NUI Galway
OÉ Gaillimh

NATIONAL UNIVERSITY OF IRELAND, GALWAY
COLLEGE OF ENGINEERING & INFORMATICS
DISCIPLINE OF ELECTRICAL & ELECTRONIC ENGINEERING

B.Eng. ELECTRONIC & COMPUTER ENGINEERING

Application of Graph Theory to Complex Dynamic Topologies in Particle Swarm Optimization

---

Student:
David Newell

Supervisor:
Dr. Enda Howley

31 March 2017

Declaration of Authorship

I, DAVID NEWELL, declare that this thesis titled, 'Application of Graph Theory to Complex Dynamic Topologies in Particle Swarm Optimization' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a bachelor's degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

Abstract

Particle swarm optimisation (PSO) is a stochastic, heuristic optimization algorithm. Its basic operation involves placing a swarm of particles in a problem space that then interact with each other to converge on a best known solution after a defined number of iterations. One of the key features of the PSO algorithm is the unique interactions between particles in the swarm. The swarm topology describes the ability of particles to share information with each other. Thus, the topological network governs the interactions between particles.

This work examines novel means of describing the operation of PSO using graph theory. The application of graph theory to the topology created by the particle swarm is particularly interesting when the structure of the graph changes over time, as in dynamic topologies. The *average path length* is a graph theory parameter that is correlated to the speed of information transfer in a graph. The application of this parameter to PSO is shown to reveal insights beyond simple measures such as convergence values. It is later used to understand the effect of the initial topology on the performance of the entire swarm. Finally, this research also proves the ability to positively influence performance of PSO by controlling the average path length parameter.

Acknowledgements

Firstly, I would like to thank my project supervisor, Enda Howley. His guidance, enthusiasm and advice throughout the project has been inspiring. His support has not gone unnoticed.
I would like to thank my classmates for their input and contribution to this thesis, and during my four year degree.
I would also like to thank my parents and my sister, Lisa, who proof read much of this thesis. Their constant interest and support while carrying out this research and throughout all my work is very much appreciated.

Contents

List of Figures

List of Tables

Abbreviations

PSO	Particle Swarm Optimization
SPSO	Standard Particle Swarm Optimization
SLR	Singly-linked Ring
FIPS	Fully Informed Particle Swarm
PSO AWL	Particle Swarm Optimization Avoidance of Worst Locations
NPSO	New Particle Swarm Optimization
NrPSO	Non-revisiting Particle Swarm Optimization
MPSO	Multi-swarm Particle Swarm Optimization
CPSO	Charged Particle Swarm Optimization
GIDN	Gradually Increasing Directed Neighbourhood
SGIDN	Sphere Gradually Increasing Directed Neighbourhood
CGIDN	Connected Gradually Increasing Directed Neighbourhood
GGIDN	Gradual Gradually Increasing Directed Neighbourhood
LGIDN	Linear Gradually Increasing Directed Neighbourhood
EMP	Enhanced Memory Particles
QSO	Quantum Swarm Optimization
APL	Average Path Length
EA	Evolutionary Algorithms
SI	Swarm Intelligence
GA	Genetic Algorithms
DE	Differential Evolution
ABC	Artificial Bee Colony
ED	Economic Dispatch
EED	Environmental/Economic Dispatch Problem
DEED	Dynamic Economic Emissions Dispatch

VRP	Vehicle Routing Problem
MPT	Modern Portfolio Theory
PO	Portfolio Optimization
NFL	No Free Lunch
ANN	Artificial Neural Network

Chapter 1

Introduction

Particle swarm optimization (PSO) is a swarm intelligence (SI) optimization algorithm. PSO can be described as a computational intelligence algorithm similar to evolutionary algorithms.

PSO was first introduced by Kennedy and Eberhart in 1995 [3] as they tried to model the social behaviour of birds flocking. They soon realised the general applicability of PSO in solving a wide range of complex and nonlinear optimization problems. PSO is favourable due to its good performance, simple implementation and low computational costs, but is not without its own limitations.

The general operation of PSO consists of placing a number of particles in a multidimensional problem space. Each particle is initialized with a certain velocity. Every particle evaluates the fitness of its current position which informs the updating of individual velocities in two ways. Firstly, each particle will update its velocity based on its own best fitness. The velocity of a particle is also influenced by the best fitness among a subset of other particles in the swarm known as its neighbourhood. The movement of particles is therefore controlled by personal and social influence [3]. This allows particles to act as individuals and as a collective swarm. The updated velocity is applied to a particles current position to calculate its new position. This method is repeated in an iterative manner. The combination of a particles personal memory of its best previously visited location and the sharing interactions between particles leads the swarm to converge on the best known solution. Earlier exploration transitions into convergence when

particles focus their search around a promising area to determine the best possible solution. It should be noted that, similar to all optimization algorithms, PSO cannot guarantee the optimum solution will be found. This downside is offset by the fact that PSO is a good general purpose problem solver that has the utility to be applied to a wide range of problems.

Over the years, significant research has been done to understand and improve the performance of PSO. Much of this work has focused on key elements of the algorithms operation such as particle initialization [4], swarm size [5], particle velocity [6][7], boundary handling [1] and swarm topology [8][9]. A variety of new PSO variations have been suggested including the use of multi-swarms [10] and avoidance strategies [11]. The Standard PSO (SPSO) was defined by Bratton [2] to act as a benchmark for all future research.

Various research has also investigated the application of PSO to a wide range of problems. These include imaging [12], portfolio management [13], power generation [14] and neural network parameter training [15].

1.1 Research Motivation

The main objective of this research is to provide a deeper insight into dynamic topologies. The intention is that their operation will be understood through a quantifiable description of the a swarms tendency to move from exploration to exploitation. Once quantified, new variations will be developed that will seek to control this parameter in an attempt to improve the overall performance of PSO.

Some of the earliest research into PSO investigated the impact of the swarm topology on performance [16]. This work mainly focused on static topologies which describes those that do not change over the course of the optimization. It was proved that only connecting particles to their neighbours either side lead to slow convergence [16]. A fully connected topology converged much quicker but risked worse performance when particles prematurely converged on a local optima. One of the key issues with static topologies in PSO is that a balance between early exploration and later convergence cannot both be facilitated by a single topology [17]. Dynamic topologies seek to address this limitation by

changing the topology of the swarm over time. During previous research, it has been proven that dynamic topologies produce better results when compared against static neighbourhood topologies [17]. This is as a result of the exploration versus exploitation tendencies of the swarm being governed by the speed of information transfer in the network, which in turn relates to the topology [18]. The aim of this thesis is to describe the transition from exploration to exploitation. Given its relationship to the topological structure of the swarm it seems logical to model the topology as a graph and to apply graph theory to the network. There has been limited research into the application of graph theory to PSO topologies to date. It is hoped that various graph theory parameters and measures can be applied to describe the interconnectivity and communication patterns within the swarm. Using this new methodology, it is envisaged that the underlying operation of dynamic topologies will be highlighted through insights gleaned from analysis of existing variations.
Understanding the underlying operation will allow for comparison across different dynamic topologies in an attempt to understand the attributes most relevant to the functioning of dynamic topologies. Finally, it is proposed that new variations will be created that seek to control these parameters. Based on insights gained through the analysis of different dynamic variations it will be beneficial to adapt existing algorithms to exploit these facts with the intent of improving the performance of PSO.

1.2 Research Questions

There are a number of questions that aim to be addressed by the research completed for this thesis:

1. Can graph theory parameters be applied to dynamic topologies to describe the change in information transfer speed within the network?
2. How does the initial topology chosen affect the early exploration of particles in PSO variations with dynamic topologies?
3. Can graph theory measures be influenced through the deliberate addition of particles to neighbourhoods to affect the performance of PSO optimization?

1.3 Research Methodology

The above research questions will be answered by designing a simulator which will allow for the implementation of standard static and dynamic PSO variants. The performance of the standard variations will be evaluated over a range of functions, which will act as benchmarks for any subsequent PSO variants.
Graph theory parameters will be applied to describe the operation of dynamic topology variants. The insights gleaned from this analysis will lead to proposed changes in existing PSO variants. The proposed PSO will be evaluated across the same set of benchmark functions. The results will be compared for statistically significant differences using appropriate tests.

1.4 Thesis Structure

The structure of this thesis is as follows:

- Chapter 2 will begin by providing an insight into the areas of optimization and computational intelligence. An overview of the terms and concepts within graph theory that are most relevant to PSO will also be covered. A detailed review into the area of Particle Swarm Optimization will follow. This will cover the origin of PSO, a general implementation, the algorithms features and its applications. The chapter will conclude by highlighting gaps in current research.

- Chapter 4 will describe and document the design of the simulator used during this research. This includes rationalization of all design and implementation decisions.

- Chapter 3 will define the standard experimental settings used for the PSO implementation. The suite of benchmark functions used to evaluate the performance of the PSO variants will also be described.

- Chapter 5 will cover the differences between static and dynamic topologies in depth including a comparison of their performance. Graph theory parameters will be applied to the dynamic variants and the results analysed in an attempt to effectively describe the reasons for enhanced performance.
- Chapter 6 will examine how changing the initial topology impacts on the performance of a PSO variation with a dynamic topology.
- Chapter 7 will seek to improve upon the existing dynamic topology PSO algorithms by influencing behaviour based off graph theory parameters.
- Chapter 8 will present conclusions formulated from the results presented during this thesis. It will also detail any potential future work that may arise as a result of this research.

Chapter 2

Literature Review

This chapter outlines the main areas and relevant research conducted to date on Particle Swarm Optimisation (PSO). Firstly a general overview of optimisation will be provided followed by an introduction into the field of Computational Intelligence. A brief overview of graph theory will also be provided due to its applicability in this research. An introduction into the PSO algorithm and its origins will then be provided. The algorithms initialization, boundary handling, topologies, swarm types and particle memory will all be examined. This chapter will conclude by focusing on the applications of PSO to date and identify gaps in existing research.

2.1 Optimisation

Optimisation involves finding a set of variables which provide the best solution to a problem. This is often known as the global optimum. In mathematics this often involves trying to minimise or maximise a given function. Optimisation is widely found in engineering design, business planning and computational problems [19]. It is useful to understand and describe optimisation in the context of the headings listed below.

2.1.1 Heuristic Optimisers

An exhaustive search of a problem space would guarantee that the best possible solution is found given all possible solutions are evaluated. This can easily be achieved for small problems but as the problem space grows large in complexity or dimensionality a solution cannot be found in an appropriate amount of time or within the resources allocated to the problem [20].

Heuristic techniques offer an alternative to exact methods whereby good solutions can be found in a reasonable amount of time but the trade-off is that the solution generated is not guaranteed to be the optimal one [19]. Therefore, heuristic methods offer speed at the expense of accuracy.

2.1.2 Constrained Optimisation

Optimisation is achieved by adjusting a problems input variables to achieve the best possible outcome. Unconstrained problems are those for which the variables can be adjusted without restriction in an attempt to produce the optimum solution.

Constrained problems also attempt to produce an optimum solution by adjusting problem parameters but involve the added task of meeting a specific set of criteria such as enforcing certain value ranges on the problem parameters [15]. Most real world problems have constraints [21]. An example is the Economic Dispatch (ED) problem where the hourly production of energy from power stations must be scheduled for periods usually up to a week in advance [22]. The aim is to optimize for a minimum cost of operating generators while also serving all of the projected customer demands.

Multiple constraints can be involved in a single problem. For the ED example this could involve minimum uptime and downtime for generators or ramping and reserve requirements. An increasing number of constraints applied to a problem corresponds to an associated increase in its complexity [23].

Equality constraints are those for which an exact value must be achieved whereas inequality constraints define an acceptable range of values [24]. Many deterministic methods cannot effectively solve nonlinear cost functions or suffer in higher

dimensions [22]. Population-based, stochastic (involving an element of randomness) search algorithms such as PSO have been effectively used on these types of problems [25][3][22].

There exists a number of different methods used to handle constrained problems. Preservation methods reduce the search space by adding boundaries to ensure generated solutions are feasible at all times [23]. Penalty functions are the most common constraint-handling techniques in evolutionary computing and act by applying a penalty to the fitness of infeasible solutions [26]. Solution feasibility is not guaranteed given a reliance on the penalty method and its application.

