious from the above observations that the most common solutions in the AMPSO search space are the ones that contain a lot of repetition. This trend is also true for binary solutions of higher dimensionality, although the frequency distribution plots become difficult to read because of the exponential growth in the number of binary solutions.

The cause of this trend is the generating function. Regularities in the shape of the generating function mean that some solutions are much easier to generate than others by varying the values of the coefficients. The solutions that are easy to generate may overshadow good solutions in the search space. Furthermore, common solutions may cause plateaus in the fitness landscape, thereby complicating the search even more.

This problem can potentially be alleviated by using a different generating function. However, it is difficult to imagine a generating function that would produce a uniform solution frequency distribution. Indeed, even for the simple case where h_3 (Equation (4.22)) is used to solve a 3-dimensional binary problem, the solution frequency distribution is also not uniform. An estimation of the frequency distribution of 3-dimensional binary solutions in the AMPSO search space — when h_3 is used as the generating function — is shown in Figure 4.10. In this case, $\mathbb{C} = \{1.00, 1.01, 1.02, \ldots, 17.0\}$. A more appropriate solution to this problem might be to replace angle modulation with a different method altogether that naturally produces a uniform solution frequency distribution.

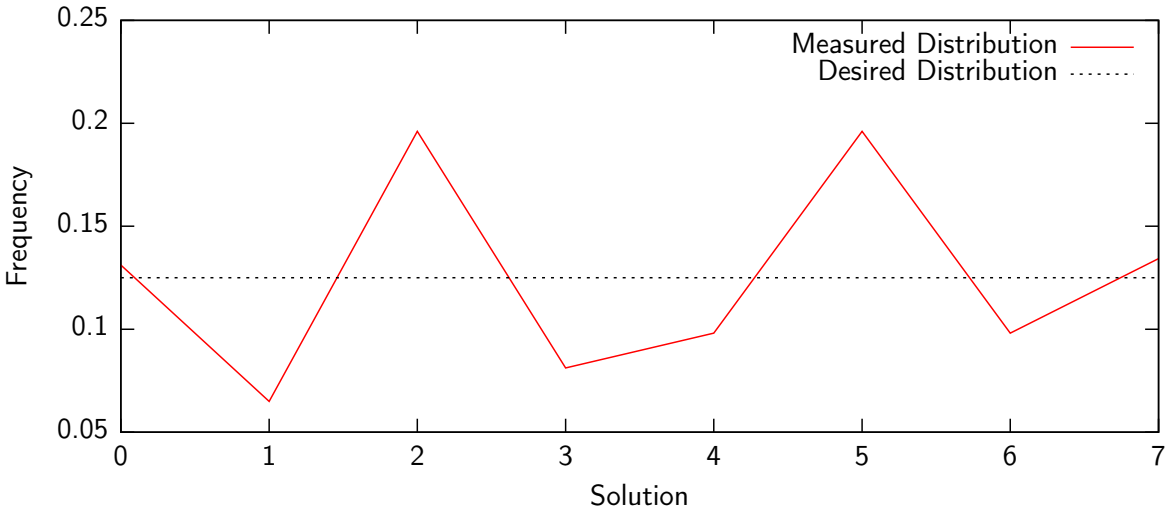


Figure 4.10: An estimation of the frequency distribution of binary solutions in the AMPSO search space for 3-dimensional binary problems, with h_3 as the generating function.

4.5.2 Obtaining A Uniform Solution Frequency Distribution with Natural Numbers

This section details a new approach to solve binary optimisation problems with PSO. The method is explained in Section 4.5.2.1, while two potential problems with the new approach are addressed in Sections 4.5.2.2 and 4.5.2.3.

4.5.2.1 PSO with Natural Numbers

To obtain a uniform solution frequency distribution, some uniform mapping from the continuous feasible space $\mathcal{F}$ to $\mathbb{B}^{n_b}$ is required. That is, a mapping $\mathcal{M} : \mathcal{F} \rightarrow \mathbb{B}^{n_b}$ is needed, such that for every $\mathcal{B} \in \mathbb{B}^{n_b}$, there is an equal number of values $\boldsymbol{x} \in \mathcal{F}$ that map to $\mathcal{B}$.

Consider the set of natural numbers $\mathbb{N}^0 = \{0, 1, 2, \ldots\}$. Every element in $\mathbb{N}^0$ can be mapped to a binary value by converting the element from base-10 to base-2. Furthermore, the mapping is a one-to-one mapping, meaning that it is always uniform. Now, clearly

$\mathbb{N}^0$ is not continuous. However, observe that

$$\lfloor x \rfloor \in \mathbb{N}^0 \; \forall \; x \in \mathbb{R}_{\geq 0}. \tag{4.33}$$

Therefore, PSO can be used to find a real value $x \in \mathbb{R}_{\geq 0}$ such that $\lfloor x \rfloor$ is the natural number that maps to the desired binary solution. Furthermore, to ensure that a mapping exists from $\mathcal{F}$ to every element $\mathcal{B} \in \mathbb{B}^{n_b}$, only a subset of $\mathbb{R}_{\geq 0}$ is required, such that $\mathcal{F} \subseteq \mathbb{R}_{\geq 0} = [0, 2^{n_b})$, and $n_x = 1$.

For example, if $n_b = 3$, then there are $2^3 = 8$ candidate binary solutions, which map to the following eight natural numbers: $\{0, 1, 2, 3, 4, 5, 6, 7\}$. Therefore, the continuous feasible search space $\mathcal{F}$, whose floored elements are equal to these eight natural numbers, is $\mathcal{F} = [0, 8)$. A one-dimensional PSO can now be used to search $\mathcal{F}$ for the natural number that maps to the optimal three-bit binary solution. This technique will henceforth be referred to as the natural numbers PSO (NNPSO). Note that the frequency distribution of binary solutions in $\mathcal{F}$ is naturally uniform in the case of NNPSO.

Because NNPSO guarantees that all the possible candidate solutions are contained in $\mathcal{F} = [0, 2^{n_b})$, particles in NNPSO should not update their personal best positions unless they are within these bounds.

4.5.2.2 Hamming Cliffs

An immediate point of concern in NNPSO is that two similar solutions in $\mathcal{F}$ could produce two dissimilar solutions in $\mathbb{B}^{n_b}$. For example, assume that $n_b = 3$ and, consequently, $\mathcal{F} = [0, 8)$. Now consider the values $3.5 \in \mathcal{F}$ and $4.2 \in \mathcal{F}$. To map these values to binary solutions, the floor function is computed before converting the value to base-2. That is,

$$\begin{aligned} \lfloor 3.5 \rfloor &= 3_{10} \rightarrow \texttt{011}_2, \text{ and} \\ \lfloor 4.2 \rfloor &= 4_{10} \rightarrow \texttt{100}_2. \end{aligned} \tag{4.34}$$

The values 3.5 and 4.2 produce consecutive natural numbers, but the Hamming distance between the two binary solutions is 3. This is clearly not desirable, because it means that similar solutions are not grouped together in the search space, which (recall from Section 4.2) is a fundamental assumption that PSO exploits to perform optimisation. Fortunately, this problem is easily addressed by using a gray code conversion, instead

of a binary conversion. Gray code ensures that consecutive values in $\mathbb{N}^0$ always produce binary strings with a Hamming distance of 1. In the case above, gray code conversion produces the following binary solutions:

$$\begin{aligned} \lfloor 3.5 \rfloor &= 3_{10} \rightarrow 010_2, \text{ and} \\ \lfloor 4.2 \rfloor &= 4_{10} \rightarrow 110_2. \end{aligned} \tag{4.35}$$

4.5.2.3 Search Space Explosion

Another problem with NNPSO is that the size of the one-dimensional search space $\mathcal{F}$ grows exponentially with n_b. Recall from Section 4.5.2.1 that $\mathcal{F} = [0, 2^{n_b})$. The effect is that the search space quickly becomes so large that one would need a huge number of particles in the swarm in order to get close to adequate coverage of the search space during swarm initialisation. As a result, PSO becomes ineffective when solving large binary problems with NNPSO.

One potential (although not foolproof) way to address this problem is to divide the binary problem into separate, smaller problems, at the expense of increased particle dimensions. For example, an 8-dimensional binary problem can be split into two four-dimensional binary problems. Then, NNPSO would be used to find *two* natural numbers $\lfloor x_1 \rfloor$ and $\lfloor x_2 \rfloor$ whose gray codes can be concatenated to produce one 8-dimensional binary solution. This obviously implies that particles in NNPSO are now two-dimensional. That is, where it used to be the case that $\mathcal{F} = [0, 256)^1$, now $\mathcal{F} = [0, 16)^2$. This approach will henceforth be referred to as ensemble NNPSO (E-NNPSO).

To formalise, the E-NNPSO algorithm can be used to solve n_b-dimensional binary problems in a ϕ-dimensional real-valued search space $\mathcal{F} = [0, 2^{n_b/\phi})^\phi$. As mentioned above, this approach is not foolproof, because the search space still grows exponentially, albeit not in a single dimension. The hope is that PSO would be able to cope better with the growth if the domain can be kept relatively small in each dimension.

4.6 Binary Solution Representations

The final aspect of AMPSO that will be considered is the influence of specific binary solution representations on the algorithm's performance. In order to illustrate the com-

plications introduced by binary solution representations, a heavy focus is placed on the n-queens problem in this section. However, the ramifications discussed here hold true for any binary problem, albeit in ways that are specific to the individual problem.

The representation of solutions in the search space is important, because the PSO algorithm makes certain assumptions. In particular, PSO assumes that similar solutions are grouped together in the search space. In general, this assumption refers to the fitnesses of different solutions. That is, solutions that return good objective function values are assumed to be grouped together. However, the assumption is actually deeper than that. Consider first the representation of solutions in a continuous search space: each variable is represented by a real number. Now consider the way in which solutions are modified by PSO during the search process: a particle is moved by a certain step size in every dimension, with small step sizes representing minor adjustments to the solution. That is, while the PSO algorithm expects to find solutions with similar objective function values close by, it generates those similar solutions by manipulating the solution representation. Therefore, not only does PSO assume that similar solutions are grouped together, but also that the representations of those solutions are similar — solutions that *look* similar *are* similar.

At this point the paragraph above may seem blindingly obvious and not at all useful, but that is partly because the assumptions are clearly true when dealing with continuous problems. It is only once binary solution representations are introduced that the assumptions can subtly break down.

Section 4.6.1 analyses the n-queens solution representation, as implemented in CIlib, and shows why the assumptions made by PSO do not necessarily hold in the binary context. In Section 4.6.2, a new n-queens solution representation is introduced, which does not violate the assumptions made by the PSO algorithm.

4.6.1 N-Queens Solution Representation in CIlib

Recall from Section 3.4.1 that the implementations of the benchmark problems used in that study was provided by CIlib. The n-queens problem was briefly discussed in Section 3.4.1.1. This section discusses the n-queens problem in more detail in order to illustrate how the solution representation in CIlib causes the assumptions made by PSO

to break down.

4.6.1.1 The chess board

In CIlib, the $n \times n$ chess board is represented as a binary string $\mathcal{B}$ of length n^2. Since each square on the board is either occupied by a queen or not, the problem lends itself well to this binary representation. For example, to represent a candidate solution to the 4-queens problem, the 4×4 chess board is represented as a 16-bit binary string, where a 0-bit indicates an empty square, and a 1-bit indicates an occupied square. Figure 4.11 illustrates two candidate solutions to the 4-queens problem, together with their binary representations.

