SWARM OPTIMIZATION USING AGENTS MODELED AS DISTRIBUTIONS

By
Nathan John Bell

A thesis submitted to
the Faculty of Graduate and Postdoctoral Affairs
in partial fulfilment of
the requirements for the degree of
Master of Computer Science

M.C.S. Computer Science
Carleton University
Ottawa, Ontario

September 2014

ABSTRACT

Particle Swarm Optimization (PSO) is a popular meta-heuristic for black-box optimization. Many variations and extensions of PSO have been developed since its creation in 1995, and the algorithm remains a popular topic of research. In this work we explore a new, abstracted, perspective of the PSO system and present the novel Particle Field Optimization (PFO) algorithm which harnesses this new perspective to achieve a behaviour distinct from traditional PSO systems.

ACKNOWLEDGEMENTS

Firstly I would like to thank my parents for their continued support and encouragement throughout the years.

I would also like to thank Dr. John Oommen for his guidance and advice during the writing of this thesis. His experience and insight was invaluable throughout all stages of research and writing.

Contents

List of Figures

ACRONYMS

SI Swarm Intelligence

PSO Particle Swarm Optimization

FIPS Fully Informed Particle Swarm

BBPS Bare Bones Particle Swarm

PFO Particle Field Optimization

Chapter 1

Introduction

1.1 Introduction

The efficient use of resources is a fundamental problem which permeates every aspect of our lives. Nearly every decision we make can be interpreted, in some way, to involve the concept of minimizing the "cost", or maximizing the "returns" of our actions. In fact, this concept can be considered to be a core challenge of life of any kind, as resources and energy are inherently limited, and must be used efficiently in order to ensure long-term survival and prosperity. Any form of "intelligence", then, *must* have the ability to deal with problems of this type.

As humans, we possess a strong natural ability to work with these problems. However, as we develop and progress, we are faced with increasingly complex problems which go far beyond our limitations. As a result, finding new tools and methods which can be applied to these problems is important for our continued advancement.

The field of function optimization presents a formalized framework for modelling and solving certain problems of this type. Given an "objective" function, which takes a number of parameters as its input, the goal is to find the combination of parameter values which returns the "best" value. This framework is abstract enough that a wide variety of different problems can be interpreted as being "function optimization" problems.

Due to this flexibility, the general problem of function optimization finds applications in an ensemble of fields, both practical and theoretical. The most common practical applications can be found in areas such as engineering design, bioinformatics and economics, where determining the optimal input to complex functions is a common problem. This is also true for many other less obvious fields, due to the flexibility of the framework.

The objective function of an optimization problem can be any function which maps any number of parameters to a single final resultant value. Because of this rather non-specific definition, we see that there are an extremely wide variety of distinct properties that these functions can have. This variety of properties presents a significant challenge when approaching function optimization. Many properties, such as discontinuities, multi-modal complexity, and elaborate constraints on the input parameter values, render "standard" optimization methods impossible or infeasible, and so currently, new methods must be created and tailored to functions with specific properties. Currently, there exist, a wide variety of methods for solving an ensemble of problems which can be classified as optimization problems. The majority of these optimization methods require that the properties of the function are known, and the function itself to be known, in detail. However, there are many cases when this information may be unavailable, or the function itself may be totally unknown.

"Black-box" optimization is a category of optimization problems dealing with functions about which no information is available. These unknown functions present a unique challenge for optimization. Without any knowledge of the function itself, algebraic manipulations (such as computing the derivatives) are impossible. As a result of this, traditional analytical methods of discovering the optimum, no longer apply. In fact, it is unknown whether or not a single global optimum exists at all, and it is impossible to verify whether any solution found is globally optimal. The task of optimization, then, must be approached from a different perspective. Finding the optimal value becomes a search process, searching through an unknown, and often infinite, solution space.

When dealing with black-box optimization, the only way of gaining information about the objective function is to generate, and evaluate potential solutions. By

observing the results of these function evaluations, some limited information can be deduced about the function, which can then be used to guide the search process. The most straight-forward approach to this would be a basic, deterministic hill-climbing algorithm. A hill climbing algorithm simply generates a number of "adjacent" solutions to the current best-found solution, and updates that best-found solution if one of the adjacent solutions is better. The process then loops until no better adjacent solution is discovered. This approach will quickly guarantee some *locally* optimal solution, but is not well suited to the problem of black-box optimization, in general. The majority of real-life applications involve functions with very complex, multi-modal solution spaces, and since nothing is known about the function being optimized, it must be assumed that these complexities are present. On functions such as these, a hill-climbing algorithm will perform poorly compared to other methods, exploiting the local information but becoming stuck in local optima.

In order to perform well on unknown functions, it is essential to avoid stagnating or converging to a suboptimal point. If we consider the hill-climbing algorithm, the opposite extreme would be an algorithm which simply generates, and evaluates, totally random solutions within the solution space. Clearly, this strategy would not become stuck in local optima as the hill-climbing strategy would. However, simply generating random solutions is no better than one which sequentially generates and tests every possible solution, and so this strategy is not feasible either, due to the existence of a potentially infinite number of points in the solution space.

We thus argue that neither the extreme *exploitation* of a hill-climber, nor the extreme *exploration* of generating totally random solutions, are suitable for the problem of black-box optimization. To effectively, and efficiently, search the unknown space of a black-box function, an algorithm must strike a *balance* between these two extremes. *Exploitation* is necessary to find the locally optimal solution in some general area, and *exploration* is necessary to escape local optima and locate more areas for potential exploitation. This strategy of balancing exploitation with random exploration describes a category of algorithms known as "Meta-Heuristics".

The term "meta-heuristic" is a combination of the modifier "meta" ("beyond"), and the word "heuristic" ("proceeding to a solution by trial-and-error or by rules that are

only loosely defined"). In general, a meta-heuristic algorithm is one which combines randomization with a higher-level guiding process to perform a search of some kind. This "higher guiding process" is what differentiates a meta-heuristic method from a *simple* heuristic method. The heuristic process within a meta-heuristic is modified by this guiding process, thus elevating the behaviour to be *beyond* a simple heuristic.

This definition of "Meta-Heuristic" describes a very broad, and vaguely-defined, categorization of search techniques and optimization algorithms. In a general sense, any algorithm or process which uses some sort of randomization with a higher level mechanism guiding it, can be considered to be a meta-heuristic. As a result of this, the variation among meta-heuristic algorithms is extremely large. Historically, one of the first meta-heuristic methods is the famous Simulated Annealing algorithm, which combined hill-climbing behaviour with a degree of randomness which is slowly decreased over time.

When investigating for new strategies for solving problems, nature provides a constant source of inspiration. This is especially true in the case of meta-heuristic methods. Randomness is inherent in nature, and many natural systems display a very strong ability to adapt and succeed despite this inherent unpredictability. It is quite easy, then, to draw parallels between these adaptive behaviours and the concept of a search through an unknown space of some kind. Because of this, there exist many meta-heuristic search and optimization algorithms which take inspiration from, or attempt to model, these natural phenomena. In fact, nearly all nature-inspired algorithms can be classified as meta-heuristics, due to the inherent randomness and necessity for adaptive behaviour.

Even among the category of nature-inspired algorithms there exist a wide variety of different approaches. These can range from algorithms modelling the process of evolution, to algorithms modelling river formation, and further to those that model the behaviour of swarms of creatures. Each of these examples model very different phenomena, but can, in turn, be described as performing a meta-heuristic optimization of some kind. This exemplifies the flexibility and variety of the processes at work within meta-heuristic algorithms. This flexibility allows for a great deal of creativity when developing new meta-heuristic algorithms.

Of particular interest to us in this thesis, are those algorithms that model so-called "swarming" behaviour. In nature, there exist many examples of social creatures that cooperatively act together and display a behaviour which, as a whole, is beyond the individual capabilities of the creatures themselves. This phenomenon is known as "Swarm Intelligence" (SI), and provides a very interesting perspective for problem solving. In an SI system, a population of rudimentary or primitive "individuals" work and interact with one another in order to solve a higher-level problem beyond the capabilities of any single individual.

One such SI algorithm is the Particle Swarm Optimization (PSO) algorithm. Created in 1995, as the result of experiments modelling a "collision-less" bird-flocking swarm behaviour, the PSO algorithm has been a very popular subject of research and development. The PSO algorithm consists of a population of extremely basic "particle" individuals, each of which exist as a point in the solution space of the objective function. These particles "fly" through this space according to a velocity, which is updated at each time step by performing a set of accelerations with added randomness. By communicating with each other, the particles are able to balance exploratory and exploitative behaviour to find "good" solutions.

The PSO algorithm is very popular, both for practical applications, and for further research and development. This is because the individual behaviour of the particles in the swarm is very basic, and the power of the algorithm arises from these basic behaviours and their interactions. As a result, implementing the PSO algorithm is extremely simple, and many opportunities exist for modifications and additions. In particular, efforts have been made to abstract the PSO behaviour, in order to render the analysis of the high-level behaviour easier. The Bare Bones Particle Swarm (BBPS) algorithm presents one such abstraction, resulting in an even simpler algorithm that possesses an easy-to-understand high level behaviour. This abstracted algorithm displays an opportunity for further abstraction, which we seek to investigate.

1.2 Motivations and Objectives

The PSO algorithm is a very simple scheme possessing a very powerful ability to optimize difficult, multi-modal functions. Due to the simplicity and strength of the PSO algorithm, it has been a popular subject for new research and development with many variations and changes proposed to improve the algorithm or change its behaviour. As mentioned above, one such variation, the BBPS algorithm, presents an abstracted perspective of the PSO algorithm which further simplifies the underlying individual particle's behaviour, while maintaining characteristics analogous with PSO at the population level. Inherent to this BBPS algorithm resides an opportunity for *further* abstraction, which, to our knowledge, has not yet been investigated. By investigating this opportunity for further abstraction, we hope to discover a new avenue for the development for particle swarm algorithms, which, in turn, presents a unique perspective on particle swarm systems.

This thesis is written with two primary objectives in mind. Our first objective is to present the previously mentioned opportunity for abstraction present in the BBPS algorithm. By exploring this new abstraction, we seek to present a new perspective on the algorithm's behaviour and, with that, to submit a number of possibly new changes and variations.

Our second objective is to present a newly-created algorithm, which we refer to as the "Particle Field Optimization" (PFO) algorithm. This novel PFO algorithm is the result of applying a number of changes to the BBPS algorithm based on the new perspective presented. With these changes, our PFO algorithm possesses a rather distinct behaviour. With this new algorithm as the basis, we then seek to investigate the range of potentials possible, and the effects of the algorithm's parameters on the corresponding behaviour.

1.3 Organization of the Thesis Proposal

The following is a brief overview of the organization of the thesis, including the content and main contributions of each subsequent chapter.

Chapter 2: This chapter is concerned with providing, for the reader, a basic framework of knowledge required to understand, and contextualize, the contributions made in this thesis. In this chapter we introduce the concept of "Swarm Intelligence" (SI) and explore the state of the art of Particle Swarm Optimization algorithms. The history of SI is detailed, beginning with observations of swarm behaviour in nature and culminating in the development of a distinct, and powerful computing paradigm. Continuing from this, we introduce the Particle Swarm Optimization algorithm, beginning with its early incarnation as a modified flocking algorithm, and detailing its development into a powerful and popular meta-heuristic method. Along with the canonical PSO algorithm, the various parameters are also discussed in detail. We then explore the various research efforts surrounding PSO, including the analysis of the algorithm's behaviour, the discovery of biases that it displays, and the wide array of new developments made since the algorithm's conception. Finally, we discuss the standard methods for evaluating meta-heuristic optimization algorithms and the particular details of evaluating PSO-based schemes.

Chapter 3: This chapter, which presents our contributions, presents our newly proposed Particle Field Optimization (PFO) algorithm and the abstractions and the new perspectives it entails. Beginning with the BBPS algorithm, we identify an opportunity for a further abstraction of the process. Using this abstraction alluded to earlier, we present a newly-discovered perspective on the behaviour of the BBPS algorithm and the processes underlying its behaviour. This new perspective allows us to highlight unique and novel opportunities for modifying the search behaviour and the underlying principles at work. We propose a number of modifications to the BBPS algorithm taking advantage of these new opportunities, which culminate in the above-mentioned distinct PFO algorithm. The chapter includes a detailed and formal description of the PFO algorithm. Finally, we also propose a number of weighting schemes that can be applied to the PFO algorithm to modify the search behaviour.

Chapter 4: This chapter presents the results of the empirical testing carried out on the newly-developed PFO algorithm. We begin by outlining the suite of test functions used, which consists of a number of benchmark test functions with a number of dimensionalities. Following this, we highlight a description of the test strategy used.

This strategy details the various parameter values chosen for testing, and the different PFO configurations, and for each case, we list the combination of chosen parameter values. The motivations behind these parameter value choices is also presented here. **Chapter 5**: This chapter concludes the thesis and presents possible opportunities for future work.

Chapter 2

Survey of Field

2.1 Introduction

In this chapter we introduce the concept of Swarm Intelligence (SI) and explore the state of the art of Particle Swarm Optimization (PSO) algorithms. We detail the history of SI, beginning with observations of swarm behaviour in nature, and culminate in the development of a distinct, and powerful problem-solving paradigm. Continuing from this, we introduce the PSO algorithm, stating with its early incarnation as a modified flocking algorithm, and proceeding with its development into a powerful and popular meta-heuristic method. We describe the canonical PSO algorithm, and discuss, in detail, the various parameters. We then explore the various research efforts surrounding the PSO, including various attempts made to analyze the algorithm's behaviour, the discovery of the biases it displays, and the wide array of new developments made since the algorithm's conception. Finally, we discuss the standard methods for evaluating meta-heuristic optimization algorithms, and the particular details associated with quantifying the performance of PSO-based algorithms.

2.2 Swarm Intelligence

2.2.1 Decentralization, Self-Organization and Emergent Behaviour

In a general sense, "Swarm intelligence" (SI) is defined as the collective behaviour of decentralized, self organized systems. It is a phenomenon which has been observed and applied in a large variety of fields. The concepts motivating SI are very broad and somewhat abstract, and so it is important to look closely at the component parts of the field to understand exactly what it is.

SI describes the behaviour of a population of individual organisms (in our setting also referred to as "agents"), adhering to some core concepts. Beginning from the ground up, we thus have a population of individuals, the "agents", which collectively constitute the "swarm".

The first core concept that we require is that of decentralization. The population must be roughly homogeneous and each agent must act individually. There can be no single "leader" or "over-mind" controlling the actions of the agents. A good way to test a population for this property is to divide the population into separate sub swarms. If the population is decentralized, each new sub-population will still function properly, although not to the same level of efficiency.

The second core concept is that of self-organization. Self-organization is defined as a phenomenon where some form of global coordination arises out of the local interactions between the components of an initially disordered system. In the case of SI, this means that the actions of the population in its entirety must arise *solely* from the interactions of the individual agents. This works in conjunction with the idea of decentralization, in that the actions of the population as a whole cannot be determined externally, or by any single agent.

Additionally, as a general rule, the agents must be relatively unsophisticated. By this we imply that SI is displayed when a population of unsophisticated, "primitive" agents interact in such a way that a relatively complex, sophisticated behaviour emerges from the population as a whole. This leads to the notion of the swarm's

so-called emergent behaviour, which is an important component of what makes SI significant.

The essence of emergent behaviour is closely linked to the concept of self-organization. A system's emergent behaviour is the one which arises from a multiplicity of relatively simple interactions. This is where the so called "intelligence" of the swarm is believed to reside. There are two varieties of emergent behaviours, i.e., weak and strong. *Weak* emergent behaviour is the most common of these two, and occurs when the behaviour of the system can be reduced down to the behaviour of the individuals. *Strong* emergent behaviour, on the other hand, occurs if the behaviour of the system cannot be directly reduced to, or deduced from, the behaviour of the individuals. The phenomenon of strong emergence supports the truism that the "whole" is greater than the sum of the strengths of its constituent parts, which is, indeed, the focus of SI.

2.2.2 Origins, Observations in Nature

The paradigm of SI, like many other paradigms, has arisen from observing nature. Consider for example, ant colonies. On an individual level, ants are extremely simple creatures. When isolated, ants seemingly move at random with no real purpose. However, observations of ant colonies display a startling level of organization, and a seemingly-intelligent behaviour of the colony is apparent. In fact, ants are one of the most successful species in the world and studies have even shown the existence of mega-colonies spanning the globe [8].

Individual ants are so basic (or "primitive") that it would be impossible for any single ant to understand the "big picture" of the colony's actions. If no ant is capable of this, it is clear that the actions or behaviour of the colony as a whole is not, and could not possibly be pre-planned. It is interesting then to perceive how it could be possible for ant colonies to display such sophisticated organization in foraging for food, building the complex colony's tunnel structures, and even waging wars on enemy colonies.

Ant colonies have been thoroughly observed and studied because of this strange decentralized intelligence and were, perhaps, the first inspiration for the phenomenon

of SI. The behaviour of the entire colony is far too complex and goes beyond our definition of SI, and so we focus on their foraging behaviour in isolation, for a simple example.

The foraging behaviour of ant colonies is a very good example of SI. On an individual level, each ant acts according to a small set of basic rules. While searching for food, ants move randomly through their environment. After finding a food source, an ant will return to the colony while laying down a pheromone. This pheromone is a chemical secretion which other ants can detect, and which slowly evaporates with time. As other ants are searching randomly for food, their movements are influenced by the presence of pheromone. Ants will be more likely to move towards a pheromone, depending on the strength of the pheromone.

The most direct consequence of these rules is that ants will be more likely to find food sources with pheromone trails and, in turn, return to the colony while leaving their own pheromone. Over time, more and more ants will find the food and the pheromone trail will become stronger and stronger, attracting more and more ants to the food. Additionally, areas with many sources of food will tend to have many pheromone trails, which will, over time, begin to merge, resulting in a branching structure of trails with massive ant "highways" leading from the colony to the general location. The creation of these "highways" is a good example of a high level emergent behaviour. Another very interesting property of these branching trails are their efficiency. The trails themselves will tend to approach an approximately shortest path over time. Nowhere in the basic operationis of the individual ant does one observe or infer anything about efficient branching structures or route planning. Rather, these seemingly intelligent behaviours emerge from the very basic interactions of the individual ants themselves. This is a prime example of the ant colony displaying a strong emergent behaviour.

For a more detailed example, we look at another colony-based insect, the termite. Termites are known for constructing vast, complex nest structures to house their colony. When one studies these structures, it is hard to believe that they were created without being pre-planned or without a central organizational control. There is no "architect" termite which intelligently designs the whole structure, communicating

those plans to all the other termites. Rather, each termite involved is roughly equal, and each termite has roughly the same power in the task of shaping the overall structure. It is the local interactions of these termites, during construction, which gives rise to the complex whole.

When the construction of a nest begins, each individual termite acts on its own, without any notion of a pre-existing plan. Termites begin creating mud balls and placing them at random. Each mud ball created is infused with a small amount of pheromone. Termites which are placing their mud balls are more likely to place them in the presence of this pheromone. The more pheromone present, the more likely it is that termites will be attracted to that location to deposit their own mud balls. From this simple interaction pillars of mud emerge. As separate pillars grow, the pheromone from one attracts termites from the other, causing the pillars to grow towards each other, creating arches and walls. In the end, these termites will have created a giant, stable structure without any central plan. This is a prime example of how a SI is capable of naturally creating complexity. Termite nests are structurally complex and very strong, despite being created solely by creatures which have no concept of what renders a structure to be sound.

Though colony-based insects are the most direct example of SI in nature, there are many other places where it can be observed. For example, large flocks of birds can be seen to display a sort of SI. Entire bird flocks can make sudden, spontaneous changes in direction without losing cohesion. These flocks also seem to travel with purpose and to make the decision to land in an instant. This is interesting because bird flocks do not have any single leader or coordinator. While individual birds are significantly more intelligent than ants and far more capable of communication, the abruptness of movement and the way that the flock remains so cohesive during such split-second maneuvers makes formal communication unlikely. Rather, it would seem that when flying in a flock, each bird acts individually based on their neighbours and surroundings. Each individual bird will attempt to keep within a comfortable range of its neighbors, and to follow where its neighbors are flying. This means that at any time, the behaviour of the flock is influenced by the decisions of any individual bird. Consider, for example, the situation where a bird near the front of the flock decides

to land. As the bird begins to change direction, all the birds next to it naturally begin to do the same due to the behaviour of following their neighbors. This, in turn, causes more birds to change direction, eventually causing the entire flock to change direction. As the flock then approaches the landing area, more birds will decide to land, since the flock is approaching the ground. Eventually, each bird will individually make the decision to land and the entire flock will land!

2.2.3 SI in Computer Systems

The idea of SI was first seen in the context of computer systems in the field of artificial life. The primary example of this is the Boids algorithm.

The Boids algorithm was published in 1986 by Craig Reynolds [23] and seeks to simulate the flocking behaviour of birds. Each agent, or "boid", acts on a very simple set of rules: separation, alignment and cohesion.

- The separation rule states that a boid will steer away from other boid or environmental obstacles if it is too close to them.
- The alignment rule states that each boid will steer towards the average heading of all other boids within a local area.
- The cohesion rule states that each boid will steer towards the center of mass of the boids within a local area.

At each step of the simulation, the individual boid updates its heading and location based on these rules. As a result of these primitive rules, boids will gather into swarms which move as though they are steered by a singular mind. The Boids algorithm has seen widespread application in computer graphics and animation, and has been used in many motion pictures.

Although the ideas and concepts behind SI had been observed and studied in nature for years, it wasn't until 1989 that the expression "Swarm Intelligence" was coined. The expression was coined by Beni and Wang in their paper on a cellular robotics system.

Since then, SI has been explored and applied to many different situations including optimization problems, routing problems, job scheduling problems etc. SI is a popular field of research in computing for many reasons. In the field of autonomous robotics, SI is a very attractive principle because it allows systems to be both naturally flexible and scalable. Additionally, adhering to the concepts of SI would imply the use of relatively less sophisticated individual robots, meaning easier production and reduced costs. In the context of practical algorithms, SI has many attractive features. Because of the decentralization of SI systems, these algorithms are naturally extendable to parallel and distributed computing environments. Additionally, as population-based methods, SI algorithms are naturally scalable.

2.2.4 SI for Optimization

SI has been applied to optimization problems as a meta-heuristic method. The term "meta-heuristic" refers to a broad classification of problem solving schemes which aim to explore the solution space in a more abstract fashion. Many meta-heuristic methods are not problem specific, and so the task of finding a good meta-heuristic algorithm can support a large number of applications.

There are a wide variety of optimization techniques based around the concept of SI, and these, in turn, have been applied to a range of problems. While many of these algorithms are based on natural phenomena, the concepts of SI have been approached in many different ways and from many other backgrounds.

One of the first examples of the concepts of SI being applied to optimization problems involved ant colony optimization. Ant colony optimization is an optimization method inspired by the above-described foraging behaviour of ants. The original ant colony optimization algorithm, published in 1991 by Dorigo, aimed to find optimal paths through a graph.

As mentioned earlier, ant trails would tend towards more efficient paths despite the incapability of individual ants making such decisions. This phenomenon is the result of two main factors. Firstly, since newer paths have weaker pheromone trails, multiple similar paths tend to appear as ants will be more likely to move away from

weak pheromones. Ants which happen to follow shorter paths will spend less time travelling, so naturally these paths will gradually receive more pheromone. Secondly, this works in conjunction with the fact that pheromone evaporates over time. Longer paths, then, will naturally have weaker pheromone levels. Over time, the shortest paths become strongest and most reinforced.

The above principles define the conceptual basis for the algorithm. Ant agents[1] are created on a starting node within a graph. Each ant then acts individually, choosing which edges to traverse randomly based on a routing table. As ants travel throughout the graph they age over time. This aging process acts to replace the evaporation aspect of pheromone in nature, and serves to safeguard the system against infinite loops. If an ant reaches a specified age without reaching the goal, it is removed. However, if the ant reaches the goal, it "deposits pheromone" along the edges it has traversed based on its age. In this context, the task of "depositing pheromone" is simulated by altering the routing table so as to increase the probability of ants following the traversed edges.

Over time, as more and more ants are created and traverse the graph, edges involved in shorter paths to the goal will become increasingly likely choices for the ants. This is because of the ants' aging process. Shorter paths deliver younger ants, which, in turn, lay stronger pheromone. Eventually, if a shortest path exists, the routing table will reflect that.

Since the original algorithm, ant colony algorithms have been thoroughly researched and extended to cover many types of problems, including dynamic load balancing of networks, routing, job scheduling and general optimization problems. In terms of optimization, ant colony optimization is generally used for combinatorial optimization problems that can be represented in terms of graphs. For example, one of the earliest ant colony optimization algorithms addressed the well-known travelling salesman problem.

Another example of an SI-based optimization method inspired by nature is the bee colony algorithm. The concept was initially proposed in 2005 by Karaboga. In nature, bees are another example of a colony-based insect possessing efficient foraging

[1] In the future, these agents will be synonymously called "ants".

behaviour. However, bees organize themselves in a more sophisticated manner than ants. Bees employ physical dances to communicate information with others in the hive. A bee which has located a source of food will return to the hive and communicate the rough location by dancing. Any idle bees observing this dance can then find the rough location of the food source.

From an abstract level, the algorithm proceeds as follows. A population of employed bee agents (also referred to synonymously as "bees") is created and each is assigned a random initial food source. Each employed bee then travels to their assigned food source and chooses a neighboring location. After determining the value of that location, the employed bees return to their hive. Each employed bee then "dances", broadcasting the location and value found. Onlooker bees waiting in the hive observe these dances and chooses one of the locations, finds a neighbour around it and evaluates the value of that neighbor. If there exists any food sources that are not not chosen by any observer bees, these sources are forgotten. Any bee whose food source is not chosen and therefore forgotten becomes a scout bee, searching randomly for a new source of food which it can present.

Bee colony optimization can be applied to any type of optimization problem where it is possible to determine neighbours for any given candidate solution. This is a good example of how SI can be applied as an abstract meta-heuristic method to operate on a very wide range of optimization problems.

2.3 Particle Swarm Optimization

2.3.1 Introduction and Basic Concept

Particle Swarm Optimization (PSO) is a Meta-Heuristic, black-box optimization algorithm first developed around 1995 and attributed to Kennedy and Eberhart [12], based on the concepts of SI. As an SI system, PSO consists of a population of agents which work together in order to find the best solution to a given black box optimization problem. These agents, or "particles", have positions representing candidate solutions and "fly" throughout the solution space. The particles are assigned values

by evaluating the black-box function with their respective position as the input variable. Each particle keeps a memory of the best position that it has found, and is capable of communicating this with the rest of the swarm, or in some cases, to a local neighbourhood. At each step, particles move through the solution space according to their velocity, while being accelerated randomly towards their personal best solution point and to the best point found by the entire swarm.

PSO was originally intended for continuous, real-valued optimization problems. Nevertheless, it has also been adapted to address combinatorial and integer-based optimization problems. This, combined with the fact that PSO can potentially optimize *any* black-box function, makes it a very flexible and popular tool for solving difficult optimization problems. The PSO algorithm has found applications in a wide variety of fields ranging from the placement of wind turbines [17] to workflow optimization within cloud computing [1].

2.3.2 Origin of PSO

The PSO algorithm came about as the result of many iterations of work stemming from the field of artificial life. The original intention was to simulate social behaviour using the Boids bird flocking algorithm as a base. Here, social behaviour refers to a more abstract concept than the physical behaviour of swarming animals or bird flocks. The "birds" of this simulation were meant to represent concepts such as "beliefs" or "attitudes" and the motivation was to simulate ways that individuals in a population conform to each other. The key difference between simulating physical bird flocks and these abstract concepts is collision. In bird flocking algorithms, each bird must maintain distance from its neighbours in order to avoid a physical collision. However, when simulating the abstract ideas of an individual's "beliefs" or "attitudes", any number of individuals can exist in the same point in space, as there is nothing stopping individuals from sharing identical "beliefs" or "attitudes". The basic goal, then, was to simulate the choreography of a bird flock using these collision free "birds".

The initial algorithm was built around two main concepts: nearest-neighbour velocity matching and so-called "craziness". The system would be initialized with agents

randomly distributed on a two dimensional toroidal grid and with random velocities. At each step of the simulation, agents would identify their nearest neighbour and match their velocity with that neighbour. This resulted in a very synchronous movement of the population, but would eventually become trapped in a single unchanging velocity. To address this problem, "craziness" was introduced. At each step of the simulation, agents velocities were randomly modified in addition to the aforementioned velocity matching. This change produced a more interesting and natural movement of the population.

The next evolution of the algorithm involved the addition of the "cornfield vector". Previous work in bird simulations utilized a so-called "roost", a point in space which would attract the agents with the ultimate goal of "landing". Building on this concept, the motivation was to simulate a flock of birds searching for food. This was achieved with the so-called "cornfield vector", a point in the two dimensional space representing a food source. At each step of the simulation, agents would evaluate their current position relative to this point according to the equation:

$$Eval = \sqrt{(presentx - fieldx)^2} + \sqrt{(presenty - fieldy)^2}$$

Expanding on this, each agent tracks the best point they have found so far, known as their *pbest* point. Agents would then broadcast their *pbest* to find the best position found by any member of the population, known as the *gbest* point. After performing these operations the agents would then modify their velocities, influenced by their *pbest* and *gbest* points. This modification was done independently for the x and y dimensions according to the following equations.

$$\text{if } presentx > pbestx \text{ then } velocityx = velocityx - U[0, p\_increment]$$
$$\text{if } presentx < pbestx \text{ then } velocityx = velocityx + U[0, p\_increment]$$
$$\text{if } presentx > gbestx \text{ then } velocityx = velocityx - U[0, g\_increment]$$
$$\text{if } presentx < gbestx \text{ then } velocityx = velocityx + U[0, g\_increment]$$

Similar modifications were performed using the y dimension. In these equations, the $p\_increment$ and $g\_increment$ variables are provided as parameters to the system and represent the maximum attractive force the personal and global best positions can

have on the agent's velocity. With these parameters set relatively low, agents would circle about the cornfield point until ultimately landing, as expected. However, with the parameters set higher, the agents would appear to be drawn directly in, and collapse on the cornfield point. It was observed, then, that the bird flock appeared to be performing optimization, as the flock would seek out the cornfield point which could be interpreted as the global minima of the cornfield function. After this was observed, motivations shifted to investigating the optimization potential of the algorithm.