2.1.3 Multi-objective Optimisation

Multi-objective problems are those for which there exists multiple functions that are to be simultaneously optimised [27]. An increasing number of objective functions corresponds to an associated increase in problem complexity as objectives are often in conflict. A simple example of a conflicting multi-objective problem would be in attempting to maximise strength and minimize weight during the design of a structure. PSO has been shown to perform well on multi-objective problems [28]. The Environmental/Economic Dispatch Problem (EED) is one such multi-objective problem where minimum power loss is required while also optimizing for cost and low fuel emissions [29][14]. Many methods have been proposed to solve multi-objective load dispatch problems such as EED and heuristic based techniques such as PSO have proved successful as of late [29][14][30].

2.1.4 Pareto-Optimality

Under most circumstances there exists no global optimum solution for multi-objective problems [29]. Trade-offs will be required to satisfy conflicting objectives. This set of compromise solutions is known as Pareto optimal solutions. Pareto optimal solutions are considered to be equally optimal when no solution dominates any other [31]. A solution is considered to be dominant if it provides a better evaluation on all of the objectives when compared to another solution. A solution is considered Pareto optimal if a change to any of the input variables of

the problem (with the intent of improving the outcome of one objective) cannot be made without decreasing the outcome of at least one other objective [32].

2.1.5 Multimodal Optimisation

Optimisation may not always be concerned with finding a single global optimum. Multimodal optimisation functions are those in which there exists multiple optima [33]. Kennedy [34] notes that the modality of a search space relates to the number of peaks it contains. A peak is a solution that is better than all of its neighbouring solutions. Therefore, the highest peak is the global optimum and all others are known as local optima. Many real-world problems such as protein design require multimodal solutions [35]. In protein design it is desirable to find a set of solutions that maximise pairwise diversity. Multimodal optimisation using computationally inexpensive functions is applied to a large set of possible solutions to generate a subset of candidate solutions. The global solution will be contained within this subset. Global optimization is later used on the reduced set of possible solutions using more realistic and thus computationally expensive functions to find the global optimum.

2.1.6 Optimisation involving uncertainty

Optimisation problems can become more difficult if certain elements are unknown. Dynamic problems are those that can randomly change over time, resulting in future optimal values that cannot be accurately predicted [10]. Many real-world problems exhibit this property and examples include portfolio optimisation where the price of stocks can change in the future which leads to the current optimum becoming suboptimal [36] or dynamic pricing where the price of online goods changes rapidly in response to changes in the market or customer demand [37].

Uncertainty may also be introduced through stochastic or random variables. The Vehicle Routing Problem (VRP) is an optimisation problem where vehicles based at a depot are required to fulfill customer demands while minimising the distance travelled and thus the cost of delivery [38]. The VRP can become a stochastic should the problem variables such as the customers, their demands or service

times become characterised by a probability distribution.
One approach to handle uncertainty is through robust optimisation, where solutions are chosen not only based on their fitness but also their robustness towards the uncertainty [36]. This can be achieved by applying a conservative overestimator or worst-case value to the uncertain variable. This approach favours solutions which are intolerant to changes in the unknown parameter but trades this robustness for a solution which may be suboptimal at the current time.

2.2 Computational Intelligence

Computational intelligence is a type of computing that exhibits some form of intelligence, alternatively described as a system that contributes to its own self-maintenance [39]. These systems often attempt to model the intelligence found in the natural world. For example, Evolutionary Algorithms (EA) [40] emulate natural evolution and Swarm Intelligence (SI) [41] algorithms model the grouping and sharing of information within natural swarms such as flocks of birds, ant colonies and schools of fish. This section describes the implementation of these types of systems and some successful applications to date.

2.2.1 Agents

Steels described an agent as a system that performs a particular function or task through relationships with its internal elements and its environment [39]. While often considered to be software agents this definition extends to biological and robotic entities. Agents must have the capabilities to act autonomously. This often requires the ability to sense or evaluate their environment and act upon it, for example an agent wanting to change their own position in a given landscape based on some external stimulus [42]. An agent must have a certain level of self-interest but must maintain itself by behaving appropriately in its environment [39]. It must be proactive in achieving these self interests but have the ability to adapt within a reasonable amount of time to changes in the environment in order to remain viable or relevant. This could be represented as an agent changing

its path or destination following a change in the environment or the agents own internal elements.

Agents also function on a group level in multi-agent systems [42]. Multiple agents must cooperate in a social manner by interacting or communicating with other agents. This is often necessary in order to survive in a particular landscape or to complete a certain task or function.

2.2.2 Evolutionary Algorithms

Evolutionary Algorithms (EA) describes a metaheuristic optimization algorithm with mechanisms inspired by evolution and natural selection that came about as a consolidation of three paradigms [40]: Evolution Strategies, Evolutionary Programming and Genetic Algorithms.

The most common type of EA are Genetic algorithms (GA) [43]. Genetic algorithms, attempt to imitate natural evolution and the principal of survival of the fittest. That is to say that over a period of time entities that are more suitably adapted to their environment or perform better under a certain evaluation criteria are more likely to survive. This results in these entities having a higher likelihood of reproducing and subsequently passing their genes on to the next generation. A GA begins with a population of typically random chromosomes that are encoded with a particular solution to the problem, usually in the form of binary digits [44]. Each of these solutions is evaluated against a particular fitness function to determine its optimality. The better the solution the more likely the chromosome is to be used during reproduction [45]. Genetic operators are then used to create new solutions from the existing population. Crossover is one such operator in which sections of two chromosomes are swapped to create two new chromosomes [45]. This process is iteratively continued until a suitable solution is achieved. Both PSO and Genetic algorithms are iterative, population based approaches to heuristic optimisation. PSO is considered a separate type of EA as it does not utilise any of the traditional genetic operators found in Genetic Algorithms including selection, mutation or crossover [44].

Differential Evolution (DE) is a type of EA that is considered very similar to PSO [25]. Both algorithms are population based, heuristic optimizers that operate iteratively over continuous problem spaces. The main difference is that the agent population of DE changes over the course of the algorithm which is not the case in PSO. As the convergence of both algorithms is relatively quick they can both be applied to complex, real-world problems such as training artificial neural networks [25].

2.2.3 Swarm Intelligence

Swarm intelligence describes a system containing a population of agents with the ability to self-organise [43]. This population is referred to as a swarm. These swarm systems are modeled off systems found in nature such as a flock of birds, bees swarming around a hive, a colony of ants or the immune system. The uniqueness in swarm intelligence is found in the sharing of information between swarm members [3]. The collective intelligence of a swarm can be quite powerful especially if individual members have limited capabilities or knowledge of their environment.

The Artificial Bee Colony (ABC) algorithm is an optimization algorithm that is modelled off the method by which bees find the best food sources [43]. There are three types of bees that each perform a given function. Scout bees begin by exploring with the intent of seeking out good positions. Employed bees return to previously visited good locations and outlook bees choose the best of these positions and remain there.

Ant colony optimisation is a swarm system modelled on the ability of ant colonies to find the shortest path to a food source [46]. The algorithm is modelled on the basis that ants release a pheromone as they traverse a given path. Paths with more pheromone result in a higher probability of another ant following that path. Pheromone levels will be highest just after being placed. This can be useful in guiding other ants to the path of shortest distance. Ant colony optimisation is thus effective in finding the shortest path through graphs [46]. As in both PSO and ABC, swarm members rely on the sharing of information to improve the collective performance of the swarm.

2.3 Graph Theory

Graph theory is a branch of mathematical arrangement problems in which certain objects can be modelled as having relationships with others [47]. Given the swarm of interconnected particles in PSO graph theory can usefully be applied to the analysis of the relationships between swarm members. This section will provide a basic overview of graph theory and detail some of its parameters relevant to PSO.

2.3.1 Graph Structure

A graph G consists of a finite, nonempty set of elements referred to as vertices and a list of element pairs, known as edges [47]. A graph can contain any number of vertices with the term *order* representing the number of vertices in a graph. Vertices can be used to represent elements including people, computer terminals, cities, or in the case of PSO, particles.
Edges describe the connections between the vertices in a graph [48]. The number of edges in a graph is known as its *size*. Edges represent a relationship between connected vertices. Relationship types are application specific but could include friendship, digital network connections or roads.

Graphs can be described in a more succinct manner if they conform to certain restrictions [49]:

1. *Simple:* A simplification to potential graphs where edges connecting a vertex to itself or multiple edges connecting the same pair of vertices are not permitted.

2. *Unweighted:* Edges are not assigned any value or level of priority. Therefore the importance of edges within the network is solely based on their positioning within the graph. An example of a weighted graph could be a communication link in which certain edges have lower transfer speeds and are thus less desirable to be used for information transport purposes.

3. *Connected:* A graph is said to be connected if every vertex can be reached from every other vertex by traversing a path consisting of a finite number of edges in the graph.

4. *Regular:* The number of edges incident with a certain vertex is known as the degree of that vertex [48]. In order to be considered a regular graph it is required that every vertex has the same degree [47] i.e the same number of edges meet at each vertex.

2.3.2 Directivity

Undirected graphs describe those for which there is no distinction between the direction of the edge between two vertices [48]. This implies that the relationship represented by the graphs edges is considered symmetric [49]. In an information network context, this can be considered as a bi-directional communication link. Directed graphs, often called digraphs, contain a similar description of vertices as in the case of unweighted graphs. Contrastingly, edges in a digraph are non-symmetric and hence must be specified by a list of ordered element pairs [50]. The edges in a directed graph are often called *arcs*, with the tail of the arc at the originating vertex and its head at the destination vertex. Considering connections as a uni-directional it can be understood how a connection from one vertex to another does not imply its reciprocal (where it always does in the case of undirected graphs).

Graphs are often presented pictorially but it can be useful to represent them mathematically as well. An *adjacency matrix* is an n x n matrix (where n is the number of vertices) that can be constructed to indicate which vertices are directly connected to each other [49].

The *degree* of a vertex is the number of edges that are incident with that vertex [48]. For a simple, unweighted graph the adjacency matrix can be used to easily calculate this measure. A single degree exists for undirected graphs, but digraphs require separate measures given the directionality of edges [47]. The *out-degree* denotes the number of arcs incident from a vertex, and the *in-degree* is the number of arcs incident to the same vertex. Given the two different degree

measures in digraphs it is further required that the in-degree match the out-degree for the graph to be considered *regular* [50].
A related parameter known as the *neighbourhood* describes the subgraph of the vertices that are adjacent to a particular vertex [48]. For an undirected graph this will include all vertices with edges at the given vertex but for digraphs it will only include the particles at the ends of the arcs incident from the vertex.

2.3.3 Average Path Length

A common and robust measure of a graph is the *average path length* (APL), sometimes called the *characteristic path length*. This is a measure of the average distance from every vertex to every other [49]. The distance between two vertices u and v, denoted by $d(u,v)$, is the shortest path length that must be traversed to reach v from u [48]. A path describes any sequence of edges that are traversed in succession to connect a pair of vertices [47]. It should be noted that the "distance" does not refer to any problem specific euclidean or space related metric, and instead is wholly dependent on the cost of traversing edges between vertices. In the case of an unweighted graph the shortest path length is the minimum number of edges that need to be traversed to reach one vertex from another given every edge has the same weight. However, in the case of digraphs this simplification does not hold. For example, it may "cost" less to reach a vertex across a series of low weight edges rather than a single heavily weighted edge.
The algorithm used during distance calculations depends entirely on the type of graph to which it is applied [50]. For example, if the vertex is part of an unweighted digraph a breadth-first algorithm is appropriate whereas a graph with non-negative weights requires the use of an alternative such as Dijkstra's algorithm. Dijkstra's algorithm operates by initially assuming infinitely positive lengths from the start vertex to all other vertices. It considers all unvisited neighbours and calculates whether the currently assigned value associated with reaching a vertex is greater than if the path was to go through the newly assessed intermediary. If yes, the distance is reduced to this new lower value i.e a shorter path.

The average path length for an entire graph is the average of the individual parameter values over all vertices in the graph [49]. Therefore the average path length of a graph can be described as the mean of the means of the shortest path lengths connecting each vertex to all others.