4.6.1.2 The objective function

The objective function in CIlib is defined as

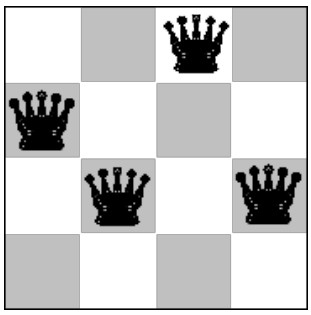
$$f(\mathcal{B}) = 1000(|n - \mathcal{Q}|) + \sum_{i=1}^{\mathcal{Q}} k_i, \tag{4.36}$$

where $\mathcal{Q}$ is the total number of queens on the board, and k_i is the total number of conflicts that queen i is involved in. Thus, the objective value for the example candidate solution in Figure 4.11(a) is $f(\mathcal{B}) = 4$. An optimal solution (one with n queens and no conflicts between any two queens), such as the solution in Figure 4.11(b), results in a fitness evaluation of 0.

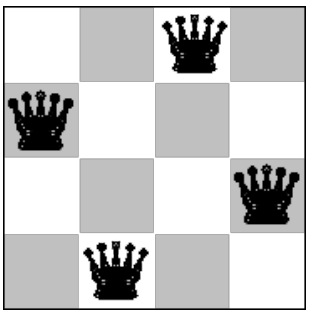
The first term in Equation (4.36) assigns large fitness values to invalid solutions (solutions that do not have exactly n queens on the board). The fitness for such solutions increases linearly for each additional missing or redundant queen.

4.6.1.3 What's the problem?

The problem with the above solution representation manifests when the PSO algorithm attempts to make adjustments to the solution representation in order to move towards a better solution. Consider the two solutions illustrated in Figure 4.11. If the queens in Figure 4.11(a) are numbered sequentially in the order that they appear in the binary representation, then moving queen 3 one row down would result in the optimal solution



(a) 0010 1000 0101 0000



(b) 0010 1000 0001 0100

Figure 4.11: Two candidate solutions to the 4-queens problem. The binary solution representation $\mathcal{B}$ for each solution is given below its corresponding figure. (Spaces are included for ease of reading).

shown in Figure 4.11(b). That is, these two solutions are conceptually adjacent: one single-square move is required to obtain the optimal solution. However, when looking at the binary representations, one notes that *two* bits have changed between the two representations. Furthermore, the two bits are far apart.

For this particular move, one could argue that both the two binary representations and the resulting solutions are still relatively close together (the Hamming distance between the two representations is only 2). However, consider what would happen if the PSO algorithm had changed two different bits from the ones in the example above. For instance, if any two 0-bits were changed to 1-bits, that would result in a vastly inferior solution, but which also has a Hamming distance of 2 from Figure 4.11(a). Alternatively, changing bits 4 and 5 results in a solution that has a Hamming distance of 2 from Figure 4.11(a), but would require multiple moves to obtain, according to the rules of chess. That is, a small change in the solution representation results in a large change in the actual solution. To make matters worse, changing any single bit in Figure 4.11(a) necessarily results in an inferior solution, even though a single bit flip implies a smaller Hamming distance than the required 2-bit change. Hence, the assumption that solutions whose representations look similar are, in fact, similar is violated. As a result, the PSO algorithm's attempts to move to better solutions by making minor adjustments to the current solution representation are flouted. The problem is subtle, but real.

4.6.2 A New Solution Representation for the N-Queens Problem

Coming up with a binary solution representation that does not violate the assumptions made by PSO might require a substantial rethinking of the particular problem. The solution will often employ some domain knowledge and therefore only apply to the specific binary problem at hand. To illustrate that it is possible, a new binary solution representation for the n-queens problem is developed here. New binary solution representations for additional problems are beyond the scope of this work.

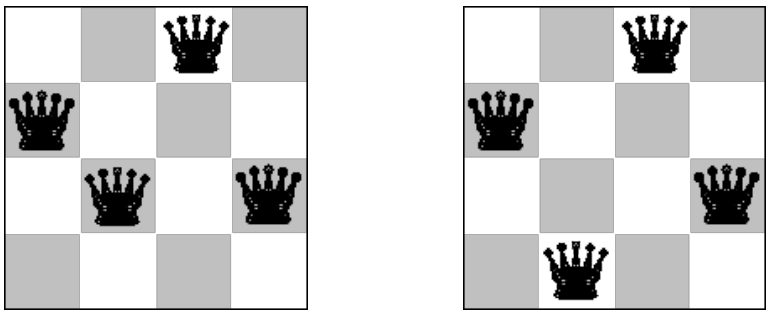
4.6.2.1 N queen positions

A considerable disadvantage of the n-queens solution representation in Section 4.6.1 is that queens can appear at any location on the board by changing 0-bits to 1-bits. Similarly, queens can disappear from any location on the board by changing 1-bits to 0-bits. This property causes single bit changes in the solution representation to produce large changes in both the actual solution (the state of the chess board), as well as the objective function values. Hence, getting rid of this characteristic seems like a good place to start.

To that end, the new binary representation will do away with the n^2-length binary string that represents the chess board. Observe instead that there are n queens on the board, and that each queen must occupy a position in a different column.[2] Therefore, the position of each queen in its column can be represented by a binary string of length $\lceil \log_2 n \rceil$, where the decimal value of the binary representation indicates the queen's position. In the case of the 4-queens example, this implies that the position of each queen is represented by $\lceil \log_2 4 \rceil = 2$ bits, as illustrated in Figure 4.12. In this representation, queens can only move up or down — never sideways.

The first obvious advantage of the new representation is that fewer bits are required to represent solutions. Additionally, queens can no longer appear and disappear. In fact, invalid solutions are only possible if n is not a power of two. This is because for

[2]This assumption is true for any optimal solution to the n-queens problem, but does not necessarily hold for other chess board problems. The n-queens problem is also rendered substantially easier by this assumption, as many sub-optimal solutions can no longer be represented.



(a) 01 10 00 10 (b) 01 11 00 10

Figure 4.12: New binary representation for the 4-queens problem. For each queen, 2 bits are used to represent the queen's position in its column.

any n that is a power of two, the resulting number of bits for each queen can represent exactly n decimal numbers. Hence, every possible binary string represents a solution with exactly n queens on the $n \times n$ chess board. However, if n is not a power of two, some queens may be moved off the board by increasing their positions to values greater than $n - 1$. For example, on a 5×5 chess board, each queen requires three bits, and a binary representation of '101', '110' or '111' for any queen would move that queen beyond the lower edge of the board. Nonetheless, this is already an improvement over the previous representation.

The next step is to ensure that solutions that are close together conceptually also have binary representations that are close together. The problem with the current representation is that the binary-to-decimal conversion creates Hamming cliffs in the resp-resentations. Consider moving a queen down its column on a 4×4 board. The queen's representation would follow the following sequence: 00 $\rightarrow$ 01 $\rightarrow$ 10 $\rightarrow$ 11. Note that the second step in this sequence requires two bits to flip. To prevent this issue, gray code will be used instead of binary to represent queen positions. The result is that every single-square move on the board implies a one-bit change in the solution representation. For example, on a 4×4 board, the sequence of representations as a queen moves down its column is now: 00 $\rightarrow$ 01 $\rightarrow$ 11 $\rightarrow$ 10.

4.6.2.2 The objective function

Finally, it is necessary to construct an objective function that takes advantage of the characteristics of the new solution representation. To achieve a sensible objective function, and to make the purpose of each term clear, the function is assembled in a stepwise fashion below. Where applicable, symbols have the same definition as in Equation (4.36).

The objective is to minimise the number of conflicts between queens on the board. Therefore, a natural function to start with is one that simply counts the number of conflicts between queens in a given solution:

$$f(\mathcal{B}) = \sum_{i=1}^{\mathcal{Q}} k_i. \tag{4.37}$$

An obvious problem with this objective function is that it does not take into account the possibility of a solution having more or fewer than exactly n queens on the board. In Equation (4.36), this problem was addressed by adding a linear term, which increased by a large constant for each additional missing or excessive queen. While the approach makes sense, it poses the problem of choosing an appropriate constant — in the case of CIlib, the value 1000 was chosen somewhat arbitrarily. Additionally, as was discussed in Section 4.6.1.3, the large penalty for invalid solutions creates very deep local optima in the search landscape. For these reasons, a quadratic term will be used instead, such that the fitnesses of solutions with missing queens still increase rapidly, but the penalty is small initially:

$$f(\mathcal{B}) = (n - \mathcal{Q})^2 + \sum_{i=1}^{\mathcal{Q}} k_i \tag{4.38}$$

The objective function is now similar in form to Equation (4.36), but it does not yet exploit any properties of the solution representation. Recall from Section 4.6.2.1 that the new representation already forces every queen to occupy a position in its own column. This was deemed an improvement to the solution representation, because any optimal solution to the n-queens problem necessarily has that property. It then follows that the promotion of solutions in which every queen occupies its own row would also be beneficial to the search, regardless of whether there are still conflicts between queens in such a solution. (Note that minimisation of conflicts is already catered for in Equation (4.38)).

To promote solutions where each queen occupies its own row, consider triangular numbers. Triangular numbers are the sequence of numbers obtained from the sequential summation of the natural numbers. Thus, the natural numbers are $\mathbb{N}^+ = \{1, 2, 3, 4, 5, 6, \ldots\}$, and the traingular numbers are $\mathbb{T} = \{1, 3, 6, 10, 15, 21, \ldots\}$. Triangular numbers are so named, because the m^{th} triangular number can be illustrated visually by arranging a number of objects in the shape of an equilateral triangle with sides of length m, as shown in Figure 4.13. The m^{th} triangular number T_m is given by

$$T_m = \frac{m(m+1)}{2}. \tag{4.39}$$

Now, let every square B_{ij} on the chess board be 0 if the square is empty, or 1 if the square is occupied by a queen. Furthermore, let

$$R = \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} r_{ij}, \tag{4.40}$$

where

$$r_{ij} = \begin{cases} i & \text{if } B_{ij} = 1 \\ 0 & \text{otherwise.} \end{cases} \tag{4.41}$$

Then, $R = T_{n-1}$ if and only if every row i on the $n \times n$ chess board contains exactly one queen. Thus, the objective function can be penalised if queens are not properly spread out on the chess board:

$$f(\mathcal{B}) = \left[(n - \mathcal{Q})^2 + \sum_{i=1}^{\mathcal{Q}} k_i \right] \times \left(|T_{n-1} - R| + 1 \right). \tag{4.42}$$

Note that if every queen occupies its own row, then Equation (4.42) reverts to Equation (4.38).

Thus, the solution representation derived in Section 4.6.2.1, combined with Equation (4.42), firstly does not violate the fundamental assumptions made by the PSO algorithm, and secondly exploits the properties of the solution representation. It should be clear that the process outlined in this section is heavily dependent on the nature of the binary problem in question, but that it is an important process to follow if one hopes to find good solutions to the binary problem using PSO.

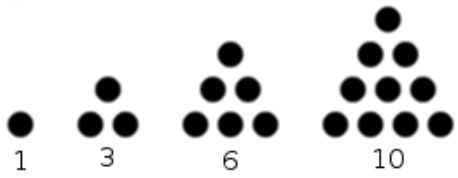


Figure 4.13: The first four triangular numbers illustrated as the number of dots needed to form equilateral triangles with sides of length 1, 2, 3, and 4, respectively.

4.7 Summary

This chapter investigated various aspects of angle modulated particle swarm optimisers in order to understand why the algorithm might fail to optimise arbitrary binary problems.

The periodicity of the generating function was investigated and found not to be a problem in the standard AMPSO algorithm or any of its existing variants. However, it was discovered that the roots of the generating function increase in frequency along the x-axis in both directions. This characteristic was shown to cause a basic assumption made by PSO (that good solutions are grouped together in the search space) to be violated. The problem is referred to as *spatial disconnect*, and was shown to manifest in all existing AMPSO variants, regardless of which binary problem is being optimised.

The first empirical analysis of particle convergence in angle modulated particle swarm optimisers was also provided. It was shown that particles in AMPSO tend to stabilise at high velocities, relative to the size of the search domain. This problem is referred to as *inadequate convergence*. The most important consequence of inadequate convergence is that particles in AMPSO are not able to exploit good solutions in the search space, because their step sizes remain too high. The problem was discussed in terms of the spatial disconnect that was discovered earlier.