The first step in adapting this simulation algorithm to a full fledged optimization technique was to identify, and strip out, components of the algorithm which did not contribute to its optimization potential. Experimentation found that the so-called "craziness" and nearest-neighbour velocity matching components did not contribute to this potential. "Craziness" was initially introduced to add some randomness to the movement of the population. However, this was made redundant with the randomness introduced in the *pbest* and *gbest* velocity modification equations. Nearest-neighbour velocity matching was observed to actually impede optimization performance, and was removed. With nearest-neighbour velocity matching removed, the visual appearance of the system changed to look much more chaotic. As such, it was decided that it would be more appropriate to refer to the population as a swarm, rather than a flock.

After the algorithm had been stripped down, it was generalized from a strictly two dimensional space to a multidimensional space. This was a simple change, as all calculations were done dimension by dimension. Experiments were then performed, applying this algorithm to non-linear optimization problems. Results were promising, showing that the algorithm was capable of optimizing many difficult problems, such as neural-network training and various extremely nonlinear benchmark functions.

The final step was a modification to the *pbest* and *gbest* components. Previously, the velocity would be modified on a per dimension basis, adding or subtracting a random amount based on whether or not the values were greater or less than the corresponding *pbest* or *gbest* values. These calculations would not take into account how close or far these values were from each other, information that intuitively seems valuable. This information was incorporated by removing the comparison operators and modifying the velocity randomly according to the difference in values, for each

dimension.

At this point, the PSO algorithm was complete in its original form. Because the visual movement of the agents was changed so much from the original bird flocking concept, it was decided that the metaphor used to describe the agents should be changed. Since it had already been decided that "flock" be replaced by "swarm", "particle" was introduced to describe agents rather than "bird". Since agents in the system were unable to collide, the concept of an arbitrarily small particle fit well.

In 1998, Shi and Eberhart would introduce a new variable to the algorithm, which would be accepted as part of the core PSO algorithm from then on [25]. Shi introduced the idea of *inertia* to the system, a variable which modified the contribution of the particle's previous velocity when calculating the updated velocity. This small change had a large impact, which will be discussed in the next section.

2.3.3 Algorithm Details

The core PSO algorithm consists of a population of particles, which act according to individual behaviour and influence each other via communication. A PSO particle i consists of a position $\vec{X}_i$ within the solution space, a velocity $\vec{V}_i$ and a personal best found point $\vec{P}_i$. The algorithm begins with an initialization stage, which is followed by an iterative simulation stage. Pseudocode for the PSO algorithm is presented in Algorithm 1. During initialization, the population of particles is created and their positions are set according to a uniform random distribution in the solution space. Upper and lower bounds for this initialization range are specified by the user. Further, particle velocities are initialized according to a uniform random distribution using the same bounds provided for the initialization of the positions. Each particle then evaluates its initial position, $\vec{X}_i$, and sets this point as its personal best solution, $\vec{P}_i$. The initial global best point, $\vec{P}_g$, is then determined from among these personal best solutions.

The simulation is then initiated and it runs indefinitely unless a termination criterion is met or the algorithm is manually stopped. At each step of the simulation, particles update their velocities, move accordingly, and evaluate their new location.

Algorithm 1 Canonical PSO Algorithm

Input:
Function $f()$ to be optimized
Initialization range $lbound$, $ubound$
Particle population size n
Inertia coefficient ω
Personal Best acceleration coefficient ϕ_p
Global Best acceleration coefficient ϕ_g

Output:
Position $\vec{P}_g$ representing the best found solution

Method:

create particle population P with size n
for each particle $i \in P$ **do**
  $\vec{X}_i \Leftarrow \vec{U}[lbound, ubound]$
  $\vec{P}_i \Leftarrow \vec{X}_i$
  if $f(\vec{P}_i) < f(\vec{P}_g)$ **then**
    $\vec{P}_g \Leftarrow \vec{P}_i$
  end if
  $\vec{V}_i \Leftarrow \vec{U}\,[-|ubound - lbound|, |ubound - lbound|]$
end for

while termination criteria not met **do**
  for each particle $i \in P$ **do**
    $\vec{V}_i \Leftarrow \omega\vec{V}_i + \vec{U}[0, \phi_p] \otimes (\vec{P}_i - \vec{X}_i) + \vec{U}[0, \phi_g] \otimes (\vec{P}_g - \vec{X}_i)$
    $\vec{X}_i \Leftarrow \vec{X}_i + \vec{V}_i$
    if $f(\vec{X}_i) < f(\vec{P}_i)$ **then**
      $\vec{P}_i \Leftarrow \vec{X}_i$
      if $f(\vec{P}_i) < f(\vec{P}_g)$ **then**
        $\vec{P}_g \Leftarrow \vec{P}_i$
      end if
    end if
  end for
end while
return $\vec{P}_g$

Their corresponding personal best solution is also updated, if necessary. The corresponding velocities are updated on a per-dimension basis based on the inertia and random accelerations toward the global best solution, and the particle's personal best solution. The concept of "inertia" refers to how much of the previous velocity carries over to the updated velocity, which we represent with the variable ω. Particles are accelerated toward their personal and global best solutions according to the following:

- The distance between itself and those points,
- The parameters representing the "power of attraction" of each best solution (ϕ_p and ϕ_g respectively), and
- Uniform random coefficients between zero and unity.

The "power of attraction" for the personal and global best solutions are supplied as parameters to the system and they combine with the uniform random coefficients. Once the velocity is updated, each particle determines its next position by adding the velocity to their current position and the simulation loop continues. A particle i updates its velocity $\vec{V_i}$, and position $\vec{X_i}$, according to the equation:

$$\vec{V_i} = \omega\vec{V_i} + \vec{U}[0, \phi_p] \otimes (\vec{P_i} - \vec{X_i}) + \vec{U}[0, \phi_g] \otimes (\vec{P_g} - \vec{X_i})$$
$$\vec{X_i} = \vec{X_i} + \vec{V_i}$$

Once a termination criterion is met, or the simulation is manually stopped, the position of the global best particle $\vec{P_g}$, which represents the best solution found by the algorithm, is returned as the output.

It is important to note that there are two distinct strategies for applying "randomness" during the velocity update function, i.e., classical and linear. In the so-called classical PSO, random weighting of vectors is affected dimension-by-dimension, generating individual random numbers for each dimension. The linear strategy, however, applies weighting to the vectors with the same random value for each dimension. These two strategies result in considerably different movement patterns and swarm behaviour. The classical strategy is, typically, the preferred strategy as it is considered to be more exploratory than the linear strategy. Consider a situation where a

particle, its personal best point $\vec{P}_i$, and the global best point $\vec{P}_g$, are co-linear along its current velocity. The linear strategy PSO will become trapped along that line unless the $\vec{P}_g$ point changes. In the same situation, however, the classical strategy PSO will not become trapped because the random weighting strategy is still capable of perturbing it off that line. Nevertheless, the classical strategy has been shown to result in a bias, which will be discussed later.

2.3.4 Parameters and Their Selection

Parameter selection has a dramatic impact on the behaviour of the particle swarm and, as a result, on the performance of any PSO algorithm on different optimization problems. The three parameters affecting the behaviour of the swarm are: Inertia (ω) and the attraction coefficients ϕ_p and ϕ_g, for the personal and global best points respectively.

The inertia parameter, ω, is a coefficient applied to the particle's previous velocity when calculating the new velocity. Values between zero and unity can be perceived to be modelling the application of a "rough" friction to the respective particles as they migrate through the solution space. Conversely, values greater than unity cause the particles to constantly accelerate forwards. Keeping in mind the accelerations toward the personal and global best points and assuming that they are comparable, it is easy to see how changes in inertia would effect the behaviour of the swarm.

As particles constantly slow down, the swarm converges to local optima. The reason for this is that without a substantial velocity, particles will not be able to escape, as easily, because of their tendency to migrate toward their personal best solutions. On the other hand, with constantly accelerating particles, the swarm shifts toward a more exploratory and divergent behaviour. Interestingly, it has been found that negative inertia values *also* perform favourably. Negative inertia values cause the particles to appear to "vibrate", bouncing back and forth due to the constantly "flipping" velocity patterns.

The personal best attraction coefficient, ϕ_p, affects the acceleration toward the particle's personal best point. Similarly, the global best attraction coefficient, ϕ_g,

affects the acceleration toward the swarm's global best point. These parameters effect the social behaviour of the agents, and as a result, different values cause different swarm behaviours. A zero value for ϕ_p causes agents to act as though they possess no memory. Consequently, the swarm will quickly collapse toward a single local optimum. A zero value for ϕ_g causes the agents to act without communication. This causes the algorithm to degenerate toward behaving more like a multiple random local search scheme than an SI algorithm. This, in turn, causes particles to become stuck in local optima almost immediately. When selecting values for these parameters it is important to find a good balance between the two.

In addition to these parameters dictating the behaviour of the swarm, there is, of course, a parameter specifying the total number of particles in the swarm. While it may seem obvious that a larger swarm is always better, in practice, it is important to keep the swarm sizes as low as possible to reduce the computational cost and the running time of the algorithm.

PSO performs very well on a large variety of optimization problems, but, in practice, the parameters must be fine-tuned to achieve this performance. As such, parameter selection has been thoroughly researched since the initial formulation of the algorithm. To this end, PSO has even been the subject of meta-optimization techniques [21]. Meta-optimization applies black-box optimization techniques as a second level on top of the original optimization algorithm. The meta-optimizer seeks to optimize the parameter values of the original optimizer as though it were a black-box function in and of itself, with the parameters being in the corresponding solution space. In fact, PSO can even be used as a meta-optimizer to optimize a second PSO, due to its flexibility.

While each optimization problem will require uniquely-tuned parameters, there are general rules which can be followed to help guide the selection process. These rules are based on different properties of the optimization problem, and so a general knowledge of the problem's solution space is extremely helpful. On problems with only a single optimum, lower inertia values and higher ϕ_g values dramatically increase the performance of PSO. This is because there is no risk of becoming stuck in local optima, though finding the global optimal point may still be difficult. In this context,

high inertia may cause particles to orbit or to overshoot the global best area of the problem. Higher inertia values become more important as problems become multimodal or noisier. With many local optima, particles require higher inertia values to avoid becoming stuck early in these local areas. However, with higher inertial values, convergence becomes less likely and the swarm may "explode", with particles constantly gaining speed. It is much more difficult to establish good rules for the ϕ_p and ϕ_g parameters for multimodal problems, as the effects of different solution space "shapes" becomes very complex. For a somewhat simple example, consider a basic single-optimum function with a superimposed sine wave noise. This problem will have many local optima, but its rough averaged "shape" will have only a single region which contains a potential global best. In this case, a higher value for ϕ_g will improve performance, as particles will be pulled more toward the good region despite the noise. As a counter example, consider a very "smooth" multimodal problem with a low number of local optima, each with similar potential for holding a global best solution. In this context, a higher ϕ_p value will improve performance, as particles will find and explore these few local optima areas more thoroughly, rather than being pulled toward a single one of these areas by the ϕ_g value. Additionally, problems which are multimodal or noisier, naturally require more particles than unimodal problems in order to effectively explore the local optima areas to find the true best.

PSO is no exception to the so-called "curse of dimensionality". The dimensionality of the problem has a large impact on parameter selection as well as on the "shape" of the solution space. As the dimensionality increases, to maintain effectiveness of the algorithm relative to lower dimensions, the number of required particles increases dramatically. The number of particles required to maintain the same performance for higher dimensionality quickly becomes infeasible, and so the parameters must be tweaked in order to make the most out of the limited swarm sizes. The implication of a limited swarm size is that each particle must cover a larger area in order to maintain its effectiveness. This, in turn, means that the PSO parameters must be tweaked in order to promote more exploratory behaviour. The number of iterations required also increases dramatically despite the tweaked parameters, as it naturally takes longer to explore the space.

2.3.5 Analysis and Inner Workings of PSO

As an SI system, PSO relies on the interactions of its individuals and the phenomena of emergent behaviour to achieve its performance. Because of the manner in which PSO was first developed, the proof of its ability to optimize, and support for its performance claims, were mostly empirical. Efforts to analyze PSO from a more theoretical standpoint have been made in order to better understand the algorithm, and to offer performance guarantees and identify biases. Due to the PSO's reliance on stochastic processes and the interactions of its population, any analysis of the algorithm is difficult and many "iterations" of work have been required to approach a full understanding of the algorithm.

The first attempt at an analysis of PSO was made in 1998 by Ozcam and Mohan [19]. This analysis considered a very stripped-down version of the algorithm in an attempt to understand the movement patterns of the individual particles. This simplified algorithm involved a system with only a *single* particle in a single dimension, with no inertial weight, in an environment of swarm stagnation, i.e. the personal and global best points were fixed and identical to each other. Additionally, the random element of the algorithm was modified to be a random constant, determined *a priori* before the beginning of the simulation rather than at each step. This analysis showed that under these conditions the particle follows a sinusoidal wave of oscillation about the global best point.

In 1999 Ozcam and Mohan extended this analysis [20]. This extended analysis generalized the results of the initial analysis to a general multidimensional space with multiple particles. However, the swarm was still assumed to be in a state of stagnation, with personal and global best points remaining constant, and with the random elements being constant throughout the simulation. This work led to the identification of different categories of oscillation for the particles, and also to strategies which identified the category of a particle's oscillation so as to modify the parameters dynamically. Despite these results, however, in practice, a PSO system, with constantly changing global and personal best points, and with an associated randomness, will have particles jumping to different sine waves so often that such an analysis will not successfully capture the true nature of the swarm.

In 2003 Clerc and Kennedy performed another analysis of the PSO system [5]. Unlike the previous analysis done by Ozcam and Mohan, this analysis was more focused on the convergent behaviour of particles, rather than on the precise movements, and the influence of the parameters upon it. This work built upon the simplified system used by Ozcam and Mohan and reintroduced elements to work toward a fully-formed PSO system. However, the PSO system used here was a modified version that used a so-called "constriction" coefficient rather than the traditional inertial weight. The variation of this constriction coefficient rendered convergent behaviour easier to understand than the traditional inertia version. This work showed that the variation of the constriction coefficient was guaranteed to converge, whereas the traditional inertial weighting strategy was not guaranteed to converge for all parameter values.

In 2005 Engelbrecht and Vandenbergh built upon the work of Clerc and Kennedy to analyze a PSO system using the traditional inertial weighting, and stochastic elements that were fully reintroduced [28]. This analysis provided enough insight into the effects of the parameters on the system so as to create a heuristic method for setting the parameters with guarantees of convergence for the traditional algorithm.

These examples of PSO analyses all look at the movements of individual particles in the swarm. However, this is not the *only* way to look at the way by which a PSO algorithm works. In 2002, Kennedy and Mendes provided a different perspective on the inner workings of PSO [13]. Their analysis considered the social behaviour of the entire swarm and the way by which information was passed between the individuals. The social aspect of PSO lies in the communication of the best found positions of the respective particles. The passing of this information can be easily represented using a graph structure where the nodes represent the particles and the edges represent the lines of communication between them. In such a graph model, the particles send their personal best value to their adjacent particles in the graph at each iteration. This implies that the information travels at a maximum rate of a single edge per step, and different graph topographies cause information to flow in different ways. In the standard PSO, all particles are immediately aware of the global best position found. Using the graph model, the standard PSO can be represented by a complete graph, because at each iteration, the particles can be considered to be sending their personal

best value to all the other particles.

Another commonly used strategy is to limit the global best communication to small local neighbourhoods, typically two or four neighbours for each particle. In the graph model, communication of this sort can be represented using a ring or square topology respectively. Kennedy and Mendes investigated the effects of several different topologies, looking for relations between the information flow and the corresponding swarm behaviours. A few topologies which performed consistently better than others were identified, and some which, surprisingly, performed very poorly. Swarms with a so-called Square topology, arranged in a 2-D toroidal mesh, performed best, along with swarm topologies which consisted of many connected cliques. Surprisingly, the worst performing topology was one close to the accepted standard "global broadcast" strategy. This was a global topology that did not permit the communication of the best point to the communicating node itself. What this means is that a particle's global best point could not be its own personal best point. Another strategy that performed extremely poorly was a star topology, in which all the particles are connected only with a single "central" particle. This result demonstrates the importance of decentralization to the power of SI systems.

2.3.6 Biases in PSO

When using or testing an optimization algorithm it is crucial to be aware of any biases the algorithm may have. This is especially important when dealing with metaheuristics or other algorithms which are not guaranteed to find the globally best solution, because these biases will have direct effects on the quality of the final result. PSO has been shown to display some biases which influence its performance in certain situations.

The first of these biases is common to most population-based algorithms. This is a bias toward the centre of the initialization area [18]. PSO will, naturally, explore the centre of its initialization area, because intuitively, individuals are more likely to pass through this area than an area on the "outskirts". In practice, knowledge of this bias can be used to increase the performance of PSO if the approximate

location of a possible global optimum is known. When testing PSO-based algorithms on benchmark functions, the initialization range should be offset from the centre in order to avoid any misleading results.

Particles in the classical PSO have been shown to favour movement parallel to the axes [27]. In this context, the reader should recall the difference between the so-called classical and linear velocity updating strategies for the primary PSO algorithm. While classical PSO generates new random values for each dimension of the update, linear schemes use the same random value to weight each dimension of the vector. This tendency of particles to move *parallel* to the axes causes a bias toward areas along an axis or to those lying on a diagonal of the solution space. However, the linear velocity updating strategy does not display this bias. Despite this bias the classical PSO is still preferred over the linear method since the basic PSO possesses a more exploratory behaviour. Again it is crucial to be aware of this bias when testing and comparing PSO algorithms, as certain benchmark functions may result in drastically different performances when rotated, yielding misleading performances.

2.4 Variations and Extensions of PSO

Due to its simple and flexible nature, there have been countless variations and extensions of the core PSO algorithm over the years. These variations and extensions vary significantly, thanks to the flexibility of the algorithm. Some researchers believe that the PSO should be simplified as much as possible, claiming that complicating the algorithm will impair the underlying SI principals at work. Other strategies involve developing a more sophisticated PSO, hybridizing the PSO with other algorithms, or applying general principals found in similar algorithms. Since PSO was initially created experimentally, the original algorithm was fairly crude with plenty of room for improvement. Modern standard PSO algorithms incorporate many of these variations and extensions to result in a more robust algorithm [6].

2.4.1 Constriction Coefficient and the Fully Informed PSO

Early experiments with PSO displayed a tendency for the swarm to "explode". That is to say, the particles would constantly gain speed and scatter, rather than converge as desired, if parameters were not carefully set and tweaked. The first attempt to address this problem was to set a hard upper bound on the velocity. If, at any time, the magnitude of a particle's velocity exceeded this maximum, the velocity would be scaled down to that maximum. This was, however, a somewhat inelegant solution, and so more attempts were made to address the problem. The inertia parameter was originally introduced as a solution, but this could still only prevent swarm explosion for certain parameter values. In order to better guarantee convergence, an alternative method of constricting particle velocities was developed, involving a so-called constriction coefficient [7]. With the standard inertial weighting, the velocity of a particle i is updated according to the equation:

$$\vec{V_i} = \omega\vec{V_i} + \vec{U}[0, \phi_p] \otimes (\vec{P_i} - \vec{X_i}) + \vec{U}[0, \phi_g] \otimes (\vec{P_g} - \vec{X_i}).$$

Rather than weighting the contribution of the particle's current velocity by an inertial coefficient, the constriction coefficient weights the new velocity according to:

$$\vec{V_i} = K\left(\vec{V_i} + \vec{U}[0, \phi_p] \otimes (\vec{P_i} - \vec{X_i}) + \vec{U}[0, \phi_g] \otimes (\vec{P_g} - \vec{X_i})\right), \quad \text{where}$$
$$K = \frac{2}{(|2 - \phi - \sqrt{\phi^2 - 4\phi}|)}, \quad \text{and}$$
$$\phi = \phi_p + \phi_g, \quad \phi > 4.$$

Unlike the inertia coefficient, which must be supplied as a parameter to the system, the constriction coefficient is a function of the "power of attraction" parameters ϕ_p and ϕ_g, and so an appropriate value is determined automatically. This is another advantage the constriction coefficient has over the inertial weighting, as it reduces the number of required parameters and makes the algorithm easier to use. This constriction coefficient is, in fact, a special case of the inertia weighting. Applying an inertia weighting equal to K will yield equivalent results and, as such, the constriction coefficient method can be used even just as a method of finding good parameter values for the standard inertia method.

Development of the constriction coefficient has also led to additional discoveries about the behaviour of PSO. The velocity updating equation in this constriction coefficient strategy can be condensed to the following formula:

$$\vec{V_i} = K\left(\vec{V_i} + \vec{U}[0,\phi] \otimes (\vec{P_m} - \vec{X_i})\right),$$

where $\vec{P_m}$, given below, is the weighted average of the personal and global best points:

$$\vec{P_m} = \frac{\phi_p \vec{P_i} + \phi_g \vec{P_g}}{\phi_p + \phi_g}.$$

Conceptually, the algorithm allows particles to randomly accelerate toward this single point of attraction, $\vec{P_m}$, which is the weighted average of the personal and global best points.

In 2004 Mendes, Kennedy and Neves built on this perspective in creating the Fully Informed Particle Swarm (FIPS) [16]. Rather than using only the personal best and global best points, the FIPS algorithm creates a point of attraction $\vec{P_m}$ using the particle's entire neighbourhood. Considering a particle with neighbourhood N, the point $\vec{P_m}$ is generated using the following equation:

$$\vec{\phi_i} = \vec{U}\left[0, \frac{\phi}{|N|}\right] \quad \forall i \in N$$
$$\vec{P_m} = \frac{\sum_{i \in N} W(i)\vec{\phi_i} \otimes \vec{P_i}}{\sum_{i \in N} W(i)\vec{\phi_k}}.$$

$\vec{P_i}$ represents the personal best point of particle k, and $W()$ is a function which assigns a weight to each point $\vec{P_i}$. Typically this function weights each point according to the fitness of the point $\vec{P_i}$. The values $\vec{\phi_i}$ replace the ϕ_p and ϕ_g values from the original algorithm and are random vectors whose sum will always be less than ϕ. Because of this, the number of parameters required by the algorithm is reduced. Additionally, since randomness has been introduced in determining the $\vec{\phi_i}$ vectors, the random weighting in the velocity updating equation can be removed, as it was essentially moved within this process of determining $\vec{P_m}$. The final velocity update equation becomes:

$$\vec{V_i} = K\left(\vec{V_i} + \phi(\vec{P_m} - \vec{X_i})\right).$$

Defining the neighbourhood N has a large effect on this algorithm. In the typical PSO, which uses global best, a particle's neighbourhood would be the set of all particles, including itself. However, the basic PSO uses only the particle's own best point and the global best point within the neighbourhood, whereas the FIPS algorithm uses all the particles within the neighbourhood. Because of this, using a neighbourhood of all the particles would result in each particle using the same information to randomly generate a point of attraction, which may not be the best strategy. To mitigate this, the analysis of different social structures in PSO, by Kennedy and Mendes [13], was valuable in determining good strategies for particle neighbourhoods in the FIPS algorithm. In fact, the global neighbourhood was consistently worse than most other neighbourhood structures, despite it being consistently better for the canonical PSO.

Initial experiments were done, comparing the canonical PSO with several variations of the FIPS algorithm using different weighting strategies. Each variation of the algorithm, including the canonical PSO, was tested with several population structures as well. The results obtained showed that the FIPS consistently performed better than the canonical PSO in all measurements.

2.4.2 Bare Bones Particle Swarm and Gaussian FIPS

In 2003, Kennedy suggested a PSO variant with the particle velocity component being entirely eliminated, and it was known as the Bare Bones Particle Swarm (BBPS) [10]. Observations of the constriction coefficient PSO showed that particles would orbit the point $\vec{P_m}$ if the global and personal best points were kept constant, eventually converging to that point. A histogram of the points evaluated by this particle over one million iterations under this condition was created. This histogram showed a tidy and distinct bell-curve distribution centred on the point $\vec{P_m}$. Based on this finding, it was theorized that particle velocity, orbiting, and all the other difficult-to-understand aspects of particle behaviour, could simply be replaced. Instead of a particle moving according to its velocity, the BBPS algorithm updates particle positions by sampling a Gaussian random distribution. The particle's velocity is removed entirely from the

algorithm and the position update equation simply becomes:

$$\vec{P_m} = \frac{\vec{P_i} + \vec{P_g}}{2},$$
$$\vec{X_i} = \vec{\mathcal{N}}(\vec{P_m}, \vec{\sigma^2}),$$

where $\vec{\mathcal{N}}$ represents a function which creates a Gaussian random vector dimension-by-dimension, centred on $\vec{P_m}$. $\vec{\sigma^2}$ is a vector of standard deviations for each dimension. In the basic case where $\vec{P_m}$ is the average of the personal and global best points, the standard deviations are typically set to be the absolute values of the difference between these points for each dimension.

This BBPS algorithm was tested against the canonical PSO and performed competitively, outperforming the canonical PSO on some functions. This confirmed the hypothesis that the velocity was not actually mandatory for a PSO system, and that the complicated analysis required for the velocity method could be simply replaced by such a random sampling. While the BBPS algorithm has a very similar high-level swarm behaviour compared to the canonical PSO, the movement of individual particles is somewhat different. With the removal of the velocity component, particles no longer "fly" through the solution space. However, the BBPS algorithm still retains enough aspects of the original algorithms to be considered a particle swarm system.

This method of updating a particle's position according to a random distribution rather than a velocity, can be easily combined with the FIPS algorithm. As well as creating the basic BBPS algorithm, Kennedy also created a BBPS-FIPS hybrid known as the Gaussian FIPS. This Gaussian FIPS algorithm simply computed $\vec{P_m}$ and the standard deviations according to the FIPS methodology and then used the position update method of the BBPS algorithm.

In 2004, Kennedy performed a deeper comparison of the BBPS and the canonical PSO algorithms [11]. This comparison found that when using the asymmetrical initialization method intended to expose certain biases, the canonical PSO performed far better than the BBPS. This result suggested that the bell-curve distribution originally observed may not, in fact, be well represented by the Gaussian distribution, despite

the visual similarity. Deeper analysis of the bell-curve histogram showed a high kurtosis[2], increasing over time. This discovery proved that a Gaussian distribution, which has a very low kurtosis, did not accurately reproduce the histogram distribution. A doubly-exponential distribution was thus proposed to replace the Gaussian distribution and tests showed that the BBPS with this distribution performed comparably with the canonical PSO even when it involved asymmetrical initialization. However, such a doubly-exponential distribution does not capture the property of an increasing kurtosis over time, and so it still did not perfectly capture the nature of the observed histogram.

2.4.3 Dynamic Parameters and Adaptive PSO

The parameters of the PSO system have dramatic effects on the algorithms behaviour and performance. In order to get the best results, these parameters must be tweaked for each problem that PSO is applied to. Additionally, different parameter values may be more valuable in certain situations, but a hindrance in others. Since, in the canonical PSO, parameters remain fixed throughout the entire search, they must be balanced so as to perform well, overall. Over the years, there have been many attempts to address this issue.

The most straightforward strategy to address this issue is to simply adjust the parameters based on time. When the inertial weighting was first introduced to PSO, it was suggested that linearly decreasing the value over the course of the search could yield better results. A higher inertia value helps prevent particles from becoming trapped in locally optimal areas during exploration, and so this is valuable in the early stages of the search in order to discover generally good areas. A lower inertia value, on the other hand, increases the particle's ability to exploit these generally good areas to determine the local optima of the area. With an inertia value decreasing over time, the hope is to benefit from a high inertia during early stages, allowing for exploration, and a lower value during the later stages, allowing the particles to converge to the optima.

[2]Kurtosis is a measurement of the "peakedness" of a random distribution and the "fatness" of its "tails". A high kurtosis corresponds with a sharper peak falling rapidly to thicker tails which extend further than a distribution with a lower kurtosis.

Experiments utilizing this paradigm showed a dramatic increase in performance in situations where the algorithm was given a limited number of iterations [26]. Similar efforts have been made with varying the acceleration constants over time [22]. If the acceleration toward the particle's personal best is greater than the acceleration toward the global best, particles will better explore the search space. Conversely, if the global best acceleration is higher, particles will converge faster to an optimum. Based on this philosophy, the acceleration coefficients were modified over time in such a way that early iterations would have a higher *personal* best acceleration, but subsequent iterations would have a higher *global* best acceleration. This strategy showed dramatic improvements on unimodal functions, but struggled on multimodal functions.