2.3.4 Clustering Coefficient

Another common parameter used in the analysis of graphs is the *clustering coefficient*. This characterises the extent to which vertices adjacent to a vertex are adjacent to each other [49]. Mathematically, the clustering coefficient for a vertex v is said to be:

$$\gamma_v = \frac{\mid E(\Gamma_v) \mid}{\binom{k_v}{2}} \tag{2.1}$$

where $\mid E(\Gamma_v) \mid$ is the number of edges in the neighbourhood of v and $\binom{k_v}{2}$ is the total number of *possible* edges in Γ_v. The *clustering coefficient* of the entire graph is the average of the clustering coefficients over each of the individual vertices in the graph [49]. This can be used as a measure of the clustering of particles in a graph. In a social networking context this may represent the *cliquishness* of people in a group.

2.4 Particle Swarm Optimisation

Particle Swarm Optimisation was originally suggested by Kennedy et al. [3] in 1995. It was described as having a simple implementation with low time and memory computational cost. The original inspiration for the algorithm arose from an attempt to model the flight of a flock of birds. The social behaviour of humans is also represented in how particles traverse a search space, sharing information in an attempt to converge on the best location.

2.4.1 Background

Earlier work by Heppner et al. [51] attempted to create a computer model of a bird flock. Kennedy and Eberhart [3] used this as a basis to develop PSO in an attempt to determine if agents could use previously visited locations to reliably determine particle velocities when searching new spaces. The intent of searching the landscape was to try to find the optimum solution with respect to some fitness value measured at particle positions. Particles were to improve their position by evaluating personal experiences and also emulating that of better group members.

Kennedy described in [3] the original concept behind a particles velocity being influenced by personal and social contributions. Each particle remembers the best location (referred to as $pBest$) it visited which acts to represent an individuals personal memory of experiences. The contribution of personal preference in updating a particles velocity is that it promotes exploration of the problem space and acts to represent an individuals self interest and free will. The dissemination of the best location (known as $gBest$) among all particles contributes towards social sharing within the swarm to aid convergence of all particles. The obvious analogy is of publicized knowledge or social norm to which particles seek to conform.

Particles are encouraged to explore the problem space during early iterations [3]. The widespread sharing of best locations will see a focus placed on searching in increasingly promising areas to extract the best location as the optimization progresses. The ultimate convergence of particles on a single location is best represented as the social behaviour of humans. If imagined as a flock of birds it is counter intuitive to think of problems beyond a three dimensional space or that multiple birds could occupy a single space at once. Two individuals can be described as occupying the same n-dimensional psychological space if they share the same beliefs and attitudes. The movement of agents through the search space is analogous to bird flight or in higher dimensions can be imagined as a visual representation of the concept of altering psychological perspectives or views.

During the conception of the algorithm the creators noted that particle behaviour

was more analogous to a swarm than a flock [3]. Five basic principles were previously identified by Millonas [52] that must be addressed by swarm intelligence systems:

1. Proximity principle: Time and space computations should be simple and executed by the population.
2. Quality principle: quality factors in the environment should be detected and responded to by the population.
3. Diverse response principle: the population should ensure diversity to avoid commitments on "excessively narrow channels" [3].
4. Stability principle: the operation of the population should not be altered by changes in the environment.
5. Adaptability principle: the population should adapt its behaviour once the environment has changed sufficiently to make it worthwhile computationally.

The authors agreed that PSO met all five principles:

1. n-dimensional space calculations are carried out on each iteration of the algorithm.
2. The velocities of the population are influenced in response to perceived environment quality factors through *pBest* and *gBest*.
3. Diversity is retained as a particles are influenced by their personal memory and through social contribution.
4. The population behaviour only changes when the *gBest* parameter changes.
5. The swarm can adapt when necessary as behaviour changes when a better *gBest* value is found.

2.4.2 Algorithm

The general algorithm originally proposed in [3] operates by initializing a set of particles in a uniform random manner throughout the search space with some initial random velocity. Each particle then calculates the fitness of its position. A particle will update its personal best position if the fitness value is higher than its current personal best. The global best of a neighbourhood is the highest personal best set by any particle in that neighbourhood. A particle then updates its velocity and position using these values. The actual equations suggested by [3] to update the velocity and position of particles were to be applied in every dimension on each iteration:

$$v_{t+1} = v_t + r_1 c_1 (pBest_t - x_t) + r_2 c_2 (gBest_t - x_t) \tag{2.2}$$

$$x_{t+1} = x_t + v_t \tag{2.3}$$

Where v_t is the particles velocity at time t, x_t is the particles position in the search space at time t, r_1 and r_2 are random variables between 0 and 1, c_1 and c_2 are acceleration constants, $pBest_t$ is the best solution found by the particle to date and $gBest_t$ is the best position found by the neighbours of a particle.

The values of the acceleration constants and random r variables have the effect of varying the position of a particle around the best position [3]. This element of randomness allows particles to search a localised promising area for potentially better solutions. The acceleration constants originally had the value 2 in [3] to ensure that agents would overshoot the target half of the time but Eberhart updated the equation in [16] to be specifiable parameters. [2] later suggested that c_1 and c_2 should remain equal regardless of their values to ensure a balance between personal and social influence over a particles proposed velocity.
It is also explained in [2] how the addition of V_{max} is used to restrict v_{t+1} velocities becoming increasing rapid and thus generating positions that are too widely spread apart. Furthermore, this prevents the potential effect of entering a state of *explosion* where particles leave the search space. A generalisable value for

V_{max} did not exist for all problem spaces as large spaces require large values to ensure adequate exploration and smaller spaces require small values to prevent explosion [2]. An *inertia weight* parameter w was introduced by Eberhart et al. a few years after the initial PSO publication [53]. This saw a change in the PSO motion equations:

$$v_{t+1} = wv_t + r_1c_1(pBest_t - x_t) + r_2c_2(gBest_t - x_t) \tag{2.4}$$

$$x_{t+1} = x_t + v_t \tag{2.5}$$

A dynamic value for w could be varied from 1.0 to less than 1.0 over successive iterations to promote exploration early on and focus search efforts to a promising area at a later stage. Controlling the *inertia* of particles meant that there was no longer a requirement for velocity limiting.

In 2002 Clerc et al. [6] introduced another method of managing explosion and stability known as *constriction*. The constriction factor χ is derived from existing variables in the velocity update equation and is defined as:

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \tag{2.6}$$

$$\varphi = c_1 + c_2 \tag{2.7}$$

It was found that for values of $\varphi < 4$ the population would slowly spiral with no guarantee of convergence. A value of $\varphi > 4$ and the swarm would guarantee convergence at a quicker rate but there existed the potential of converging upon a local optima. Clerc suggested a value of $\varphi = 4.1$ as this avoided explosion and guaranteed convergence. While it is possible to adjust the values of c_1 and c_2 it was recommended to have values of $c_1 = c_2 = 2.05$ to balance personal and social influence of a particles velocity. Using these values the constriction factor $\chi \approx 0.72984$ [2]. The updated velocity equation is defined as:

$$v_{t+1} = \chi(v_t + r_1 c_1(pBest_t - x_t) + r_2 c_2(gBest_t - x_t)) \tag{2.8}$$

This velocity update equation is considered favourable over the inertia weight method [2]. Explosion of particles is avoided and convergence is aided without the need for a V_{max} term.

2.5 Initialization

2.5.1 Position

Assuming no known knowledge of the search space in advance of performing PSO, the obvious approach to particle positioning is to place particles within the bounds of the feasible search space with a random distribution [2]. This ensures a high degree of diversity among particles which promotes exploration in the early stages of the algorithms execution.

2.5.2 Swarm Size

The swarm size dictates how many particles should be initialized into the problem space. Too small a number of particles and there is a risk of the swarm becoming trapped in a local optima but excessive amounts of particles may lead to unnecessarily high computation times [54]. The work of Shi et al. [5] investigated the use of swarm sizes between 20 and 160 particles and found that all sizes have similar performance levels. Further research by Li-ping [55] in 2005 found that the performance is insensitive to population sizes larger than 50 particles in problems of higher dimensions while populations of between 20-50 is appropriate for lower dimensional problems. It is shown in [2] that the optimal swarm size depends on a given problem but that all swarm sizes perform acceptably on standard benchmarks. Supporting [55], it is said that there exists no superiority or inferiority in terms of results produced when using swarm sizes between 20-100 particles.

2.5.3 Velocity

Much research and debate has gone into the the initialization of particles. The work of Engelbrecht [4] describes the three primary methods used to initialize velocities used in current literature.

The simplest means of initializing particles is to set the initial velocity of all particles to 0. The main argument against this method is that it limits early exploration of the problem space. Alternatively, particles can be initialized with a random velocity value between 0 and some defined limit. While this method increases initial diversity the consequence of particles moving with larger step sizes is that it causes a greater number of particles to leave the search space [4]. Particles that violate the problem space boundaries are termed *roaming particles*. A third method attempts to increase diversity while simultaneously avoiding boundary violations by initializing particles with small random velocity values. Appropriate values will depend on the problem being optimized, and more particularly, the size of the problem space which makes calibration for general use difficult.

The main aim of Engelbrecht's work was to show that the initialization of velocities with uniform random values was not the preferable option [4]. While it was found that the initial swarm diversity was increased, the convergence time suffered as a result of an increase in the number of particles violating boundaries and hence wasting effort searching outside the bounds of the problem.

Helwig et al. [7] also proved theoretically that uniform velocity initialization causes not only the best but all particles to leave the search space. This prompted Helwig to investigate an alternative initialization method known as the *Half-diff* method:

$$v_{i,0} = \frac{1}{2}(y_i - x_{i,0}) \tag{2.9}$$

Where $v_{i,0}$ is the initial velocity in dimension i, $x_{i,0}$ is the initial position in the same dimension and y_i is a randomly chosen value between the upper and lower boundaries of the problem space.

Given zero initialization stifled initial exploration and uniform initialization caused

all particles to leave the search space, Helwig recommended the *Half-diff* method but noted that no strategy managed to stop particles leaving the search space.

2.6 Bound Handling

The are a number of motivations for applyting boundary constraints to a problem. Boundaries can act to simplify problems as the number of local optima within a region can be reduced. Infeasible solutions that exist outside the boundary can be ignored which is particularly beneficial if there is a high computation cost associated with evaluating the fitness. This section will examine the effects of high dimensions on boundaries as well as the different methods of handling particles which go beyond a boundary in a given dimension.

2.6.1 Curse of Dimensionality

The Curse of Dimensionality is a general reference coined by Bellman [56] to describe the unintuitive phenomena that arise in problems and applications in high dimensions. When analysing the performance of PSO in high dimensional problem spaces Hendtlass [57] found that performance can decrease as the number of dimensions in a problem space increases. He noted that PSO exhibits dimensionally separate behaviour, while many of the problems it is used on are not dimensionally separate. He stated that while a swarm may be converging (or have converged) in one dimension it may still be exploring another. For non-separable problems, the effect on the fitness from a change in one dimension depends not only on the magnitude of this change but also on the position in other dimensions. The fact that all dimensions do not converge simultaneously means that the effective problem space is restricted [57].

Helwig et al. [7] proved theoretically that all particles in a high-dimensional search space are initialized very close to at least one border with an extremely high probability. This is understandable given most of the search space in high dimensional problems is near at least one border. As a result of this it is also very likely that particles will leave the search space on the first iteration. Engelbrecht

later confirmed experimentally that particles often violate the bounds of high-dimensional problems and thus bound handling directly affects the quality of the algorithms output [58]. If no boundary conditions are applied he found that, as a result of these boundary violations, a large proportion of the time the global best found was outside of the boundary. This had the effect of attracting further particles outside of the boundary through the social contribution of the global position to a particles velocity. This can lead to wasted effort while searching or an optimum solution being found in infeasible space. Hence, bound handling or lack thereof strongly impacts swarm behavior and should be considered appropriately.

2.6.2 Boundary conditions

The constriction factor introduced in [6] acts to limit the maximum velocity of particles but cannot prevent them for exiting the bounds of the feasible solution space. Hence, infeasible solutions can be generated if particles are permitted to leave the problem space.

In 2007 Xu et al. [1] researched six different boundary conditions to evaluate their performance. These boundary conditions define what happens when a particle flies outside the feasible solution space in any one of the dimensions. The first four conditions are comprised of applying basic operations:

1. Absorbing: the particle is relocated at the boundary in that dimension, with the velocity component of that dimension being zeroed.