The problems discovered in the initial parts of this chapter were all associated with the generating function. In particular, spatial disconnect seemed to be the cause of these problems. Thus, it was hypothesised that the best way to address the problems would be to replace the generating function with one that does not cause spatial disconnect. To this end, the chapter outlined the first formal definition to quantify the ability of arbitrary generating functions to solve arbitrary binary problems. This quantity is referred to as the *potential* of the generating function. Using the definition of generating

function potential, a new generating function was constructed to solve binary problems with fewer than four dimensions. This new generating function has a single coefficient, meaning that a reduction in dimensionality when solving simple binary problems using AMPSO was made possible for the first time. Furthermore, the use of multiple generating functions in AMPSO was proposed as a method to solve arbitrarily high-dimensional binary problems. Generating functions consisting of multiple functions that solve parts of the binary problem are referred to as *ensemble generating functions*.

Additionally, it was shown that the frequency distribution of binary solutions in the AMPSO search space is not uniform. Consequently, some solutions might be harder to find than other solutions, simply because they are not as common in the search space. This problem was addressed by proposing a novel PSO variant, called NNPSO, which guarantees that solutions are distributed uniformly in the feasible search space.

Finally, this chapter discussed the importance of good solution representations when solving binary problems with PSO. Some focus was placed on the n-queens problem to illustrate how naive solution representations can subtly break the assumptions made by PSO. A new solution representation for the n-queens was developed in order to ensure that no known assumptions are violated.

The insights gained from this chapter pave the way for various additional studies, as well as many more potential improvements to angle modulated particle swarm optimisers. The next chapter provides empirical investigations into some of the proposals that were made in this chapter. Investigating every proposal thoroughly could easily result in a vast number of experiments. In order to keep the scope manageable, only those proposals that seem most promising, when considered in the greater context this chapter, are examined further.

Derived Publications

- The analyses presented in Sections 4.1 – 4.4 were published in [28].
- The analyses presented in Sections 4.5 and 4.6 formed part of the work that was published in [27].

Chapter 5

Empirical Analysis

> *Science, my lad, is made up of mistakes, but they are mistakes which it is useful to make, because they lead little by little to the truth.*
>
> *Jules Verne*

This chapter provides an empirical analysis in which some of the suggestions that were made throughout this dissertation are investigated. The aim is not to draw conclusive findings on the capabilities of the various approaches, but rather to substantiate or refute the suggestions with initial empirical evidence, whatever the case may be. The work presented here will serve as a guideline for constructing meaningful, in-depth empirical studies in future research endeavours.

Section 5.1 provides brief summaries of the algorithms used in this section. An overview of the problems is given in Section 5.2, while the measurements are listed in Section 5.3. The results are discussed in Section 5.4, and the chapter is concluded in Section 5.5.

5.1 Algorithms

This section describes the algorithms that were compared in the experimental analysis. Each of the algorithms has already been discussed in detail in previous chapters. Therefore only brief descriptions, together with references will be provided here. Section 5.1.1

describes the algorithms used in the experiments. For those algorithms that did not form part of the experimental study, justifications are provided in Section 5.1.2, while the parameters used in the experiments are given in Section 5.1.3.

5.1.1 Algorithm Descriptions

The algorithms used in the empirical analysis are listed below. Algorithms are shown in bold, while a brief explanation of each algorithm is printed in normal font beside each entry:

- **BPSO**: a variant of PSO, designed to work on binary optimisation problems. The algorithm interprets particle velocities as probabilities in order to determine how particles move in binary space. The algorithm is discussed in detail in Section 2.3.
- **AMPSO**: an algorithm which solves binary optimisation problems via an intermediary continuous optimisation problem. PSO is applied to find the optimal coefficients of a continuous generating function, such that the generating function can be used to produce the optimal binary solution. For a complete overview, see Sections 3.1 and 3.2.
- **AMPSO variants**
  - **A-AMPSO**: adds an additional coefficient to the AMPSO algorithm in order to control the amplitude of the generating function. An overview is presented in Section 3.3.1.
  - **MM-AMPSO**: augments particles in AMPSO with two additional dimensions to control the sampling range of the generating function. A discussion is provided is Section 3.3.2.
- **E-AMPSO**: divides the binary optimisation problem into equal parts. Each part is then assigned its own generating function, and the coeffiecients of all generating functions are optimised simultaneously by PSO at the expense of increased particle dimensions, when compared to AMPSO. The concept is fully explained in Section 4.4.4.2.

- **E-NNPSO**: uses NNPSO (see Section 4.5.2.1) to solve binary problems by dividing the problem into equal parts. For each part of the problem, the partial binary solution is found by searching for a natural number with PSO. The natural numbers are all optimised simultaneously. The algorithm is introduced in Section 4.5.2.3.

5.1.2 Justifications for Omitted Algorithms

The following is a list of algorithms that were discussed in this dissertation, but omitted from the empirical analysis. The algorithm is printed in bold below, together with the justification for its omition in normal font:

- **PSO**: this algorithm is fundamental to understanding how the AMPSO algorithm and all of its variants function. Therefore, PSO was discussed in detail in Section 2.2. However, the PSO algorithm is a continuous algorithm and cannot solve binary problems in its original form. For this reason PSO is not considered in the analysis performed in this chapter.

- **ID-AMPSO**: the study presented in Chapter 3 found no difference in performance between AMPSO and ID-AMPSO (see Section 3.5). This result, combined with the fact that the variation proposed by the algorithm is extremely minimal, provides reasonable grounds to not consider the algorithm in further studies.

- **NNPSO**: as discussed in Section 4.5.2.3, the one-dimensional search space in NNPSO grows exponentially with n_b. In practise, this quickly becomes a problem as the search space becomes so large that overflows occur in particles' positions. For this reason the performance of NNPSO could not be accurately assessed, and the algorithm was therefore not considered. In the case of E-NNPSO, the search space grows in multiple dimensions, preventing the overlow issues.

5.1.3 Algorithm parameters

For all algorithms, the parameters listed in Table 2.1 were used. In the case of BPSO, particle velocities were clamped at a maximum of 4.

5.2 Problems

This section describes the binary problems that were used to assess the performance of the various algorithms listed in Section 5.1.1. For ease of reference, a brief overview of each binary problem is presented in its own section below.

5.2.1 N-Queens

The n-queens problem requires one to place n queens on an $n \times n$ chess board in such a way that no two queens are in conflict, according to the standard rules of chess. The n-queens problem was discussed in great detail in this dissertation. Sections 4.6.1 and 4.6.2 discuss the CIlib implementation and a newly developed representation of the problem, respectively. In this analysis, the representation developed in Section 4.6.2 was used. Therefore, the objective function is defined as follows (Equation (4.42)):

$$f(\mathcal{B}) = \left[(n - \mathcal{Q})^2 + \sum_{i=1}^{\mathcal{Q}} k_i\right] \times (|T_{n-1} - R| + 1)\,.$$

All algorithms were compared on the n-queens problem with $n = 10$, 15, 20, and 25. Given the solution representation, this resulted in problem dimensions of 40, 60, 100, and 125, respectively. For a complete description of this problem, refer to Section 4.6.2.

In the case of E-AMPSO and E-NNPSO, this problem was divided into n parts. That is, in each case, E-AMPSO optimised n generating functions to find the position of the queen in each column. Similarly, E-NNPSO searched for n natural numbers that mapped to the optimal queen positions.

5.2.2 Knights' Coverage

The knights' coverage problem requires the placement of the minimum number of knights on an $n \times n$ chess board, such that the maximum numbers of squares on the chess board are covered. For a full description of the term "covered", refer to Section 3.4.1.2. The objective function used in this analysis was provided by the CIlib library and is defined as follows:

$$f(\mathcal{B}) = \frac{\mathcal{K}}{\mathcal{C}+1} + \frac{n^2 - \mathcal{C}}{\mathcal{K}+1}, \tag{5.1}$$

where $\mathcal{K}$ is the number of knights on the chess board, and $\mathcal{C}$ is the number of covered squares on the chess board. The knights' coverage problem was investigated for $n = 10, 15, 20,$ and 25, which gave rise to problem dimensionalities of 100, 225, 400, and 625, respectively.

As was the case for the n-queens problem above, this problem was divided into n equal parts when either E-AMPSO or E-NNPSO was used.

5.2.3 Random Bit String Matching

For this problem, random target bit strings of length 50, 100, 200, 300 and 500 were generated. The objective is to find a bit string $\mathcal{B}$, such that the Hamming distance between $\mathcal{B}$ and the target bit string is minimised. The optimal solution has a Hamming distance of 0 to the target bit string.

In the case of E-AMPSO and E-NNPSO, this problem was divided into 5-bit parts in all cases.

5.3 Measurements

In addition to the global best fitness, the measurements used in Section 4.3.1 were also recorded at every iteration. The complete set of measurements are as follows:

- global best fitness,
- expected average magnitude of particle velocities (Equation (2.6)),
- actual average magnitude of particle velocities, and
- whether the global best fitness had changed since the previous iteration.

5.4 Results and Discussion

This section discusses the results obtained from the experiments outlined above. The discussion is split into two parts. Section 5.4.1 focusses on the fitness profiles of the algorithms on the various problem cases, while Section 5.4.2 investigates the velocities

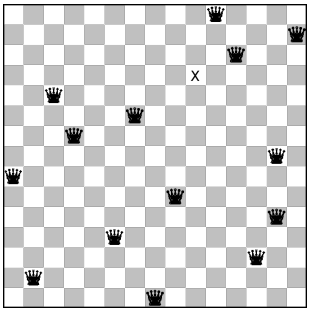
of particles in E-NNPSO during the search process. Finally, the discussion is concluded in Section 5.4.3.

5.4.1 Fitness Results

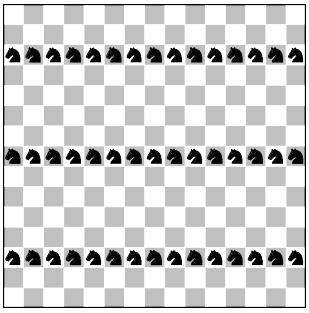
Graphs 5.1 to 5.4 show the average fitness profiles for all algorithms for the n-queens problems. For all three problem cases, E-NNPSO obtained a much lower average fitness than all the other algorithms. To understand why E-NNPSO obtained the lowest average, it is helpful to visualise a solution to the n-queens problem. Figure 5.1(a) shows the best solution found by the E-NNPSO algorithm on the n-queens problem with $n = 15$. The solution is not complete. An "X" in the figure shows the location where the final queen could have been placed. However, a queen in that position would cause a conflict with another queen to the lower right. Note that there is no obvious repeating pattern in this solution to the n-queens problem. While the solution in Figure 5.1(a) is not optimal, it resembles the typical non-repetitive pattern that solutions to the n-queens problem exhibit — at least, when they are encoded as binary strings.

Recall from Section 4.5.1 that candidate solutions that contain repetition are relatively common in the AMPSO search space. Therefore, AMPSO and all of its variants have a very hard time in finding good solutions to the n-queens problem and tend to converge on sub-optimal solutions instead. Essentially, the AMPSO algorithms already fail during the exploration phase of the search. The same argument explains why BPSO obtained a lower average than AMPSO. However, the lack of sensible guides in BPSO ultimately means that the algorithm cannot effectively exploit the search space. In contrast, the E-AMPSO algorithm is able to effectively explore and exploit the solution space, given the new binary representation for this problem. Unfortunately, inescapable local optima are still present in the search landscape, as is evident by the fact the solution depicted in Figure 5.1(a) is still not optimal. Even if a queen is placed at the position of the "X" in the figure, there are still conflicts. Thus, finding the optimal solution would require many moves to make space for the queen that is currently missing.