While these methods work well when there is a hard limit to the number of iterations allowed, they are essentially blind to the state of the population. It is simply assumed or hoped that as the parameters shift the swarm to a more convergent behaviour, the population will have identified good areas to exploit. A more sophisticated approach to this is to adapt parameters based on properties of the swarm itself. An example of this is the so-called Adaptive PSO proposed by Zhan *et al* in 2009 [30]. The Adaptive PSO dynamically adjusts the inertia and acceleration coefficients using so-called "Evolutionary State Estimation" (ESE). ESE is a process of determining a populations "evolutionary factor" based on the distribution of particles. The "evolutionary factor", f, is a ratio of the average Euclidean distance of all the particles from the global best solution and the range of average Euclidean distances of all the particles to one another. This is formulated by:

$$f = \frac{d_g - d_{min}}{d_{max} - d_{min}} \in [0, 1],$$

where d_g is the average Euclidean distance of all the particles from the global best, d_{min} is the smallest average-distanced particle, and d_{max} is the greatest average-distanced particle. This "evolutionary factor" value is then classified into one of four sets which represent four "evolutionary states". These states are "Exploration", "Exploitation", "Convergence" and "Jumping Out". "Jumping Out" refers to a state where most of the swarm is clustered around a global best solution with some particles

being trapped in the local optima. In this situation the particles trapped in the local optima must "jump out" to join the rest of the swarm. These states are classified using fuzzy membership functions established through experimentation. In general, the lowest values of f correspond to exploration, followed by exploitation, convergence and jumping out as f increases.

Values for the inertial coefficient and the acceleration coefficients are set, at each iteration, according to the value of f, in order to adapt the swarm's behaviour to its current state. The inertial coefficient, ω, is set directly according to a sigmoidal function of f where lower values of f correspond to lower values of ω and higher values of f to higher values of ω.

Values for the acceleration coefficients are initialized to 2 and are adjusted based on the evolutionary state determined by the fuzzy membership function of f. If the swarm is determined to be in the Exploration state, the personal best acceleration is increased while the global best acceleration is decreased. This helps particles better explore their own areas and avoid local optima. In the state of Exploitation the personal best acceleration is increased marginally, while the global best acceleration is decreased marginally. In the Convergence state, both the personal best and global best acceleration coefficients are increased marginally. This is done to increase the speed of convergence of particles toward the global best solution without causing the personal best to be ignored entirely. Finally, in the Jumping-Out state the personal best acceleration is decreased while the global best acceleration is increased. This shifts the particle's attraction away from its personal best to a more drastic degree, allowing particles to escape their local optima and join the majority of the swarm around the global best, leading to better convergence.

This method of adapting the parameters was tested by combining it with a number of popular PSO variants and comparing its performance with the respective original algorithms. Tests showed that applying the adaptive parameter method resulted in drastic performance boosts on the majority of tested PSO variants, with some converging to the global optima up to twenty five times faster.

2.4.4 Hybrid PSO

Due to its relative simplicity and flexibility, PSO has been successfully hybridized with many other algorithms over the years. The most popular PSO hybridization introduces concepts from Genetic Algorithms (GA), due to the structural similarities shared with PSO. Both PSO and GA are population-based meta-heuristics for black-box optimization and both follow the same general process of iteratively updating their populations to search for optimal solutions. The main difference between the two algorithms lies in the methods and principles used to update the population at each iteration. PSO individuals fly through the solution space, effected by their own memory and the other individuals in the population. In GA however, a new population is generated for each iteration using the process of selection, breeding and mutation; inspired by the theory of evolution. At a glance, a new generation of individuals is created by randomly combining individuals of the previous generation based on their relative fitness values. The importance is that the movement and dynamics of the populations of the two systems are very different, each with its own strengths and weaknesses.

Hybridization of PSO and GA was first done in 1998 by Angeline, who applied the GA concept of selection to the PSO population [2]. Selection is a concept central to GA which models the natural phenomenon of survival of the fittest. In GA, selection is used to determine which individuals in the population are chosen to breed and pass their "genes" to the next generation. Individuals with higher values are generally more likely to be selected than others, depending on the specific method of selection. This causes more individuals to be created in generally good areas of the solution space, which causes the population to move toward these areas. In PSO, these generally good areas are found as a result of the global, or neighbourhood best points of attraction pulling more particles toward them. However, applying selection to the PSO population can improve this behaviour and increase the rate of convergence for some problems.

In the original work by Angeline, this concept was applied using the so-called tournament selection method, commonly used in GA. At each iteration of the PSO algorithm, after all particles have been evaluated, particles are randomly separated

into a number of tournament groups. Within these tournament groups, each particle is then assigned a value based on the number of particles in that tournament group with values lower than its own. The entire population is then sorted using these new values, and so relatively good particles tend to be ranked higher than relatively bad particles, but not strictly so. The top half of this sorted population is then "selected" and these selected particles will be updated according to the standard PSO velocity and position update methods. In GA systems, individuals which are not selected are simply discarded. However, in PSO, discarding individuals would cause the population to shrink exponentially, and so Angeline suggested a slightly different method. Particles which have not been selected are randomly assigned a position and velocity of a selected particle, but retain their own personal best found points. The personal best points are not copied from the selected particles in order to maintain swarm diversity. Experiments performed on this selection PSO showed a dramatic increase in performance on most test functions, but not on all. The selection based PSO seemed to identify generally good areas much faster than the standard PSO, but was less effective at finding the best point within those general areas.

Other GA concepts, such as mutation and breeding, have also been hybridized with PSO. A PSO with breeding was first presented in 2001 by Lovbjerg, Rasmussen and Krink [15]. Breeding, via the process of "cross-over", is a method of creating a new individual modelled after the recombination of DNA in nature to create offspring. In GA systems, cross-over is used to create the next generation of individuals. There are various methods of cross-over, but in general, a new individual is created by randomly combining two individuals of the current generation. This method of "movement" is quite different from the one invoked by the PSO; it is capable of making large leaps through the solution space, and so a hybrid method may help prevent PSO populations from becoming trapped in local optima.

In the original work by Lovbjerg *et al*, cross-over was implemented as a low probability event meant to allow particles the ability to "jump out" of local areas rather than a main form of movement. The Cross-Over PSO algorithm is identical to the standard PSO with the additional incorporation of the breeding phase in each iteration. After updating all particle velocities and positions, the breeding phase begins.

Each particle in the population has a small chance to be selected for breeding. Among the particles selected for breeding, two are randomly selected, cross-over is performed to create two offspring and the two original particles are discarded. This is done until there are no longer any particles marked for breeding, after which the algorithm continues on to the next iteration. The cross-over method used is the typical one applicable for real-valued GAs. Positions of child particles are created on a per dimension basis, as randomly weighted averages of its parents, effectively sampling the hypercube defined by the two parents. The velocities of the offspring particles is similarly set to a randomly weighted average of the parent particles velocities, but the result is normalized to the magnitude of one of the parent velocities. The experimental results showed this cross-over PSO did not, in fact, perform provably better than the canonical PSO.

Mutation is a concept used in GA which models the natural process of genetic mutation. Mutation is essential to GA, as genetic cross-over can only create new positions lying between currently existing positions. When creating the next generation of individuals in a GA system, mutation is applied as a small chance for random change. This is typically done on a per-dimension basis, making random modifications to randomly chosen dimensions. Higher rates of mutation lead to a more randomized search and to a greater variety among the population, while lower rates lead to faster convergence and less variety in the GA. In a PSO system, the randomness of accelerations can be thought of as a form of mutation, but a key difference is that the random acceleration cannot mutate to move away from the point of attraction.

Mutation was introduced into PSO by Higashi and Iba in 2003 [9]. Higashi and Iba proposed adding mutation to PSO as a tool to help particles escape local optima and increase diversity in the swarm. In their model, particles are randomly selected for mutation based on a mutation rate which decreases linearly over the course of the run. This decreasing mutation rate means that the early stages of the algorithm will be more exploratory, as particles are more likely to move randomly. As the algorithm reaches the end of its run, the lower mutation rate allows particles to then converge without randomly moving away from the optima. This gives particles a chance to escape from local optima that they may have been trapped in during the early stages,

but still allows for rapid convergence. A particle chosen for mutation will have a randomly chosen dimension, d, mutated according to the equation:

$$\vec{X}_d = \vec{X}_d * (1 + \mathcal{N}(\alpha)),$$

where $\mathcal{N}$ is a Gaussian distribution and α is the amplitude of mutation. In Higashi and Iba's work, this amplitude was set to be one tenth of the range of the search space in the chosen dimension. Experimental results showed that the addition of this mutation step increased the overall performance considerably, outperforming the canonical PSO in most tests performed.

The next logical step is to combine all three of these key GA concepts to create a truly hybridized PSO/GA algorithm. An example of this is the so called "Breeding Swarms" hybrid algorithm proposed by Settles and Soule in 2005 [24]. The Breeding Swarms algorithm fully hybridizes the GA and PSO algorithms, updating the population of individuals using equal parts PSO and GA methodology. Individuals are particles as in PSO, with positions, velocities and which track their previous best found locations. At each iteration of the algorithm, a percentage of the population is updated according to the canonical PSO velocity/position update method while the remaining individuals are updated according to GA methods. The percentage of the population to be updated with GA methods is referred to as the "Breeding Ratio" (ψ) and is provided as a parameter. After evaluating the individuals, the worst ψ percent of the population is discarded. The remaining individuals are then updated according to the canonical PSO velocity method. After these individuals have been updated, the discarded individuals are replaced using the GA process of selection and breeding. Individuals are selected for breeding using a standard tournament selection, but cross-over between breeding individuals is performed in a specific manner to promote swarm diversity. The crossover operation introduced, known as "Velocity Propelled Averaged Crossover", creates two children, c_1 and c_2 from two parents p_1 and p_2. Each child has its position initialized as the midpoint between the two parents. These positions are then offset according to a random negative weighting of the respective parents velocities. These points are offset by such a negative weighting of the velocity with the hope of increasing the swarm's diversity by pushing children away from the

areas that the parents are exploring. The children further inherit velocities from their respective parents, but do not inherit the "memory" of their parents, as the previous best points for each child are initialized to that child's initial position. Mutation is then applied to the entire population as proposed by Higashi and Iba [9]. This Breeding Swarms algorithm was tested against basic GA and PSO systems on a variety of problems. The Breeding Swarms algorithm performed somewhat better than the canonical PSO and much better than the basic GA. Particularly, it was successful in avoiding local optima in which the basic PSO and GA would become trapped in.

Since these initial efforts, many new strategies of creating hybrid PSO/GA algorithm have been attempted. One such example is a hybrid algorithm which applies fuzzy logic to dynamically combine the population update strategies of PSO and GA algorithms [4].

2.5 Evaluation of Particle Swarm Optimization Algorithms

PSO and its variants are stochastic meta-heuristics for black-box optimization, which are not guaranteed to find the optimal solution. Because of this, evaluation and comparison of these algorithms is largely empirical. The strength and motivation of meta-heuristic algorithms is their flexibility, and their ability to operate on black-box functions without a knowledge of the function itself. Therefore, for a meta-heuristic to be considered "good", it should perform well on a wide variety of functions, or very well on *some* subset of desired functions.

Such an evaluation phase is complicated by virtue of the so-called "No Free Lunch" theorem. The "No Free Lunch" theorem is a piece of mathematical folklore, which when applied to the field of meta-heuristic optimization, implies that there cannot be a *single* best meta-heuristic optimization algorithm. The "No Free Lunch" theorem for meta-heuristic optimization was proven in 1997 by Wolpert and Macready [29] who showed that the performance of all meta-heuristic algorithms must be equal when averaged over *all* possible functions. From a naive perspective, it would appear

as if this conclusion implies that it is futile to work on improving or creating new meta-heuristic algorithms, because any new meta-heuristic algorithm would have the same performance as any existing algorithm over all possible functions. This is not necessarily true in practice, however, and meta-heuristics remain a popular field of study. Though meta-heuristics must perform equally well when averaged over *all* possible functions, the fact is that any given meta-heuristic could perform better than any other on a certain desired *subset* of functions. Because of this, the motivation for developing or creating new meta-heuristics is not based on the urge to determine a meta-heuristic which is superior to all other meta-heuristics, but rather to find a meta-heuristic which performs well on a pre-defined subset of desired functions, or on functions which possess certain properties.

By virtue of this goal, empirical testing of meta-heuristics must be carried out on a carefully selected suite of test functions. Performance metrics are typically gathered for each test function in the test suite, and the algorithm is evaluated by drawing conclusions from these results.

2.5.1 Metrics

Black-box optimization algorithms search for the optimum value by exploring the solution space of the objective function. In order to explore this space, these algorithms must perform function evaluations to assign values to the points in that space. As such, when performing empirical analysis on black-box optimization algorithms, the main performance metric used is the number of function evaluations performed. In the case of population-based algorithms, instead of directly measuring the number of function evaluations performed, one quantifies this by instead measuring the number of iterations executed. This is because in most population-based algorithms, each individual in the population is evaluated exactly once per iteration and population sizes remain constant, and so the number of iterations provides a more intuitive measurement.

Meta-heuristics are not guaranteed to find the global optimum of the objective

function, and so, typically, a maximum number of iterations is specified as a termination condition. Because of this, for difficult functions where the algorithm cannot consistently find the optimum, the measurement of iterations is not sufficient. Usually, one uses another primary metric, which is the best value the algorithm can find within the number of allowed iterations.

Many meta-heuristic algorithms employ stochastic processes to allow a more exploratory search. Because of this, it is not sufficient to run single tests or "cherry pick" the best results. Rather, tests must be repeated a sufficient number of times and the results analyzed stochastically to obtain a more meaningful measurement of the algorithm's performance. Typically, this simply means evaluating based on the mean value of each gathered metric.

Additionally, for these reasons, meta-heuristics are not typically evaluated in terms of the computational complexity of the algorithm, as it is generally assumed that the cost of evaluating the objective function dominates this complexity.

Due to the implications of the "No Free Lunch" theorem, when creating a new meta-heuristic, the goal is not always to simply outperform the previous algorithms on the selected test functions. A secondary motivation of empirical testing is often to identify the properties or biases of the new algorithm, or to discover which classes of problems the algorithm is able to solve better than previous algorithms. To discover these traits, the selection of a good test suite is essential. When testing to see which of these test functions the algorithm performs well on, instead of simply recording the average best value, or the average number of iterations required, a graph is also plotted. This graph plots the average best value found by the algorithm at each iteration. Plotting and comparing algorithms on this graph can provide insight into the behaviours and properties of each algorithm, which cannot be easily deduced from the other metrics. For example, properties such as a tendency to become trapped in local optima can be seen in a graph which improves quickly to some non-optimal value and then sharply plateaus.

Many meta-heuristics involve parameters which can be used to tweak their performance and increase their flexibility. In practice, a large number of parameters makes the algorithm more difficult to use, as each parameter must be understood

and adjusted properly to get the most out of the algorithm. Because of this, having a lower number of parameters, without destroying the flexibility of the algorithm, can be considered as an improvement over a competing algorithm possessing more parameters. However, this is largely a secondary consideration.

2.5.2 PSO Specific Testing Concerns

When evaluating algorithms based on, or extending, the canonical PSO algorithm, it is important to account for biases. The canonical PSO has been shown to display a number of biases, which must be considered when performing empirical tests. When evaluating an algorithm which extends the canonical PSO, it is important to determine whether or not the algorithm has inherited any of these biases. The biases displayed by the canonical PSO were exposed by performing specific empirical tests,and consequently, determining whether or not a PSO-based algorithm has inherited these biases is as simple as repeating these tests. The canonical PSO has been shown to display two main biases:

- A bias toward the centre of the initialization range, and
- Rotational variance, resulting in a bias toward areas lying on or near axis or diagonals of the solution space.

Each of these biases can be tested for with some simple test strategies. Testing for a bias toward the centre of the initialization range can be done using an asymmetrical initialization strategy. Many optimization benchmark functions have solutions lying at the origin of the space for the sake of simplicity and clarity. Rotational variance can be exposed by rotating the solution space. These specific tests are typically performed on each test function and compared to the results of more general tests done on those test functions. If the algorithm is not significantly biased, the performance on these specific tests should be roughly the same as the general tests.

2.6 Test Functions

The evaluation and comparison of meta-heuristic algorithms relies mostly on empirical testing. Because of this, and the implications of the "No Free Lunch" theorem, a good suite of test functions is essential to allow meaningful claims of a meta-heuristic's performance. Function optimization has been studied extensively for decades and there have been countless functions created for the purpose of testing these methods and algorithms. Some of these test functions are inspired by, or taken directly from, practical sources, while most are created specifically to provide challenges to optimization algorithms. Many of these test functions are designed to specifically hold certain traits or properties which have been determined to highlight, or expose, different behaviours of the algorithms being tested on them. These traits can be used to classify test functions into many different groupings. Selecting an excessive number of test functions from within common groupings may result in redundancy, as an algorithm which works well on a function with some specific trait is likely to work well on other functions sharing that trait. Creating a good test suite, then, requires test functions to be selected such that a variety of desired traits are covered, rather than many similar test functions.

However, many of these test functions also display common properties which may be easily exploited by an optimization algorithm, leading to misleading performance results. Such properties include symmetries, regular spacing of local optima or optima lying at the origin of the space. To this end, processes known as "function generators" have been investigated to produce a wider array of test functions while minimizing these undesired properties [3].

PSO algorithms have been a popular topic of research for many years, and as a result, a number of test functions have become accepted as the standard baseline or benchmark set of tests for these algorithms. A common set of functions is desirable for two reasons. First, these common test functions have already been determined to provide good insight into PSO-based algorithms. Secondly, having a common set of test functions allows for easier comparison of related algorithms without requiring these algorithms to be implemented and tested on new test functions every time.

These benchmark test functions are described here.

2.6.1 The Sphere Function

The Sphere function is a commonly used test function for evaluating PSO algorithms and almost always included when performing tests. The Sphere function is unimodal and radially symmetric, with only a single optimum. This function is a very "easy" test function, as the absence of other local optima means that any reasonable optimization algorithm should find the approximate global optimum point without trouble. Because of this, on its own, the function does not provide a good measurement of the optimizing "power" of an algorithm. However, the Sphere function can provide a good measurement of how *fast* an algorithm will converge to the optimum in a trivial situation.

The Sphere function is simply the squared Euclidean distance from the origin, formulated as:

$$f(\vec{x}) = \sum_{i=1}^{d} x_i^2$$

Typically search boundaries are set to the range of $x_i \in [-600, 600]$. The global optimum has value 0 at the point $\vec{x} = [0, 0, ..., 0]^T$. A plot of the function in two dimensions is given in Figure 2.1.

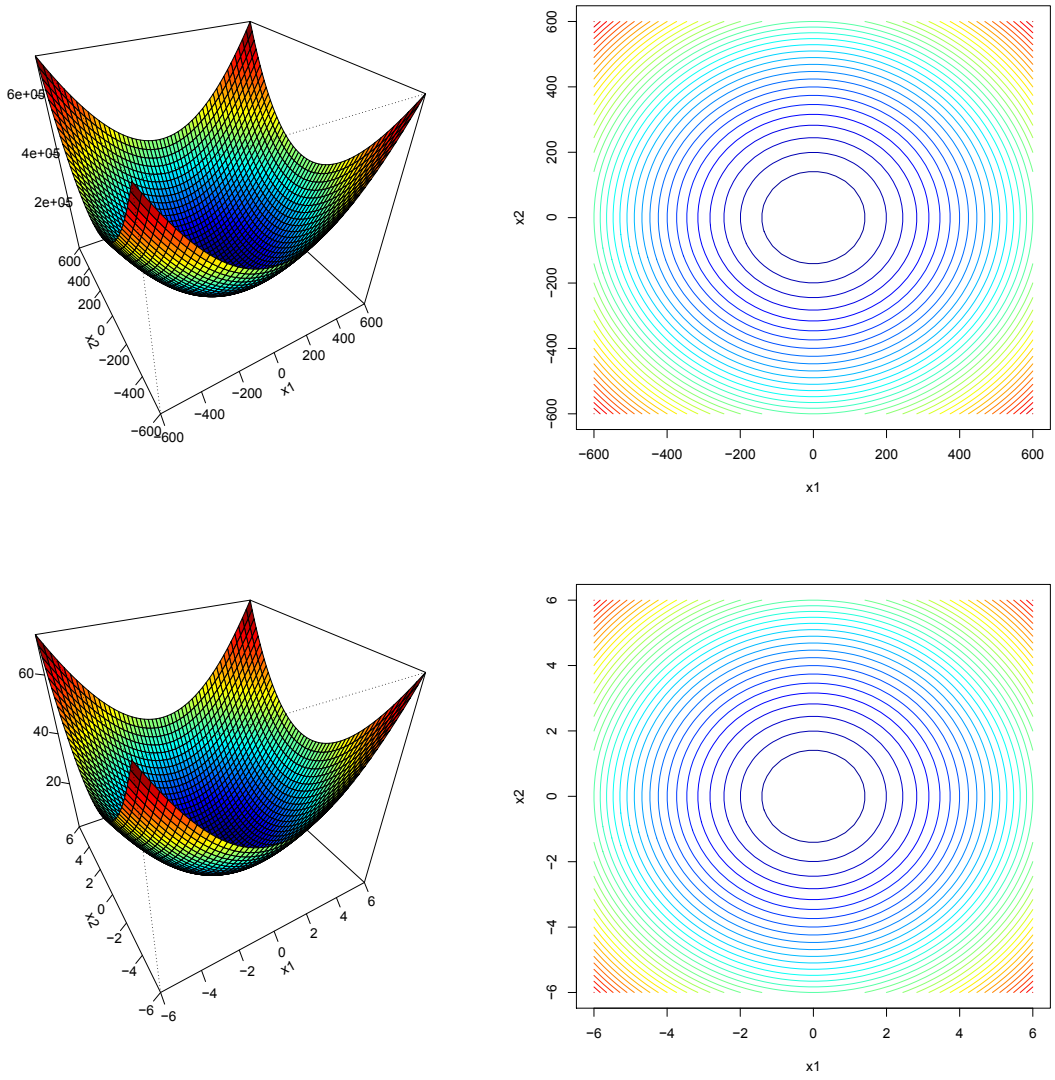


Figure 2.1: Surface and contour plot of the Sphere function in two dimensions, and a fine-grained view of the areas near the optimum.

2.6.2 The Rosenbrock Function

The Rosenbrock function is another commonly used test function. This function is two dimensional, though generalized versions can be created by taking the sum, or average, of multiple Rosenbrock functions coupled over an even number of dimensions. The Rosenbrock function is unimodal, with a single optimum. However, this optimum lies in a long, relatively "flat" "banana shaped" valley. While it is trivial to locate this valley, locating the true optimum within it is more difficult. In order to reliably find this optimum the optimizer must both quickly identify the generally good area of the valley, and effectively search that relatively "flat" area without getting stuck. Another minor detail is that the curve of this valley near the optimum is not parallel, or roughly parallel, to either axis. This detail is of interest when testing an optimizer related to PSO, which has been shown to favour movement parallel to an axis.

The Rosenbrock function is formulated as:

$$f(x_1, x_2) = (1 - x_1)^2 + 100(x_2 - x_1^2)^2$$

The search boundaries for the Rosenbrock function are typically set to $x_1 \in [-2, 2], x_2 \in [0, 3]$ and the global optimum has value 0, lying at the point $(x_1, x_2) = [1, 1]$. A plot of the Rosenbrock function is given in Figure 2.2.

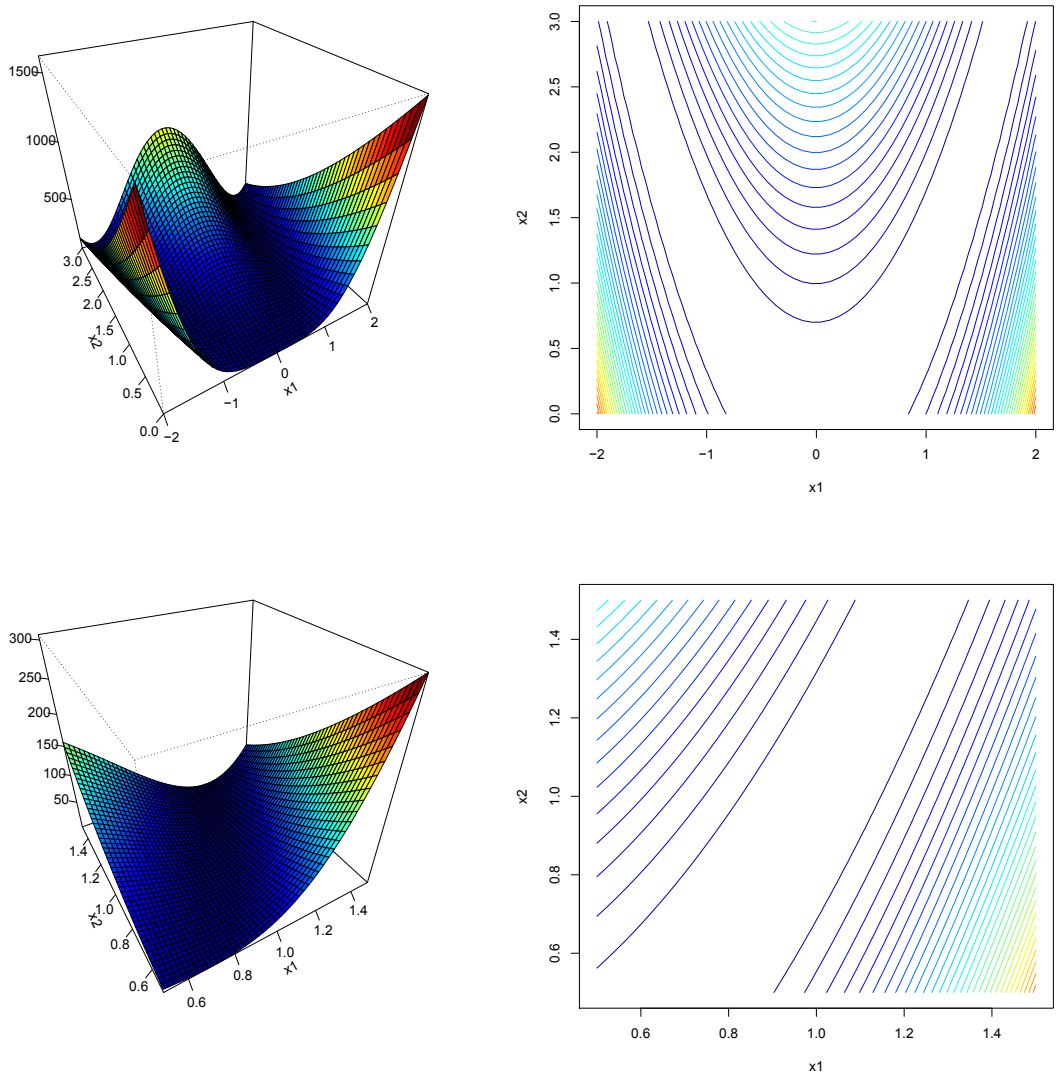


Figure 2.2: Surface and contour plot of the Rosenbrock function, and a fine-grained view of the areas near the optimum.

2.6.3 The Rastrigin Function

The Rastrigin function is a difficult function which is used in nearly all test suites for PSO-based algorithms. This function is a Sphere function with sinusoidal waves that are added parallel to the axes, creating many local optima. Further, the function is symmetrical about each axis. In particular, the globally optimal area is surrounded by a number of local optima lying along the axes. These local optima are very close in value to that of the global optimum. It has been observed that the canonical PSO algorithm performs exceptionally well on this function compared to other popular meta-heuristics. This may be a result of the surface of the function which favours movements parallel to the axes, which is a bias held by the canonical PSO.

The Rastrigin function is formulated as:

$$f(\vec{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10cos(2\pi x_i)\right]$$

The search boundaries for the Rastrigin function are typically set to $x_i \in [-5.12, 5.12]$ and the global optimum has value 0, lying at the point $\vec{x} = [0, 0, ..., 0]^T$. A plot of the function in two dimensions is given in Figure 2.3.

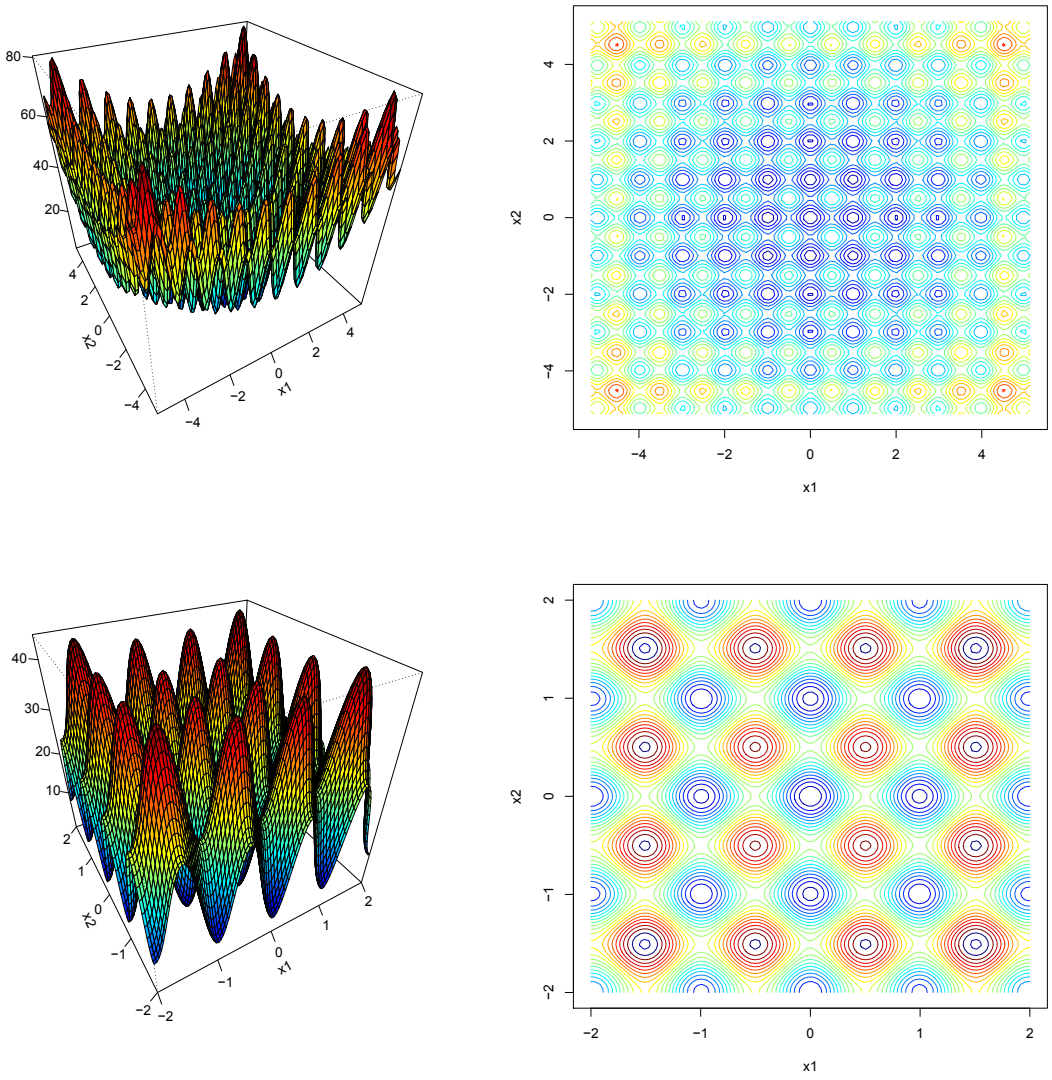


Figure 2.3: Surface and contour plot of the Rastrigin function in two dimensions, and a fine-grained view of the areas near the optimum.