2. Reflecting: the particle is relocated at the boundary in that dimension, with the sign of the velocity component in that dimension changed.

3. Damping: the particle is relocated at the boundary of that dimension with the velocity component sign being changed with a random factor between 0 and 1.

4. Invisible: the particle remains outside the boundary but the fitness evaluation is skipped and a bad fitness is assigned.

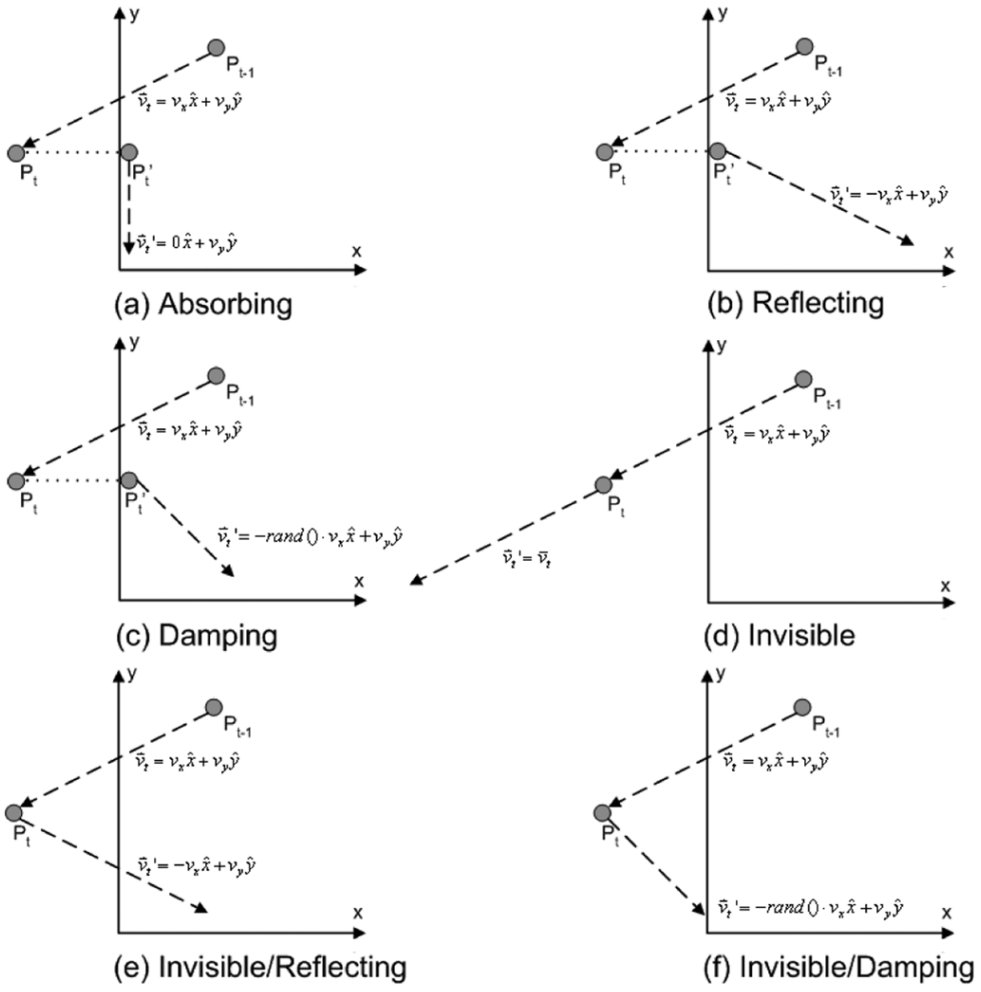


FIGURE 2.1: Boundary Conditions [1]

Xu categorized the boundary conditions into two groups. Restricted boundary conditions including absorbing, reflecting and damping, are those which relocate the roaming particle inside the feasible search space. Unrestricted boundary conditions such as invisible, simply skip fitness evaluations to prevent the generation of invalid solutions. Given no fitness values are assigned, a particles personal and global best counteract its current momentum and eventually pull the particle back into the search space.

Xu also introduced two hybrid boundary conditions that attempted to add the favourable characteristics of the reflecting and damping boundary conditions to the invisible boundary condition. The Invisible/Reflecting condition allows particles to stay outside the boundary but the sign of the velocity in that direction is changed to accelerate it back towards the solution space. Invisible/Damping

uses a similar method but the reversed velocity is multiplied by a random factor between 0 and 1 to dampen the acceleration of a particle.

Xu states that the performance of different boundary conditions depends greatly on the fitness function, the dimensionality and the location of the global optimum. These conditions are unique to each problem and thus concludes, in line with [59], that there is no single "best" boundary handling strategy and that its choice depends on the problem specific application. Xu went further by stating that the damping boundary condition was advantageous for safe performance while the invisible boundary had potential for efficient performance.

2.7 Topology

PSO is unique due to the dynamic interactions between particles [3]. A single particle or group of unconnected particles are incapable of solving any problem without the use of an exhaustive search. It is therefore implied that value arises from the interconnections and sharing of information among particles. A topology defines the way in which particles in the population are connected and a neighbourhood describes the particles within a swarm to which a given particle can communicate [16]. The chosen topology therefore plays an important role in the success of the algorithm as it defines the means of information sharing between particles [60].

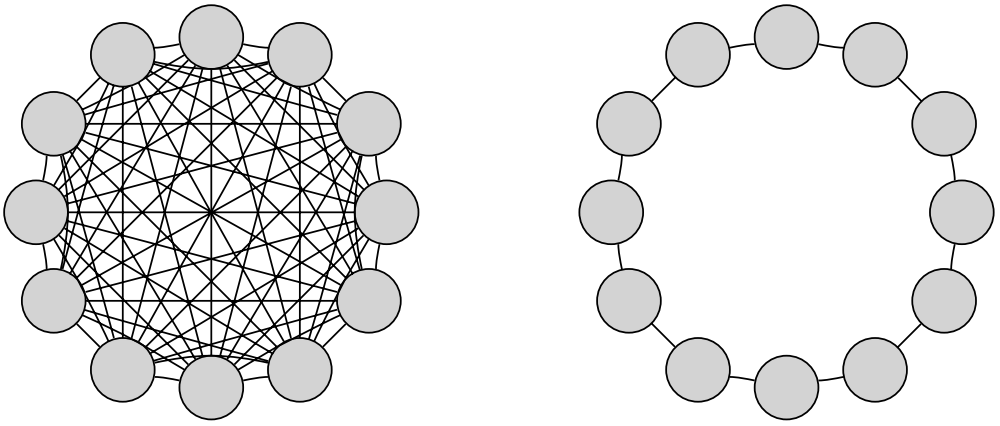
2.7.1 Static Topologies

Shortly after the initial publications on PSO, Eberhart et al. [16] reviewed the relationships between particles in a swarm. They compared the performance of PSO using two different topologies. The *gBest* topology was used in the original implementation of PSO [3]. In this topology every particle has a connection to every other. Individual particles are thus attracted to the best solution found by any member of the entire population through the social influence parameter in the velocity update equation. The sharing of the same global best position among all particles in the swarm leads it to converge quickly on positions found during early iterations.

Social simulations, where entities only communicate with a small portion of a population, was the motivation for introducing topology variants [16]. By reducing the number of connections in the swarm topology the social contribution to a particles velocity can only come from a subset of the entire population. This subset is referred to as the particles neighbourhood. The ring topology, often called *lBest*, is the often considered the simplest local topology and has a neighbourhood size of two [2]. It creates the slowest communication path between particles [9]. Each particle k is only directly connected to its immediately adjacent neighbours, *k+1* and *k-1*. Likewise *k+1* and *k-1* have particle k as one of their neighbours.
It was concluded in [16] that the *gBest* variation converged quicker but that *lBest* was more robust and resilient to becoming trapped in local optima.
The later work of Bratton and Kennedy [2] favoured the use of the *lBest* topology. They agreed that faster convergence does not directly correlate to better performance and that the slower convergence speed is due to an increased amount of diversity and exploration within the population at early stages of the algorithm. This aids in avoiding convergence on local optima, particularly in multimodal problems.

Kennedy and Mendes [9] too realised that the topology greatly influenced the performance of the algorithm. They therefore systematically investigated a number of topologies in an attempt to define the best structures. Their experiments included:

1. *gBest*: the entire population was treated as a particles neighbourhood.
2. *lBest*: adjacent particles were treated as an individuals neighbourhood.
3. Pyrimid: a three-dimensional wire-frame triangle.
4. Star: one central particle influences and is influenced by the rest of the population.
5. Small: particles were arranged into small cliques and isolates.
6. von Neumann: particles above, below and on either side of a two-dimensional lattice were connected.



(a) The gbest topology

(b) The lbest ring topology

FIGURE 2.2: Static Global Ring Topologies [2]

They found that the topologies with a greater number of connections tended to converge rapidly. This is in line with earlier findings of Eberhart which claimed that the medium number of iterations to convergence for the *gBest* topology is less than that of the *lBest* [16]. This can be conceptualized as information flowing quickest between connected particles, but that is buffered when passing through intermediaries [9]. If there exists a short communication path between particles then the population tends to favour solutions found in early iterations and rapidly converge towards them. This behaviour is beneficial if the population was initialized in a good region as it was likely that the global optimum will be found quicker. However, if particles are initialized badly there is a high likelihood that the swarm will converge on a local optima. This is particularly prevalent in landscapes of complex functions as particles may fail to explore outside "locally optimal regions" of the problem space [9]. Conversely, inhibiting communication between particles through a reduced number of connections results in the inefficient spread of information among the swarm. This is beneficial at first to promote early exploration of the problem space but becomes undesirable as optimization progresses as particles waste time aimlessly exploring suboptimal regions of the problem space when more promising regions have been found by others.

Kennedy and Mendes favoured the von Neumann topology as it best balanced exploration and convergence [9]. It favoured well in meeting the criteria within a reasonable number of iterations. It was not consistently the best performing

topology but achieved the best overall performance over the range of problems.

In 2009 Muoz-Zavala et al. [61] wished to improve upon the ring topology. The mutual attraction between neighbouring particles in the canonical ring topology can be represented as a doubly-linked list. A singly-linked ring (SLR) topology structure was introduced where particle k has particles *k+1* and *k-2* as neighbours. Likewise particle *k-1* has *k-3* and k as neighbours. This has the effect that k attracts *k-1*, but *k-1* can only attract k through particle *k+1*. An intermediary particle is cancelling the mutual attraction between particles. This further reduces the transfer of information between particles during early exploration while simultaneously increasing exploitation at later stages. The new singly-linked structure was statistically proved to be better than both the ring topology and von Neumann structures over the majority of functions tested. These results are not conclusive and require further experimentation.

From the above analysis of all static topologies it can clearly be seen that the number of connections in the swarm largely dictates the speed of information transfer [9]. This explains the relationship between the number of connections and the tendencies of the swarm to explore or exploit the problem space. The swarms performance is thus largely affected by its topological structure in PSO [62].

2.7.2 Dynamic Topologies

Dynamic topologies are an attempt to achieve a better balance between exploration and exploitation by changing the swarm topology over the course of optimization, usually once every iteration [17]. This is in contrast with static topologies which do not change as the algorithm executes.
It is known that the *gBest* topology promotes high levels of information flow as every particle shares information with every other particle, but the resultant quick convergence introduces greatly increased risk of particles becoming trapped in a local optima [16][9]. The *lBest* topology only connects a particle to its two adjacent neighbours which reduces information flow and thus the risk of premature convergence on local optima. The trade-off is that the convergence speed is slower than in the case of *gBest*.

Liu et al. [17] was prompted by a recognition of the need for early exploration of the problem space and quicker convergence during the exploitation phase to develop the Gradually Increasing Directed Neighbourhood PSO (GIDN PSO). The addition of particles in the swarms neighbourhoods over the course of the optimization saw the GIDN PSO algorithm meet the requirements for exploration in the early phases of optimization and exploitation of promising areas at a later stage. The algorithm emulated the positive performance of both *gBest* and *lBest* topology variations to produce a productive balance between early exploration and later exploitation. It was statistically proven that this resulted in an improved performance over the static *gBest* and *lBest* topologies.