The fitness profiles for the knights' coverage problems are shown in Graphs 5.5 to 5.8. For this problem, the MM-AMPSO and E-AMPSO algorithms obtained the lowest average fitness in all cases. Again, a visualisation of the solution is helpful. Figure 5.1(b)



(a) E-NNPSO solution

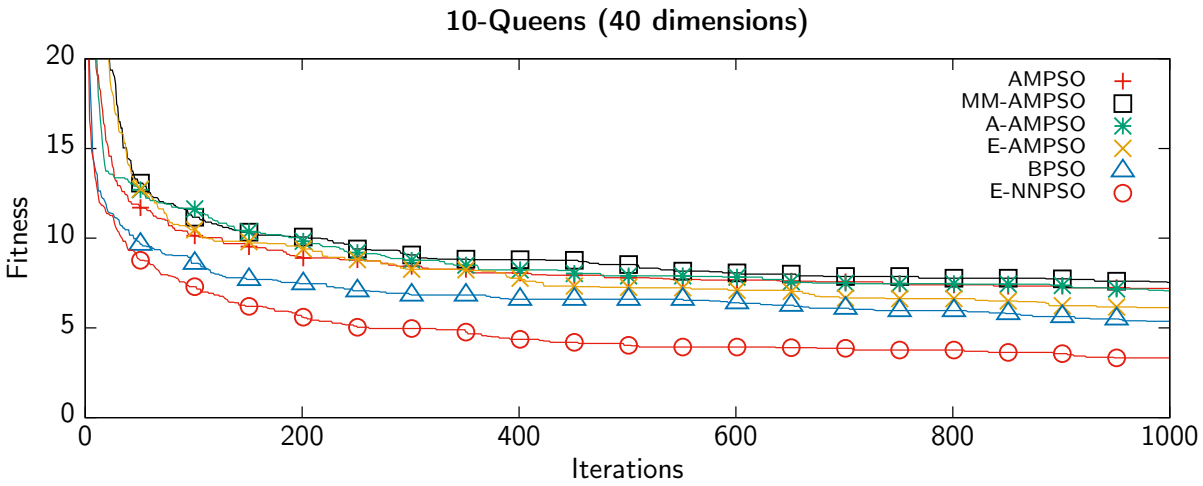


(b) E-AMPSO solution

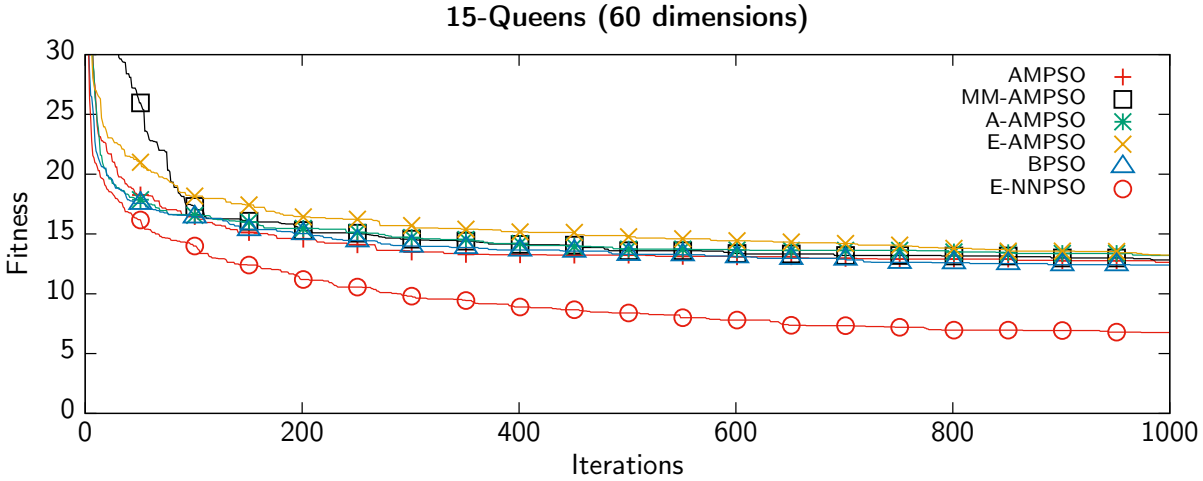
Figure 5.1: Figures (a) and (b) show the best solutions found for n-queens and knights' coverage, respectively. In both cases, $n = 15$. The "X" in Figure (a) marks the location where the final queen could have been placed.

shows the best solution obtained by E-AMPSO on the knights' coverage problem, with $n = 15$. The repetitive pattern in the solution is immediately obvious. This solution also happens to be optimal. The AMPSO algorithms have an advantage on this problem, because the optimal solutions happen to be common in the AMPSO search space. E-AMPSO obtained a lower average fitness in all cases, because the algorithm breaks the problem up into individual columns. In that case, each column's solution is a bit-string of either only '1' bits, or only '0' bits. Recall from Section 4.5.1 that, in the 8-dimensional case, those solutions make up 20% of the search space. While the exact proportion of the search space made up by these solutions might differ for higher dimensions, they do still dominate. In the case of MM-AMPSO, the algorithm is able to isolate parts of the generating function that exhibit the required pattern and thus obtained a lower average fitness than either AMPSO or A-AMPSO.

Finally, Graphs 5.9 to 5.13 show the fitness profiles for the bit string mapping problems. The AMPSO algorithms obtained the highest average fitness values across all problem cases. This was expected, because random bit strings are not likely to contain patterns. E-NNPSO obtained the lowest average fitness on 50-, 100-, and 200-dimensional problem cases. This was also expected for the reasons given above. However, for the 300-dimensional case, E-NNPSO obtained the highest average fitness, while the lowest average fitness was obtained by BPSO. The reason for this dramatic change



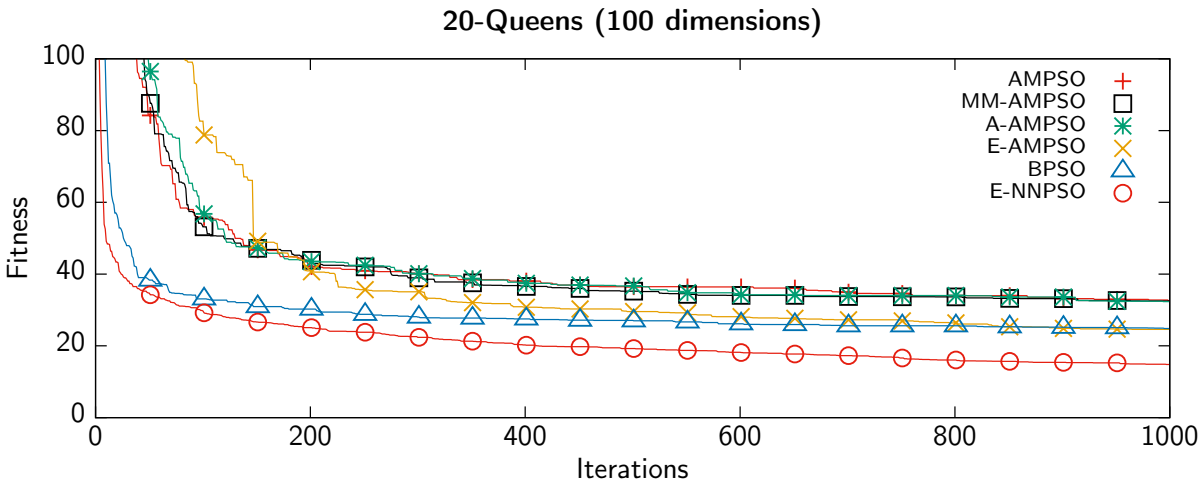
Graph 5.1: Fitness profiles for n-queens problem with $n = 10$.



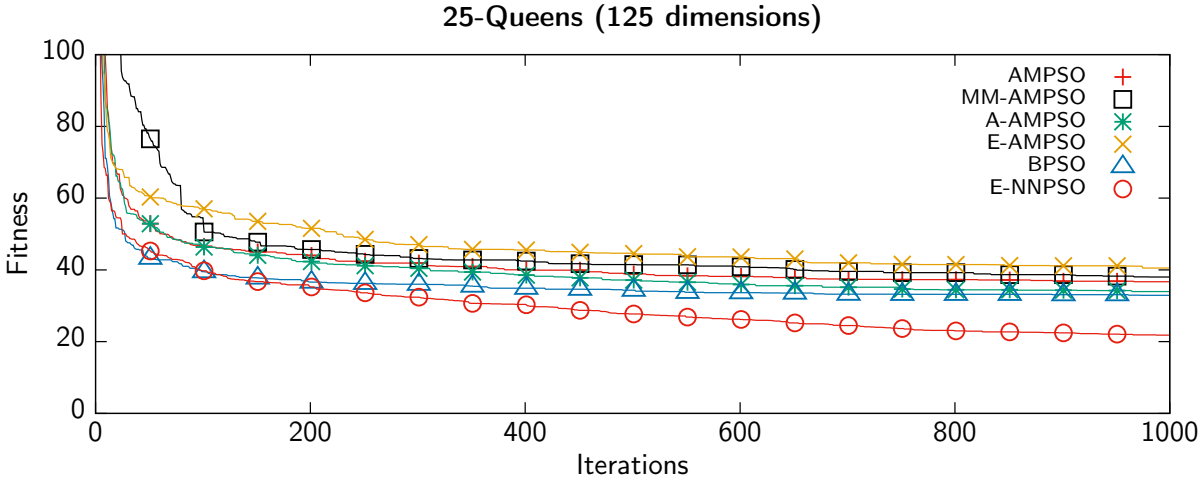
Graph 5.2: Fitness profiles for n-queens problem with $n = 15$.

in the relative performance of E-NNPSO is not currently clear, but warrants further investigation. One possibility is that the division of the problem into 5-bit parts pushes the dimensionality of the resulting PSO too far. However, finding optimal splits for this problem across multiple dimensionalities is beyond the scope of this work. The performance of E-NNPSO deteriorates even further in the 500-dimensional case, with almost no improvements being made on average for the entire duration of the search.

The performance of BPSO on the bit string matching problems is also better than

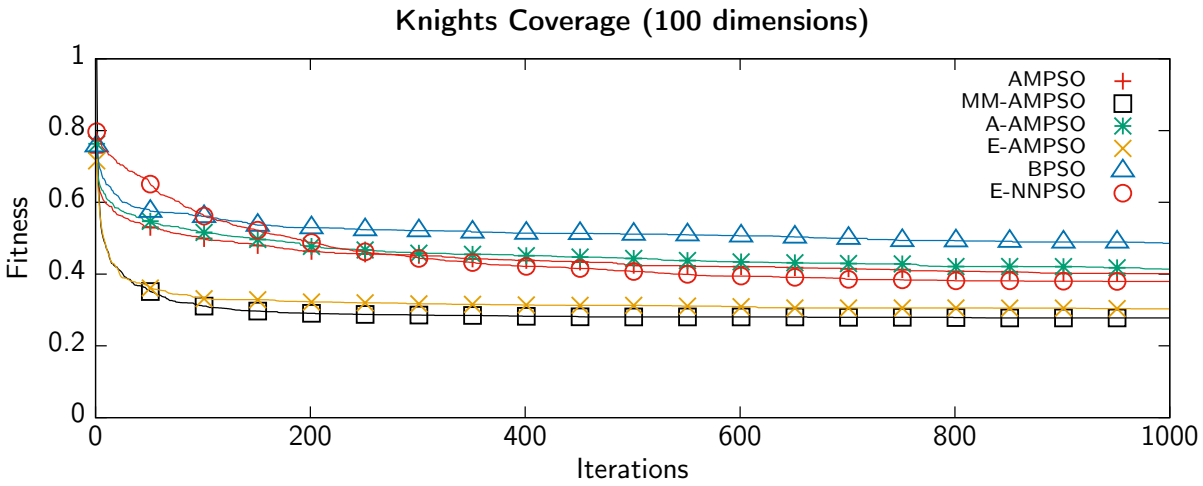


Graph 5.3: Fitness profiles for n-queens problem with $n = 20$.