2.6.4 The Griewank Function

The Griewank function is another popular test function for testing optimization algorithms. Similar to the Rastrigin function, the Griewank function has a general spherical shape with an added wave-shaped noise. However, the Griewank function has a much larger number of local optima surrounding the optimal area, increasing exponentially with the number of dimensions. Interestingly, it has been shown that the Griewank function actually becomes somewhat "easier" as the number of dimensions is increased, despite the exponentially increasing number of local optima [14].

The Griewank function is formulated as:

$$f(\vec{x}) = 1 + \frac{1}{4000}\sum_{i=1}^{d} x_i^2 - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right)$$

The search boundaries for the Griewank function, plotted in Figure 2.4, are typically set to $x_i \in [-600, 600]$, and the global optimum has value 0, lying at the point $\vec{x} = [0, 0, ..., 0]^T$.

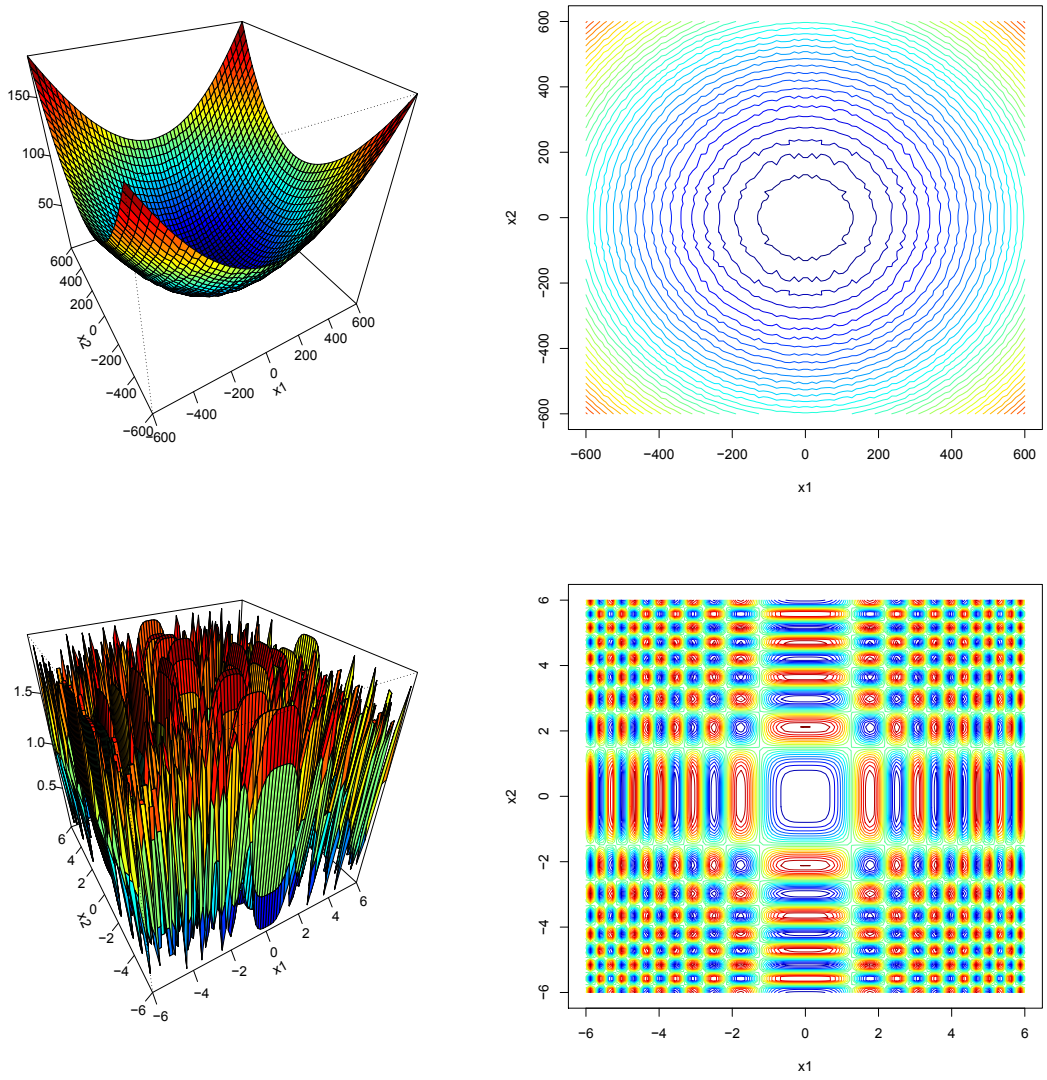


Figure 2.4: Surface and contour plot of the Griewank function in two dimensions, and a fine-grained view of the areas near the optimum.

2.6.5 The Ackley Function

Ackley's function is a test function with many local optima. Similar to the Rastrigin function, the Ackley function consists of a basic unimodal surface with added sinusoidal waves to create the local optima. However, the basic surface to which the noise is added is significantly different. This shape is a fairly flat surface with the global optimum lying within a sharp pit. The result of this is that the global optimum is fairly easy to find within the generally good area around it, but that "generally good area" is "generally more difficult" to locate than in the case of the Rastrigin or Griewank functions.

The Ackley function is formulated as:

$$f(\vec{x}) = 20 + e - 20e^{-\frac{1}{5}\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2}} - e^{-\frac{1}{d}\sum_{i=1}^{d} cos\left(2\pi x_i^2\right)}$$

The search boundaries for the Ackley function have a large impact on the difficulty of the function, as a larger search range makes the optimal area more difficult to find. Typically the boundaries are set to $x_i \in [-30, 30]$. The global optima has the value 0, lying at the point $\vec{x} = [0, 0, ..., 0]^T$. A plot of the function in two dimensions is given in Figure 2.5.

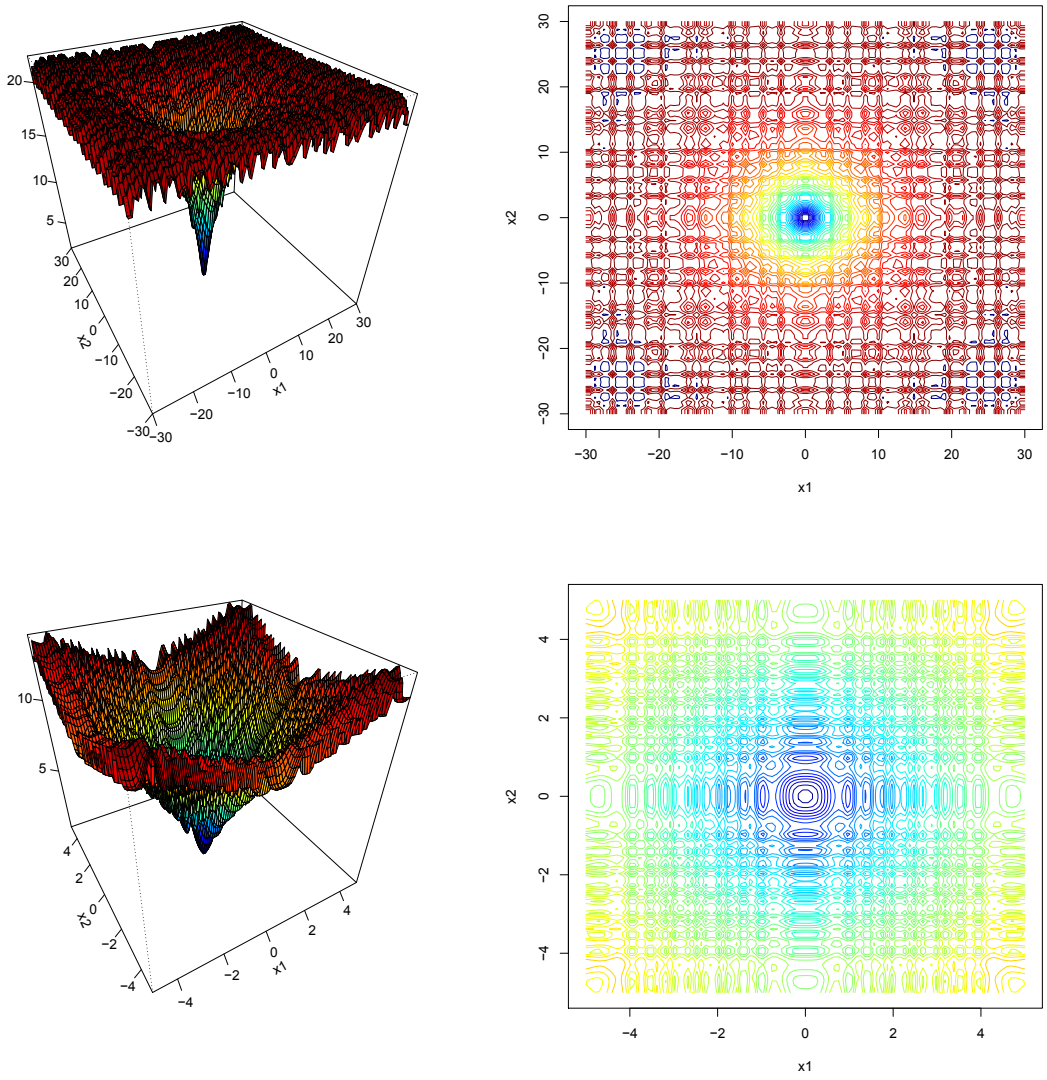


Figure 2.5: Surface and contour plot of the Ackley function in two dimensions, and a fine-grained view of the areas near the optimum.

2.6.6 The Schaffer F6 Function

The Schaffer F6 function is a difficult optimization test function used in many test suites. Unlike many other optimization test functions, the Schaffer F6 function does not have a roughly spherical or parabolic surface. It rather consists of a flat plane with an added sinusoidal wave radiating from the optimal area. This wave dramatically increases in amplitude as it approaches the global optimum, resulting in high frequency circular hills and valleys, where the valley containing the global optimum is surrounded by the highest peaks in the function. As in the case of the Rosenbrock function, the Schaffer F6 function is only two dimensional, but higher dimensional generalizations can be made by using the sum or average of multiple Schaffer F6 functions. Another property of the Schaffer F6 function worth noting is that it is radially symmetric.

The Schaffer F6 function is formulated as:

$$f(x_1, x_2) = 0.5 + \frac{sin^2(\sqrt{x_1^2 + x_2^2}) - 0.5}{[1 + 0.001 * (x_1^2 + x_2^2)]^2}$$

The search boundaries for the Schaffer F6 function are typically set to $x_1 \in [-50, 50], x_2 \in [-50, 50]$ and the global optimum has value 0, lying at the point $(x_1, x_2) = [0, 0]$. A plot of the function in two dimensions is given in Figure 2.6.

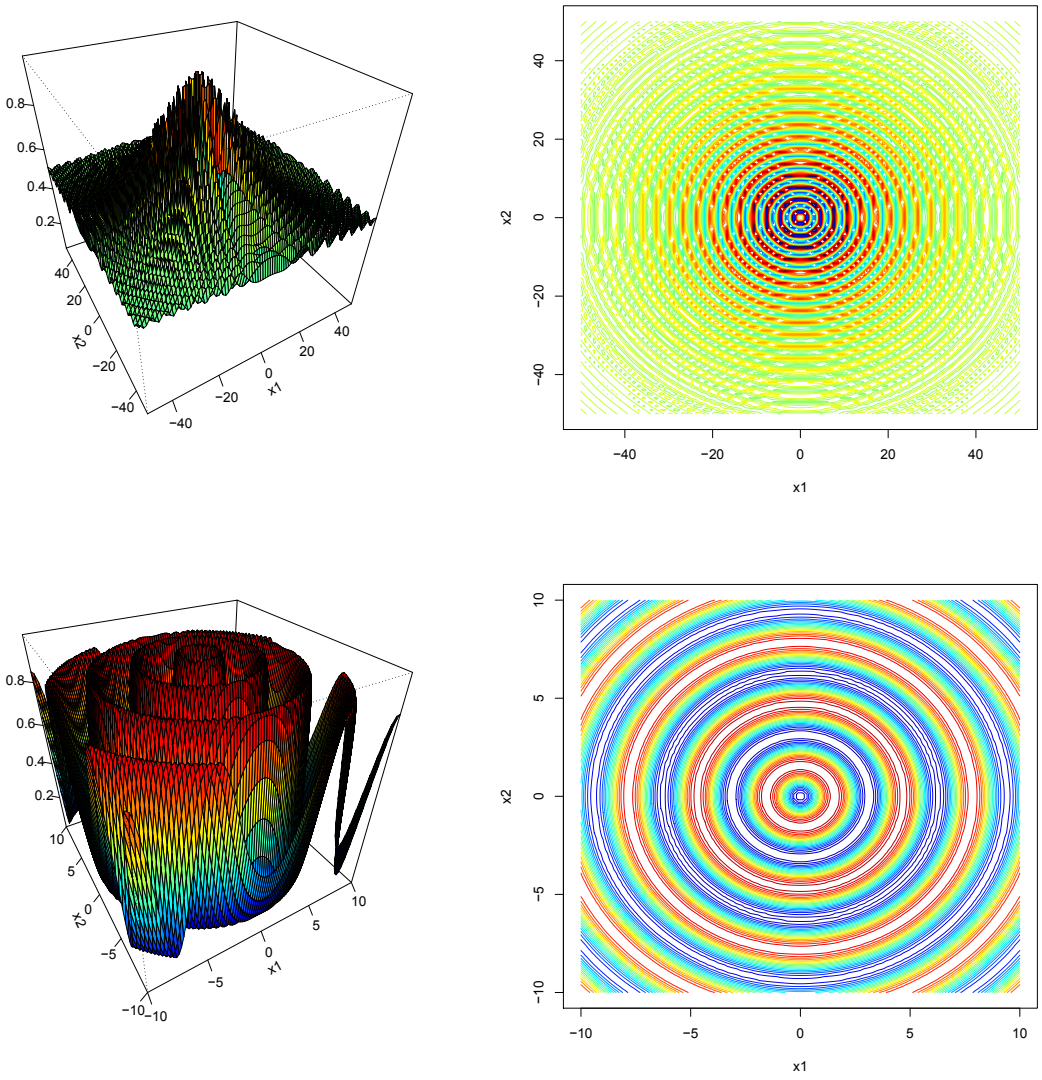


Figure 2.6: Surface and contour plot of the Schaffer F6 function, and a fine-grained view of the areas near the optimum.

2.7 Conclusions

In this chapter we have introduced the concept of Swarm Intelligence, a natural phenomena which has inspired a field of study in computation. In particular, we have explored the creation and history of the Particle Swarm Optimization algorithm, which has remained a popular subject of research and development since its inception in 1995. Notable advances in the field, including FIPS, BBPS and Hybrid-PSO have been discussed, as well as various efforts made to analyze the inner workings of the PSO system. Finally, we have discussed the evaluation of meta-heuristic algorithms, and stated the problems and limitations of these algorithms. We have also visited the testing process used to achieve these evaluations as well as common functions used in this testing process. These functions include the Sphere function, the Rosenbrock function, the Rastrigin function, the Griewank function, the Ackley function and the Schaffer F6 function.

Chapter 3

Novel Contributions of the Thesis

3.1 Introduction

This chapter, which deals with our fundamental contributions, presents our newly proposed Particle Field Optimization (PFO) algorithm and the abstractions and new perspectives involved. Beginning with the Bare Bones Particle Swarm (BBPS) algorithm, we identify an opportunity for a further abstraction of the process. Using this abstraction, we present a newly-discovered perspective on the behaviour of the BBPS algorithm and the processes underlying its behaviour. This new perspective allows us to discover new opportunities for modifying the search behaviour and the underlying principles at work. We, consequently, propose a number of modifications to the BBPS algorithm, taking advantage of these novel and unexplored opportunities, and this effort culminates in the distinct PFO algorithm. The PFO algorithm itself is discussed and formalized here. Finally, we also highlight a number of weighting schemes. These weighting schemes can be applied to the PFO algorithm to further modify the search behaviour.

3.2 Abstracted Bare Bones PSO

3.2.1 The Bare Bones Particle Swarm Model

The Bare Bones Particle Swarm (BBPS) algorithm [10] is a PSO variant which seeks to emulate the high-level behaviour of the basic PSO algorithm, while simultaneously using a much simpler particle update strategy. Observations of the basic PSO system suggested that very similar behavioural patterns could be achieved without the complicated velocity *and* acceleration components. Further, observations of a *single* particle using the standard velocity strategy, with the system in a state of swarm stagnation (i.e. when the personal and global best points remain constant), produced a histogram which displayed a distinct and tidy bell-curved shape. This phenomenon led to the conjecture that rather than dealing with the particle velocities and accelerations, each particle could be updated by merely sampling a random distribution constructed to approximate the observed histogram.

The BBPS algorithm entirely eliminates the velocity component of the particle and merely determines the particle's next position by sampling a Gaussian random distribution. This Gaussian distribution is constructed and sampled individually for each dimension. The mean of this distribution is set to be the midpoint between the global best found point and the particle's own best found point, and the variance is set as the absolute distance between the global and personal best points for each dimension. Given a particle i with position $\vec{X}_i$, whose personal best point is $\vec{P}_i$ and for which the global best point is $\vec{P}_g$, the particle's next position is determined according to:

$$\vec{P_m} = \frac{\vec{P}_i + \vec{P}_g}{2},$$
$$\vec{X}_i = \vec{\mathcal{N}}(\vec{P_m}, \vec{\sigma^2}),$$

where $\vec{\mathcal{N}}$ represents a function which creates a Gaussian random vector dimension-by-dimension, centred on $\vec{P_m}$. $\vec{\sigma^2}$ is a vector of standard deviations for each dimension, set to be the absolute values of the difference between the $\vec{P}_i$ and $\vec{P}_g$ points for each dimension.

This update strategy is marginally more abstract than the basic PSO velocity strategy, since the particles no longer "fly" through the space, but the general behaviour of the swarm remains the same. At a higher level, the algorithm still involves a population of particles which move through the solution space, and which are influenced by the memories of their best found positions and the communication of the global best found position.

3.2.2 Abstracting the BBPS Model

The BBPS model discards the velocity component of the basic PSO model and introduces a more abstract method for updating the positions of the particles. However, if we take a closer look at the consequences of this change, it is possible to take the abstraction to a level that is even higher, whence one discovers a completely new perspective and new avenues for development.

To motivate this abstraction, we first consider the issue of what constitutes a single particle in these two models. In the basic PSO model, an individual particle consists of:

- A Current position,
- A Velocity,
- A Personal best found position, and
- Knowledge of the Global best found position.

Each of these components is required in order to determine the subsequent position of the particle for the next iteration. The particle's position in the next iteration depends upon its current position and the particle's velocity. The updated velocity, in turn, depends on the particle's current velocity, the particle's personal best found position and the population's global best found position. Each component of the particle contributes to the update function and must be maintained for use in the next iteration.

In the BBPS model, the velocity term is removed and an individual particle consists of:

- A Current position,
- A Personal best found position, and
- Knowledge of the Global best found position.

However, unlike the basic PSO model, updating the particle does not require all of these components. Specifically, the position of the particle in the next iteration is independent of the position of the particle in the current iteration. When updating a particle, the next position is generated by *sampling* a Gaussian random distribution, which, in turn, is constructed using the particle's personal best, and the global best points. After being created, the position is used for only *one* purpose, which is the updating of the particle's personal best found point. If a new position is generated which possesses a value better than the particle's current best found point, the best found point is updated to be the new position. If the newly generated position's value is worse than the particle's current personal best, it is "discarded" and not used for anything at all. The global best point is, thereafter, updated using the population's personal best found points, and so does not rely *directly* on the particle's current position.

The fact that the particle's next position is independent of its current position, and the fact that the current position is only used to update the personal best point means that it is possible to remove the particle's position component from the model entirely. Rather than storing and maintaining the particle's current position, we can modify the update operations to work directly with the particle's personal best point while maintaining equivalent behaviour. The particle update strategy remains roughly the same as before. A new point is generated by sampling the constructed random distribution. If the new point has a better value than the particle's personal best found point, the best found point is set equal to the new point. The only difference is that this newly generated point is discarded after the update is completed rather than being stored and maintained for the next iteration. The behaviour of the

algorithm with this abstracted model remains unchanged, but the metaphors and concepts of the original PSO may require re-examination.

After removing the particle's current position component, an individual particle now consists of only:

- A Personal best found position, and
- Knowledge of the Global best found position.

The particle no longer exists as an explicit point in space. Conceptually, the particle's personal best point is a memory that the particle maintains of the best point it has evaluated so far, and the global best position can be thought of as the collective memory of the population. These two "memories" define the random distribution used to update the particle. Though the particle no longer exists as an explicit point in space, we can, instead, think of the particle's "position" in that space as being defined by the random *distribution* itself. This is because this distribution represents the probability field of possible possible positions for the particle. From this perspective, the particle exists as a random field defined by its own memory and the collective memory of the population.

With this new, more abstract, concept of a particle existing as a random field of possible positions, we re-examine the phenomena behind the particle update. Rather than updating a particle's position, we are now simply sampling the distribution and updating the particle's personal best found position, if necessary. Conceptually, we are evaluating the particle's "position" by sampling the distribution defining it. These "particle fields" move through the space by evaluating their current "position" and updating their memories.

With the new concept of what the population individuals are, and how they move, the high-level perspectives of the algorithm change drastically. The metaphor of a swarm of particles "flying" through space no longer aptly describes the high-level concept of the algorithm. Rather, the algorithm now consists of a population of "particle fields" which move throughout the space in a different way. Because the "positions" of these "particle fields" are defined as random distributions, "evaluating" a "particle field's" current "position" is non-deterministic, and so these "particle fields"

do not necessarily have to "move" to explore new points. These "particle fields" remain "stationary" in the space until either the individual's personal best point changes, or the population's global best point changes. This population of "particle fields" can, itself, be perceived as a random field of particles defined as the sum of each individual's random field. This complex random field, defined as the sum of each individual's random fields, can be thought of as an abstract representation of a particle swarm, where the random fields represent a probability distribution of all possible particle locations at each iteration of the algorithm. This perspective of the high-level concept is significantly different than the BBPS algorithm, although the behaviour of the two is identical. However, with this new perspective, it is possible to explore new directions in improving or changing the behaviour of the algorithm.

3.3 Toward the New Model

With this new, abstracted, perspective of the BBPS algorithm we can begin taking steps towards designing an algorithm with this abstracted philosophy, with potentially distinct behavioural patterns.

First, we consider the general task of updating the population. At each iteration, the population consists of a number of individuals, but these individuals no longer directly represent candidate solutions. Each individual maintains a "memory" of the best point that it has found in the same way as the PSO particle does, but it possesses no explicit position in the solution space. Because the individual's personal best point is only updated when a superior point is located, the best point found by the algorithm so far can still be determined easily by considering each individual, as in traditional PSO systems. Using their own "memory", and the communication of the population's global best found point, each individual defines its "position" in the space as a random distribution constructed using these two points. In the abstracted BBPS model, each of these individuals would generate and evaluate a single new point per iteration so as to maintain the concept of an individual representing a single particle. However, it is not necessary to maintain this metaphor in our new model. Because each individual particle field represents a probability *field* of possible particle positions, there is no

need for each individual particle field to represent a single particle. Consequently, we can now approach the generation and evaluation of candidate solutions from a perspective that is distinct from what is done for traditional PSO algorithms.

Rather than have each individual directly representing a candidate solution in and of itself as in traditional PSO algorithms, we use the population in a more indirect way to explore the solution space. Each individual's "position" is a random distribution representing a probability field of potential particle locations. Collectively, the population represents a complex, finite mixture distribution made up of an underlying set of simple multivariate Gaussian distributions, which are defined by the particle field individuals. This complex distribution represents the probability field of all possible particle locations for the current iteration and is used to guide the search.

Candidate solutions are generated by sampling this population-level distribution. Although the population-level distribution is complex, the sampling process is simple. The population-level distribution is a finite mixture distribution with known component distributions. Therefore, sampling this complex population-level distribution is a simple matter of sampling one of the underlying component distributions, selected at random. In the context of the population, this means selecting a particle field individual and then sampling the multivariate Gaussian distribution defining that particle field's "position".

Once the candidate solution points have been generated and evaluated, each particle field individual in the population is updated. Each individual is updated using the candidate solutions generated from the individual's own distribution. If a candidate solution is better than the individual's own personal best point, the personal best point is set equal to that solution. In this way, the population defines a random distribution which guides the search and generation of candidate solutions, which are then used to update the population and redefine the search area for the next iteration.

This concept of the population guiding the search leads us to the next step towards a new, distinct algorithm. Because the complex distribution created by the population is a finite mixture distribution, it is possible to apply a simple weighting scheme. By weighting the contribution of each particle field's distribution to the population-level mixture distribution, it is possible to incorporate additional information into the

search process, independent of the underlying PSO processes.

Another way by which this population-wide distribution can be modified is to control the flow of information between the individuals. The application of communication topologies to traditional PSO algorithms has been shown to improve the performance of those algorithms and has become commonplace. In the traditional PSO, particles accelerate towards two points, their own personal best point, and the population's best point. Applying a population topology, which limits the communication of this "global best" point to a "neighbourhood best" point modifies the behaviour of the PSO system. This concept can be extracted from the PSO model and applied directly to this new model with no modification. This is because the population of particle field individuals in the new model retains the concept of both the personal and the global best points. Applying different topologies to the new model has dramatic effects on the complex distribution generated by the population, and thus, a dramatic effect on the behaviour of the algorithm itself.

Finally, since the population of the particle field individuals no longer directly represents candidate solutions, it is no longer necessary that the number of candidate solutions generated and evaluated at each iteration equal the size of the population. Because of this, it is possible to further modify the behaviour by using different relative population and point pool sizes. Because the population-wide distribution is the sum of a number of simple distributions, changing the number of particle field individuals in the population effects the "resolution" of the population-wide distribution. The consequence of this is that a larger population will result in a more complex population-wide distribution, while a smaller population will result in a less complex population-wide distribution. Additionally, since particle field individuals are only updated if they are used to generate candidate solutions, individuals in a smaller population will be more likely to update their personal best points. In short, a smaller population will lead to faster convergence, but a larger population may better explore the solution space.

3.3.1 The Particle Field Optimization Algorithm

With these changes taken into account, we have now moved away from the traditional PSO paradigm and arrived at a new, distinct algorithm, which will be hereafter referred to as Particle *Field* Optimization (PFO). This algorithm consists of a population of "particle field" individuals and a " point pool" of candidate solution points. The population of particle field individuals uses PSO principles to guide the search of the solution space, which is carried out by generating and evaluating, the pool of candidate solution points. Analogous to traditional PSO algorithms, the PFO algorithm consists of an initialization phase and a simulation phase which loops until some termination criteria is met, at which point the best solution found by the algorithm is returned as output. As its parameters, the algorithm takes an initialization range, a population size and a pool size described below.

- The population size parameter specifies the number of particle field individuals which form the population used to guide the search.
- The pool size parameter specifies the number of candidate solutions to be generated and evaluated, at each step of the simulation.
- Additionally, a weighting function may be specified, which weights the contribution of each individual to the population-wide distribution.

The initialization phase initializes the population of particle field individuals. A particle field individual is fairly abstract and stores only a personal best found position, and so the initialization of these individuals is simple. Each individual is assigned an initial personal best point by sampling a uniform random distribution defined by the initialization range. These personal best points are then evaluated to assign initial values to the individuals. With this population initialized the simulation phase can begin.

The simulation phase loops until a termination criteria is met, which is typically, the maximum number of iterations. Each iteration consists of two phases. In the first phase, candidate solutions are generated. The generation of these candidate solutions is guided by the distribution defined by the population of particle fields. Candidate

solutions are generated by sampling the mixture distribution defined by the population of particle field individuals. For each point in the point pool we do the following: A particle field individual is selected at random from the population, according to some weighting scheme. Then, the point's position is generated by sampling the random distribution defined by the selected individual. This random distribution is constructed using the individual's personal best, and the global (or neighbourhood) best points in the same way as the BBPS method does. Given a particle field with personal best point $\vec{P}_i$ and for which the global best (or neighbourhood best), point is $\vec{P}_g$, the position of the candidate solution point, $\vec{c}$, is determined according to:

$$\begin{aligned} \vec{P_m} &= \frac{\vec{P}_i + \vec{P}_g}{2}, \\ \vec{\sigma^2} &= |\vec{P}_i - \vec{P}_g| \\ \vec{c} &= \vec{\mathcal{N}}(\vec{P_m}, \vec{\sigma^2}), \end{aligned}$$

Once the candidate solution has been generated, the objective function is evaluated, using this generated point as input, in order to assign it a value. After each candidate solution in the point pool has been generated, the second phase begins. In the second phase, the population of particle field individuals is updated. Each individual updates its own best found point using the set of candidate solutions generated from its own distribution. Each individual selects the best point from the set of associated candidate solutions. If the best associated candidate solution is better than the individual's personal best found point, the individual sets its personal best found point to be equal to that candidate solution point. The pool of candidate solutions is then "emptied", and the simulation continues to the next iteration.