2.7.3 Small World

In 1999 Kennedy [60] investigated the application of social network structures to PSO topologies. In social networks it is common to have clusters of strongly connected individuals, some of which possess "weak ties" with individuals in other clusters. These links serve as a connection between two individuals in separate clusters who are not directly linked. The ability to transport information through distant acquaintances is vitally important in social networks as it is a means of sharing information that otherwise would be unknown by a clique or cluster.

Kennedy's research attempted to add these "weak ties" or "small-world" shortcuts to various topologies in particle swarm optimisation. He assumed that these links would reduce the average distance between particles while maintaining a high level of clustering. An example of this can be applied to the commonly used *ring topology* where for a small percentage of particles their second link would not be connected to their nearest neighbour but rather a distant acquaintance. By reducing the distance between neighbours through shortcuts it was expected that the swarm would converge faster. The results of experimentation were inconclusive in terms of improving performance. "Weak ties" introduced to different topologies had different effects depending on the function being optimised. Kennedy could only conclude that the sociometry of particles greatly impacts performance but that the optimal topology depends on the problem being solved.

2.8 Standard PSO

In 2007 Bratton and Kennedy [2] introduced the Standard PSO (SPSO). The intention for SPSO is to act as a representative baseline of the PSO algorithm by taking advantage of the generally applicable improvements in the ten years since its original publication. The standard algorithm was characterised by:

1. a local ring topology,
2. the constricted update rule in Equation 2.8 for velocity updates,
3. a swarm of 50 particles,
4. non-uniform particle initialization,
5. an invisible boundary condition.

It was noted that the defined SPSO is not optimally configured for all problem sets. There are many different configurations under which PSO may perform preferably for a given problem. SPSO is intended to serve as a default for algorithm comparison and a standard representation of a modern PSO which can be used in research or as a starting point for certain applications or problems.

2.9 Fully Informed Particle Swarm

The work of Mendes et al. [62] introduced the idea of informing a particles velocity using information from all particles in their neighbourhood, not simply the best performer. This resulted in the creation of Fully Informed Particle Swarm (FIPS) variant of PSO. As a particles velocity is influenced by all members of its neighbourhood and not just the best performer, the topological structure has a critical impact on the performance of the algorithm. A number of different variations of FIPS were tested and each outperformed the canonical PSO with the wFIPS variation (where a weighting was applied to a neighbours contribution based on the fitness of their personal best) performing best overall. Questions still remain surrounding the effect of weighting on performance but it was suggested that the extra computational cost of weighting may not be justifiable

if unweighted versions of FIPS performed well enough. Experimentation also proved that FIPS performed best when every particle had a small number of neighbours rather than the *gBest* topology. Research conducted by Montes de Oca et al. [8] attempted to understand whether the performance decrease when the *gBest* topology was implemented with the FIPS algorithm was as a result of random behaviour being induced in the swarm. It was questioned whether the simultaneous influence of all particles when updating the velocity of a particle could be causing particles to move randomly. It was discovered that this was not the case but that the increased connectivity was causing the particles to spatially converge too rapidly, as was seen in [9] and [16].

2.10 Memory

The social contribution of the best known position within a particles neighbourhood promotes the convergence of particles [3]. This is balanced by a particles memory of their previous personal best position to encourage exploration of the problem space. Therefore a particles personal memory is essential to the cognitive role of defining its velocity and to ensure diversity within the swarm.

2.10.1 Enhanced Particle Memory

Research conducted by Broderick et al. [63] aimed to enhance the cognitive influence on a particles velocity by remembering multiple previous successful positions (PSO EMP). Their research aimed to show that additional positions should improve swarm diversity and improve the performance over the standard PSO. They found that retaining the two best positions yielded the appropriate amount of additional exploration without hindering convergence. A Roulette Wheel selection approach where positions are weighted with a higher probability based on their fitness was best as it performed like the standard PSO in most iteration which allowed for minimal variance. It was found that the new PSO EMP had a significant performance improvement over the standard PSO due to the increased diversity of particles.

2.10.2 Avoidance Strategies

Yang et al. [64] were the first to introduce the idea of using personal worst known locations and worst global locations to inform a particles updated velocity. Although the new particle swarm optimization method (NPSO) did not improve upon the canonical PSO algorithm it nonetheless introduced the idea of area avoidance rather than attraction. Jayabarathi et al. [65] combined the NPSO and canonical PSO algorithms to influence a particles velocity based on its personal best and worst positions as well as that in the particles neighbourhood:

$$\begin{aligned} v_{t+1} = wv_t + r_1c_1(pBest_t - x_t) + r_2c_2(gBest_t - x_t) \\ + r_3c_3(x_t - pWorst_t) + r_4c_4(x_t - gWorst_t) \end{aligned} \tag{2.10}$$

Analysing the associated velocity equations it is evident that the least amount of extra velocity is added when the worst location is at a maximum. One would expect the particle to move faster in known bad areas to reduce wasteful time exploring. Mason et al. [11] recognised this flaw and aimed to correct it in a new PSO variant with Avoidance of Worst Locations (PSO AWL). The research showed significant improvements over the standard PSO and vast improvements on any previous avoidance strategy variations of PSO. PSO AWL performed particularly well on functions where the optimum was located near the boundary given particles had less velocity as they moved away from the worst location which better enabled convergence.

2.10.3 Non-Revisiting

In 2008 Chow et al. [66] introduced a Non-revisiting PSO (NrPSO) which ensures that a particles new position has not previously been visited or evaluated. This approach eradicates the cost of recomputing previously visited positions which is particularly beneficial if the fitness function evaluation is time or memory expensive. It also aids in avoiding premature convergence of particles. A binary search tree is used to store all visited locations. When a particles updated position has already been visited a velocity is applied to the particle so that it

escapes that region. The results proved that NrPSO outperformed four standard PSO variants in both uni-modal and multimodal problems up to 40 dimensions. The insignificance of overhead and archive size are also said to make it suitable for real world problems.

2.11 Multi-swarm PSO

Arising from the need to adapt PSO to achieve optimal results on dynamic problems, Blackwell et al. [10] developed the Multi-swarm PSO (MPSO). MPSO is an adaptation of a previous variation on PSO developed in [67]. Particles within traditional swarms converge on a given subsection of the problem space and begin exploitation after a period of exploration [17]. This can have a number of consequences in dynamic problems depending on the movement of the optimum solution. Should the optimum move within the subsection particles are exploiting, the swarm will be able to efficiently track the movement and converge upon it. However, if the optimum shifts significantly far away from the converging swarm their reduced velocities in the exploitation phase will prevent adequate tracking and give rise to linear collapse [67]. For this reason Blackwell developed the Charged PSO algorithm (CPSO) in [67]. A number of charged, mutually repelling particles are added to a swarm. On commencement of convergence the swarm can be imagined as a nucleus while the charged particles act as an orbit around the nucleus. Forcing the charged particles to search the area outside the optima ensures that diversity is retained in the population and thus the changing function can quickly be recognised and adapted to by the swarm.

In adapting the CPSO to be used with multiple swarms Blackwell recognised and found solutions to a number of difficulties. As neutral particles do not repel, multiple swarms could converge on a single peak. To remedy this Blackwell added an 'exclusion radius'. Should one swarm enter a predefined radius of another the swarm with the highest global fitness continues exploitation while the other swarm is randomly regenerated. There is a high computational cost involved in computing the repulsion of charged particles to the order of $O(N^2)$. Blackwell correctly presumed that the success of CPSO over PSO was due to the increased diversity of the orbiting particles rather than particles finding better

solutions within the nucleus. Therefore he replaced the computationally expensive repelling orbit with a 'quantum cloud'. In this model called Quantum Swarm Optimization (QSO) charged particles are randomized in a ball some set radius away from the swarms global optimum.
Blackwell found improved performance over PSO when MPSO or QSO was employed. QSO was considered to be preferable as it is scalable due to a linear complexity.

2.12 Applications

The "No Free Lunch" theorem (NFL) states that there is no one optimization algorithm that performs best on every function possible [68]. It is suggested that deceptive and random functions are those on which optimisation algorithms often fail as there is no gradient or structural indication of where the solution will be found. Mendes et al. [62] suggest that there exists a third class of functions for which regularities in the fitness landscape offer clues as to the location of a solution. Heuristic optimisers often exploit predictability in landscapes making them particularly good candidates for finding global optimums in these types of structured problems. In spite of their complexity, real world problems often have structure to them. This has lead to the successful application of PSO to many real world problems, some of which are detailed below.

2.12.1 Imaging

In 2014 Lahmiri et al. [69] applied PSO to search for optimal threshold values to split an image into distinct regions using image thresholding when performing image segmentation in image processing or computer vision applications. Their research confirmed that all PSO variants achieved improved performance over traditional techniques. This result was achieved as the PSO variants formulated a multidimensional optimization problem to find the optimal thresholds.

PSO has also proved highly successful when used for face detection [12]. Traditional approaches require exhaustive search. This is computationally expensive

as faces appear in four dimensions: size, orientation and position (both vertically and horizontally). The research of Marami et al. [12] found that using PSO significantly improved classification accuracy under *real world* conditions over a widely used computer vision software library. It also reduced the time needed for detection by a factor of 200 as it only examines approximately 0.4% of all possible image positions.

2.12.2 Neural Networks

Artificial Neural Networks (ANN) are learning algorithms modelled on the human brain [70]. They consist of a number of layers of interconnected neurons whose output depends on a specific weight assigned to the neuron and on inputs to the neuron, be they external or from other neurons. Training an ANN (by adjusting the weights of its neurons) is known to be a nonlinear, multidimensional, dynamic problem to which Engelbrecht et al. [3] applied PSO in their original publication on the algorithm. Experimentation found that PSO could be used to train ANN weights at least as effectively as the commonly used backpropagation algorithm. Informally they found that PSO occasionally produced an ANN with superior performance on a test set over the equivalent generated using backpropagation.

2.12.3 Portfolio Management

In 1952 Harry Markowitz [71] introduced Modern Portfolio Theory (MPT) which involves the allocation and management of financial funds to a number of assets within a portfolio. Portfolio Optimisation (PO) describes the assumed behaviour that an investor wants to find the best possible combination of these assets based on their predicted future performances. Since the future performance of a given asset is uncertain it is necessary for investors to diversity their choice of assets in order to reduce risks.
Reid et al. [13] applied PSO to a multi-constrained version of the PO problem, adding real world factors such as taxation, transactional costs and market impacts. They found that PSO was suitable for finding optimal portfolios with significant long term improvements.

Wang et al. [72] achieved similarly good results by applying PSO to the PO problem. By gradually decreasing the number of dimensions, which corresponded to a gradual decrease in the number of selected assets, a resulting reduction in problem complexity was achieved. This proved successful in solving larger-scale portfolio optimization on which traditional methods failed.

2.12.4 Economic Emissions Dispatch

The Economic Emissions Dispatch (EED) is an optimization problem where the main objectives are to allocate the optimal power generation from different units with the lowest emissions and cost possible while meeting all system constraints [29]. It is known to be a highly nonlinear constrained multiobjective optimization problem with conflicting objectives [29]. A dynamic version of the problem has also been proposed in [73] which considers ramp rate limits of the generating units. It is shown in [29] that PSO can be applied to the EED problem to generate the Pareto optimal solution set. The results proved the effectiveness and robustness of PSO in finding optimal or near optimal solutions to the EED problem.

Abido [14] showed similarly promising results of PSO on the EED problem, emphasising the need for a single simulation run to generate multiple Pareto-optimal solutions. He noted that redefining PSO to operate on multiobjective problems was nontrivial as a single best local or global solution does not exist, but rather a set of non-dominated solutions must be found. Many real world problems are known to be multiobjective and thus PSO appears to perform well on these types of problems.

2.13 Gaps in Research

This chapter provided an overview of the main research into PSO to date. During this analysis it became apparent how the dynamic interactions between swarm members produces unique operation in the PSO algorithm [9]. It seems that the swarm topology is the best means of influencing these interactions [62].