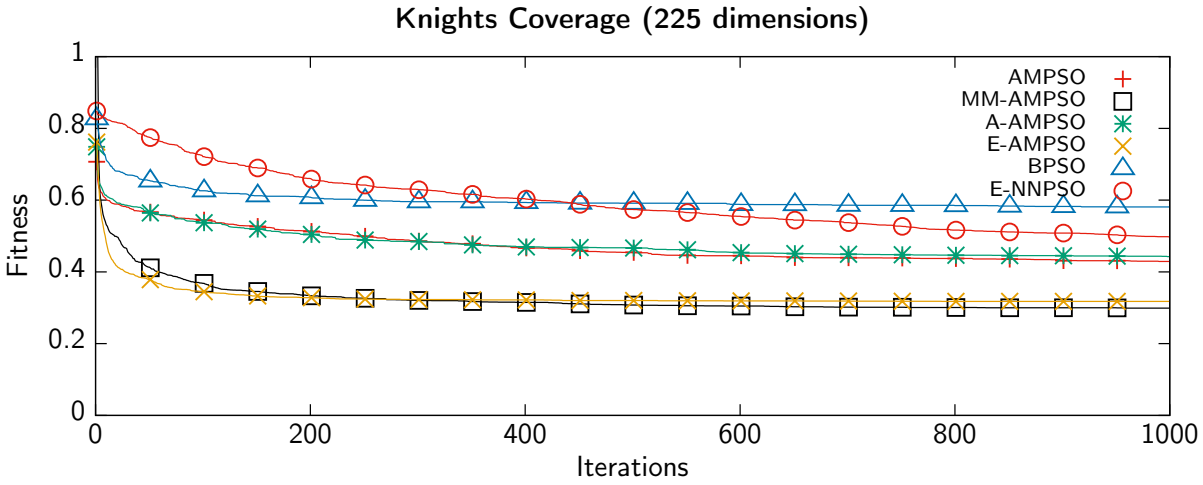


Graph 5.4: Fitness profiles for n-queens problem with $n = 25$.

AMPSO and all its variants in all cases. While the bad performance of the AMPSO algorithms on this problem is now understood, explaining why BPSO performs relatively well remains a difficult task. This is, in part, due to the fact that neither the PSO theory used, nor any of the analyses performed in this dissertation apply to BPSO, as was discussed at length in Section 2.3.2. A thorough theoretical investigation of the BPSO algorithm is beyond the scope of this work.



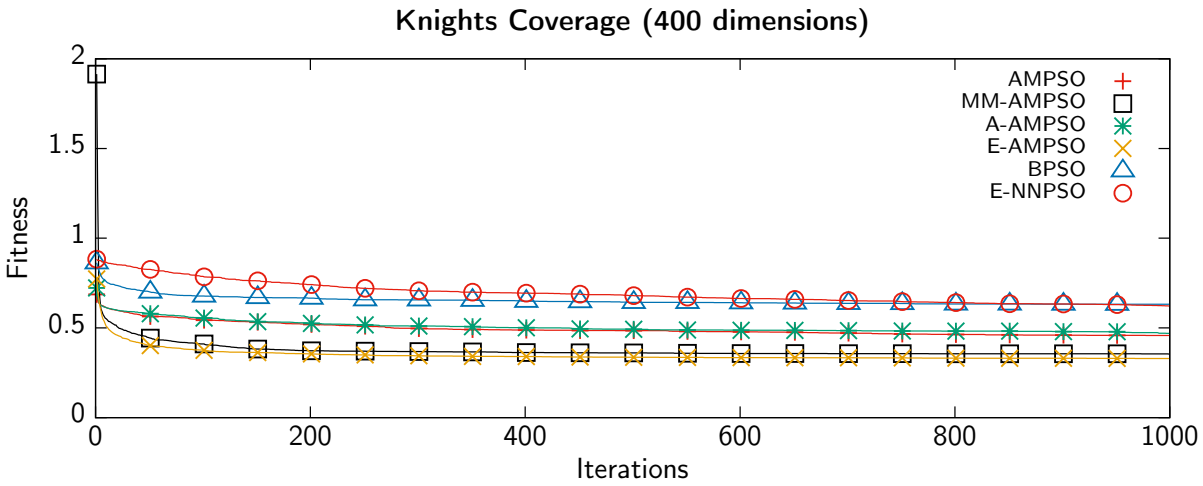
Graph 5.5: Fitness profiles for knights' coverage with $n = 10$.



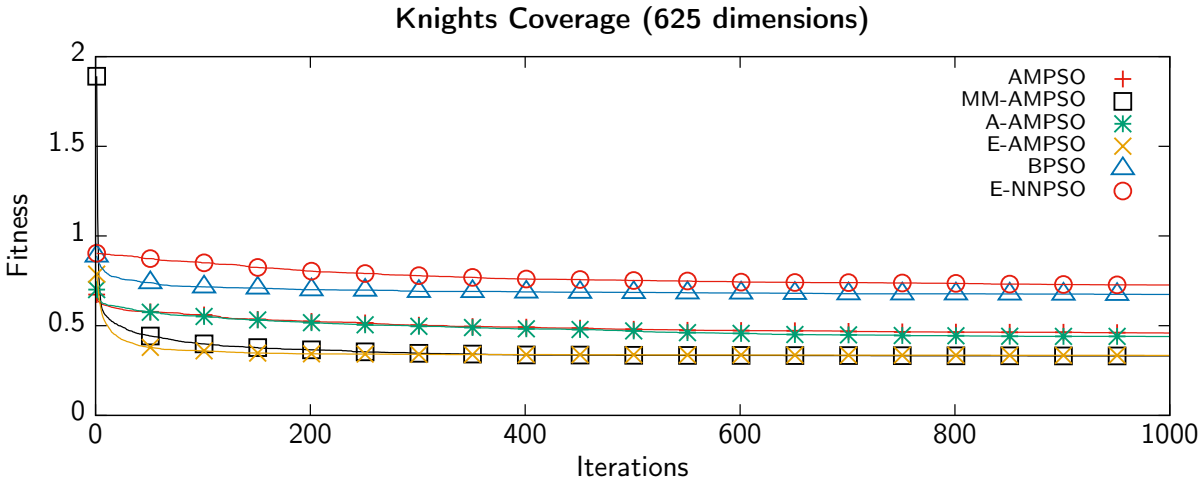
Graph 5.6: Fitness profiles for knights' coverage with $n = 15$.

5.4.2 Particle Velocities in E-NNPSO

The typical average particle velocity profiles for E-NNPSO are depicted in Graphs 5.14 to 5.16. As was the case when studying particle velocities in AMPSO in Section 4.3, each velocity profile shown here is the behaviour of a single execution of the algorithm. However, the results reflect the behaviour observed across multiple executions. Furthermore, the results shown here were all recorded on the n-queens problem. The reason no other problems were considered here is that the n-queens problem is the only prob-



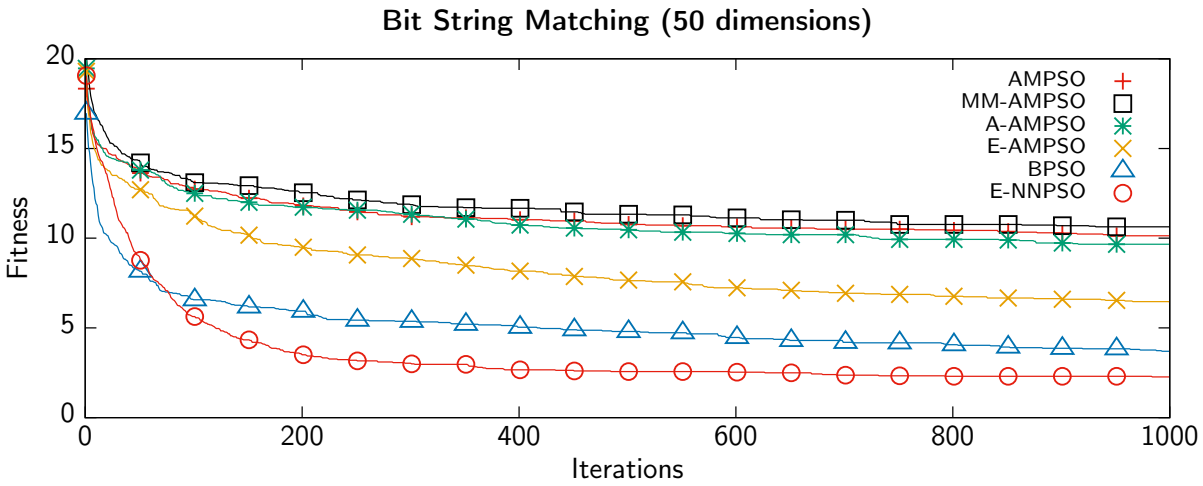
Graph 5.7: Fitness profiles for knights' coverage with $n = 20$.



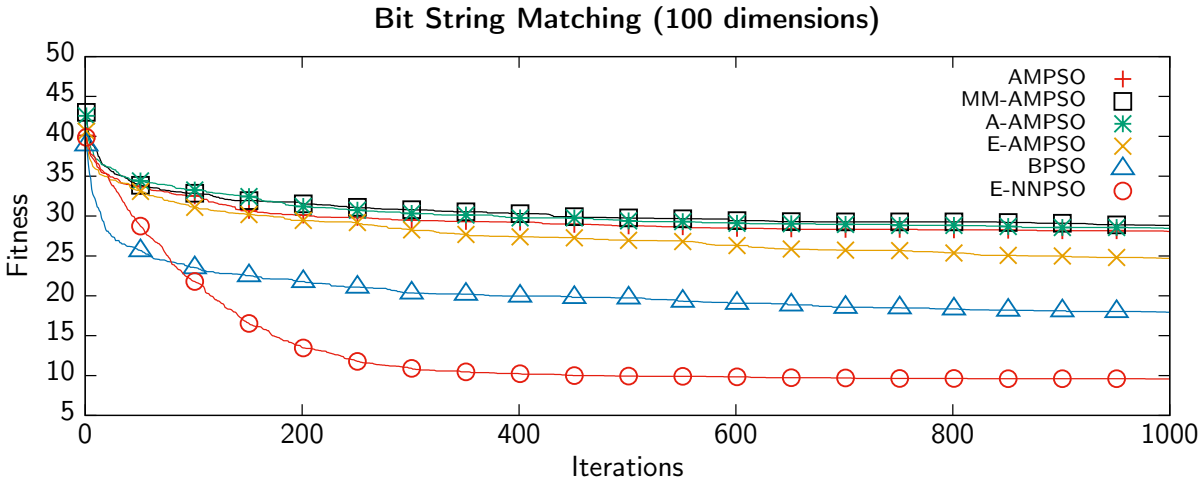
Graph 5.8: Fitness profiles for knights' coverage with $n = 25$.

lem whose solution representation was studied and rectified in this dissertation, such that it does not violate any known assumptions made by the PSO algorithm (see Section 4.6). Such violations may affect the behaviour of particle velocities, and could affect the interpretation of the results significantly.