Once the termination criteria has been met, the global best found point is returned as output of the algorithm. The formal algorithm follows in Algorithm 2.

The computational complexity of the PFO algorithm's simulation step is roughly the same as that associated with the BBPS algorithm and the canonical PSO algorithm. The BBPS and PSO algorithms simply update the characteristics associated with each particle in their population, which involves a process that has a constant time for each particle. They then determine the global best found point among each

particle, which is linear in the number of particles. This results in a complexity which is linear with respect to the population size. On the other hand, the PFO algorithm generates candidate solutions by randomly selecting particle field individuals from the population of particle fields and then sampling the associated distributions. The process of random selection can be done in constant time, and so the complexity of the process involving generating candidate solutions is linear with respect to the size of the pool of candidate solutions. The particle field individuals are updated by choosing the best candidate solution from among those generated using the individual's distribution and then by comparing it with their previously best found position. Because candidate solutions can be associated with *only* a single particle field individual, this process is also linear with respect to the size of the pool of candidate solutions. Each particle field individual must be assigned a weighting value, and this weighting process is linear with respect to the particle field population size. By considering the overall processes, the simulation step of the PFO has a computational complexity which is linear with respect to both the candidate solution pool size and the particle field population size.

Algorithm 2 Particle Field Optimization Algorithm

Input:

Function $f()$ to be optimized

Initialization range $lbound$, $ubound$

Particle field population size n_{pop}

Candidate solution point pool size n_{pool}

Weighting function $w()$

Output:

Point $\vec{P}_g$ representing best found solution

Method:

create particle field population P with size n_{pop}
create candidate solution point pool C with size n_{pool}
for each particle field $i \in P$ **do**
  $\vec{P}_i \Leftarrow \vec{U}[lbound, ubound]$
  if $f(\vec{P}_i) < f(\vec{P}_g)$ **then**
    $\vec{P}_g \Leftarrow \vec{P}_i$
  end if
end for

while termination criteria not met **do**
  for each candidate solution point $\vec{c} \in C$ **do**
    select particle field $i \in P$ with probability $\frac{w(i)}{\sum_{p \in P} w(p)}$
    $\vec{c} \Leftarrow \vec{\mathcal{N}}\left(\frac{\vec{P}_i + \vec{P}_g}{2}, |\vec{P}_i - \vec{P}_g|\right)$
    $S_i \Leftarrow S_i \cup \vec{c}$
  end for
  for each particle field $i \in P$ **do**
    choose point $\vec{c_{min}}$ from S_i which minimizes $f(\vec{c_{min}})$
    if $f(\vec{c_{min}}) < f(\vec{P}_i)$ **then**
      $\vec{P}_i \Leftarrow \vec{c_{min}}$
      if $f(\vec{P}_i) < f(\vec{P}_g)$ **then**
        $\vec{P}_g \Leftarrow \vec{P}_i$
      end if
    end if
    $S_i \Leftarrow \emptyset$
  end for
end while
return $\vec{P}_g$

3.3.2 Magnification of a Subtle BBPS Issue

The BBPS algorithm, on which the PFO algorithm is based, displays a subtle issue which becomes magnified in the PFO algorithm. This issue is also present in the original PSO algorithm, to a degree, although the impact it has is relatively insignificant. However, this issue has a *critical* impact on the performance of the PFO algorithm and must be addressed for PFO to be effective.

This issue arises in the BBPS algorithm when a particle's personal best point is equivalent to the global best point. In the BBPS algorithm, the particle's position is updated by sampling a Gaussian random distribution for each dimension. This distribution is constructed, for each dimension, with mean equal to the midpoint between the particle's personal best point and the global best point, and with a variance equal to the distance between these two points in that dimension. In the case where the personal best point and the global best point are equal, the sampled Gaussian distribution will have a mean equal to both of these positions, and a variance of zero. Sampling such a distribution, with a variance of zero, will generate *only* points equal to the mean. In the case of the BBPS algorithm, the result of this is that the global best particle will remain *static*, evaluating the same point each iteration. If another particle discovers a new global best point, the previously best particle will begin to generate new points again, but this new global best particle will become static. This situation can also occur if multiple particles have exactly the same position. However, when considering a multi-dimensional, real-valued solution space, the probability of this occurring is insignificant in terms of overall performance.

In the case of the PFO algorithm, each particle field individual constructs a random distribution in the same way as the BBPS algorithm. As a result, an individual with a personal best found point equal to the global best found point will suffer from the same "collapse" as in the BBPS algorithm. The impact this has on the performance of the PFO algorithm is magnified by the fact that individuals are not limited to generating only one new candidate solution for each iteration. Individuals with a higher assigned weighting will generate, on average, more candidate solutions. Any basic weighting scheme that assigns weighting values based on an individual's personal best found value will likely assign the highest weighting to the globally best

individual. This means that the individual which is *guaranteed* to suffer from this "collapse" is likely to be assigned more candidate solutions than any other individual. This means that many, or even the majority, of all function evaluations allowed for each iteration will be wasted evaluating the same point.

Fortunately, this issue can be easily addressed. To solve this problem, the particle field individual with the globally best found point must use some other particle field individual as the global best point when constructing the random distribution. The most straight-forward way of achieving this would be to have the individual with the globally best found point use the individual with the globally second-best found point to construct its distribution. This can be seen as applying a neighbourhood communication topology to the population, which is a common practice in traditional PSO algorithms. Considering a population which uses a topology to restrict communication to neighbourhood bests, enforcing a simple trait in this topology will solve the collapse problem. As long as a particle's neighbourhood topology does not include that particle itself, this problem cannot occur. There is still the chance that two particle field individuals have personal best found points which share the exact same position in the solution space, but as previously discussed, such a case occurs with insignificant probability, and thus has insignificant impact on the overall performance.

There are many alternative solutions to this problem as well, if application of a neighbourhood topology is undesirable. For example, another solution would be to simply select a particle at random for use as the best neighbour of the global best particle. This method can be seen as the global best particle defining its own random distribution as the union of the distributions of its neighbours. In this sense, the global best particle will generate a much wider variety of candidate solutions than any other particle, and may lead to beneficial exploration behaviour. Another potential solution would be to simply assign the globally best point a weighting of zero, in order to ensure no candidate solutions are generated using the "collapsed" distribution.

3.4 Particle Field Weighting Schemes

The addition of a weighting scheme to the construction of the population-wide mixture distribution makes it possible to manipulate the search process without disrupting the PSO principals at work in the particle field population. Four different weighting schemes have been created for testing and are presented here. For simplicity, the weighting schemes here are described for the case of maximization. In the case of minimization, weighting values are simply negated and normalized.

3.4.1 Personal Best Weighting Scheme

The "Personal Best" weighting scheme simply assigns a weighting to a particle field individual according to the value of that particle field's best found value. The motivation behind this weighting scheme is to provide a very basic scheme which does not attempt to introduce any new information into the search process. The Personal Best weighting scheme is formulated simply as:

$$w(i) = f(\vec{P}_i)$$

3.4.2 Average Value Weighting Scheme

The "Average" weighting scheme assigns a weighting according to the average value of the points which are directly related to the particle field individual. This includes the individual's personal best point, the global, or neighbourhood, best point, and any new candidate solution points generated using that individual during the current iteration. The average value of these points represents the information learned about the average value of the area of the solution space "covered" by the particle field' at the given iteration. The motivation behind this weighting scheme is to provide a more "intelligent" version of the personal best weighting scheme. The Average weighting scheme is formulated as:

$$w(i) = \frac{f(\vec{P}_i) + f(\vec{P}_g) + \sum_{\vec{c} \in S_i} f(\vec{c})}{2 + |S_i|}$$

Where S_i is the set of candidate solution points generated by individual P_i.

3.4.3 Relative Average Weighting Scheme

The "Relative Average" weighting scheme attempts to use the same information as the Average scheme, but to produce a significantly different effect. Rather than weighting according to the average value of the related points, this weighting scheme uses the average value of these points *relative* to the particle field's own personal best found value. The motivation behind this weighting scheme is to concentrate the search into areas which show the greatest increase in performance. The Relative Average weighting scheme is formulated as:

$$w(i) = f(\vec{P}_i) - \frac{f(\vec{P}_g) + \sum_{\vec{c} \in S_i} f(\vec{c})}{1 + |S_i|}$$

Where S_i is the set of candidate solution points generated by individual P_i.

3.4.4 Improvement Percentage Weighting Scheme

The improvement percentage scheme aims to assign a weighting to the particle field individual according to the percentage of related points which are better than it's own personal best found point. This weighting scheme is similar to the Relative Average scheme, but the actual values of the related points to not factor into the weighting. In this sense, the Improvement Percentage scheme considers all improvements equally. The Improvement Percentage scheme is formulated as:

$$w(i) = \frac{1 + b(\vec{P}_g) + \sum_{\vec{c} \in S_i} b(\vec{c})}{2 + |S_i|}$$

$$b(\vec{X}) = \begin{cases} 0, & \text{if } f(\vec{X}) > f(\vec{P}_i) \\ 1, & \text{otherwise} \end{cases}$$

Where S_i is the set of candidate solution points generated by individual P_i.

3.5 Conclusion

In this chapter we have proposed a new algorithm which has been referred to as the Particle Field Optimization (PFO) algorithm. The distinguishing feature of the PFO is that each particle is not just merely characterized by a position, velocity etc. Rather each particle is characterized by a random distribution, which is also referred to as a "field". The chapter depicts the development process of the PFO algorithm, building from the traditional PSO algorithm to the new paradigm. By considering the BBPS algorithm, which is an abstraction of the canonical PSO algorithm, we have developed an additional level of abstraction, leading to a new high-level perspective of the algorithm. By concentrating on this perspective, we can obtain a number of variations and modifications, each with its own new and distinct behaviour. The final PFO algorithm is a consequence of these. We have described the PFO algorithm itself. The behaviour of the PFO algorithm has been detailed, and a formalized algorithm has been presented. Finally, a number of weighting schemes have been presented which can be applied to the PFO algorithm to modify the search process.

Chapter 4

Empirical Results

4.1 Introduction

This chapter presents the results of the empirical testing carried out on the newly-developed PFO algorithm proposed in Chapter 3. We begin by outlining the specific suite of test functions used. This test suite, surveyed in Chapter 2, consists of a number of popular test functions commonly used for the evaluation of particle swarm algorithms. Following this, we submit a description of the test strategy used. The test strategy describes the various parameter values chosen for testing, and the justifications for these specific values. The test results themselves are presented, as appropriate, in both table and graph forms. Tabulated results represent the average best final values discovered by each algorithmic configuration, and the graphed results plot the average best values found as a function of the number of iterations. Further, the results are also separated and grouped based on the test function used, and the results for each function are discussed briefly. Following this, we discuss these results as a whole, identifying patterns and establishing correlations between the parameters and the respective performances and behaviours of the configurations involved. Finally, we provide a retrospective of the PFO algorithm, discussing its benefits and drawbacks, as well as the new phenomena exhibited.

Table 4.1: Test Function Suite

Function	Dim	Formula
Rosenbrock	2	$(1-x_1)^2+100(x_2-x_1^2)^2$
Schaffer F6	2	$0.5+\frac{sin^2(\sqrt{x_1^2+x_2^2})-0.5}{[1+0.001*(x_1^2+x_2^2)]^2}$
Sphere	10	$\sum_{i=1}^{d} x_i^2$
Ackley	10	$20+e-20e^{-\frac{1}{5}\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}}-e^{-\frac{1}{d}\sum_{i=1}^{d}cos\left(2\pi x_i^2\right)}$
Rastrigin	2	$10d+\sum_{i=1}^{d}[x_i^2-10cos(2\pi x_i)]$
	10	
	20	
Griewank	2	$1+\frac{1}{4000}\sum_{i=1}^{d}x_i^2-\prod_{i=1}^{d}\cos\left(\frac{x_i}{\sqrt{i}}\right)$
	10	
	20	

4.2 Test Suite

In order to accurately measure the performance of a black-box optimization algorithm, empirical testing must be carried out on a variety of test functions. The test suite selected to evaluate the PFO algorithm consists of a variety of different test functions commonly used for evaluating PSO related algorithms. The suite of test functions used for general testing is presented in Table 4.1, and each algorithm is described, in detail, in Section 2.6.

4.3 Overview of Testing Strategy

Testing was carried out on the PFO algorithm in order to investigate the possible range of behaviour of the algorithm and the effects of each parameter on that behaviour. The PFO algorithm was created by introducing a number of changes to the BBPS algorithm. As a result, the BBPS algorithm provides the best baseline point of comparison for assessing the effects of these changes. In addition to this, the BBPS algorithm takes only one parameter, which is the population size parameter. For reasons explained below, this value was kept static for each test function. Because of this, the BBPS algorithm required no parameter optimization, and so provides a very straightforward and dependable baseline.

The PFO algorithm can be configured using three primary parameters which modify the algorithm's behaviour. In order to thoroughly investigate the effects of these parameters, it is necessary to decide on a variety of values to be tested for each parameter. In addition to this, the effects of the parameters on an algorithm's performance is likely co-variant, and so all combinations of the chosen values for each parameter must be tested in order to ensure thoroughness. These primary parameters are the "pool size" parameter, the "population size" parameter and the weighting scheme used by the algorithm.

The "pool size" parameter specifies the number of candidate solutions to be generated and evaluated by the PFO algorithm for each iteration. When comparing the performance of the PFO algorithm to the baseline provided by the BBPS algorithm, it is important that the number of candidate solutions evaluated at each iteration by the algorithms is maintained to be consistent between them. For this reason, the pool size parameter of the PFO algorithm, and the population size of the BBPS algorithm are set equal to one another for each test function. This value is chosen for each test function based on the dimensionality of that function. On functions with two dimensions, a pool size of twenty was used. On functions with ten dimensions, a pool size of fifty was used, and on functions with twenty dimensions a pool size of one hundred was used. Only one value for these parameters was tested for each function, as testing multiple values would exponentially increase the number of tests required

to cover all combinations selected PFO parameter values.

The "population size" parameter specifies the size of the particle field population which guides the search process. Unlike more traditional population based algorithms, this population size is not directly tied to the number of candidate solutions evaluated per iteration. As a result, it is possible to test various population sizes against the baseline provided by the BBPS. For each test function, three population sizes were tested, determined by the chosen pool size for that particular test function. Population sizes of half the pool size, equal to the pool size, and double the pool size were tested. This strategy was used in order to investigate the relationship between the pool size and the population size, as these two parameters are directly related.

The "weighting scheme" parameter is a function which assigns weightings to the particle field individuals that make up the particle field population. These weightings effect the population-level mixture distribution which is generated at each iteration to guide the search, and so different weighting functions are expected to produce different search behaviours. The weighting schemes tested the Personal Best, Average Value, Relative Average and Improvement Percent schemes, as described in Section 3.4. In addition to these weighting schemes, PFO configurations which did not use any weighting scheme were also tested in order to provide a baseline.

In total, for each test function, the PFO algorithm was tested using all combinations of the outlined parameter values. To provide a baseline point of reference, the BBPS algorithm was tested on each of these test functions, with a population size equal to the PFO pool size used on that function. Each of these tests consisted of running the algorithm, with the chosen parameters, on the chosen test function for a specified number of iterations. The number of iterations was chosen based on the complexity and dimensionality of the test function. When comparing optimization algorithms in this way, it is important for the number of function evaluations to be consistent between the algorithms.

During the initialization phase of the PFO algorithm, a single function evaluation was performed for each particle field individual in the particle field population. Since the particle field population size can be larger or smaller than the candidate solution pool size, this can lead to a small discrepancy between the total number of function

evaluations between different PFO configurations. In order to counteract this, PFO configurations with population sizes equal to twice the pool size were given one less iteration, and PFO configurations with population sizes equal to half the pool size were allowed two extra iterations. This ensured that the number of function evaluations was equal for all the algorithms tested.

In addition to this general testing, a subset of the test function suite was used to investigate the effect of applying a square communication topology to the particle field population. This subset consisted of the Rastrigin function on 10 and 20 dimensions, the Schaffer F6 function and the Rosenbrock function. On these functions, tests were repeated using the same set of parameter values, but with a square communication topology applied to the population.

4.4 Test Results

4.4.1 Sphere Function

The Sphere function, described in Section 2.6.1, is a unimodal and radially symmetric test function, without any local optima. As a generalized function, the Sphere function can be configured with any number of dimensions.

Tests were performed on the Sphere function using 10 dimensions. Particle field population sizes of 25, 50 and 100 were tested, with a pool size of 50. The baseline BBPS was run with a population size of 50 to match the pool size of the PFO. Each configuration was run for 800 iterations, and each test was repeated 500 times. Plotted results of the average best-found value by each configuration against the number of iterations are presented in Figure 4.1, and average final best-found values are tabulated in Table 4.2.

PFO configurations with particle field population size of 25 performed best on the Sphere function. The performance varied among the weighting schemes a fair amount, with all weighting schemes performing better than the PFO configuration that utilized no weighting scheme.

With a population size of 50, the performance was, on average, slightly worse than

Table 4.2: The Performance Results of the Various Algorithms for the Sphere Function when the Number of Dimensions was 10.

Optimizer		**Average Result After 800 Iterations**		
BBPS: Population 50, Global		$5.528287E^{-66}$		
	Weighting Scheme	**PF Population size**		
		25	50	100
PFO Pool Size 50 Global	none	$1.392424E^{-87}$	$5.124157E^{-62}$	$4.218981E^{-40}$
	pbest	$4.826226E^{-89}$	$2.330228E^{-63}$	$3.608431E^{-41}$
	average	$\mathbf{1.329902E^{-89}}$	$2.211993E^{-62}$	$5.911718E^{-41}$
	relative	$2.042696E^{-88}$	$6.51605E^{-63}$	$7.310571E^{-41}$
	percent	$1.409235E^{-89}$	$\mathbf{3.659147E^{-66}}$	$\mathbf{2.663403E^{-43}}$

the baseline BBPS. There was a small amount of variance in performance among the weighting schemes, with the Improvement Index weighting scheme performing best by a fairly significant margin over the rest.

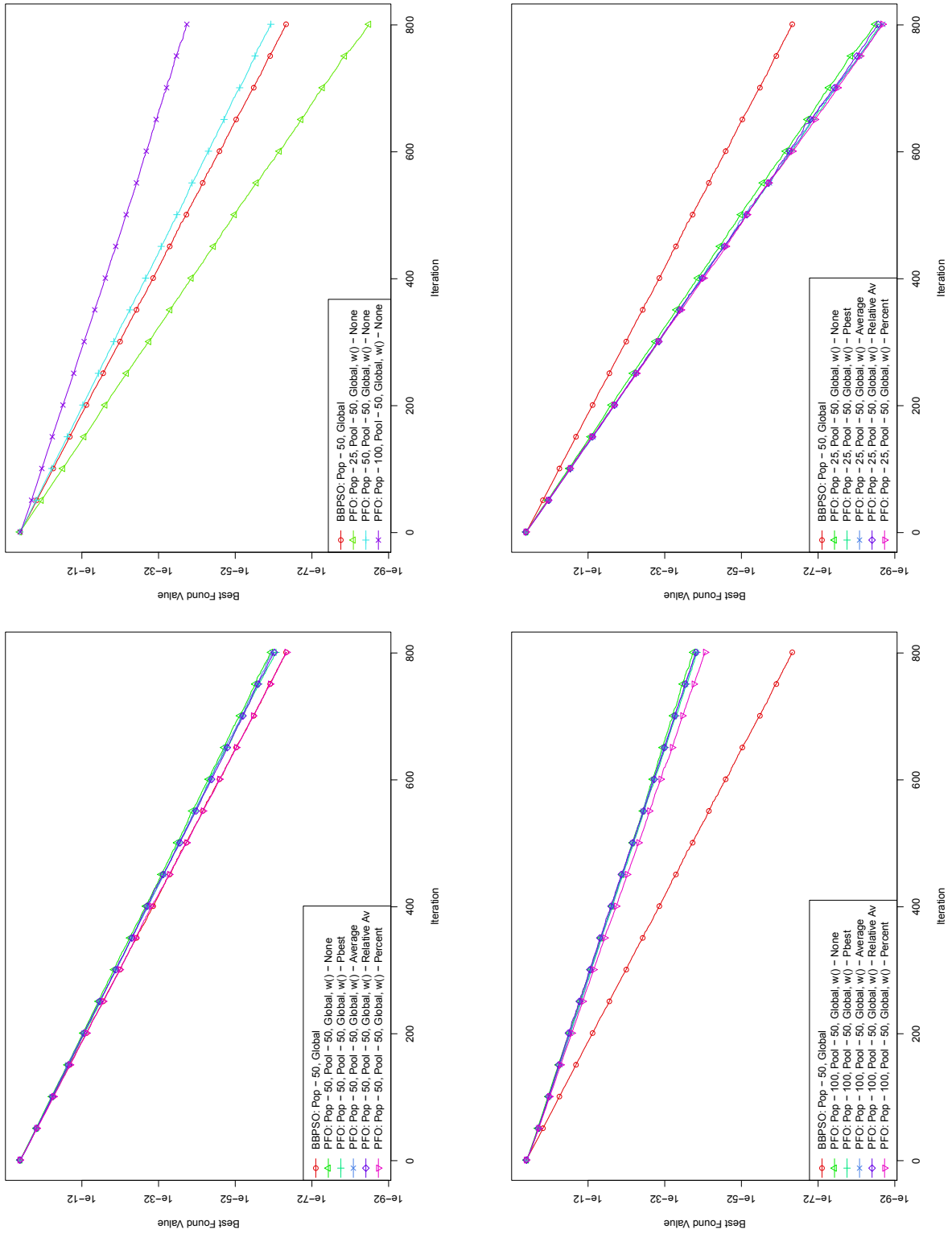
The configurations with a population size of 100 performed far worse than others. Similar to the "equal size" configuration results, the Improvement Index showed a distinct improvement over the other weighting schemes.

4.4.2 Rosenbrock

The Rosenbrock function, depicted in Section 2.6.2, is unimodal, with a single optimum. However, this optimum lies in a long, relatively "flat" "banana shaped" valley. While it is trivial to locate this valley, locating the true optimum within it, is more difficult.

As a two dimensional function, population sizes of 5, 10 and 20 were tested on the Rosenbrock function, with a pool size set to 10. The baseline BBPS was configured with a population of 10 to match the PFO's pool size of 10. Each configuration was run for 300 iterations and each test was repeated 500 times. Plotted results of the average best-found value by each configuration against the number of iterations, are presented in Figure 4.2, and average final best-found values are tabulated in Table 4.3.

Figure 4.1: The Performance Results of the Various Algorithms for the Sphere Function when the Number of Dimensions was 10.



PFO configurations using a population size of half the pool size performed poorly when compared to the baseline BBPS. Despite the Rosenbrock function being unimodal, these configurations quickly became stagnant and stuck in a sub-optimal area within the first 50 iterations. Variations in the performance among the weighting schemes with this population size was also considerable, but with no particular outliers. The best performing weighting scheme was the Improvement Performance scheme.

Configurations with a population size of 10 performed, on average, considerably better than the baseline BBPS. The variation in performance among the weighting schemes with this population size was significant, with the Average weighting scheme performing dramatically worse than the others. The best performing weighting scheme was the Improvement Percentage weighting scheme.

All the PFO configurations with a population size of 20 performed dramatically better than the baseline BBPS. Among these configurations, two weighting schemes showed a considerable improvement over the others. These weighting schemes were the Personal Best and Average weighting schemes.

Additional testing was performed on the Rosenbrock function in order to investigate the effects of a square population topology. Overall, there was no significant performance difference observed on this function between the configurations with and without the square topology.

Table 4.3: The Performance Results of the Various Algorithms for the Rosenbrock Function when the Number of Dimensions was 2.

Optimizer		**Average Result After 300 Iterations**		
BBPS: Population 10, Global		0.0170631		
PFO Pool Size 10 Global	**Weighting Scheme**	**PF Population size**		
		5	10	20
	none	0.2615312	0.001772619	0.0004318705
	pbest	0.4201835	0.002465582	**0.000081423**
	average	0.3127475	0.01488313	0.000116974
	relative	0.4980816	0.003692096	0.0004330519
	percent	**0.2392649**	**0.001124253**	0.0005015844
BBPS: Population 10, Square		0.019571364		
PFO Pool Size 10 Square	**Weighting Scheme**	**PF Population size**		
		5	10	20
	none	0.277132132	**0.002332304**	**0.000424214**
	pbest	0.398186151	0.013917394	0.000597541
	average	0.294996534	0.009457424	0.000763763
	relative	0.464259223	0.017161443	0.000666798
	percent	**0.231612039**	0.003179736	0.000585061

4.4.3 Schaffer F6

Unlike many other optimization test functions, the Schaffer F6 function, described in Section 2.6.6, does not have a roughly spherical or parabolic surface. It, rather, consists of a flat plane with an added sinusoidal wave radiating from the optimal area. This wave dramatically increases in amplitude as it approaches the global optimum, resulting in high frequency circular hills and valleys, where the valley containing the global optimum is surrounded by the highest peaks in the function.

As a two dimensional function, population sizes of 5, 10 and 20 were tested on the Schaffer F6 function, with a pool size of 10 across all configurations. A baseline was provided by a BBPS with a population of 10, to match the pool size of the PFO configurations. Each configuration was run for 300 iterations and each test was repeated 500 times. Plotted results of the average best-found value by each configuration against the number of iterations are presented in Figure 4.4, and average final best-found values are tabulated in Table 4.4.3.

PFO configurations using a particle field population size of 5 performed poorly compared to the baseline BBPS. The variation in performance among weighting schemes in this group was fairly significant, with two performing much worse than the others. These were the Personal Best and Relative Average weighting schemes. The best performing weighting scheme was the Improvement Percentage scheme.

Configurations with a population size of 10 showed, on average, a small improvement over the baseline BBPS. The variation in performance among weighting schemes in this group was insignificant, with the exception of one out-lier. This out-lier was the Average weighting scheme, performing significantly worse than the other configurations. The best performing weighting scheme was the Relative Average scheme.

With a population size of 20, there was evidently no significant improvement over the baseline BBPS. The performance of these configurations was very similar to those which used a population size of 10, but was on average slightly better. Variance in performance among weighting schemes in this group was insignificant, with no outliers. The best performing weighting scheme was the Improvement Percentage scheme.

Additional tests were carried out on the Schaffer F6 function in order to investigate

Figure 4.2: The Performance Results of the Various Algorithms for the Rosenbrock Function when the Number of Dimensions was 2.

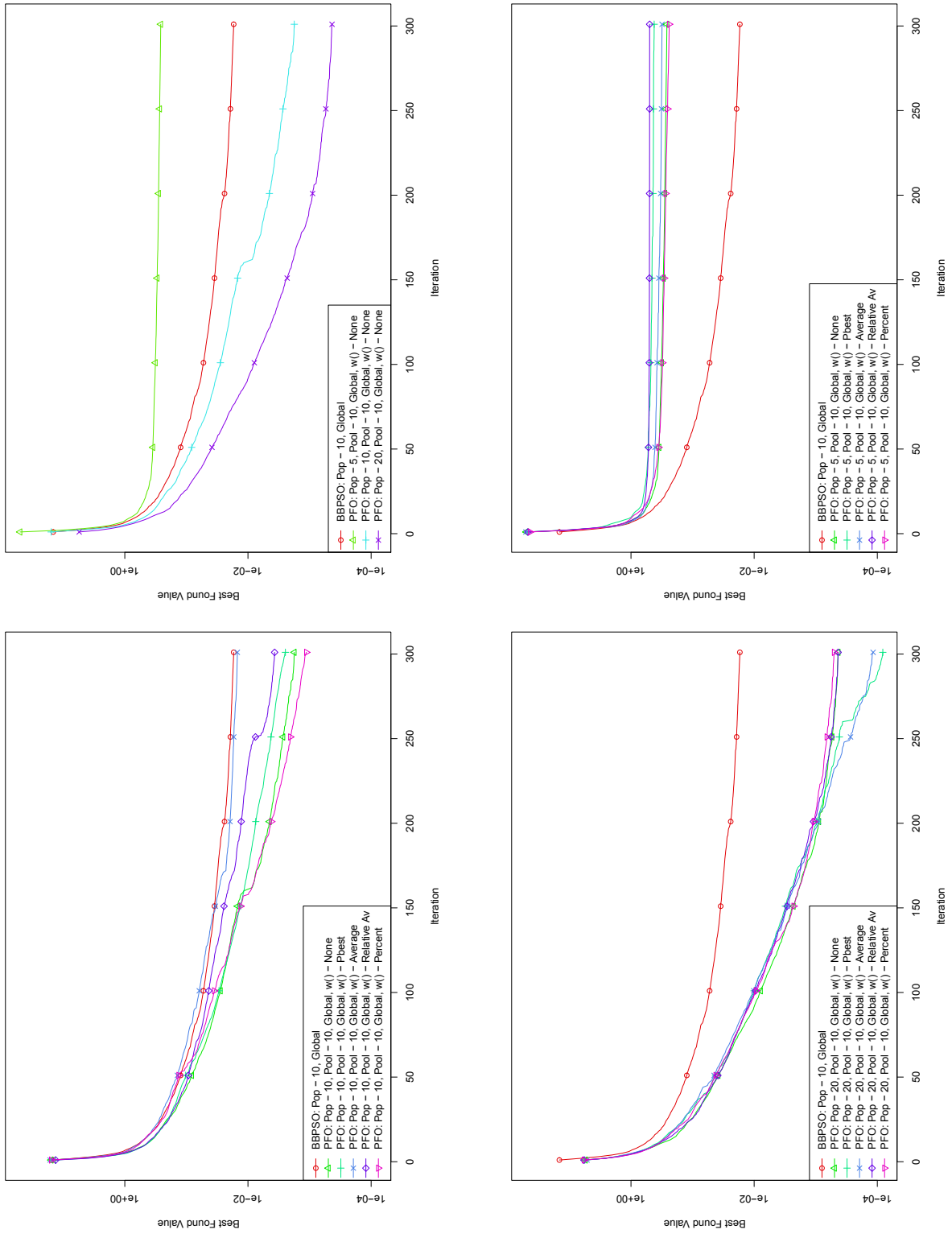


Figure 4.3: The Performance Results of the Various Algorithms for the Rosenbrock Function using the Square Topology when the Number of Dimensions was 2.