There has been a limited amount of research carried out into the use of dynamic

topologies, which have been shown to improve the balance between exploration and exploitation within a swarm during optimization [17]. This thesis will investigate the applicability of graph theory to indicate the topological changes during optimization with the intention of finding novel and insightful means of describing the operation of PSO beyond simple convergence graphs. The focus when applying graph theory will be in finding an appropriate means of describing the speed of information transfer with the swarm and the effect the topology have on this parameter [9].

This thesis also aims to investigate the impact of the initial topology on the performance of the swarm. Much of the research into topology structure before this has focused on static topologies with little emphasis on the initial structure of dynamic topologies. It has already been shown that a reduction in the speed of information transfer within a network improved the performance of PSO [61]. The goal will be to identify the initial topologies that produce the best performance and to discuss the resulting speed of information transfer.

Finally, using the insights into the operation of PSO variations with dynamic topologies, it will be investigate whether the performance of PSO can be positively influenced by controlling the information transfer speed appropriately during optimization.

Chapter 3

Simulator Design

This chapter will detail the design of the simulator built to implement PSO variations and assess performance. A description of the simulator requirements, pseudocode algorithm and class diagrams are provided. The simulator was written in Java. The object-orientated nature and authors familiarity with the Java language were the main reasons for its use.

3.1 Design Requirements

The basic operation of the simulator requires that PSO variants can be applied to a number of different search spaces to optimize certain functions. In this respect the general requirements are:

- A defined number of particles can be created.
- Particles are initialized to a search space with a position and velocity.
- The fitness of particle positions can be evaluated with respect to a function.
- Information exchange between particles facilitates updating of particles velocities and hence positions.
- The swarm converges on a solution over a given number of iterations

To accurately test the operation of PSO variations the simulator compares performance across 32 benchmark functions. Due to a generic structure, benchmark functions can be implemented using inheritance. A loop can be used to cycle through individual functions and polymorphically reference them to automate the running of optimization on multiple functions using the same PSO variation.

For similar reasons to functions, PSO variations should be modelled as children of an abstract parent class. By providing a generic template, future variants can easily be developed thanks to extensibility.

The simulator needs to record the best result for each variant and function combination and output the results to a file. The random initialization and stochastic nature of PSO can heavily influence the performance of PSO for a given variant and function on a single run. Therefore results need to be computed over multiple runs with the mean and standard deviation of the multiple runs also being stored in the file.

In order to understand a PSO variant in more detail it is common to plot convergence graphs. Convergence graphs are a plot of the best fitness versus the iteration number. They are most useful in comparing the convergence characteristics of different variants over the same functions. The actual plots for each run are stored as an image file and generalized plots are created by averaging the convergence values for every iteration over each of the multiple runs. The raw data values associated with each of the points plotted are also written to a file. Plots for the graph parameters average path length and number of disconnected paths are included for variants with dynamic topologies. These were constructed in the same manner used to produce convergence graphs.

3.2 Default Parameters

A number of consistent parameters were implemented and used during simulations.
Particles are randomly initialized within the bounds of the problem space and their initial velocity is set using the half-diff method [7].

The simulator uses an invisible boundary [1]. This means that no action is applied to a particle when it exits the search space other than to ignore any fitness values produced outside the boundaries of the problem space. Given positions outside the search space are not evaluated the particle will eventually be dragged back into the search space through the contribution of its personal and global best positions on its velocity.

3.3 Algorithm

Algorithm 1 and Figure 3.1 outline the basic operation of a general PSO implementation originally suggested in [3]. A set number of N particles are randomly initialized within the search space and are assigned certain velocities. For each iteration the velocity and position of every particle is updated based off knowledge of good positions. The fitness of a particles new position is evaluated after each change in position. If a particles current fitness is better than its personal best or the global best the respective values are updated to reflect this fact. This process repeats for a defined number of iterations (i_{max}). The *gBest* position is

output as the best solution found once the maximum iteration has been reached.

Algorithm 1: Standard PSO Algorithm

Step 1: Initialize N particles with a position and velocity

Step 2: For each iteration:

while $i < i_{max}$ **do**
    for *particle* p=i ***to*** N **do**
        Update velocity of particle p
        Update position of particle p
        Calculate *fitness* of particle p
        if *fitness* $<$*pBest* **then**
            $pBest = fitness$
            if *fitness* $<$*gBest* **then**
                $gBest = fitness$
            end
        end
    end
end

Step 3: Output *gBest*

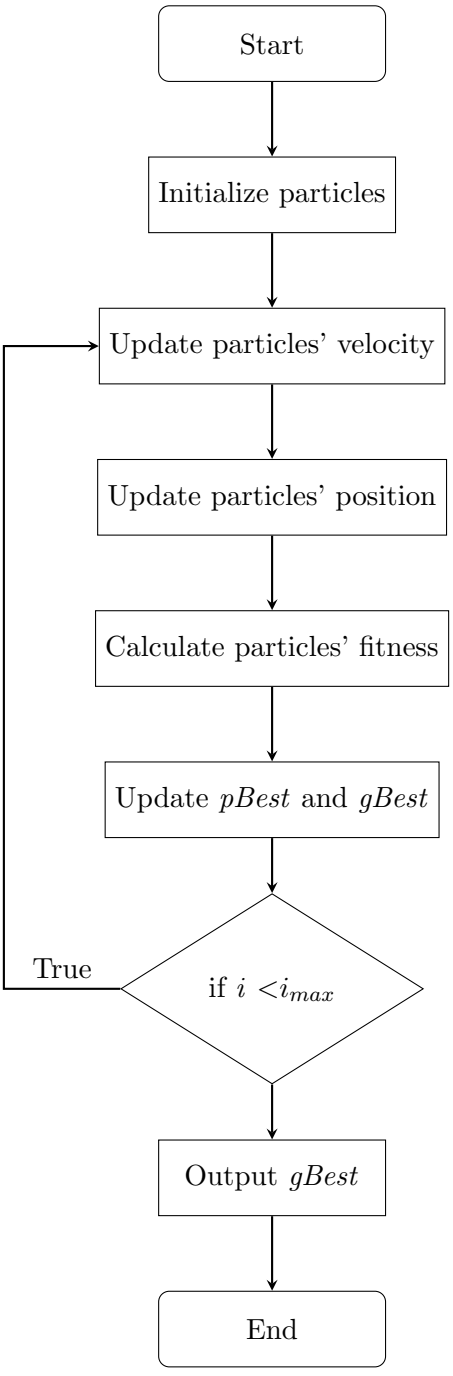


FIGURE 3.1: PSO Algorithm flowchart

An overview of the class structure and relationships within the Java simulator are shown in Figure 3.2 with a more detailed description of the methods contained in each class provided in Figure 3.3.

The abstract PSO class is used to define the basic structure of the algorithms operation. Subclasses of this class are created to implement different PSO variations. Each subclass will override the *run* method to reflect the exact implementation of a variation. This level of extensibility makes it easy to design and add new variations. The PSO class initializes a number of Particle objects (instances of the Particle class). Each PSO object has a single Function object (an instance of the Function class). Each of the 32 benchmark functions are extensions of the abstract Function class with custom implementations of the *evaluate* method to evaluate the fitness of a given position with respect to the function. The equations to mathematically describe function evaluation are explained in Chapter 3. Function objects also define the dimensions of the problem space and boundaries within which solutions must be contained.

The Particle class stores all information pertaining to an individual particle including its position, velocity, the best position found by the particle and the fitness at that position. It also contains a method to determine whether the particle is within the bounds for a given function.
Each particle contains a neighbourhood object (an instance of the Neighbourhood class). This defines the list of other particles in its neighbourhood. It also stores and updates the best position and fitness of the neighbourhood which is used as the social contribution when updating that particles velocity. A particles neighbourhood can easily be increased using the *addToNeighbourhood* method. Adding certain particles to the neighbourhood during the operation of the algorithm is necessary in dynamic topologies, which are covered in Chapter 5. The GIDN PSO is an example of a PSO variation with a dynamic topology [17]. It would be implemented as a subclass of PSO that contains a new method to increase particle neighbourhoods whenever necessary.

The Main class is used to instantiate a subclass implementation of the abstract PSO class. The PSO class initializes a swarm of Particle objects to perform the optimization. The PSO class is provided with a Function object which defines the

problem to be evaluated. The global best position and fitness found by the swarm during optimization is stored in the PSO class and returned on completion of the algorithm. The Main class specifies the function, dimensions and number of iterations on which the algorithm should operate. It calculates a set of results for each algorithm on every function over multiple runs. These results are outputted to the console and stored in files along with associated graphs.