The first noteworthy observation is that, in general, there is a noticeable downwards trend in the velocities of particles as the search progresses. This is a good result, indicating that particles' personal best positions and the global best position gradually move

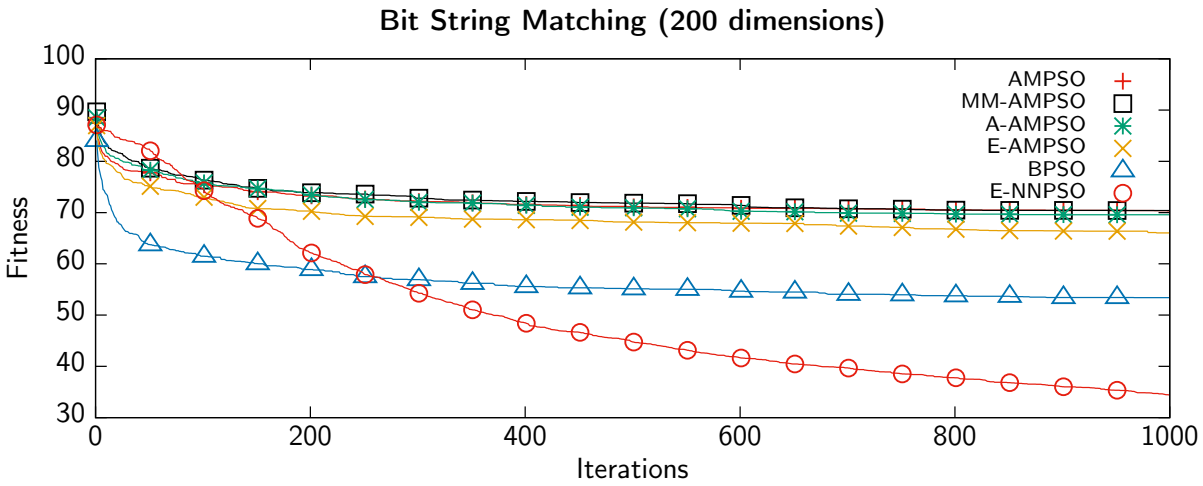


Graph 5.9: Fitness profiles for Random Bit String Matching with $n_b = 50$.

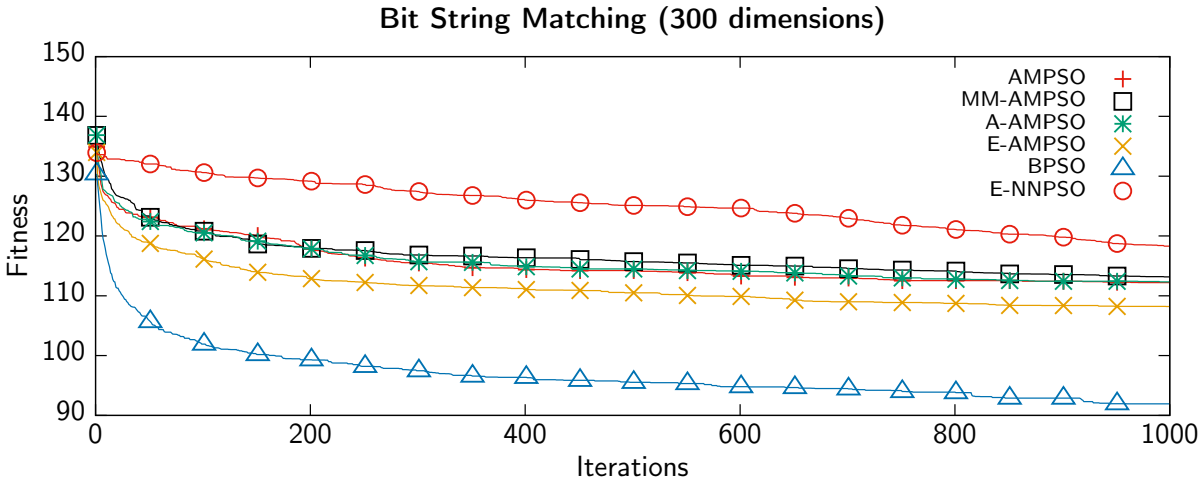


Graph 5.10: Fitness profiles for Random Bit String Matching with $n_b = 100$.

closer together throughout the search process. This scenario is shown in Graph 5.14. Additionally, Graphs 5.15 and 5.16 show that jumps in the average magnitude of particle velocities do occur in E-NNPSO, but that the velocities generally recover to lower magnitudes after these events occur. In contrast, in the case of AMPSO, particle velocities would typically stabilise at higher average magnitudes after similar events occurred. In AMPSO, that behaviour was attributed to the problem of spatial disconnect (see Section 4.2). Spatial disconnect seems to be eliminated in the case of E-NNPSO, which

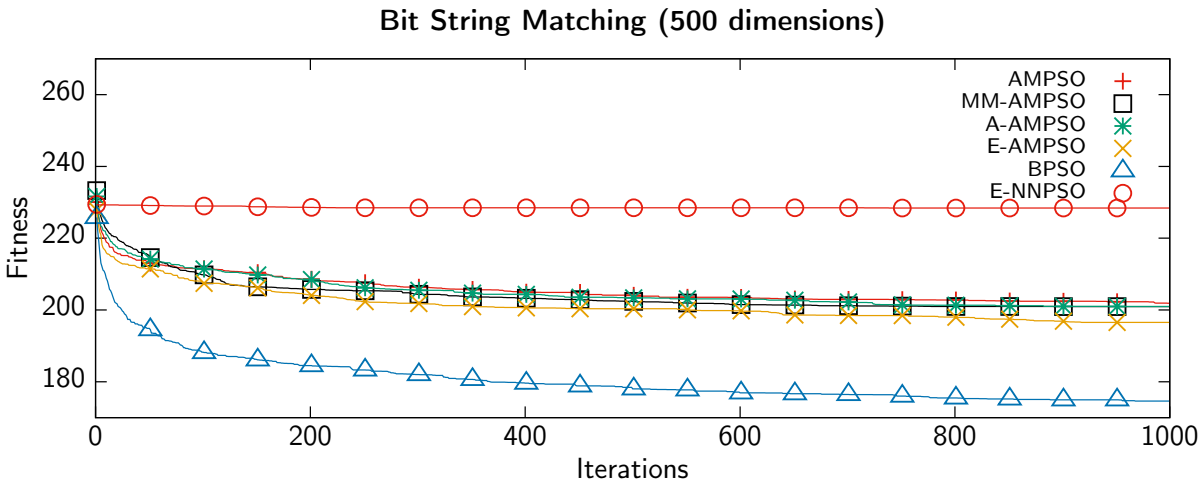


Graph 5.11: Fitness profiles for Random Bit String Matching with $n_b = 200$.

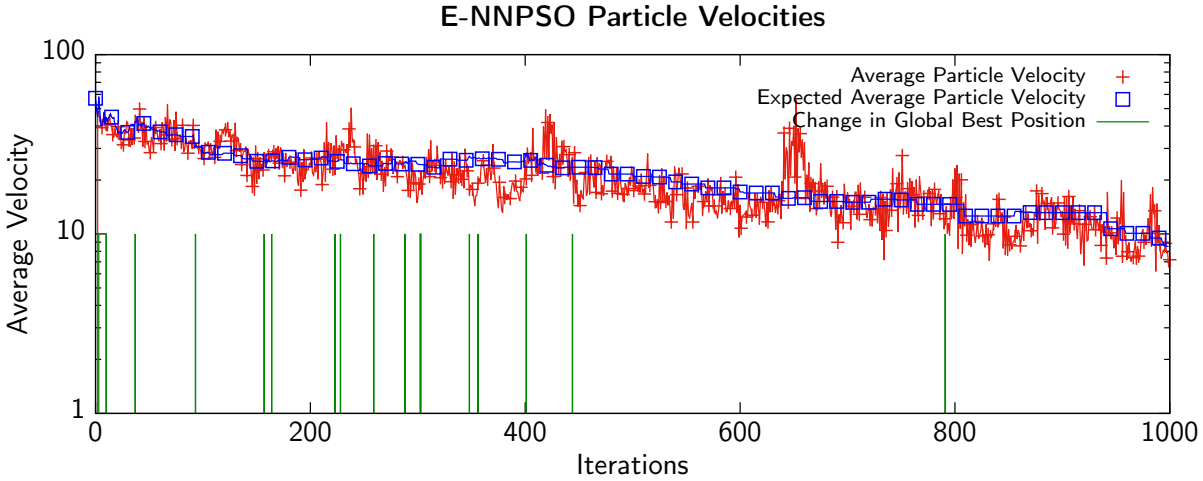


Graph 5.12: Fitness profiles for Random Bit String Matching with $n_b = 300$.

makes sense, because the problem is known to be caused by the generating function, and no such function exists in E-NNPSO. One could well argue that the mapping from grey code to binary is itself a generating function, but then it is necessary to concede that the nature of this mapping is fundamentally different from that of the generating function in AMPSO. Specifically, increasing the dimensionality n_b of the binary problem also increases the size of the E-NNPSO search space. Therefore, new binary solutions occupy new locations in the search space. In contrast, the AMPSO generating function



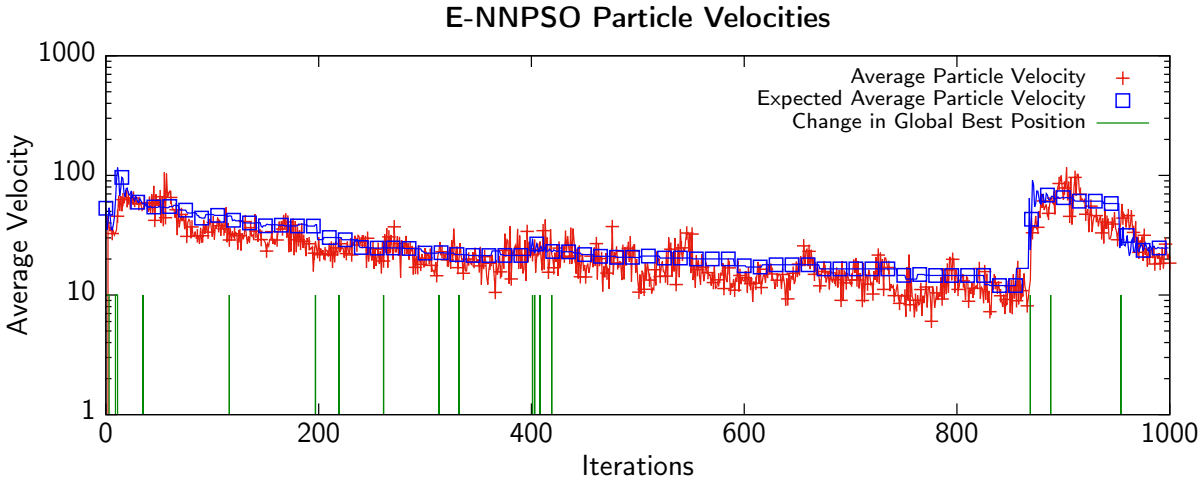
Graph 5.13: Fitness profiles for Random Bit String Matching with $n_b = 500$.



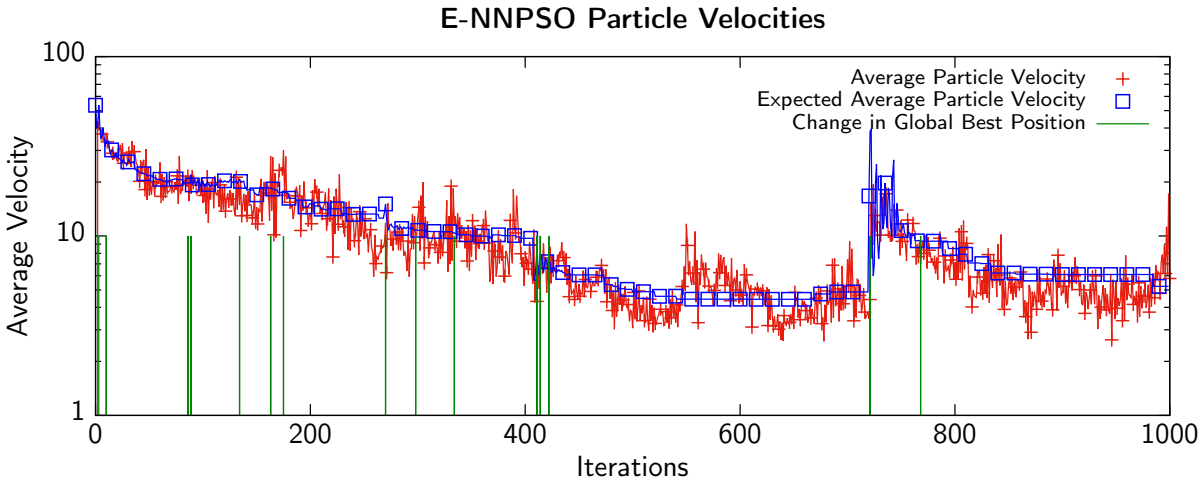
Graph 5.14: Particle velocities gradually decrease over the course of the search process

would merely be sampled at additional sampling points to generate a higher-dimensional binary solution, and the increasing frequency of the generating function is what causes the problem.