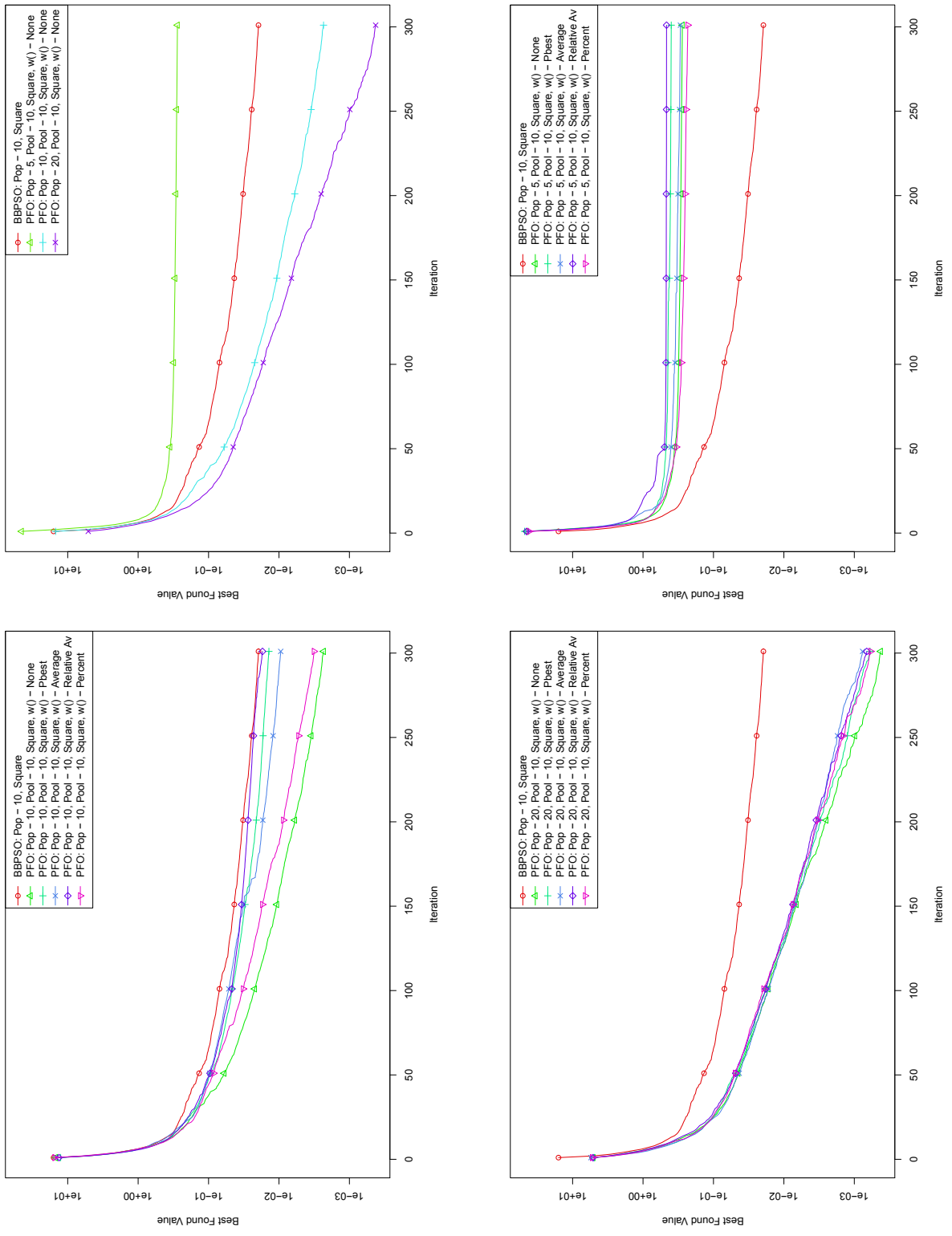


Table 4.4: The Performance Results of the Various Algorithms for the Schaffer F6 Function when the Number of Dimensions was 2.

Optimizer		**Average Result After 300 Iterations**		
BBPS: Population 10, Global		0.010073551		
	Weighting	**PF Population size**		
	Scheme	5	10	20
PFO	none	0.026353959	0.008570938	0.007864256
Pool Size 10	pbest	0.032898468	0.009100150	0.007944492
Global	average	0.025614011	0.010458391	0.007982574
	relative	0.032047888	**0.008528701**	0.008047997
	percent	**0.023505543**	0.008829997	**0.007617807**
BBPS: Population 10, Square		0.008933520		
	Weighting	**PF Population size**		
	Scheme	5	10	20
PFO	none	0.025009582	0.008855935	**0.007670143**
Pool Size 10	pbest	0.021154261	0.008757860	0.007677467
Square	average	0.022108461	0.009139542	0.008247457
	relative	**0.021056724**	0.009043058	0.008617698
	percent	0.023566949	**0.008641752**	0.007693947

the effects of a square population topology. The square topology did not produce significant improvements, showing decreased performance, on average, for configurations with population sizes of 10 and 20.

4.4.4 Rastrigin Function

The Rastrigin function, as depicted in Section 2.6.3, has a general spherical shape with added sinusoidal waves parallel to the axes. The global optimal area is surrounded by a number of local optima which have values very close to the global optima, making the function quite difficult. As a generalized function, the Rastrigin function can be configured with any arbitrary number of dimensions. Rastrigin functions of dimension 2, 10 and 20 were used for testing.

Figure 4.4: The Performance Results of the Various Algorithms for the Schaffer F6 Function when the Number of Dimensions was 2.

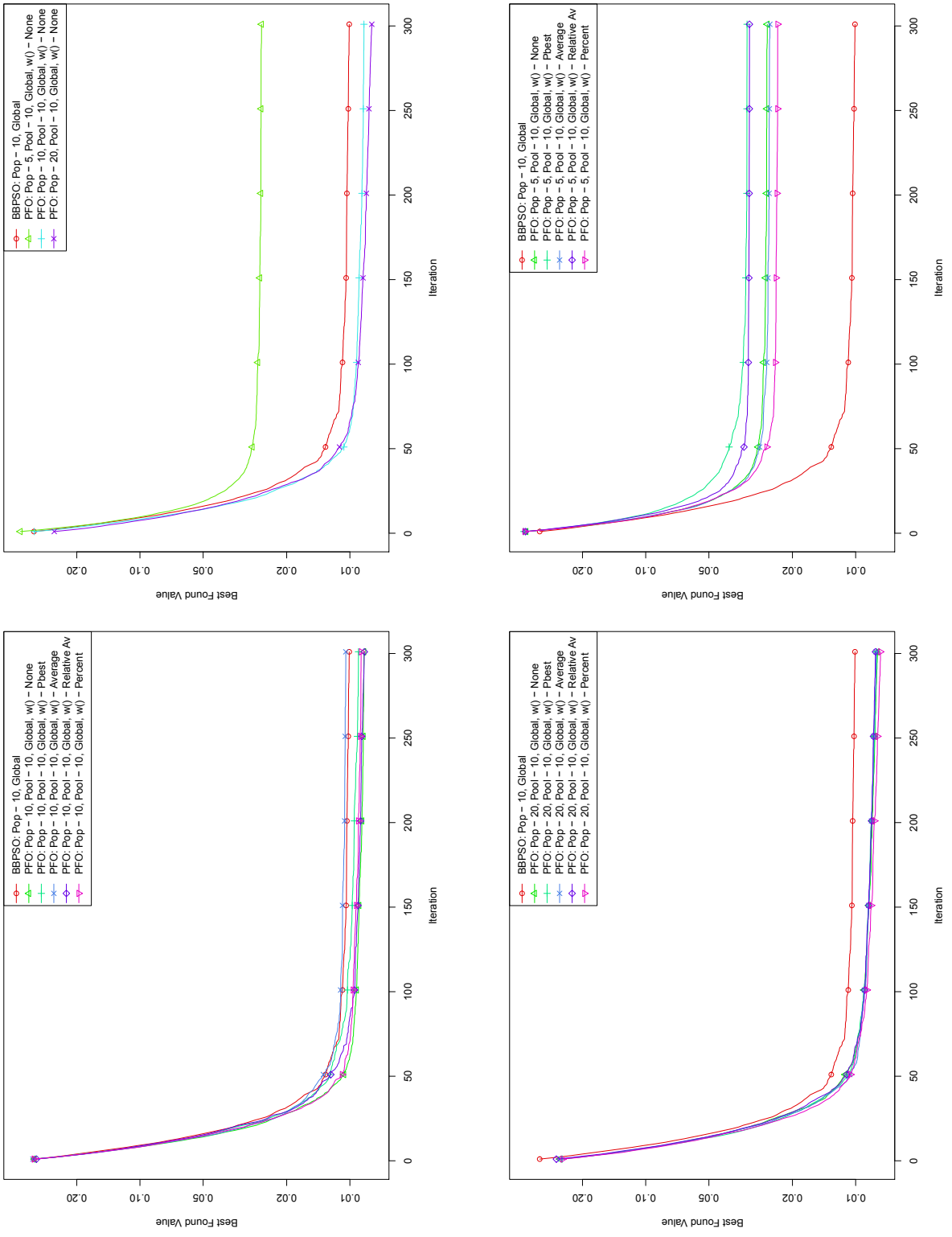
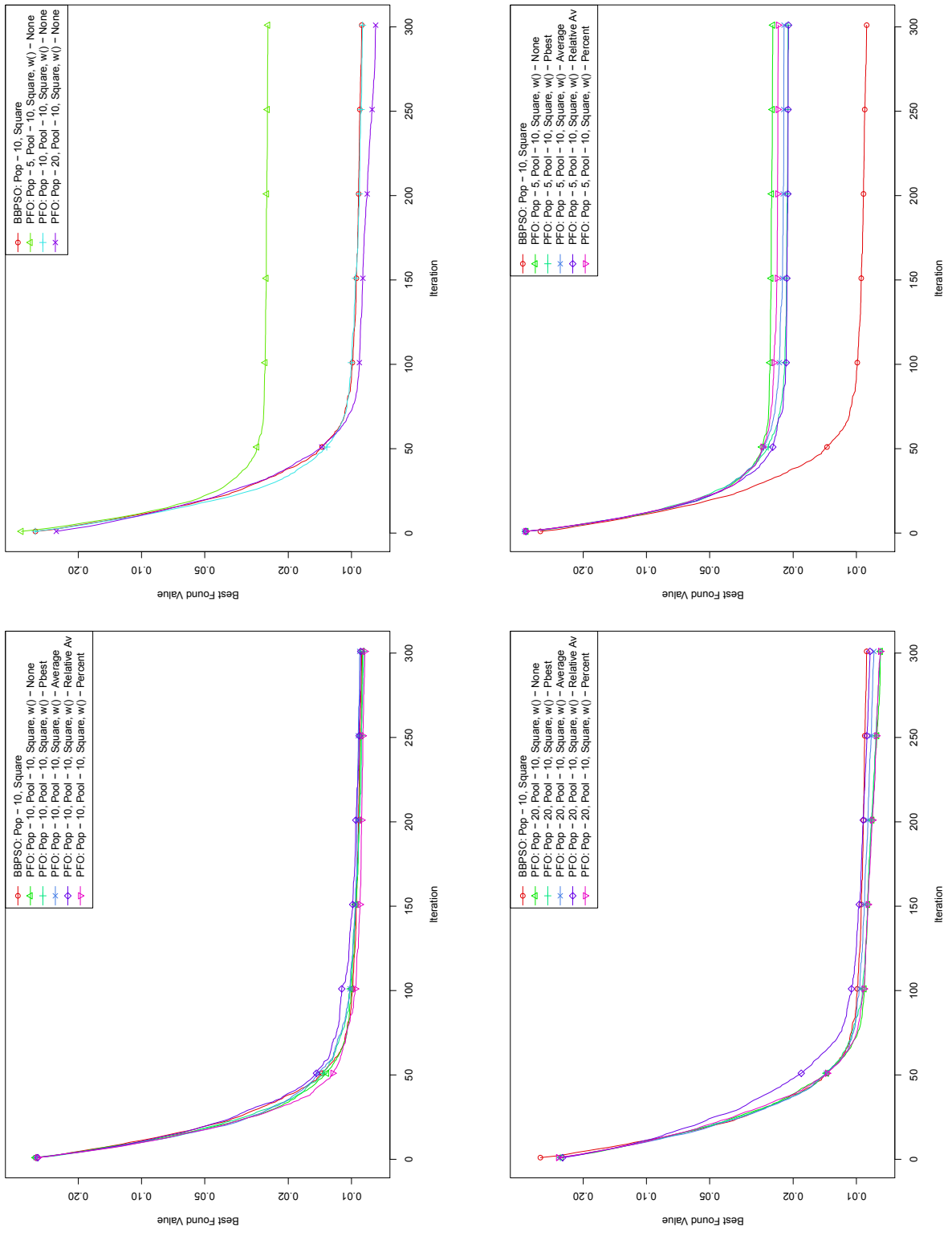


Figure 4.5: The Performance Results of the Various Algorithms for the Schaffer F6 Function using the Square Topology when the Number of Dimensions was 2.



Population sizes of 5, 10 and 20 were tested on the Rastrigin 2 function, with a pool size of 10 across all configurations. A baseline was provided by a BBPS with a population of 10, to match the pool size of the PFO configurations. Each configuration was run for 300 iterations and each test was repeated 500 times. Plotted results of the average best-found value by each configuration against iteration are presented in Figures 4.6, 4.7 and 4.9. The average final best-found values are tabulated in Tables 4.5, 4.6 and 4.7.

PFO configurations using a particle field population size of 5 performed poorly compared to the baseline BBPS. Variation in performance among weighting schemes in this group was small, with no significant outliers. The best performing weighting scheme was the Personal Best scheme.

Configurations with a population size of 10 performed, on average, about the same as the baseline BBPS. Variation in performance among weighting schemes in this group was insignificant, with the exception of one out-lier which was, in fact, the configuration with no weighting scheme, which performed the best by a large margin.

Configurations which used a population size of 20 performed much better than the baseline BBPS. Variation in performance among weighting schemes in this group was, indeed, significant. Interestingly, all weighting schemes performed worse than the configuration with no weighting scheme in this case.

For the Rastrigin 10 function, population sizes of 25, 50 and 100 were used, with a pool size of 50. The baseline BBPS was configured with a population of 50 to match the pool size. On the Rastrigin 10 function, each configuration was run for 1500 iterations and each test was repeated 500 times. Plotted results of the average best-found value by each configuration against the number of iterations are presented in Figure 4.9 and average final best-found values are tabulated in Table 4.6.

The performance of the PFO algorithm on the Rastrigin 10 function was roughly similar to the performance on the Rastrigin 2 function. All configurations, grouped by particle field population size, showed only small variance in performance among the different weighting schemes. In all cases, the Improvement Percentage scheme performed best.

Results on the Rastrigin 20 function were similar to those of the Rastrigin 10

function. Particle field population sizes of 50, 100 and 200 were used, with a pool size of 100. The baseline BBPS was configured with a population of 100 to match the pool size. On the Rastrigin 20 function each configuration was run for 3000 iterations and each test was repeated 500 times. Plotted results of the average best-found value by each configuration against the number of iterations are presented in Figure 4.9 and average final best-found values are tabulated in Table 4.7. All configurations, grouped by particle field population size, showed only a small variance in the performance, as in the 10-dimensional case. However, on the 20-dimensional function, the Personal Best weighting scheme performed best.

Additional testing was performed on the Rastrigin 10 and 20 functions in order to investigate the effects of a square population topology. In both cases, there was significant improvement on PFO configurations using the smaller population size. PFO configurations using a population size equal to the pool size saw no significant improvement, on average, and configurations using a population size equal to twice the pool size saw a decrease in performance. This decrease in performance was the result of a dramatically slower rate of convergence, and it is likely that these configurations would show an improvement if allowed many more iterations. With one exception, all configurations using the Improvement Percentage weighting scheme performed best on their respective function, within their respective population size. The exception to this was the PFO configurations using a population of 200, on the Rastrigin 20 function, on which the Average weighting scheme performed better.

4.4.5 Griewank Function

The Griewank function, described in Section 2.6.4, has a general spherical shape with an added wave-shaped noise with many local optima surrounding the global optima. The Griewank function can be generalized to any arbitrary dimension, though it becomes "easier" as the number of dimensions increases. Tests were performed on the Griewank function with 2, 10 and 20 dimensions.

Particle field population sizes of 5, 10 and 20 were tested on the Griewank 2 function, with a candidate solution pool size of 10 across all configurations. A baseline

Table 4.5: The Performance Results of the Various Algorithms for the Rastrigin Function when the Number of Dimensions was 2.

Optimizer		**Average Result After 300 Iterations**		
BBPS: Population 10, Global		0.35890299		
	Weighting Scheme	**PF Population size**		
		5	10	20
PFO Pool Size 10 Global	none	1.11259831	**0.2673588**	**0.06705732**
	pbest	**1.05692345**	0.35313462	0.07601928
	average	1.26427503	0.34896154	0.11465541
	relative	1.2443847	0.37585698	0.10368333
	percent	1.09066552	0.37100784	0.10168929

Table 4.6: The Performance Results of the Various Algorithms for the Rastrigin Function when the Number of Dimensions was 10.

Optimizer		**Average Result After 1500 Iterations**		
BBPS: Population 50, Global		4.466349		
	Weighting Scheme	**PF Population size**		
		25	50	100
PFO Pool Size 50 Global	none	6.478197	4.346715	2.833591
	pbest	6.579886	4.029685	2.942540
	average	6.458258	4.362666	2.989045
	relative	6.320912	4.211014	2.761618
	percent	**6.169141**	**4.013764**	**2.760911**
BBPS: Population 50, Square		3.937005		
	Weighting Scheme	**PF Population size**		
		25	50	100
PFO Pool Size 50 Square	none	4.393429	3.407534	3.152126
	pbest	4.961148	3.465975	2.936005
	average	4.964573	3.274318	**2.875819**
	relative	4.735141	3.546900	3.165446
	percent	**4.368536**	**3.154454**	3.043883

Figure 4.6: The Performance Results of the Various Algorithms for the Rastrigin Function when the Number of Dimensions was 2.

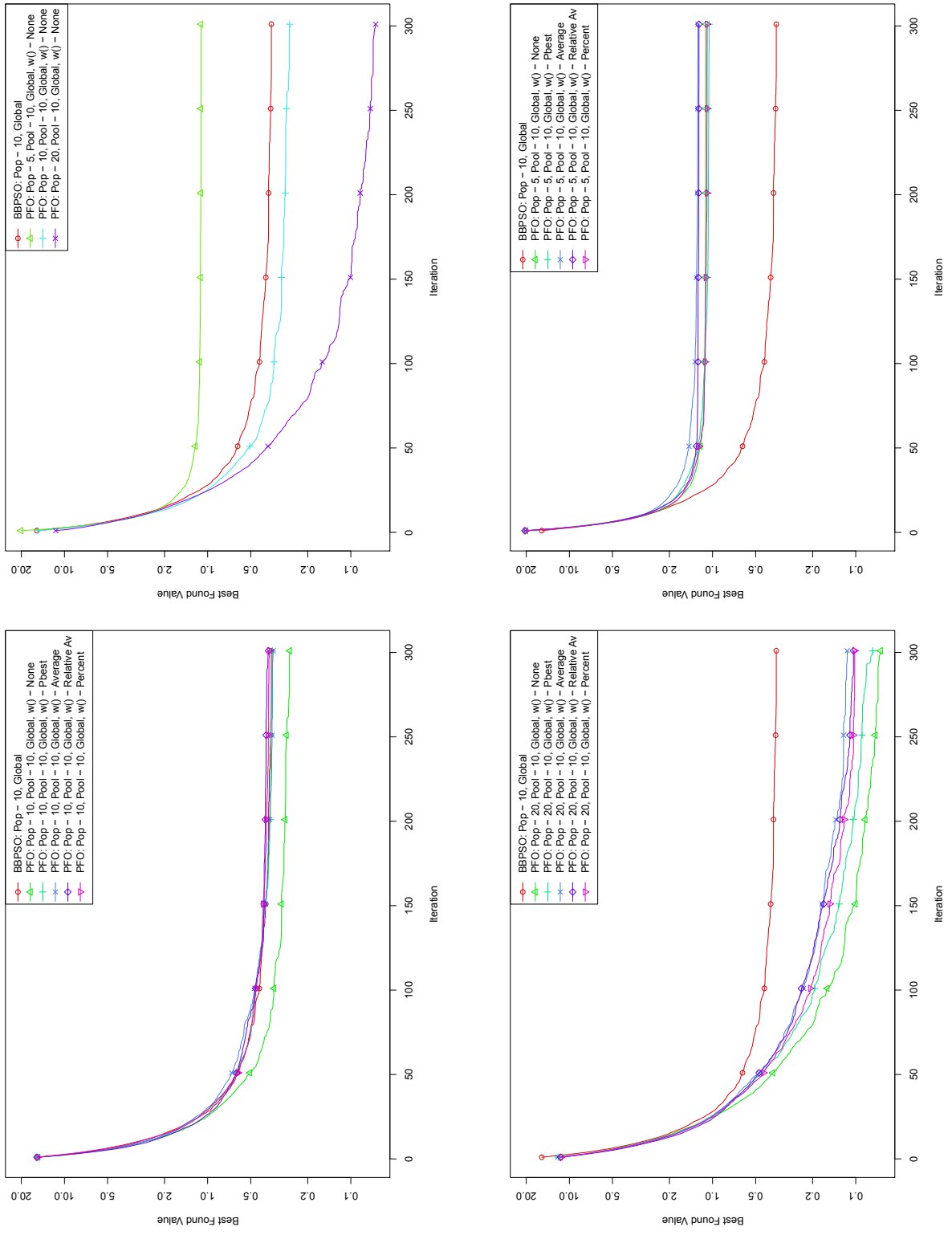


Figure 4.7: The Performance Results of the Various Algorithms for the Rastrigin Function when the Number of Dimensions was 10.

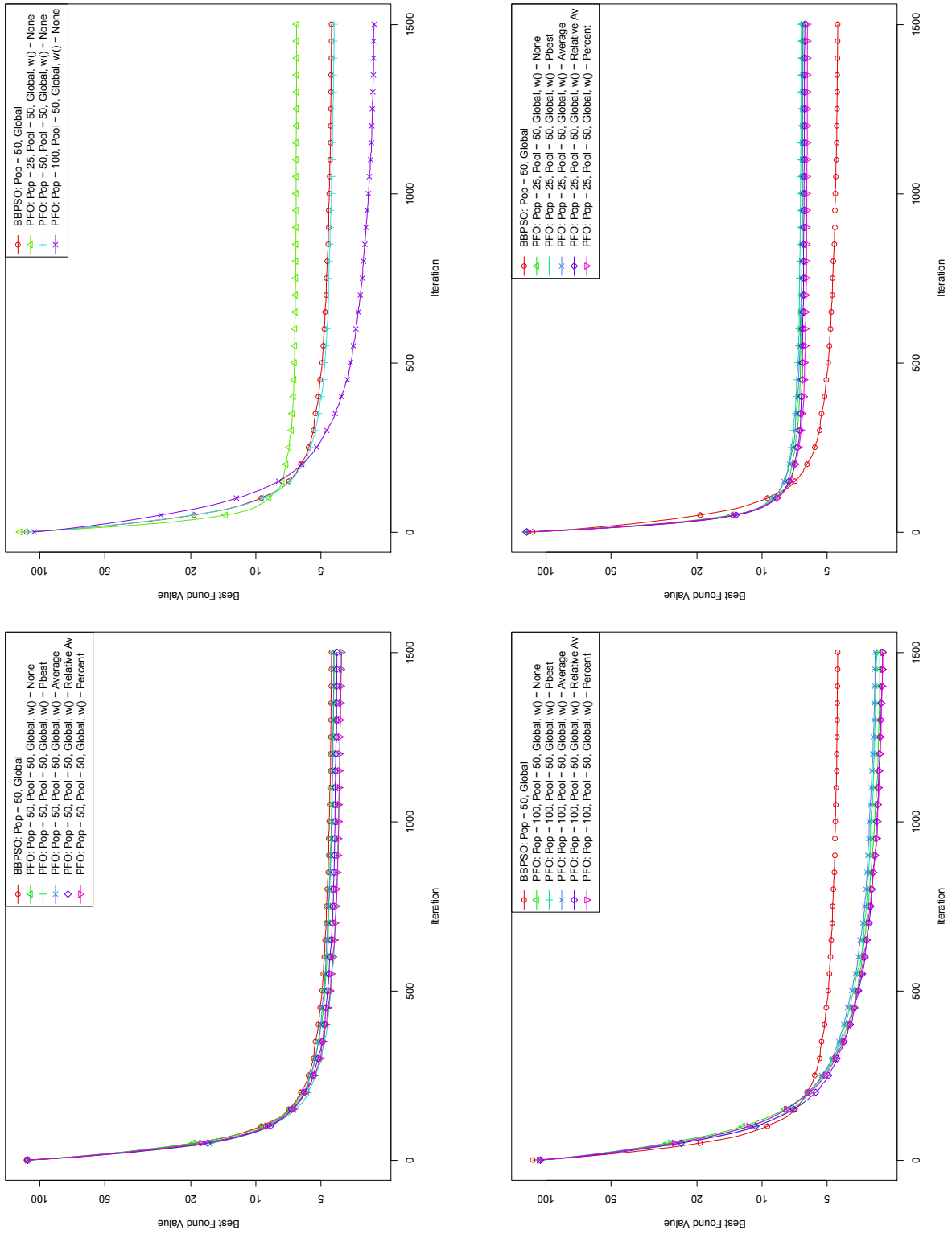


Figure 4.8: The Performance Results of the Various Algorithms for the Rastrigin Function using the Square Topology when the Number of Dimensions was 10.

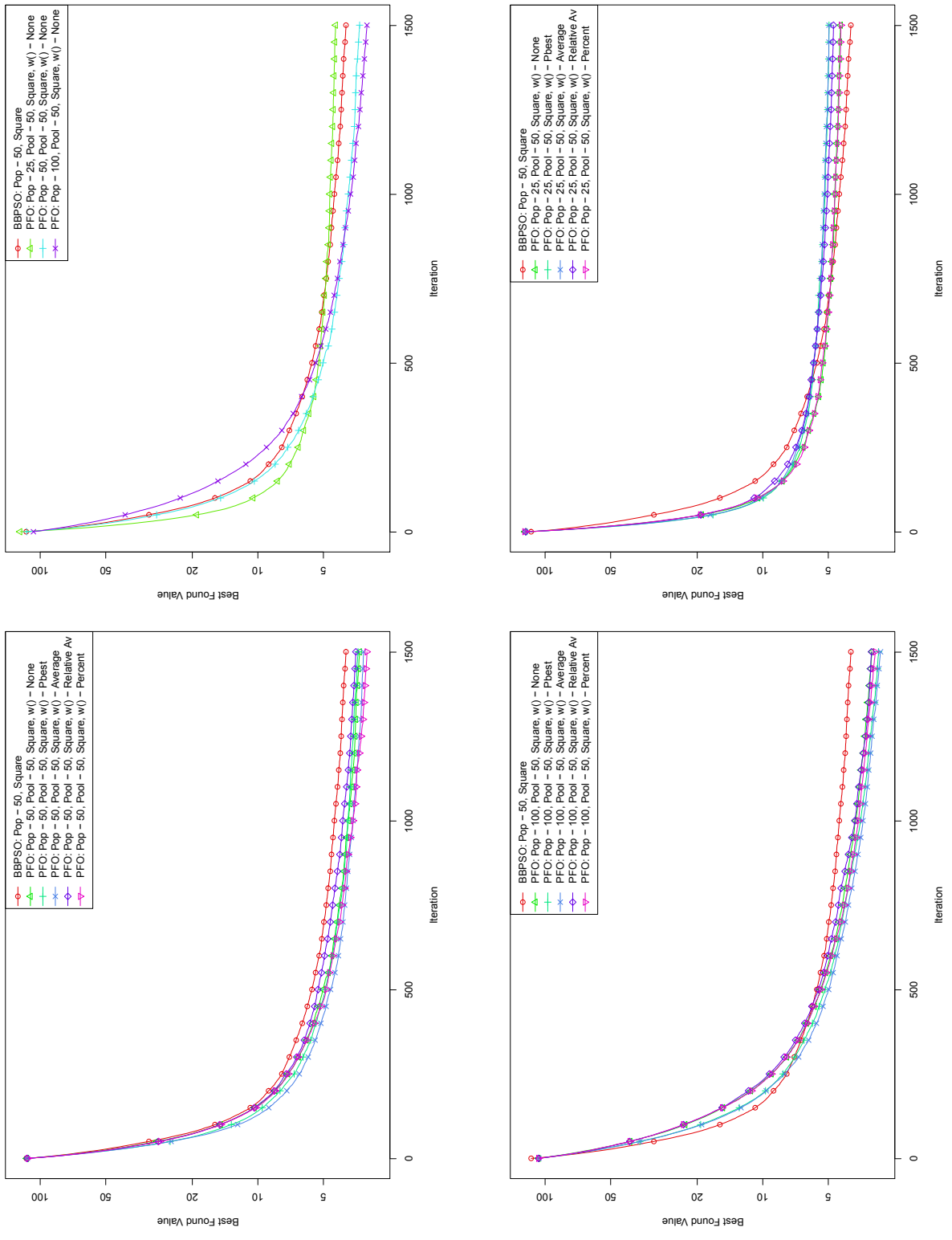


Table 4.7: The Performance Results of the Various Algorithms for the Rastrigin Function when the Number of Dimensions was 20.