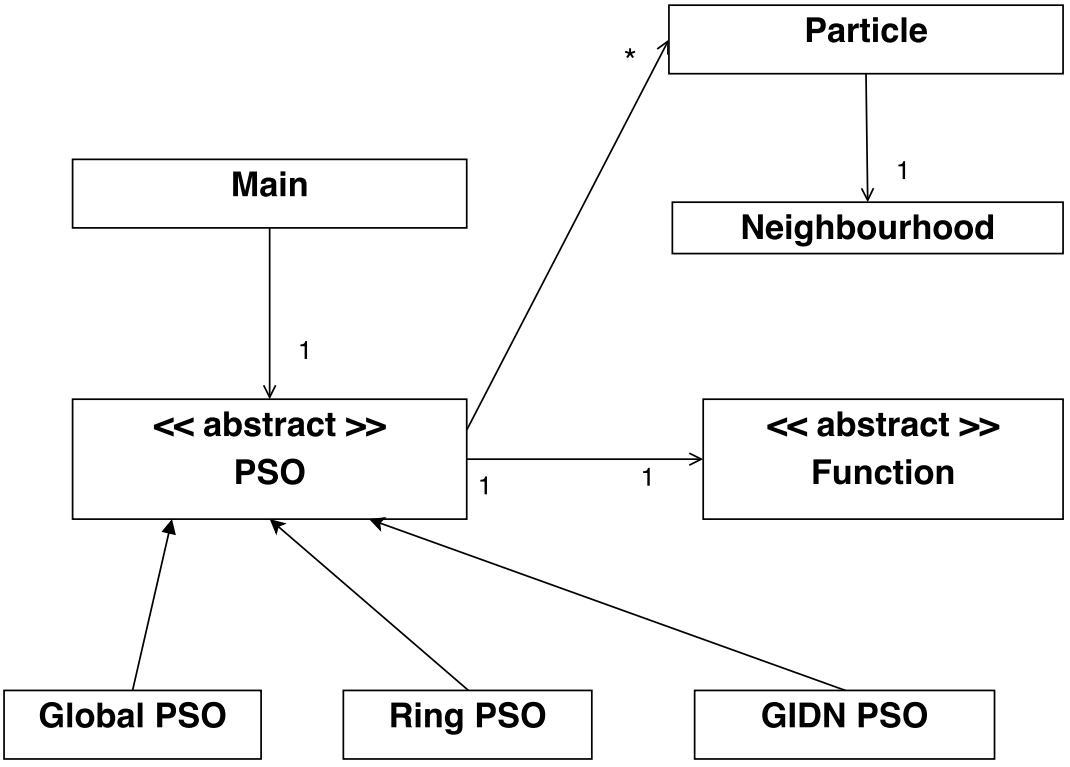


FIGURE 3.2: PSO Simulator Structure

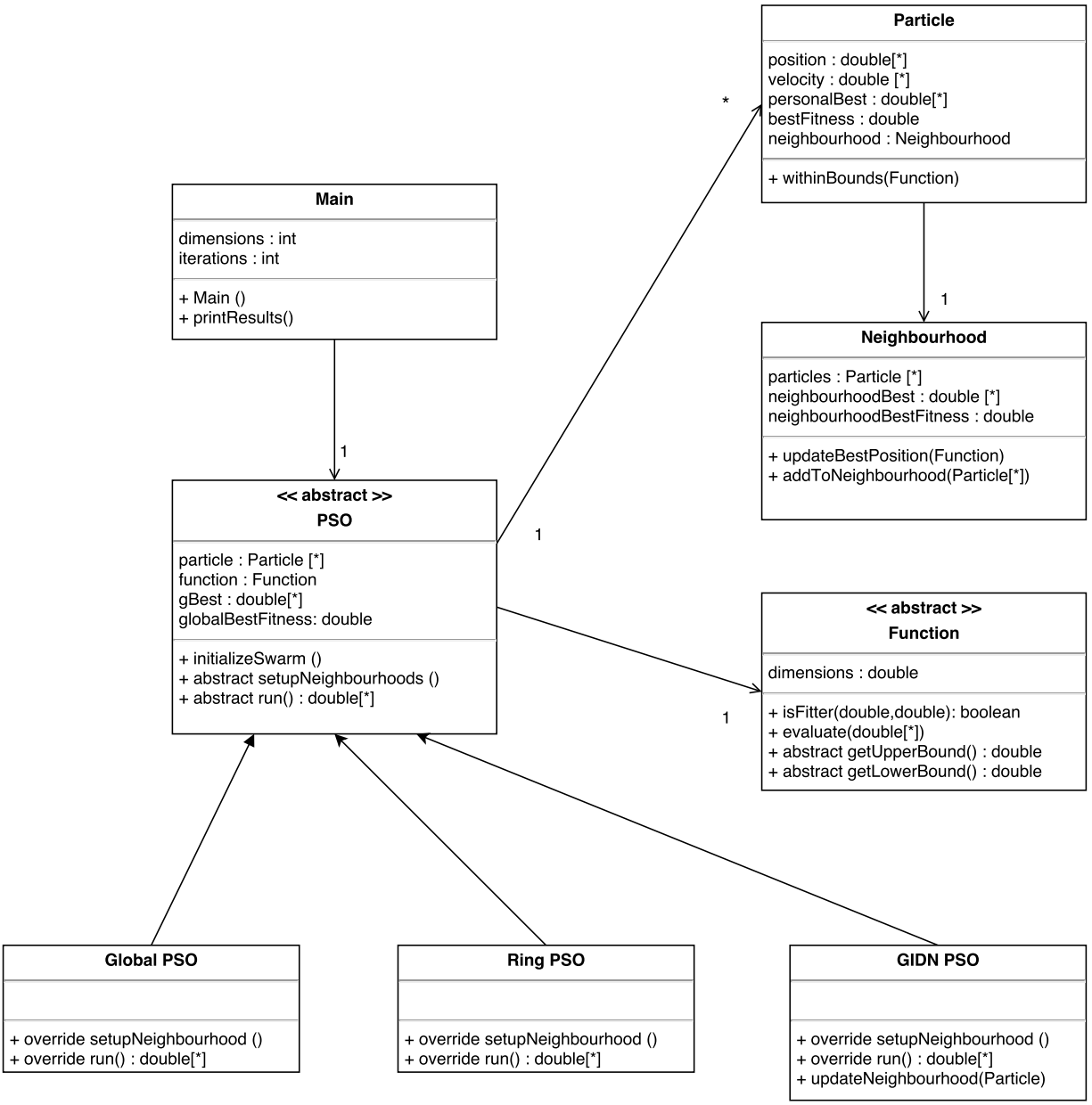


FIGURE 3.3: PSO Class Diagram

Chapter 4

Experimental Benchmarking

This chapter describes the standard PSO variants implemented that should be used as benchmarks when evaluating the performance of alternatives developed. The suite of test functions used the evaluate the performance of each PSO implementation is also described.

4.1 Experimental Parameters

The research described in Chapter 2 produced a number of equations and parameters that are now considered the standard for PSO implementation [2]. Particles are defined in a position x_t with a velocity v_t at a given time t. The velocity and position update equations are given in [6] as:

$$v_{t+1} = \chi(v_t + r_1 c_1(pBest_t - x_t) + r_2 c_2(gBest_t - x_t)) \tag{4.1}$$

$$x_{t+1} = x_t + v_t \tag{4.2}$$

The values for r_1 and r_2 are random numbers between 0 and 1 which are regenerated in each dimension. The values of c_1 and c_2 define the personal and social contribution to a particles velocity and are often equal for simplicity [2]. The *pBest* value for a particle is the best location it has found during the previous

iterations and *gBest* is the best location found by any particle in its neighbourhood. χ is known as the constriction factor and is defined in [6] as:

$$\chi = \frac{2}{\left|2 - \varphi - \sqrt{\varphi^2 - 4\varphi}\right|} \tag{4.3}$$

$$\varphi = c_1 + c_2 \tag{4.4}$$

Values of $c_1 = c_2 = 2.05$ balances the contribution personal and social contribution to a particles velocity while also ensuring convergence [2]. This results in $\chi \approx 0.72984$.

The above described method for updating particle positions will be used to evaluate PSO performance across the test functions using the *gBest* and *lBest* topologies described in [2], as well as the GIDN PSO in [17].
An invisible boundary condition is used [1]. When a particle goes outside the boundaries of the problem space in functions with the optimum within the bounds, the fitness values are not recorded. This results in particles outside the problem space being pulled back in by the personal and social contribution in the velocity equation.
Particles are initialized at random positions in the problem space and the *Half-Diff* method [7] is used to initialise their velocities.

4.2 Test Functions

A series of benchmark functions are used to evaluate the performance of PSO variations under a number of different conditions. A list of functions tested is provided in Table 4.1. All functions are evaluated in 30 dimensions, except for the Schaffer2D and Griewank10D functions which require evaluation in 2 and 10 dimensions respectively.
Different combinations of the first seven functions are often used to benchmark the performance of optimisation algorithms [74][17][2]. Functions *f1* through

TABLE 4.1: 32 Benchmark Functions

Function	Type	Bounds	Optimum
Sphere	Uni	[-5.12,5.12]	0.0
Rosenbrock	Uni	[-2.048,2.048]	0.0
Ackley	Multi	[-32,32]	0.0
Griewank	Multi	[-600,600]	0.0
Rastrigin	Multi	[-5.12,5.12]	0.0
Schaffer2D	Multi	[-100,100]	0.0
Griewank10D	Multi	[-600,600]	0.0
f1 - Shifted Sphere	Uni	[-100,100]	-450
f2 - Shifted Schwefel	Uni	[-100,100]	-450
f3 - Shifted Rotated Elliptic	Uni	[-100,100]	-450
f4 - Shifted Schwefel Noise	Uni	[-100,100]	-450
f5 - Shifted Schwefel with Global Optimum on Bounds	Uni	[-100,100]	-310
f6 - Shifted Rosenbrock	Multi	[-100,100]	390
f7 - Shifted Rotated Griewank	Multi	[0,600]	-180
f8 - Shifted Rotated Ackley with Global Optimum on Bounds	Multi	[-32,32]	-140
f9 - Shifted Rastrigin	Multi	[-5,5]	-330
f10 - Shifted Rotated Rastrigin	Multi	[-5,5]	-330
f11 - Shifted Rotated Weierstrass	Multi	[-0.5,0.5]	90
f12 - Schwefel	Multi	[$-\pi$,pi]	-460
f13 - Shifted Expanded Griewank plus Rosenbrock	Multi	[-5,5]	-130
f14 - Shifted Rotated Expanded Scaffer's F6	Multi	[-100,100]	-300
f15 - Hybrid Composition Function 1	Multi	[-5,5]	120
f16 - Rotated f15	Multi	[-5,5]	120
f17 - f16 with Noise	Multi	[-5,5]	120
f18 - Rotated Hybrid Composition Function 2	Multi	[-5,5]	10
f19 - f18 with Narrow Basin Global Optimum	Multi	[-5,5]	10
f20 - f18 with Global Optimimum on Bounds	Multi	[-5,5]	10
f21 - Rotated Hybrid Composition Function 3	Multi	[-5,5]	360
f22 - f21 with High Condition Number Matrix	Multi	[-5,5]	360
f23 - Non-Continuous f21	Multi	[-5,5]	360
f24 - Rotated Hybrid Composition Function 4	Multi	[-5,5]	260
f25 - f24 without bounds	Multi	[2,5]	260

f25 are described in [75]. These functions consist of numerous attributes including shifting, rotating, expansion, noise, optimum on the bounds, optimum outside the initialization region, non-continuous and consisting of numerous subfunctions. This wide range of function characteristics means that PSO algorithms can be more comprehensively compared. Each of the functions are continuous with the exception of *f23*.

The bounds of the function define the highest and lowest valid positional values and hence describe the size of the problem space. The optimum indicates the lowest possible fitness value of any valid position. The fitness is a single value measure of how close a candidate solution is to the optimal value [76]. Thus it can be understood that the global optimum is located at the position for which the fitness equals to optimum. Functions are considered unimodal if they contain no local optima or multimodal if they consist of multiple local optima [33].

Given particles are randomly initialized the result achieved on a specific run will depend greatly on the starting positions of particles. As a result the mean and standard deviation will be calculated over 50 runs for each function. The number of iterations in each run is 10,000 and 50 particles are used in the swarm. The mean and standard deviation of the best fitness values found during each run will be used to assess the performance of a PSO variation on a given function.

4.3 Statistical Comparisons

The Wilcoxon signed rank test [77] is used to compare the performance of various PSO implementations on each of the benchmark functions. Applying this test will determine whether there exists statistical significance in the performance difference between two PSO implementations. Some of the 25 runs on each function may result in particles converging on a local optima, the likelihood of which increases with an increasing number of local optima in a function. A high number of local optima within a function will likely produce a discretely distributed set of results. The Wilcoxon test does not assume an underlying normal distribution which makes it a valid choice of statistical test. A p-value of 0.05 is used to measure statistical significance at a 95% confidence interval.

Chapter 5

Application of Graph Theory to Dynamic Topologies in PSO

This chapter examines the main difference between static and dynamic topologies. The aim is to critically assess the performance of these two types of topologies on a set of 32 benchmark functions. A description of the graph theory parameters relevant to the operation of dynamic topologies is also provided. These parameters are then applied to PSO variations with dynamic topologies in an attempt to better describe the operation of the algorithm.

5.1 Static Topologies

The topology of a swarm is a description of the connections between its particles [9]. These connections are uni-directional which means that they originate at one particle and go to another. Connections permit particles to communicate their best personal fitness values to others through the social contribution in the velocity update equation [3]. The particles from which a given particle receives information are said to be its neighbours [16]. The neighbours of a particle make up its neighbourhood. Due to the uni-directionality of connections it is not necessary for mutual particles to be in each others neighbourhoods [9].

Static topologies are those that do not change over the course of the optimisation [17]. Static topologies are easier to implement due to their defined nature

and hence are more commonly used in PSO implementations. Examples include global, ring and von Neumann topologies [9]. Global (also called *gBest*) sees every particle connected to every other in the swarm, ring (also called *lBest*) sees the neighbourhood of a particle contain the two particles either side of it and von Neumann connects every particle to its four nearest neighbours to create a lattice structure. Early research examined the effect of connecting every particle in the swarm (*gBest*) versus limiting connections to a particles two nearest neighbours (*lBest*) [16]. It was found that the *gBest* implementation led to quick convergence on the best solutions found in early iterations as the global best location was shared with all particles in the swarm [9]. Quick convergence is desirable in unimodal problems but can result in particles converging on local optima in more complex multimodal problems.

As connections are limited the spread of information with the network becomes slower and particles explore the problem space much more [9]. As a result particles were more robust to local optima when the *lBest* or von Neumann topologies were used compared to *gBest*. There is a clear trade-off between the opposing characteristics of exploration and speed of convergence. Research into a number of static topologies conducted by Kennedy and Mendes [9] found that none performed optimally over a range of problems. It was the von Neumann topology that performed best over all the functions, but not optimally on any.

5.2 Dynamic Topologies

Dynamic topologies can be used in an attempt to achieve a better balance between the exploration and exploitation characteristics of a PSO variation [17]. Increasing the size of particles' neighbourhoods over the course of the algorithm gradually changes the tendency of the swarm from exploration to exploitation. This complexity is added to achieve a greater degree of exploration in the early stages of optimisation as a result of reduced neighbourhood sizes and improved exploitation tendencies at a later stages by increasing the size of each particles neighbourhood [9].

5.2.1 Gradually Increasing Directed Network

Research conducted by Liu et al. [17] developed the Gradually Increasing Directed Network PSO (GIDN PSO) which is an example of how the neighbourhood of a particle can be increased in proportion to the current iteration in an attempt to yield better exploration versus exploitation characteristics. The number of neighbours for each particle is iteration dependent and defined as:

$$\mid H_t^+(p_i) \mid = \lfloor (\frac{t}{max_t})^\gamma * N + b \rfloor \tag{5.1}$$

Where t is the current iteration, max_t is the maximum number of iterations, b is the number of neighbours a particle begins with, N is total number of particles in a swarm and γ is a constant used to control the speed at which the neighbourhood size increases. $\mid H_t^+(p_i) \mid$ is the *in-neighbourhood* set of a particle (p_i) which describes the number of particles in the neighbourhood of p_i. The general operation of the GIDN algorithm is provided as pseudocode below.