However, while the scenario depicted in Graph 5.17 was extremely rare, it did occur in a handful of test cases. In this situation, the velocities of particles gradually decreased during the first 550 iterations. At this point, a change in the global best position caused particle velocities to increase, and subsequently the swarm failed to recover back to lower

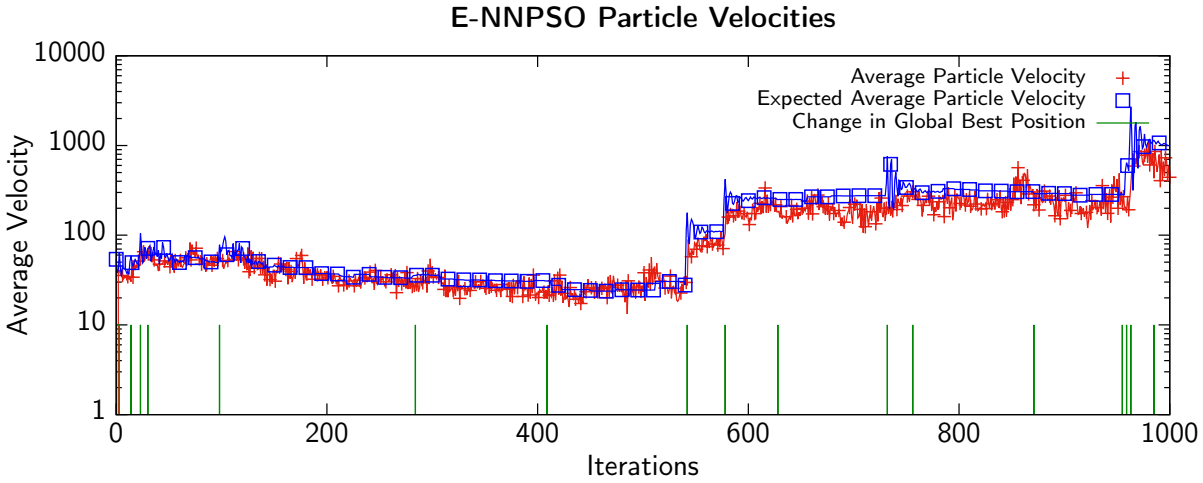


Graph 5.15: Particle velocities generally decrease. When jumps in the average magnitude of particle velocities occur, the swarm subsequently recovers to lower velocities again.



Graph 5.16: Particle velocities generally decrease. When jumps in the average magnitude of particle velocities occur, the swarm subsequently recovers to lower velocities again.

particle velocities. Note that the average magnitude of particle velocities eventually reached a value of 1000, which is much higher than what was observed in the case of AMPSO. Given that the nature of the E-NNPSO algorithm is not that different from the standard PSO algorithm, this result may suggest that such cases also occur in PSO in general, albeit probably not very frequently. While understanding exactly why this result occurred would be fascinating, such an investigation falls outside the scope of this



Graph 5.17: Particle velocities initially decrease, but a destabilising change in the global best position eventually causes velocities to increase. After this initial increase, the swarm is not able to recover, and stabilises at higher velocities instead.

dissertation.

5.4.3 Conclusion

Given the results discussed above, one can reasonably conclude that further research into the E-NNPSO is justified, since it outperformed AMPSO and all of its variants on a large number of test cases. The E-AMPSO algorithm, while obtaining better average fitness values than all the other AMPSO algorithms, generally lagged behind E-NNPSO or BPSO or both. The exception to this was the performance of E-AMPSO on the knights' coverage problem, but only because solutions to that problem happens to contain a lot of repetition. Therefore, it seems natural to consider E-AMPSO to be merely another AMPSO variant. Further justification for this choice is that E-AMPSO still suffers from spatial disconnect, albeit to a lesser extent. The only real advantage that E-AMPSO has over other AMPSO variants is that the potential of the ensemble generating function is known to be 1, as long as the binary problem is divided into parts of length 16 or less. However, even in those cases, E-AMPSO failed to produce the best results.

In addition to further studying E-NNPSO, it would be useful to compare the results to particle velocities in the standard PSO algorithm. If the scenario shown if Graph 5.17

also occurs in the standard PSO algorithm when applied to continuous problems, there are likely underlying reasons for the effect that do not have anything to do with the application of PSO to solve binary problems. Alternatively, if no evidence can be found of this effect in the continuous case, then at least the cause can be attributed to the mapping from the continuous search space to the binary solution space. In either case, further research is needed to understand the cause.

The use of new generating functions was deemed superfluous for the purpose of this work, and indeed was not even considered in this experimental chapter. The main reason is that different trigonometric functions would almost certainly also cause spatial disconnect in AMPSO. The only way to prevent spatial disconnect is to use a periodic function, which would cause repeating patterns in the generated binary solutions. Therefore, a new generating function that does not have either of these problems — if such a function exists — would likely not be trigonometric. Chaotic maps come to mind, and might be worth investigating, but such a venture was deemed beyond the scope of this work. Nonetheless, if further research in this direction is undertaken, the potential of any new generating function can — and should — be evaluated, using the work presented in Section 4.4 as a guide.

5.5 Summary

This chapter presented an empirical investigation to evaluate the merits of some of the recommendations that were made throughout this dissertation. The algorithms considered in this investigation were BPSO, AMPSO, A-AMPSO, MM-AMPSO, E-AMPSO, and E-NNPSO. Each of these algorithms were tested on the n-queens, knights' coverage, and random bit string matching problems in various dimensions. It was observed that E-NNPSO obtained the best average fitness in the majority of test cases. In the case of the n-queens problem, the success of E-NNPSO was attributed to the algorithm's uniform candidate solution distribution, as opposed to AMPSO and its variants, whose solution spaces are dominated by binary solutions that contain repetition. For the knights' coverage problems, AMPSO or one of the AMPSO variants generally outperformed all other algorithms. It was shown that the optimal solutions to the knights' coverage problem

contain a lot of repetition, and therefore AMPSO and its variants can generally find those solutions easily.

In addition to measuring performance, the velocities of particles in E-NNPSO were also studied. Observations showed that, in general, velocities in E-NNPSO tend to decrease during the search process. This result was deemed an improvement over AMPSO and was explained by the fact that E-NNPSO lacks a generating function, which is the cause of spatial disconnect in AMPSO. It was concluded that further research into the performance of E-NNPSO is justified.

Further research into the use of ensemble generating functions, or replacing g with new generating functions in AMPSO was discouraged, because of the difficulty of conceiving a generating function that is not periodic, and at the same time would not cause spatial disconnect. However, it was noted that non-trigonometric functions may provide a solution, and that the work on generating function potential that was presented in this dissertation should be applied if any such research is pursued.

Derived publications

- Parts of the empirical analysis presented in this chapter was published in [27].

Chapter 6

Conclusions

> *We can judge our progress by the courage of our questions and the depth of our answers, our willingness to embrace what is true rather than what feels good.*
>
> *Carl Sagan*

This chapter gives an overview of the conclusions arrived at in this dissertation. For ease of reference the research objectives given at the outset of this work are reiterated in Section 6.1. A summary of the novel contributions made by this work is presented in Section 6.2, while the experimental findings are outlined in Section 6.3. Finally, potential future research topics are listed in Section 6.4.

6.1 Research Objectives

The research objectives are as follows:

- To provide evidence of cases where the AMPSO algorithm fails and/or where the behaviour of the algorithm cannot readily be explained based on current knowledge.
- To identify a range of potential underlying causes of failure.
- To investigate the identified aspects theoretically and/or empirically, as deemed appropriate.

- To provide recommendations on how any identified limitations of the AMPSO algorithm could be overcome.
- To provide empirical analyses of the most promising recommendations.

6.2 Summary of Contributions

The first contribution made in this dissertation was to introduce three variants of the AMPSO algorithm that were intended to overcome a number of perceived limitations of the original AMPSO algorithm. It was shown in Chapter 3 that the new variants achieved statistically significant improvements over AMPSO on a number of benchmark problems. However, it was noted that the perceived limitations were presented in the absence of any empirical and/or theoretical evidence, and that the performance gains made by some of the variants could potentially be better understood, given a deeper understanding of the inner workings of the AMPSO algorithm. To this end, a critical analysis of AMPSO was presented in Chapter 4.

The analysis of AMPSO started out by investigating the periodicity of the generating function. This was deemed important, because a periodic generating function could potentially produce binary solutions that contain repeating patterns. It was shown that the standard AMPSO generating function is not periodic. However, the proof uncovered an alarming characteristic of the generating function: that the frequency of the function increases indefinitely as $|x| \to \infty$.

Arguably the most important contribution made by this work was the discovery in Section 4.2 that the increasing frequency of the generating function gives rise to a problem dubbed spatial disconnect. Essentially, spatial disconnect implies that small step sizes in the coefficient space do not translate into small step sizes in the solution space. Specifically, in the case of AMPSO, it was shown that bits in the lower dimensions of the binary problem stabilise first, and that high dimensional bits only stabilise at much lower particle velocities. The consequence of spatial disconnect is that AMPSO cannot effectively exploit the binary solution space.

During the investigation of spatial disconnect, the question arose whether particle velocities in AMPSO eventually slow down enough that high-dimensional bits in the

binary solution can also be exploited. What followed in Section 4.3 was the first empirical investigation into the convergence behaviour of particles in AMPSO. It was shown that particles in AMPSO tend to converge at high velocities relative to the size of the search space. Additionally, it was observed that the velocities at which the swarms converged were generally too high for even the lower-dimensional bits in the binary solution to stabilise. This was the second most important contribution made in this dissertation.

Because spatial disconnect was now known to be caused by the generating function, the next question was whether a different generating function could be used in order to prevent spatial disconnect from manifesting. Section 4.4 addressed the problem of finding a suitable generating function. It was noted that, thus far, it was not even known whether the current generating function could generate any arbitrary binary solution. To find out, the concept of generating function potential was developed. The new definition was then applied to introduce a novel generating function for use with low-dimensional binary problems. In order to solve arbitrarily high-dimensional binary problems with AMPSO, the concept of ensemble generating functions was introduced in Section 4.4.4.2.

Another novel contribution was to show in Section 4.5 that candidate binary solutions do not appear in the binary solution space with uniform frequency in AMPSO. The reason is that the generating function produced non-uniform mapping from the coefficient space to the solution space, and consequently some solutions would exist with a much higher frequency than others. As a solution to this problem, a novel PSO variant was introduced. The variant was named NNPSO, and makes use of natural numbers to produce a uniform solution frequency distribution.

The final contribution was to investigate how the binary solution representation could affect the performance of AMPSO. It was shown that poorly constructed binary representations could violate the assumptions made by the PSO algorithm. As a consequence, the ability to exploit the solution space is hindered. To provide an example of how this problem can be overcome, a new solution representation for the n-queens problem was developed in Section 4.6.2.