Optimizer		**Average Result After 3000 Iterations**		
BBPS: Population 100, Global		17.55725		
PFO Pool Size 100 Global	**Weighting Scheme**	**PF Population size**		
		50	100	200
	none	20.85061	15.37354	11.39027
	pbest	**20.8465**	15.22319	**11.04656**
	average	21.33815	15.52382	11.20758
	relative	20.9652	15.93398	11.59487
	percent	21.02209	**14.49605**	11.14769
BBPS: Population 100, Square		16.23535		
PFO Pool Size 100 Square	**Weighting Scheme**	**PF Population size**		
		50	100	200
	none	16.96687	15.01759	13.7362
	pbest	17.63669	15.30112	13.75336
	average	17.40359	14.80903	**12.95031**
	relative	18.46826	16.02153	14.75224
	percent	**15.98447**	**14.37205**	13.44679

Figure 4.9: The Performance Results of the Various Algorithms for the Rastrigin Function when the Number of Dimensions was 20.

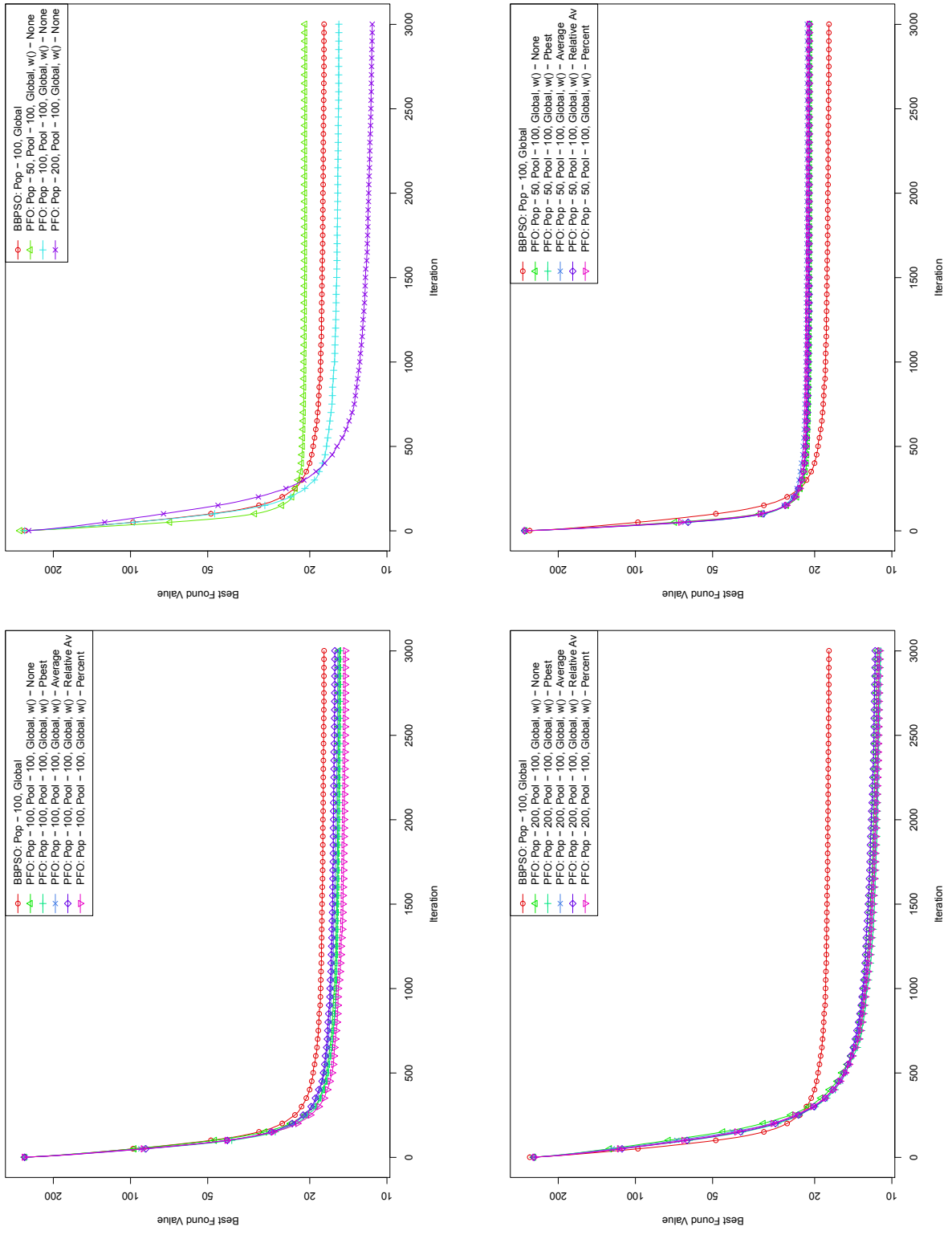
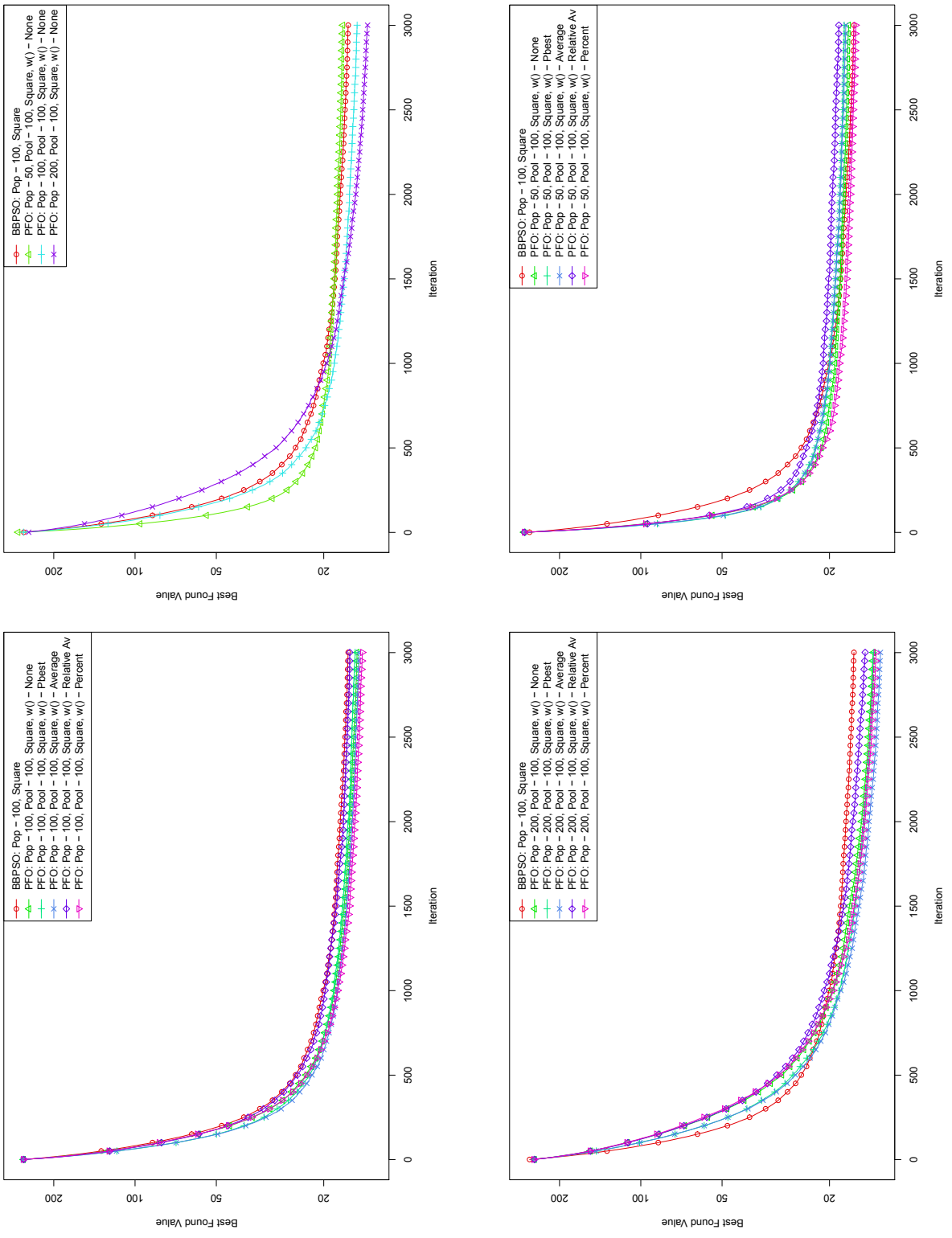


Figure 4.10: The Performance Results of the Various Algorithms for the Rastrigin Function using the Square Topology when the Number of Dimensions was 20.



was provided by the BBPS algorithm configured with a population of 10, to match the pool size of the PFO configurations. On the Griewank 2 function each configuration was run for 300 iterations and each test was repeated 500 times. The plotted results of the average best-found value by each configuration against the number of iterations are presented in Figure 4.11. Average final best-found values are tabulated in Table 4.8.

PFO configurations using a particle field population size of 5 performed poorly compared to the baseline BBPS. Variation in performance among the weighting schemes in this group was small, with one significant out-lier. This out-lier was the Relative Average weighting scheme, performing much worse than other weighting schemes. The best performing weighting scheme was the Average scheme.

Configurations with a population size of 10 performed comparably to the baseline BBPS. The variation in performance among the weighting schemes in this group was small, with no significant outliers. All weighting schemes performed worse than the PFO configuration that used no weighting scheme in this group.

With a population size of 20, the PFO algorithm performed much better than the baseline BBPS. Variation in performance among weighting schemes in this group was relatively small, with the best performing weighting scheme being the Relative Average scheme.

In the case of the Griewank 10 function, the PFO algorithm was tested using population sizes of 25, 50 and 100, with a pool size of 50 across all configurations. The Baseline BBPS was configured with a population of 50, to match the pool size of the PFO configurations. Each configuration was allowed 800 iterations and each test was repeated 500 times. The plotted results of the average best-found value by each configuration against the number of iterations are presented in Figure 4.12 and average final best-found values are tabulated in Table 4.9.

In terms of the final overall results, the PFO configurations with 25, 50 and 100 population sizes performed approximately the same as one another on the 10-dimensional Griewank function. This result is unique to the Griewank function, as results on all other test functions showed that the population size did have a dramatic impact. There is still a small impact, however, with the population size of

Table 4.8: The Performance Results of the Various Algorithms for the Griewank Function when the Number of Dimensions was 2.

Optimizer		**Average Result After 300 Iterations**		
BBPS: Population 10, Global		0.01291898		
	Weighting Scheme	**PF Population size**		
		5	10	20
PFO Pool Size 10 Global	none	0.069519851	**0.009907508**	0.00620303
	pbest	0.059657239	0.014337017	0.006538943
	average	**0.054423059**	0.012170993	0.007341048
	relative	0.259916617	0.011722682	**0.005651862**
	percent	0.080863936	0.011272416	0.006324045

25 performing, on average, very slightly worse than the population size of 50, which were likewise slightly worse than the population size of 100.

Finally, on the Griewank 20 function, population sizes of 50, 100 and 200 were tested, with a pool size of 100 across all configurations. The baseline BBPS was configured with a population size of 100, to match the pool size of the PFO configurations. On the Griewank 20 function each configuration was run for 1,000 iterations and each test was repeated 500 times. Plotted results of the average best-found value by each configuration against the number of iterations are presented in Figure 4.13 and average final best-found values are tabulated in Table 4.10.

Similar to the Griewank 10 function, on the 20-dimensional Griewank function, the configurations with 50, 100 and 200 population sizes performed about the same as one another. However, in this case, the population size of 50 performed slightly *better* than the population size of 100, which, in turn, performed slightly better than the population size of 200. This would suggest that as the dimensionality of the Griewank function increases, the larger population size becomes less beneficial, and more detrimental. Considering this along with the results of the Sphere function, this observation is in line with the idea that the Griewank function becomes "easier" as the number of dimensions is increased.

Figure 4.11: The Performance Results of the Various Algorithms for the Griewank Function when the Number of Dimensions was 2.

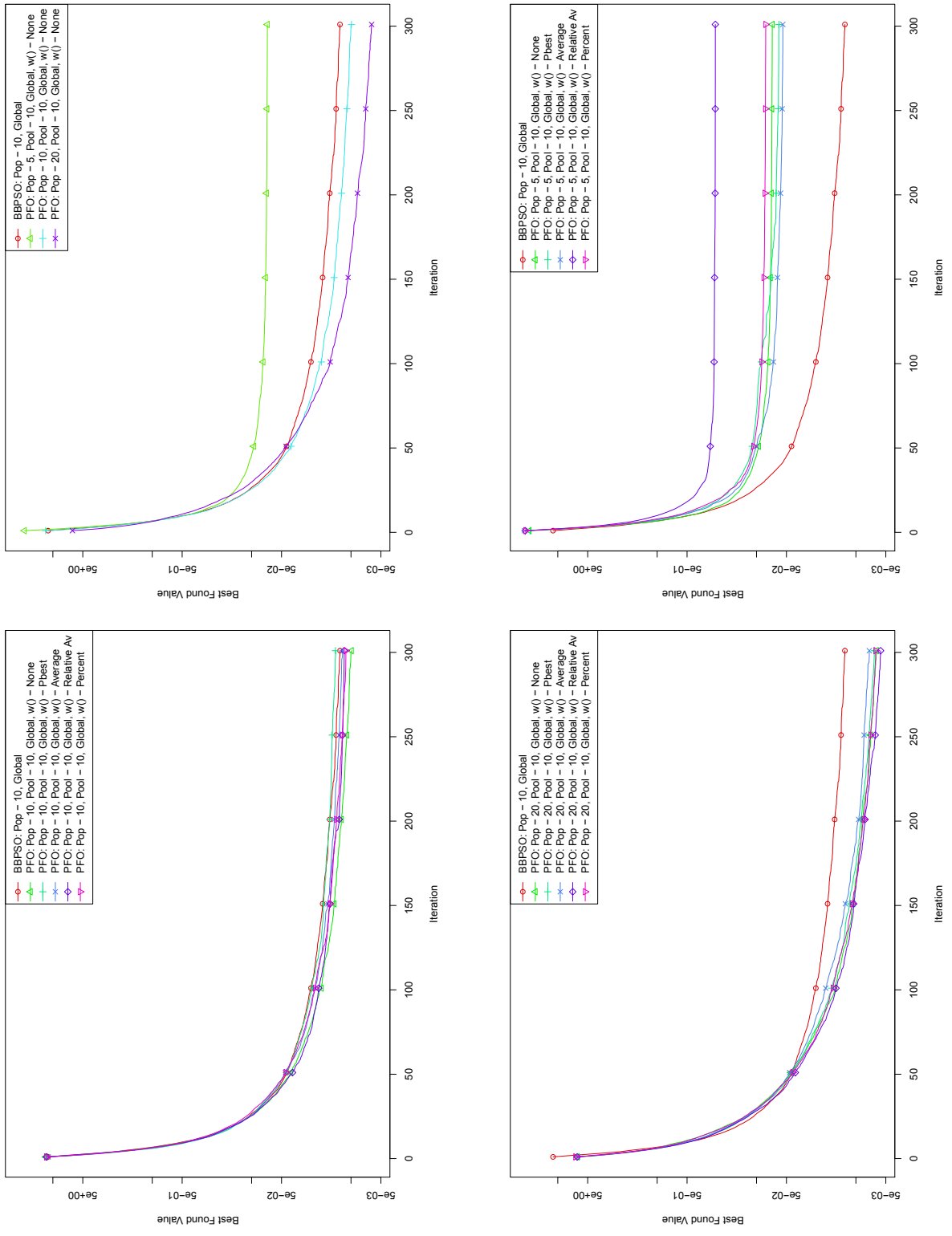


Table 4.9: The Performance Results of the Various Algorithms for the Griewank Function when the Number of Dimensions was 10.

Optimizer		**Average Result After 800 Iterations**		
BBPS: Population 50, Global		0.07132231		
	Weighting Scheme	**PF Population size**		
		25	50	100
PFO Pool Size 50 Global	none	**0.07650848**	0.06979642	0.06905533
	pbest	0.07663484	**0.06890227**	0.07101425
	average	0.07989598	0.07209505	0.07024574
	relative	0.07997927	0.07221773	**0.06830714**
	percent	0.07901374	0.0711286	0.06917764

Table 4.10: The Performance Results of the Various Algorithms for the Griewank Function when the Number of Dimensions was 20.

Optimizer		**Average Result After 1000 Iterations**		
BBPS: Population 100, Global		0.02426064		
	Weighting Scheme	**PF Population size**		
		50	100	200
PFO Pool Size 100 Global	none	0.02216336	0.02434762	**0.0266703**
	pbest	**0.02069583**	0.02453049	0.02803368
	average	0.02228252	0.02466496	0.02811329
	relative	0.02367083	0.02447077	0.02680295
	percent	0.02233828	**0.02335999**	0.02826242

Figure 4.12: The Performance Results of the Various Algorithms for the Griewank Function when the Number of Dimensions was 10.

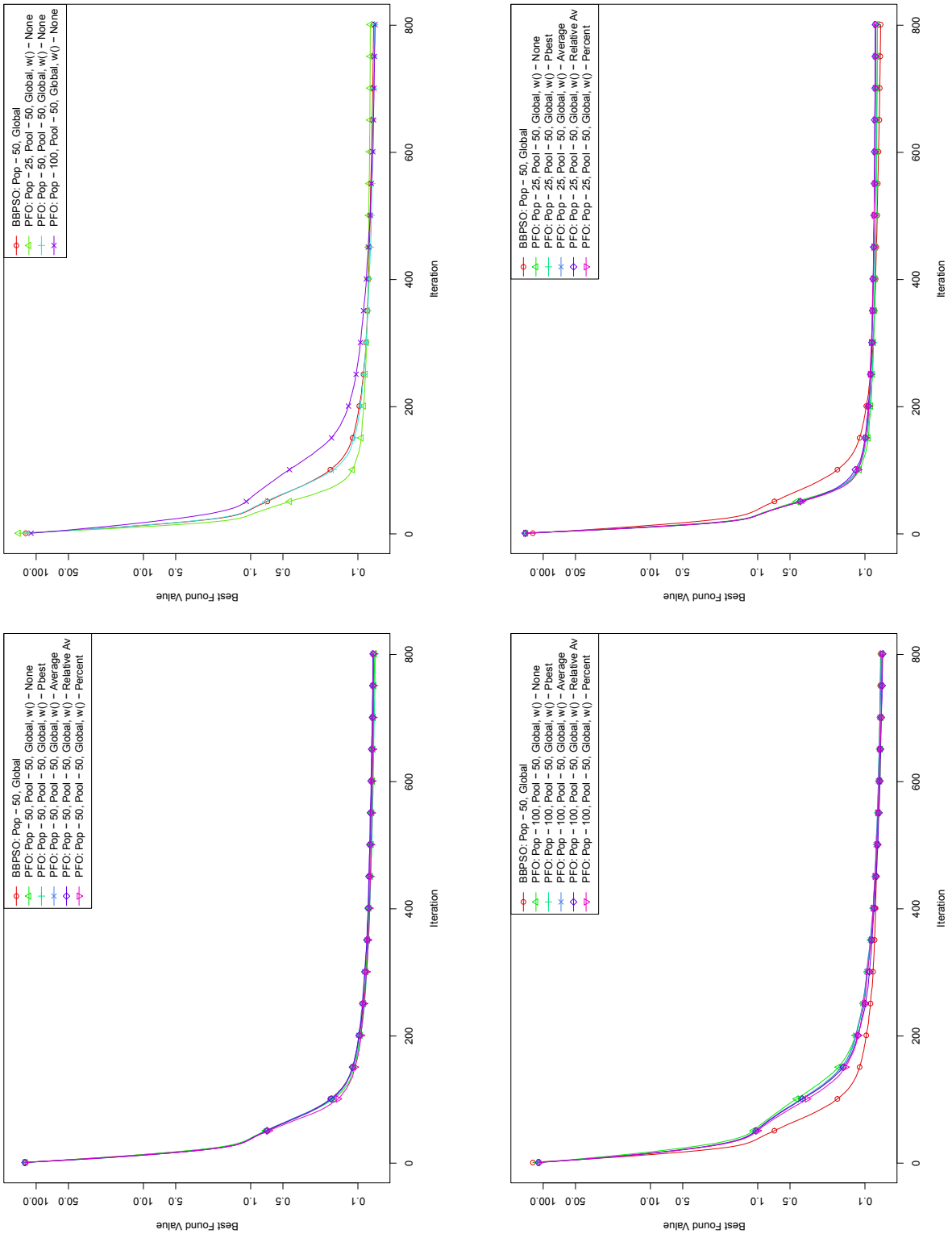
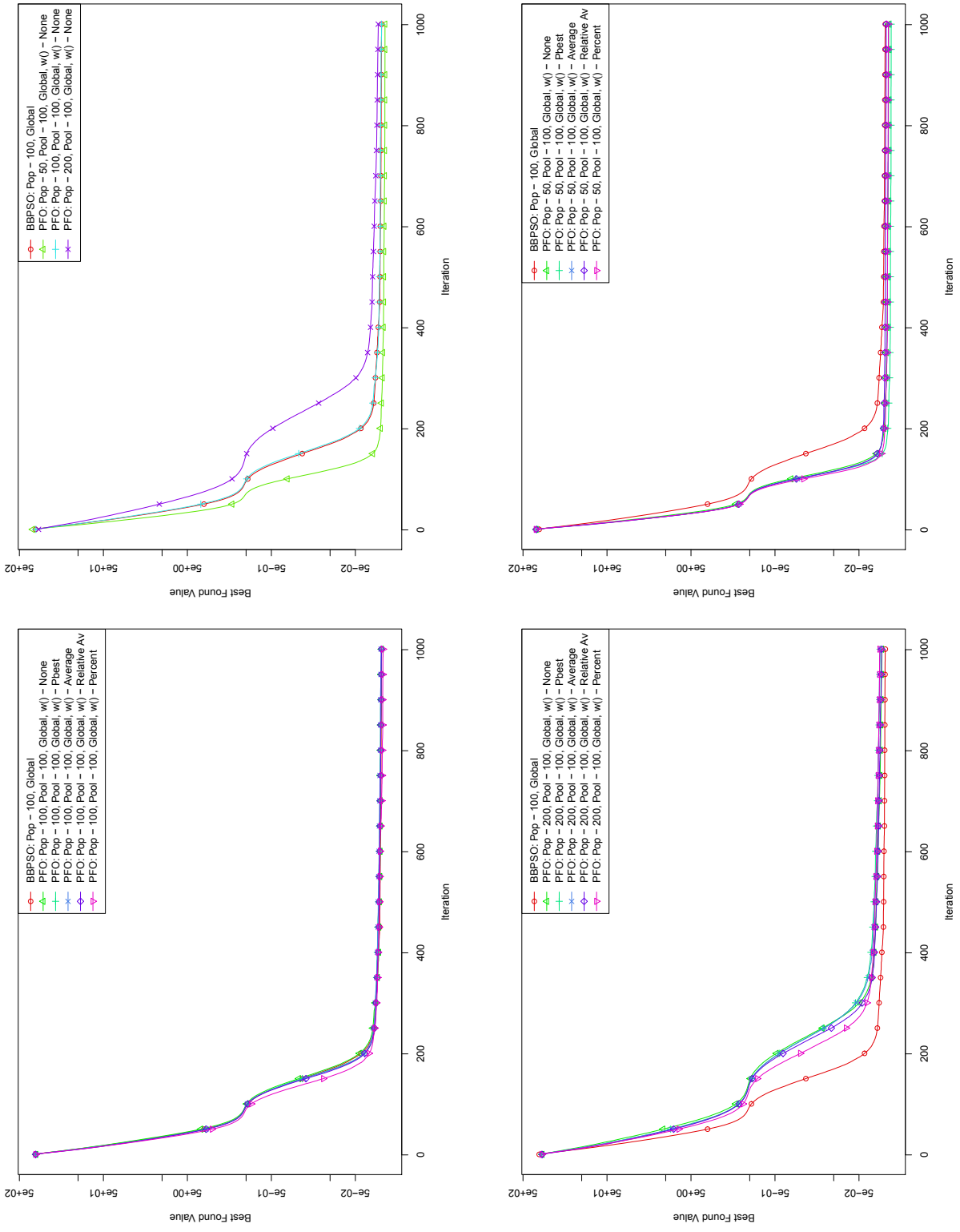


Figure 4.13: The Performance Results of the Various Algorithms for the Griewank Function when the Number of Dimensions was 20.



4.4.6 Ackley Function

Ackley's function is a test function with many local optima, as described in Section 2.6.5. Similar to the Rastrigin function, the Ackley function consists of a basic unimodal surface with added sinusoidal waves to create the local optima. However, the basic surface to which the noise is added is significantly different. This shape is a fairly flat surface with the global optimum lying within a sharp pit. The result of this is that the global optimum is fairly easy to find within the generally good area around it, but that "generally good area" is "generally more difficult" to locate than in the case of the Rastrigin or Griewank functions. Tests were performed on the Ackley function with 10 dimensions, and so particle field population sizes of 25, 50 and 100 were used with a pool size of 50. The baseline BBPS was configured with a population of 50 to match the PFO pool size. Each configuration was run for 800 iterations.

Due to the nature of the function, 1,000 test repetitions were performed for each test rather than the 500 used for other functions. This is because there exists a very small chance that the random initialization of the optimizer populations would result in the population becoming stuck in an extremely bad local optimum. When this did not happen, however, the optimizers easily converged to relatively good optima. Because of this, the test results can be interpreted as the ability of the algorithm to avoid these extremely bad areas when that situation arose. Plotted results of the average best-found value by each configuration against the number of iterations are presented in Figure 4.14, and average final best-found values are tabulated in Table 4.11.

PFO configurations using a population size of 25 performed the worst by a large margin. Variation among weighting schemes was small, except for a single outlier, which was the configuration that used no weighting scheme. This configuration performed significantly worse than the others. The best performing weighting scheme was the Average scheme, by a small margin.

With a population size of 50, the PFO algorithm performed slightly better than the baseline BBPS on average. Similarily to the configurations using a population size of 25, the variation among weighting schemes was small, with a single out-lier. In

Table 4.11: The Performance Results of the Various Algorithms for the Ackley Function when the Number of Dimensions was 10.

Optimizer		**Average Result After 800 Iterations**		
BBPS: Population 50, Global		0.0089554		
	Weighting Scheme	PF Population size		
		25	50	100
PFO Pool Size 50 Global	none	6.45359	0.004453149	$3.909314E^{-15}$
	pbest	0.09868803	0.007922931	$3.89509E^{-15}$
	average	**0.08097863**	0.02330885	$3.848843E^{-15}$
	relative	0.09156425	**0.002804872**	$\mathbf{3.841728E^{-15}}$
	percent	0.09419716	0.0089554	$3.919986E^{-15}$

this case, the out-lier was the configuration which used the Average weighting scheme, which performed notably worse than the others. The best performing scheme in this group was the Relative Average scheme.

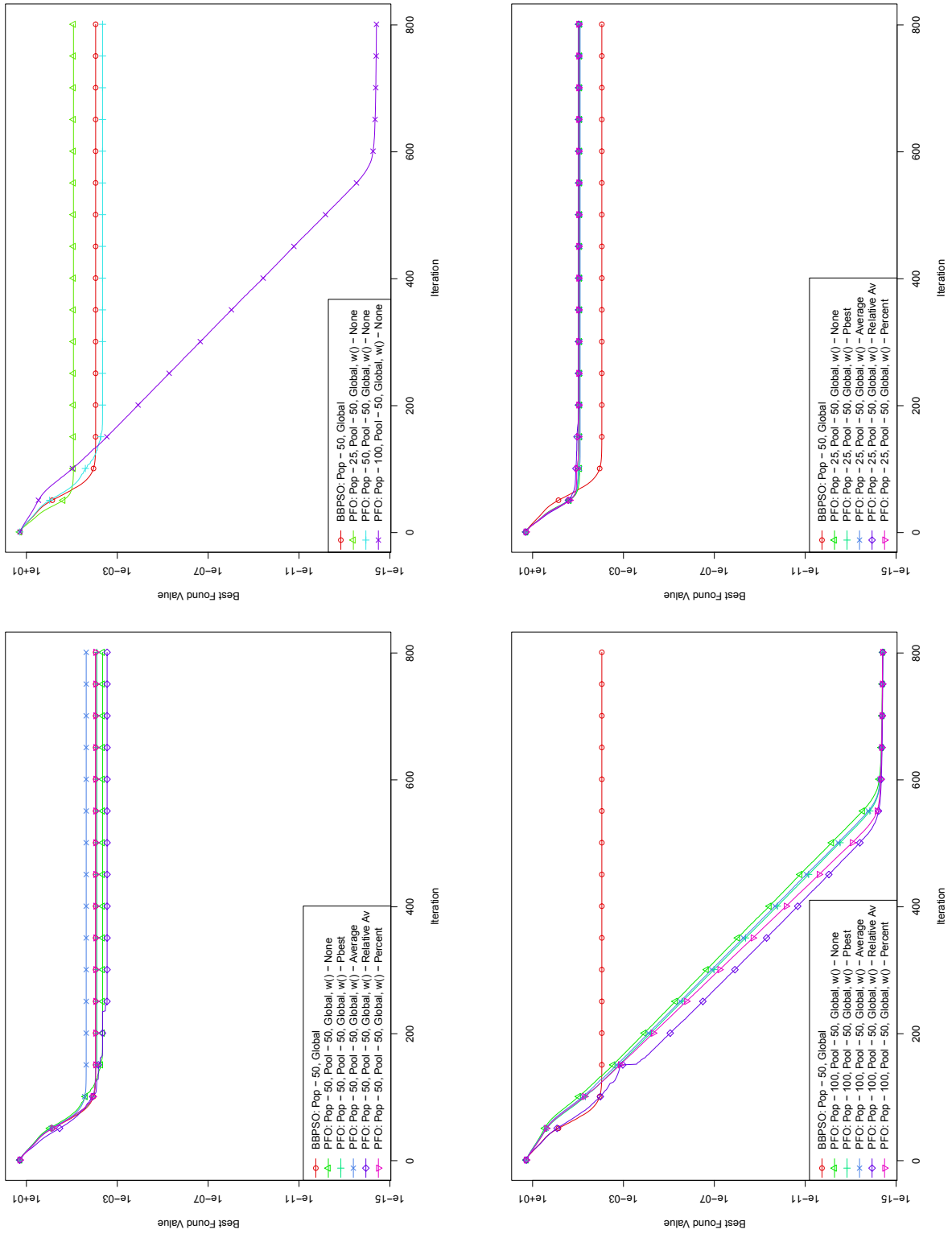
Configurations with a population size of 100 out-performed all others to an extreme degree, with variation among the weighting schemes being insignificant. This dramatic result shows the effect that a "bad initialization" can have on the algorithm's performance on the Ackley function. With twice the population size, the chance for the entire population to be initialized in a bad area is dramatically reduced, which in turn reduces the rate of occurrence of extremely poor final values. The best performing scheme, among these configurations, was the Relative Average scheme.

4.5 Discussion of Results

4.5.1 Overall Conclusions

From analyzing the results presented in the previous sections, it is possible to identify some patterns and gain insight into the behaviour of the PFO algorithm, and the effects of the algorithm's parameters. From a cursory glance, the test results show a great amount of variation among the different configurations of the PFO algorithm on each test function, with many configurations performing significantly better than the

Figure 4.14: The Performance Results of the Various Algorithms for the Ackley Function when the Number of Dimensions was 10.



baseline BBPS, and some performing significantly worse. This high variance among results shows that the different parameter values chosen had a significant impact on the behaviour and performance, of the PFO algorithm.

Among these results, a clear pattern emerges, suggesting a significant correlation between the particle field population size and the behaviour of the PFO algorithm. On all test function, with the exception of the Griewank 10 and 20 functions, the final results are divided by the particle field population size into distinct clusters. With the pool size parameter set as a single value for each function, the difference in the PFO configurations within these clusters is only the weighting scheme used. This suggests that the particle field population sizes chosen for each test "dominated" the choice of the weighting scheme, as there was no overlap between these groups. In addition to this distinct grouping, a correlation between the relative order of these groups and the particle field population size can be observed. With the exception of the Sphere function, a larger particle field population size was correlated with a better best value found. On the Sphere function, the opposite is observed, where a smaller population size is correlated with a better best value found. In addition to this exception we observe the previously mentioned exceptions of the Griewank 10 and 20 functions, on which the particle field population size did not show such a distinct clustering on the final results. These exceptions can be explained when considering the plotted test results showing the average best value found at each iteration. Rather than make a correlation between the particle field population size and the final result, it is more accurate to correlate the particle field population size with the "exploratory" behaviour of the algorithm. This correlation is in line with results on *all* test functions, including the Sphere and Griewank 10 and 20 functions.

In the case of the Sphere function, a higher "exploratory" behaviour would show poor performance compared to a more "exploitative" behaviour, due to the simplicity of the function. This is in line with the plotted results, as configurations with smaller particle field population sizes were observed to converge much faster than configurations with larger population sizes.

The results on the Griewank 10 and 20 functions also suggest this correlation, though not as clearly. The plotted results show this correlation in the early iterations,

although the final values are not distinctly separated by the particle field population sizes. It has been shown that the Griewank function becomes "easier" as the number of dimensions are increased, and so taking this into account, these "strange" results on the Griewank 10 and 20 functions can be explained. Results on the Griewank 2 function support the correlation between the particle field population size and the "exploratory" behaviour, with larger population sizes yielding dramatically better final results. On the Griewank 10 function, the final results are not distinctly separated by the population size, with some configurations using a smaller population size outperforming configurations using larger population sizes and vice-versa. However, when grouping the results based on the population sizes, and comparing the average final result between these groups, it is still the case that the configurations with larger population sizes perform slightly better, on average, than those with smaller population sizes. When comparing these observations with the results of tests on the Griewank 20 function, a possible pattern emerges which explains the observed exception. Similar to the Griewank 10 function, the results on the Griewank 20 function are not distinctly separated by particle field population sizes. However, the observed relation between the average final value for each group is, in fact, the opposite of the observed relation on the Griewank 10 function. In this case, the PFO configurations with a smaller population size performed, on average, slightly better than those with a larger population size. With the knowledge that the Griewank function becomes "easier" as the number of dimensions are increased, and recalling the results of tests on the Sphere function, a possible explanation for the results can be made. As the number of dimensions increases, the Griewank function becomes "easier", and so as the number of dimensions increases, "exploitative" behaviour becomes more valuable than "exploratory" behaviour. With a correlation between larger population sizes being much more "exploratory", and smaller population sizes being much more "exploitative", the difference in performance between the two on the Griewank function as the dimension increases would, intuitively, shrink, until the "exploitative" behaviour of small populations begins to yield better results than the "exploratory" behaviour of large populations. This is in line with the observed results on the Griewank function.

Results on the Griewank 2 function show "exploratory" large populations performing much better than "exploitative" small populations. Results on the Griewank 10 function show the performance difference approaching zero. Finally, results on the Griewank 20 function show the "exploitative" small populations out-performing the "exploratory" large populations by a slight margin.

The weighting scheme was expected to have a large impact on the PFO algorithm's behaviour. Test results show, however, that the weighting scheme chosen had a relatively small impact when compared with the particle field population size parameter. By grouping PFO configurations according to particle field population size, the effects of the weighting schemes can be compared within these groups. In addition to the tested weighting schemes, PFO configurations with no weighting scheme were tested to provide a baseline point of comparison. The effectiveness of the weighting schemes varied greatly across the test functions and population size groups. On some functions there was a significant variation in performance between weighting functions, and on some the variation was very small. Also, the relative performance of the weighting schemes within their groupings was inconsistent over the different test functions, with no significant patterns suggesting any clear correlation between the weighting schemes and specific behaviours. In addition to this, there were many cases where some weighting schemes performed worse than the configuration with no weighting scheme, and in some of these cases, this configuration was observed to perform significantly better than all of the weighting schemes. These observations suggest that the effects of the weighting scheme on the behaviour of the algorithm are quite unintuitive, and so deciding on what weighting scheme to use for a particular problem may require an extensive trial and error approach.

4.5.2 Effects of Square Population Topology

On a selected subset of test functions, tests were repeated with a "square" communication topology applied to the population of particle field individuals in order to investigate the effects of such a topology on the PFO algorithm's behaviour. The square topology restricts communication between particles to a toroidal grid, where

each particle has exactly four neighbours. Application of the square communication topology to traditional particle swarm algorithms has the effect of increasing the exploratory behaviour of the swarm, slowing convergence while decreasing the chance of the population becoming stuck in local optima. It was expected that applying a square topology to the PFO algorithm would lead to similar effects, increasing the exploratory behaviour, and the empirical test results confirm this.

On the Rastrigin 10 and 20 functions, the square topology showed a dramatic impact in performance when combined with a smaller particle field population size. This is because the Rastrigin function is multi-modal, with many local optima, and so requires a higher degree of exploratory behaviour to effectively optimize. However, with larger population sizes, the application of the square topology was observed to, in fact, result in poorer final values. Thus, when considering the plotted results of these cases, it becomes evident that the improvement of the best value found had not yet plateaued when reaching the maximum allowed number of iterations. With the global topology, the maximum number of iterations chosen was sufficient to allow the average best value found to plateau, but it appears that the same was not true when combining the square topology with larger population sizes. As a result of this, it is possible that the square topology will produce improved results in these cases if the maximum number of iterations is increased significantly.

On the Schaffer F6 and Rosenbrock functions, the square topology did not show an improvement in performance. The Schaffer F6 function is a difficult function, and so it was expected that the increase in exploratory behaviour would increase performance further. This was not the case, however, as the impact of the square topology on the final best values found was insignificant, though the plotted results show a slowed rate of convergence, which suggests that the exploratory behaviour was, indeed, increased. Results on the Rosenbrock function showed a decrease in performance, with poorer final values found by the configurations using the square topology. This was expected to a degree, as the Rosenbrock function is unimodal, and so pushing the exploratory behaviour further may not be beneficial. However, despite the lack of improvement in terms of final values, the plotted results show that the application of the square topology allowed many configurations to avoid plateauing

within the maximum number of allowed iterations.

4.6 Discussion of the PFO Algorithm

4.6.1 Expected Behaviour and Observed Behaviour

Empirical testing was carried out on the PFO algorithm with the motivation of studying the effects of different parameter values on the search behaviour. As such, it was not the motivation of this testing process to determine the best performing configuration of the PFO algorithm for each test function. Rather, the motivation of the testing process was to investigate the behaviour of the PFO algorithm, and the effects of the different parameters. From these test results we can make an appraisal of the PFO algorithm in terms of its expected behaviour against observed behaviour.

At a high level, the PFO algorithm performed as well as expected, displaying a wide range of behaviours for different parameter configurations, and with many configurations proving very effective for each test function. When taking a closer look at the behaviour of the scheme, there were, however, some surprises. As previously discussed in Section 4.5.1, the weighting schemes tested with the PFO algorithm proved to have a relatively small impact on its high-level behaviour. This result was unexpected, as the originating motivation behind the application of a weighting scheme was to introduce significant changes to the search process. One possible explanation for this result lies in how the weights were determined for each particle field individual. All weighting schemes tested applied a weight based only on the information from the current iteration, which may be insufficient to produce a wide variety of weights. The question of determining other weight-assigning strategies is unsolved.

The particle field population size had a dramatic impact on the exploratory behaviour of the algorithm. More specifically, it was the ratio of the particle field population size and the candidate solution pool size which had this effect. This ratio effects the rate at which particle field individuals are updated at each iteration. With a particle field population size larger than the candidate solution pool size, the

particle field individuals are less likely to be used to generate new points, and so will their values will be involved in an update less often. With a particle field population size smaller than the candidate solution pool size, the particle field individuals are more likely to be used to generate multiple new points, and so they will be updated more often. A higher rate of updating leads to a faster convergence of the population, which, in turn, displays a more exploitative behaviour. A lower rate of updating leads to slower convergence of the population which displays a more exploratory behaviour.

4.6.2 Benefits and Drawbacks of the PFO Algorithm

From the test results, various observations and conjectures were made regarding the effects that each parameter had on the behaviour of the PFO algorithm. In particular, the particle field population size parameter was observed to display a direct correlation with the balance between exploitative and exploratory behaviour of the algorithm. What this correlation implies, is that this parameter provides an *intuitive* control over the algorithm's behaviour. The parameters of traditional PSO algorithms, which control inertial and accelerational weighting, are fairly unintuitive to manipulate. In many cases, these parameters are simply given default values which have been deemed to be "generally good". Because of the direct control that the particle field population size parameter displays over the exploitative and exploratory behaviour of the PFO algorithm, it may be easier to use in practice, as determining what parameter values result in good results on an unknown function will be more intuitive and require less trial and error.

However, not all parameters of the PFO system proved to be as intuitive. Several different weighting schemes were tested, each designed to promote a specific behaviour, as described in Section 3.4. Unfortunately, the effects of each weighting scheme on the algorithm's performance varied significantly between the different test functions, showing no clear patterns. In fact, on some test functions, all weighting schemes resulted in poor performances when compared to the baseline weight-free PFO configuration. What this implies, is that choosing a weighting scheme will, in

practice, be quite unintuitive, as the effect that each has on the algorithm's performance appears to depend almost entirely on the function itself.

One attractive property of the canonical PSO algorithm is the simplicity of designing it. Indeed, it can be implemented in very few lines of code, without any complex data structures or additional modules. This simplicity allows the PSO algorithm to be applied to many problems, or included as a small part of a larger system without difficulty. The PFO algorithm is also fairly simple to implement, as it is made up of very simple component parts. However, it is still relatively complex when compared with the canonical PSO algorithm. This is not a primary concern when appraising the PFO algorithm, but it is worth noting, because despite the multitude of advances and developments made to improve the PSO algorithm, the basic canonical PSO algorithm remains a very popular tool due to this ease of implementation.

4.6.3 A Hybrid Perspective of the PFO Algorithm

The PFO algorithm was created by introducing a number of changes and additions to the BBPS algorithm, and as such, the two algorithms are closely related. Although abstracted from the traditional particle swarm model itself, the BBPS algorithm still adheres closely to the core "particle swarm" paradigm in which population of particles, each representing candidate solutions, explore the solution space, communicating and influencing one another in order to search for the global optimum. The changes made to the BBPS algorithm to produce the PFO algorithm introduce a significantly different *perspective* to the process, which does not necessarily adhere completely to the core particle swarm paradigm. This brings up a philosophical question as to whether or not the PFO algorithm should be considered to be a "pure" particle swarm algorithm, or a hybrid algorithm of some kind.

The PFO algorithm is distinct from traditional particle swarm models primarily because of the method by which candidate solutions are managed. Unlike traditional particle swarm algorithms, the population of particle field individuals within the PFO algorithm do not directly represent candidate solutions themselves. The particle field individuals, which make up this population, behave in a way which is very similar to

the BBPS algorithm, but instead of each individual maintaining a position as a candidate solution point, each individual instead maintains a position in terms of a more abstract random distribution of potential candidate solution points. This population of particle field individuals does not directly perform the search itself, as would be the case in traditional particle swarm algorithms. Instead, this population is used to define a mixture distribution, which is then sampled a number of times to generate new candidate solution points. This abstraction is what blurs the formulation of it being a particle swarm algorithm.

If we move our focus from the underlying particle swarm principles at work within the particle field population, the algorithm can be conceptualized as proceeding in the following way. A random distribution is created, which represents how the solution space should be explored in the current iteration. Following this, a number of candidate solutions are sampled from this distribution, and assigned values by evaluating them using the objective function. Finally, these candidate solutions are used to update the random distribution for the next iteration. From this perspective, the PFO algorithm proceeds in a way similar to Bayesian learning methods such as Thompson Sampling or Bayesian Optimization. From an abstract perspective, these algorithms begin with a prior distribution, sample that distribution, and then use the knowledge gained from the sampled point, in order to form a posterior distribution. However, a key distinction exists between the PFO algorithm and these methods, which is that the PFO algorithm does not use Bayesian inference to update this distribution, but rather uses particle swarm principles as the underlying process instead. As such, the PFO algorithm can be considered, in some ways, as a hybridization of the particle swarm paradigm with the general inclusion of a Bayesian learning strategy.

4.6.4 Potential Violation of SI Principles

The PFO algorithm is similar to the BBPS algorithm on which it is based, but the changes and abstractions made result in a considerably different high-level concept. The PSO and BBPS algorithms are both concretely SI algorithms and adhere strictly to the concepts of self-organization and decentralization. In particular, the PFO

algorithm abstracts the system into two populations. The population of particle field individuals acts similarly to the population of the BBPS algorithm, and it, in and of itself, adheres strictly to the SI definition. However, this population can now be considered to be *guiding* the search, rather than carrying out the search itself. The second population consists of the pool of candidate solutions to be generated and evaluated at each iteration. The generation of these candidate solutions is *guided* by the mixture distribution defined by the population of particle field individuals. It is the act of separating the population of candidate solutions which has introduced a potential violation, or blurring, of SI principles.

Each candidate solution is generated by sampling the random mixture distribution defined by the population of particle field individuals. This is achieved, for each candidate solution, by randomly selecting a particle field individual from the population according to assigned weightings, and then generating a new point based on that selected individual. In order to achieve this random selection, the candidate solutions must each be *globally aware* of the population of particle fields. This introduces a potential violation of SI principles, as such a global awareness, which, in turn, implies a certain degree of centralization. Specifically, it is the link between these two populations which is problematic.

This is not much of an issue when considering the standard computational model, but presents a problem when considering the application of the algorithm to alternative models of computation. Since the population of candidate solutions requires such a global awareness of the population of particle field individuals, it is not as flexible as an algorithm which adheres strictly to the SI principles. This flexibility is a desirable trait of SI algorithms, and so restoring a strict adherence to SI principles may be desirable.

A similar blurring of SI principles is, in fact, also present in the canonical, global best PSO model. Since each particle in the canonical PSO population must be aware of the global best-found point, a similar issue of global awareness is present. However, this issue has been addressed in the PSO model by applying communication topologies to the population. Restricting communication of best points to neighbourhoods adds enough flexibility to the PSO algorithm that it can be considered to be a strictly SI

algorithm.

Application of communication topologies can potentially be used to address this problem in the PFO algorithm as well. The core issue arises because of the global awareness of the candidate solution points. Applying a communication topology to the generation of candidate solution points could potentially address this problem, but such a communication topology may not be in line with the high level concept of the PFO algorithm. The PFO algorithm works by using a population of particle field individuals which invoke PSO principles to define a search area as a random mixture distribution. This mixture distribution is then sampled in order to explore the space. To sample this complex distribution, the candidate solutions must be aware of the global population of particle field individuals. As a result, applying a communication topology to the candidate solutions, limiting which particle field individuals each is aware of, changes the high-level concept somewhat, as the distribution of generated points will not match the overall distribution defined by the entire population of particle field individuals.

4.7 Conclusions

In this chapter we have presented the empirical testing process carried out on the PFO algorithm, as well the results of each test performed. With these results, we have identified correlations between the parameters of the PFO algorithm and the range of the behaviours observed. Finally, we have discussed the PFO algorithm itself in the context of the observed test results, and included discussions of the benefits and drawbacks of the algorithm, as well as its distinction from traditional particle swarm algorithms.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Particle swarm algorithms have been a popular topic of research and development since the inception of the original Particle Swarm Optimization (PSO) algorithm in 1995. Many different strategies have been explored to change, or improve, the PSO algorithm, with distinct motivations and results. The initial stages of our research into these various strategies led us to study the Bare Bones Particle Swarm (BBPS) algorithm, which simplified and abstracted the PSO algorithm. Upon discovering the potential for a further level of abstraction, our research shifted to investigating this new abstraction, and the implications of the new perspective it provided.

This thesis was written with two primary objectives in mind. The first objective was to present an abstracted perspective of the behaviour of the BBPS algorithm. With this abstracted perspective came many new opportunities for the development of the algorithm. The second objective of this thesis was to present the newly created PFO algorithm, which was designed by exploring some of the new opportunities presented by this abstracted perspective of the BBPS algorithm.

Both of these objectives have been thoroughly achieved and explored. The abstracted perspective of the behaviour of the BBPS algorithm presents many new opportunities for development which are not evident with the traditional particle swarm perspective. With this perspective as a foundation, we have explored a number of

these new opportunities, applying a number of changes and additions to the BBPS algorithm, resulting in the distinct Particle Field Optimization (PFO) algorithm. The PFO algorithm itself was a success, proving to be effective in the optimization of a variety of different functions of high and low dimensionalities and differing complexities. The PFO algorithm also presents a rich framework for further development and further exploration of the new, abstracted perspective presented.

5.2 Summary of Work Done

Chapter 1: We began this thesis with an introduction of the problem of optimization, the concept of a black-box function, and the unique difficulties associated with optimizing these functions. Next, we presented the strategy of meta-heuristic optimization and the motivations behind it. Finally, we detailed the motivations behind the research presented in this thesis, and our objectives for this research.

Chapter 2: Starting with the core concept of Swarm Intelligence (SI), we presented a basic framework of knowledge required to understand our contributions. From the observations of SI in nature, to the application of SI principles to problem solving, the history of the concept was detailed. Building from this, the fundamental particle swarm paradigms and principles were described, and a survey of the state-of-the-art of particle swarm algorithms was presented.

Chapter 3: With this framework in place, the abstracted perspective of the BBPS algorithm was then described. The justification for this abstraction was presented, followed by the step-by-step process of leveraging this abstraction to discover a new perspective on the behaviour of the BBPS algorithm. From this new perspective, a number of modifications and additions were presented, along with the motivations behind each decision. These modifications and additions culminated in the distinct PFO algorithm, which was then finalized and formalized.

Chapter 4: With the PFO algorithm formalized, we then carried out rigorous empirical testing with the motivation of investigating its behaviour and the effect that its parameters had on that behaviour. From the results of these tests, the effects of each parameter were discussed, and conjectures made about the impact that each

had on the behaviour of the PFO algorithm. With these observations, we submitted a final discussion and appraisal of the PFO algorithm. These discussions included its benefits and drawbacks, the observed *versus* expected behaviours, and a potential perspective of the algorithm as a hybrid scheme.

5.3 Future Work

5.3.1 Application of Particle Swarm Developments to the PFO Algorithm

The PFO algorithm has an overall behaviour which is distinct from traditional particle swarm algorithms, but the core particle swarm framework is still present within the population of particle field individuals. Because of this, it is possible to incorporate many developments originally created for traditional particle swarm algorithms into the PFO algorithm. In particular, one such particle swarm development presents an interesting possibility for integration into the PFO algorithm. This is the so-called Fully Informed Particle Swarm (FIPS) algorithm, described in Section 2.4.1. The FIPS algorithm has already been shown to integrate naturally with the BBPS algorithm, resulting in the Gaussian FIPS algorithm, as described in Section 2.4.2. This Gaussian FIPS algorithm modifies the process involved in the construction of the multivariate Gaussian distributions from which each BBPS particle samples its next position. Rather than choosing a single neighbouring particle, the Gaussian FIPS algorithm uses a weighted average of all neighbouring particles to construct this distribution. This process can be directly applied to the population of particle field individuals within the PFO algorithm, as these particle field individuals construct distributions in the same way as the BBPS particles. Application of the Gaussian FIPS process to the PFO algorithm would drastically modify the mixture distribution created by the particle field population at each stage of the algorithm, and so presents a clear opportunity for further development.

Introducing dynamic control of parameters to the PSO algorithm has been a popular topic of research and development, and has resulted in dramatic performance

increases in some cases. A basic example of this is a standard PSO algorithm with a dynamically controlled inertial weighting parameter. In this algorithm, the PSO system is initialized with a high inertia value, which is decreased over the course of the simulation. The motivation for this is to promote exploratory behaviour in the early stages of the algorithm, but to then shift toward exploitative behaviour in the later stages. It is very likely that applying a similar control to the PFO algorithm's particle field population size parameter would result in a similar increase of performance. Empirical testing performed on the PFO algorithm suggests that there is a direct correlation between the particle field population size and the balance between the exploratory and exploitative behaviours, and so this parameter is a good candidate for such a dynamic control.

5.3.2 Improvement of Weighting Schemes

The weighting scheme parameter of the PFO algorithm was originally intended as a way to introduce a dramatic change in the search behaviour without interfering directly with the underlying particle swarm principles at work. As previously discussed, however, the impact of the different weighting schemes tested was much less significant than hoped, as well as it proving to be very problem specific. Because of this, the improvement of the weighting scheme portion of the PFO algorithm presents a clear opportunity for further research and development. The observed effects of the weighting schemes were surprising, and so it is not yet clear *why* these different weighting schemes did not show the expected impact. Further investigation into the weighting schemes themselves, then, is required to understand this.

In addition to the basic improvement of the weighting schemes themselves, the process of assigning weight values to each particle field individual itself provides an opportunity for further investigation. The weighting schemes proposed and tested were all very simple, but with the amount of information available to the system, much more sophisticated methods are possible. Indeed, a wide range of methods are possible, as the assignment of weighting values operates without directly impacting

the particle swarm principles at work within the population of particle field individuals. In particular, it would be possible to create weighting schemes which consider the full history of each particle field individual, rather than relying only on the information contained in the current iteration. Such weighting schemes may be able to provide a greater variety of different weighting values and more effectively identify "good" individuals on which to focus the search process.

Bibliography

[1] T. Achalakul, O. Udomkasemsub, T. Tirapat, and L. Xiaorong. Cost optimization for scientific workflow execution on cloud computing. *International Conference on Parallel and Distributed Systems*, pp. 663–668, 2013.

[2] P. J. Angeline. Using selection to improve particle swarm optimization. *IEEE International Conference on Computational Intelligence*, 1998.

[3] J. Barrera and C. A. Coello. Test function generators for assessing the performance of PSO algorithms in multimodal optimization *Handbook of Swarm Intelligence*, Vol. 8 pp. 89–117, 2011.

[4] O. Castillo, P. Melin and F. Valdez. Fuzzy control of parameters to dynamically adapt the PSO and GA algorithms. *IEEE International Conference on Fuzzy Systems*, pp. 1–8, 2010.

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

[6] M. Clerc, R. Rojas and M. Zambrano-Bigiarini. Standard particle swarm optimisation 2011 at CEC-2013: A baseline for future PSO improvements. *IEEE Congress on Evolutionary Computation*, pp. 2337–2344, 2013.

[7] R. Eberhart and Y. Shi. Comparing inertia weights and constriction factors in particle swarm optimization. *Congress on Evolutionary Computation*, Vol. 1 pp. 84–88, 2000.

[8] T. Giraud, J. S. Pedersen, and L Keller. Evolution of supercolonies: The argentine ants of southern europe. *Proceedings of the National Academy of Sciences*, Vol. 99 pp. 6075–6079, 2002.

[9] N. Higashi and H. Iba. Particle swarm optimization with gaussian mutation. *IEEE Swarm Intelligence Symposium*, pp. 72–79, 2003.

[10] J. Kennedy. Bare bones particle swarms. *IEEE Swarm Intelligence Symposium*, pp. 80–87, 2003.

[11] J. Kennedy. Probability and dynamics in the particle swarm. *Congress on Evolutionary Computation*, Vol. 1 pp. 340–347, 2004.

[12] J. Kennedy and R. Eberhart. Particle swarm optimization. *IEEE International Conference on Neural Networks*, Vol. 4 pp. 1942–1948, 1995.

[13] J. Kennedy and R. Mendes. Population structure and particle swarm performance. *Congress on Evolutionary Computation*, Vol. 2 pp. 1671–1676, 2002.

[14] M. Locatelli. A note on the griewank test function. *Journal of Global Optimization*, Vol. 25 pp. 169–174, 2003.

[15] M. Lovbjerg, T. K. Rasmussen, and T. Krink. Hybrid particle swarm optimiser with breeding and subpopulations. *Genetic and Evolutionary Computation Conference*, pp. 469–476, 2001.

[16] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, Vol. 8 pp. 204–210, 2004.

[17] G. Mokryani and P. Siano. Assessing wind turbines placement in a distribution market environment by using particle swarm optimization. *IEEE Transactions on Power Systems*, Vol. 28 pp. 3852–3864, 2013.

[18] C. K. Monson and K. D. Seppi. Exposing origin-seeking bias in pso. *Conference on Genetic and Evolutionary Computation*, GECCO '05, pp. 241–248, 2005.

[19] E. Ozcan and C. K. Mohan. Analysis of a simple particle swarm optimization system. *Intelligent engineering systems through artificial neural networks*, Vol. 8 pp. 253–258, 1998.

[20] E. Ozcan and C. K. Mohan. Particle swarm optimization: Surfing the waves. *Congress on Evolutionary Computation*, Vol. 3 pp. 1939–1944, 1999.

[21] M. E. H. Pedersen. Good parameters for particle swarm optimization. *Hvass Lab., Copenhagen, Denmark, Tech. Rep. HL1001*, 2010.

[22] A. Ratnaweera, S. Halgamuge, and H. C. Watson. Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, Vol. 8 pp. 240–255, 2004.

[23] C. W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pp. 25–34, 1987.

[24] M. Settles and T. Soule. Breeding swarms: a ga/pso hybrid. *Conference on Genetic and Evolutionary Computation*, GECCO '05, pp. 161–168, 2005.

[25] Y. Shi and R. Eberhart. A modified particle swarm optimizer. *IEEE World Congress on Computational Intelligence*, pp. 69–73, 1998.

[26] Y. Shi and R. Eberhart. Empirical study of particle swarm optimization. *Congress on Evolutionary Computation*, Vol. 3 pp. 1945–1950, 1999.

[27] W. M. Spears, D. T. Green, and D. F. Spears. Biases in particle swarm optimization. *Int. J. Swarm. Intell. Res.*, Vol. 1 pp. 34–57, April 2010.

[28] F. van den Bergh and A. P. Engelbrecht. A study of particle swarm optimization particle trajectories. *Information Sciences: an International Journal*, Vol. 176 pp. 937–971, 2006.

[29] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, Vol. 1 pp. 67–82, 1997.

[30] Z. H. Zhan, J. Zhang, and H. S. H. Chung. Adaptive particle swarm optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, Vol. 39 pp. 1362–1381, 2009.