6.3 Summary of Experimental Findings

An empirical analysis was performed in Chapter 5. The purpose of the analysis was to support the most promising recommendations made throughout the dissertation with initial empirical evidence. The BPSO, AMPSO, A-AMPSO, MM-AMPSO, E-AMPSO, and E-NNPSO algorithms were compared on three benchmark problems. The three problems were n-queens, knights' coverage, and random bit string matching, all with various dimensionalities. It was observed that E-NNPSO obtained the best average fitness values on the n-queens and bit string matching problems, with the exception of high-dimensional bit string matching problems. The success of E-NNPSO on these problem cases were attributed to the lack of a generating function that could cause spatial disconnect. On the knights' coverage problems, the AMPSO algorithms generally obtained the best average fitness values. This was explained by the fact that optimal solutions to the knights' coverage problem contain a lot of repetition. It was shown in Section 4.5.1 that solutions containing repetition dominate the solution space in AMPSO, thus making these solutions relatively easy to find.

The velocities of particles in the E-NNPSO algorithm were also investigated. It was shown that particle velocities tend to decrease during the search process. This result, combined with the good performance of E-NNPSO led to the conclusion that future research into E-NNPSO is justified.

The use of ensemble generating functions in AMPSO did not provide a sufficient increase in performance to justify much further research. Similarly, an investigation into alternative generating functions was not performed, because of the difficulty in designing a function that does not cause spatial disconnect, while also not producing repeating patterns in generated binary solutions. Furthermore, the non-uniform solution frequency distribution is also caused by the generating function, and would manifest for any function that does not guarantee that an equal number of permutations of its coefficients will map to each possible binary solution.

6.4 Future Work

The following is a list of potential future studies that could follow from the work presented in this dissertation:

- A thorough investigation of the magnitude of velocities at the time of convergence in PSO could shed some light on the convergence behaviour of E-NNPSO.
- An investigation into the solution representations of a wider range of binary benchmark problems is required. This study should determine whether the solution representations violate any of the assumptions made by the PSO algorithm, and rectify any such cases before any PSO-based algorithm is applied the problems.
- An analysis of E-NNPSO on a wider range of binary benchmark problems is required.
- The use of non-trigonometric generating functions can be investigated. Care should be taken to avoid any of the detrimental characteristics of the standard generating function that were discovered in this dissertation.

Bibliography

[1] P. Balaprakash, M. Birattari, and T. Stützle. Improvement strategies for the f-race algorithm: Sampling design and iterative refinement. In *Hybrid Metaheuristics*, pages 108–122. Springer, 2007.

[2] M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Genetic and Evolutionary Computation Conference*, pages 11–18, 2002.

[3] Gordon E Carlson. *Signal and linear system analysis*. John Wiley, 1998.

[4] C.W. Cleghorn and A.P. Engelbrecht. A generalized theoretical deterministic particle swarm model. *Swarm Intelligence*, 8(1):35–59, 2014.

[5] M. Clerc. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 3, pages 1951–1957. IEEE, 1999.

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

[7] T. Cloete, A.P. Engelbrecht, and G. Pamparà. Cilib: A collaborative framework for computational intelligence algorithms - part II. In *IEEE International Joint Conference on Neural Networks*, pages 1764–1773, 2008.

[8] S. Dirakkhunakon and Y. Suansook. Simulated annealing with iterative improvement. In *International Conference on Signal Processing Systems*, pages 302–306, 2009.

[9] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science*, volume 1, pages 39–43, 1995.

[10] R. Eberhart, P. Simpson, and R. Dobbins. *Computational intelligence PC tools*. Academic Press Professional, Inc., 1996.

[11] R.C. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the Congress on Evolutionary Computation*, volume 1, pages 84–88. IEEE, 2000.

[12] A.P. Engelbrecht. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2006.

[13] A.P. Engelbrecht. *Computational intelligence: an introduction*. John Wiley & Sons, 2007.

[14] A.P. Engelbrecht. Particle swarm optimization: Velocity initialization. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2012.

[15] A.P. Engelbrecht. Roaming behavior of unconstrained particles. In *Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 104–111. IEEE, 2013.

[16] A.P. Engelbrecht and G. Pamparà. Binary differential evolution strategies. In *IEEE Congress on Evolutionary Computation*, pages 1942–1947. IEEE, 2007.

[17] D. Fisher. On the nxn knight cover problem. *Ars Combinatoria*, 69:255–274, 2003.

[18] N. Franken. Pso-based coevolutionary game learning. Master's thesis, University of Pretoria, 2004.

[19] D. Goldberg. Simple genetic algorithms and the minimal, deceptive problem. *Genetic algorithms and simulated annealing*, page 88, 1987.

[20] V. Gordon and T. Slocum. The knight's tour - evolutionary vs. depth-first search. In *IEEE Congress on Evolutionary Computation*, volume 2, pages 1435–1440, 2004.

[21] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, 1995.

[22] J. Kennedy and R. Eberhart. A discrete binary version of the particle swarm algorithm. In *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, volume 5, pages 4104–4108. IEEE, 1997.

[23] James Kennedy, James F Kennedy, and Russell C Eberhart. *Swarm intelligence*. Morgan Kaufmann, 2001.

[24] M.A. Khanesar, M. Teshnehlab, and M.A. Shoorehdeli. A novel binary particle swarm optimization. In *Mediterranean Conference on Control & Automation*, pages 1–6. IEEE, 2007.

[25] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simmulated annealing. *Science*, 220(4598):671–680, 1983.

[26] B.J. Leonard and A.P. Engelbrecht. Angle modulated particle swarm variants. In *Swarm Intelligence*, volume 8667, pages 38–49, 2014.

[27] B.J. Leonard and A.P. Engelbrecht. Frequency distribution of candidate solutions in angle modulated particle swarms. In *IEEE Symposium on Swarm Intelligence*, pages 251–258, 2015.

[28] B.J. Leonard, A.P. Engelbrecht, and C.W. Cleghorn. Critical considerations on angle modulated particle swarm optimisers. *Swarm Intelligence*, pages 291–314, 2015.

[29] I. Martinjak and M. Golub. Comparison of heuristic algorithms for the n-queen problem. In *29th International Conference on Information Technology Interfaces.*, pages 759–764, 2007.

[30] G. Pamparà. Angle modulated population based algorithms to solve binary problems. Master's thesis, University of Pretoria, 2013.

[31] G. Pamparà, A. Engelbrecht, and N. Franken. Binary differential evolution. In *IEEE Congress on Evolutionary Computation*, pages 1873–1879, 2006.

[32] G. Pamparà and A.P. Engelbrecht. Binary artificial bee colony optimization. In *IEEE Symposium on Swarm Intelligence (SIS)*, pages 1–8. IEEE, 2011.

[33] G. Pamparà, A.P. Engelbrecht, and T. Cloete. Cilib: A collaborative framework for computational intelligence algorithms – part I. In *Proceedings of the International Joint Conference on Neural Networks*, pages 1750–1757. IEEE, 2008.

[34] G. Pamparà, N. Franken, and A.P. Engelbrecht. Combining particle swarm optimisation with angle modulation to solve binary problems. In *IEEE Congress on Evolutionary Computation, 2005*, volume 1, pages 89–96, 2005.

[35] R. Poli. Mean and variance of the sampling distribution of particle swarm optimizers during stagnation. *IEEE Transactions on Evolutionary Computation*, 13(4):712–721, 2009.

[36] John G Proakis and Masoud Salehi. *Communication systems engineering*, volume 2.

[37] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 69–73. IEEE, 2002.

[38] A.M. Turky and A. Ahmad. Using genetic algorithm for solving n-queens problem. In *International Symposium in Information Technology*, volume 2, pages 745–747, 2010.

[39] F. Van den Bergh and A.P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences*, 176(8):937–971, 2006.

[40] Jakob Vesterstrøm and René Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Congress on Evolutionary Computation Evolutionary Computation*, volume 2, pages 1980–1987. IEEE, 2004.

Appendix A

Acronyms

This appendix lists all the acronyms used in this dissertation. Acronyms are listed alphabetically. Each acronym is typeset in bold, with the meaning of the acronym printed alongside it in normal font.

A-AMPSO	Amplitude AMPSO
ABC	Artificial Bee Colony Optimisation
AMPSO	Angle Modulated Particle Swarm Optimisation
BPSO	Binary Particle Swarm Optimisation
CIlib	Computational Intelligence Library
DE	Differential Evolution
E-AMPSO	Ensemble AMPSO
E-NNPSO	Ensemble NNPSO
GA	Genetic Algorithm
ID-AMPSO	Increased-Domain AMPSO
MM-AMPSO	Min-Max AMPSO
NNPSO	Natural Numbers PSO

PSO	Particle Swarm Optimisation
SA	Simulated Annealing

Appendix B

Symbols

This appendix provides an exhaustive list of symbols used throughout this dissertation. Symbols are listed by chapter and sorted alphabetically. Where applicable, Roman symbols are listed first, followed by Greek symbols. In cases where symbols are re-used, they are re-defined under the relevant chapters.

B.1 Chapter 2: Optimisation Problems and Particle Swarm Optimisers

$\mathbb{B}$	The set of binary numbers
c	Sum of the social- and cognitive acceleration coefficients
c_1	Cognitive acceleration coefficient
c_2	Social acceleration coefficient
$\mathbb{D}$	A discrete subset of $\mathbb{R}$
$E[\bullet]$	An arbitrary expected value
$\mathcal{F}$	Feasible space
f	Objective function
i	Particle index
j	Dimension index
n_s	Swarm size

n_x	Dimensionality of a continuous optimisation problem
$\boldsymbol{r}$	Random vector
$\mathbb{R}$	The set of real numbers
$\mathcal{S}$	Search space
t	A specific time step during the search process
v_{avg}	Average magnitude of particle velocities
$\boldsymbol{v}_i$	Velocity of particle i
$\boldsymbol{x}$	Input to an objective function
$\boldsymbol{x}_i$	Position of particle i
$\hat{\boldsymbol{y}}$	Global best position
$\boldsymbol{y}_i$	Personal best position of particle i
$\boldsymbol{\mu}$	Mean of a particle's positions
σ	Standard deviation

B.2 Chapter 3: Angle Modulated Particle Swarm Optimisation

a, b, c, d, e	Coefficients of the AMPSO generating function
$\mathcal{B}$	Binary solution
g	AMPSO generating function
n_b	Dimensionality of a binary optimisation problem
α	Coefficient of the AMPSO generating function
α_ℓ	Lower bound
α_u	Upper bound
δ	Sampling interval

B.3 Chapter 4: Critical Considerations on Angle Modulated Particle Swarm Optimisers

b_i	n_b-dimensional bit string
C	Number of coefficients
h	Generating function
I	Interval
J_k, K_k	Sequences of intervals
k	Arbitrary positive natural number
$\mathbb{N}^0$	Set of natural numbers, including 0
$\mathbb{N}^+$	Set of positive natural numbers
$\mathcal{O}(\bullet)$	Big O notation
P	Generating function potential
$\mathbb{P}$	Set of permutations
$\mathbf{p}_i$	Random vector
s	Step size
$\mathbf{s}$	Uniform random vector with length s
T	Period of a continuous function
v	Frequency coefficient
$\mathbb{Z}$	Set of integers
Θ	Ensemble generating function
$\Pi(\bullet)$	Number of roots of g in an arbitrary interval
Υ	Arbitrary generating function
φ	Number of generating functions in Θ
ψ	Coefficient of an ensemble generating function

Appendix C

Derived Publications

The following three publications were derived from the work presented in this dissertation:

- B.J. Leonard and A.P. Engelbrecht. Angle modulated particle swarm variants. In *Swarm Intelligence*, volume 8667, pages 38–49, 2014.
- B.J. Leonard, A.P. Engelbrecht, and C.W. Cleghorn. Critical considerations on angle modulated particle swarm optimisers. *Swarm Intelligence*, pages 291–314, 2015.
- B.J. Leonard and A.P. Engelbrecht. Frequency distribution of candidate solutions in angle modulated particle swarms. In *IEEE Symposium on Swarm Intelligence*, pages 251–258, 2015.