Algorithm 2: PSO GIDN

Step 1: Randomly initialize particle positions and velocities

Step 2: Calculate initial neighbourhood number for each particle using Equation 5.1

Step 3: Update each particles neighbourhood as follows:

for p=i *to* N **do**
  if $\mid H_t^+(p_i) \mid > \mid H_{t-1}^+(p_i) \mid$ **then**
    Randomly choose ($\mid H_t^+(p_i) \mid$ - $\mid H_{t-1}^+(p_i) \mid$) distinct particles that still do not have a connection with particle p_i as its new neighbours
  else
end

Step 4: Evaluate each particles fitness according to the fitness function

Step 5: Update particles' personal and neighbourhood best position

Step 6: Update each particle's position and velocity

Step 7: Check termination condition: If not return to **Step 3**, otherwise output best position

In order to answer the first research question relating to the effects of dynamic topologies on PSO, an implementation of the GIDN topology was tested with

the neighbourhood expansion speed parameter $\gamma = 2$ and the number of initial neighbourhood particles $b = 3$ as previous experimentation found these values produced the best results [17]. The performance results of the GIDN PSO compared to the static *gBest* and *lBest* topologies are shown in Table 5.1. The convergence graphs for each of the three variations are also included in Figure 5.1.

Table 5.1: GIDN PSO vs PSO with static topologies

Function	GIDN Mean	GIDN std	Global Mean	Global std	Sphere Mean	Sphere std
Sphere	6.85E-128	4.49E-127	**2.59E-220**	0.00E+00	1.42E-90	2.89E-90
Rosenbrock	9.30E+00	2.01E+00	**2.61E+00**	2.46E+00	1.51E+01	2.02E+00
Ackley	**6.70E-15**	1.52E-15	8.34E-01	8.62E-01	7.34E-15	8.44E-16
Griewank	3.79E-03	7.26E-03	1.15E-02	1.48E-02	**1.48E-04**	1.04E-03
Rastrigin	**4.19E+01**	1.39E+01	5.36E+01	1.10E+01	6.07E+01	1.16E+01
Schaffer2D	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Griewank10D	**1.56E-02**	1.26E-02	6.99E-02	3.69E-02	2.18E-02	1.42E-02
f1	**-4.50E+02**	3.86E-14	-4.50E+02	1.97E-14	-4.50E+02	5.57E-14
f2	-4.50E+02	6.53E-12	**-4.50E+02**	4.64E-13	-4.50E+02	3.05E-04
f3	7.54E+05	4.03E+05	**6.18E+05**	2.82E+05	1.04E+06	3.79E+05
f4	**-4.46E+02**	5.77E+00	-4.38E+02	2.97E+01	4.66E+03	2.16E+03
f5	**4.17E+03**	7.68E+02	4.57E+03	1.12E+03	4.81E+03	8.77E+02
f6	4.08E+02	2.67E+01	**3.95E+02**	9.06E+00	4.10E+02	3.33E+01
f7	-1.80E+02	1.23E-02	-1.80E+02	1.52E-02	**-1.80E+02**	8.97E-03
f8	-1.19E+02	6.00E-02	-1.19E+02	6.99E-02	**-1.19E+02**	6.49E-02
f9	**-2.80E+02**	1.59E+01	-2.49E+02	2.02E+01	-2.41E+02	1.80E+01
f10	**-2.43E+02**	3.34E+01	-1.62E+02	5.19E+01	-2.26E+02	2.11E+01
f11	**1.18E+02**	2.87E+00	1.20E+02	3.07E+00	1.20E+02	2.41E+00
f12	9.19E+03	1.05E+04	1.57E+04	1.86E+04	**3.55E+03**	2.98E+03
f13	**-1.26E+02**	7.69E-01	-1.24E+02	1.86E+00	-1.24E+02	1.61E+00
f14	**-2.88E+02**	4.24E-01	-2.87E+02	5.00E-01	-2.87E+02	3.84E-01
f15	4.75E+02	1.16E+02	4.97E+02	1.02E+02	**4.53E+02**	5.92E+01
f16	**3.20E+02**	1.14E+02	4.58E+02	1.49E+02	3.32E+02	5.94E+01
f17	**3.44E+02**	1.24E+02	4.45E+02	1.46E+02	3.88E+02	7.69E+01
f18	9.27E+02	2.53E+01	9.68E+02	5.51E+01	**9.26E+02**	3.47E+01
f19	9.30E+02	2.55E+01	9.83E+02	4.71E+01	**9.21E+02**	4.20E+01
f20	**9.28E+02**	2.53E+01	9.81E+02	6.04E+01	9.29E+02	3.13E+01
f21	9.91E+02	2.56E+02	1.24E+03	3.29E+02	**9.09E+02**	1.51E+02
f22	**1.31E+03**	2.58E+01	1.42E+03	6.18E+01	1.38E+03	3.17E+01
f23	9.82E+02	1.95E+02	1.16E+03	2.95E+02	**8.97E+02**	8.13E+00
f24	5.28E+02	2.48E+02	6.08E+02	3.68E+02	**4.60E+02**	6.91E-13
f25	4.77E+02	4.04E+01	4.94E+02	1.44E+02	**4.75E+02**	8.70E-01

Best	15		5		12	
Worst	1		19		11	

GIDN Statistically Better			16		13	
GIDN Statistically Worse			4		5	
GIDN Statistically Equal			12		14	

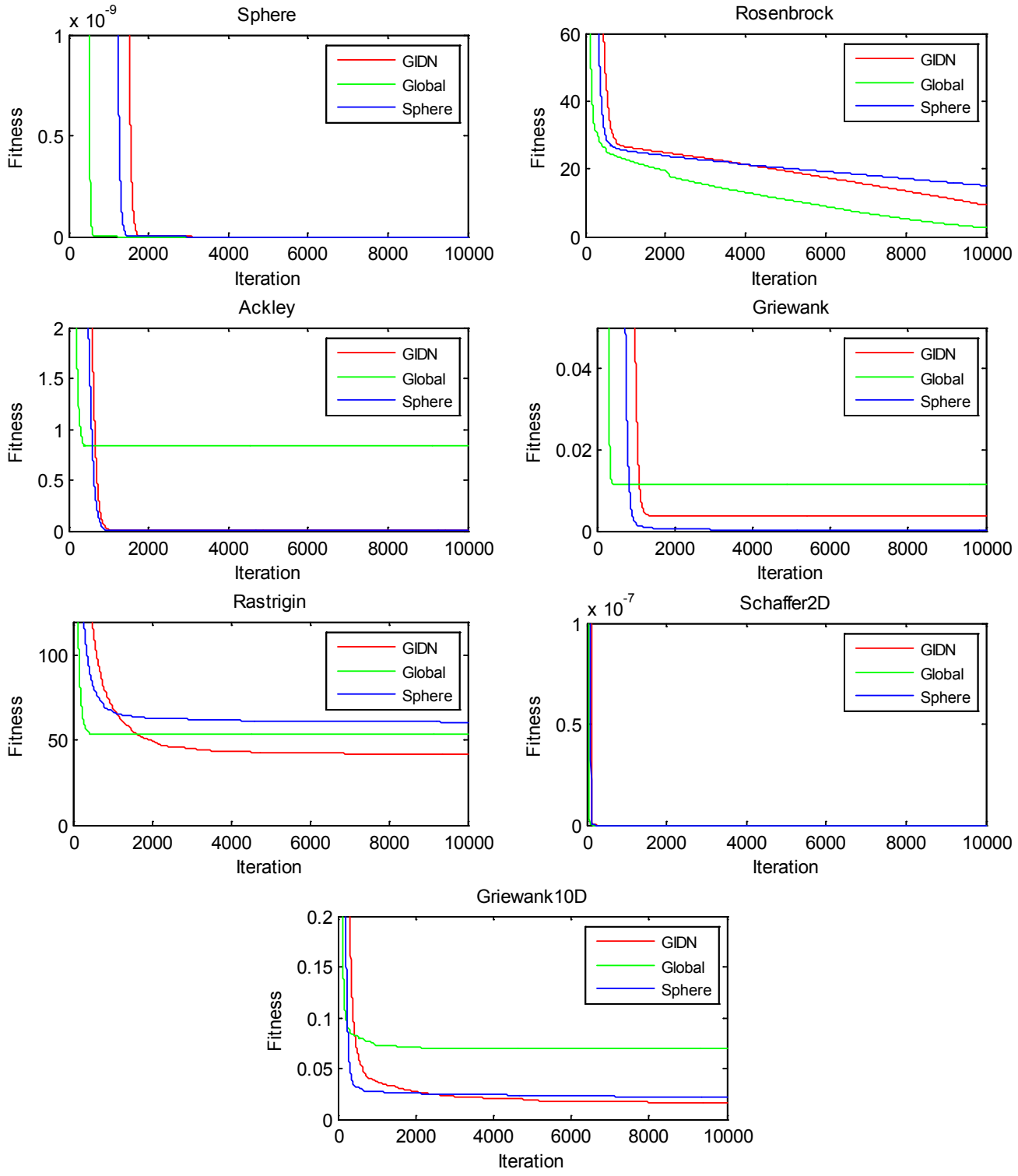


FIGURE 5.1: GIDN PSO versus static topologies - Convergence

5.2.2 GIDN Evaluation

Analysing the results of the static topologies, the same conclusions as [16] can be drawn. The *gBest* performs well on the earlier functions. As indicated in previous research, this suggests quick convergence and hence increased exploitation time on the simpler functions [9]. However it seems to also lead to premature convergence on later functions, negatively impacting performance.
The converse is true of *lBest*. It rarely performs well on simple functions but does well on the later functions. This suggests that a reduction in the speed of information transfer favours exploration over exploitation [17].
From the above results it can be seen that the GIDN PSO performs favourably when compared to either *gBest* or *lBest*. It performed the best across the most amount of functions and worst on only a single function. When compared statistically the improved performance still applied as GIDN performed better on 16 and worse on 4 functions compared to the *gBest* topology and better on 13 and worse on 5 compared to the *lBest* topology.
It seems that GIDN's favourable performance across a wide range of the 32 benchmark functions is due to its dynamic topology which has the ability to transition from early exploration to later convergence [17].

5.3 Graph Theory in PSO

The goal of this section is to provide an overview of graph theory and its relevancy to PSO. A description will be provided into the means by which graph theory will be applied to PSO and the benefits of doing so.

5.3.1 Graph theory applicability

The nature of the graph created by the topology of a PSO swarm is revealed in more clarity when analysed using the descriptions of graphs provided in Section 2.3.
Graphs are mathematical structures that model pairwise relationships between objects [47]. The finite set of objects within a graph are known as vertices.

The relationships between vertices are described using interconnections known as edges. PSO consists of a group of particles that communicate with each other and it is therefore easy to see the applicability of graph theory to PSO. The finite set of particles in a swarm is analogous to the set of vertices required to create a graph. These particles can communicate with each other based on the swarm topology [16]. It is therefore reasonable to model the relationship between a pair of particles using edges between them where a particle can only communicate with another in the swarm if an edge exists in the graph. Therefore, the edges emanating from a particle will travel to all neighbours of that particle.

Simple graphs are a subset of potential graphs in which edges do not connect a particle to itself or multiple edges do not connect the same particle pairs [49]. Edge connections are used to describe particle neighbourhoods and therefore should be examined to determine if the graph created by a PSO swarm is *simple*. The neighbourhood of a particle is used in the calculation of the social contribution to a particles velocity during the velocity update equation (Equation 4.1). It is thus reasonable to assume that a particles neighbourhood should not contain the particle itself as this would cause the social contribution to be based on a particles own best fitness in cases where that fitness is better than any of its neighbours. In a similar fashion, there is no benefit achieved from having the same particle in a neighbourhood multiple times as the single best fitness is chosen from the neighbourhood to influence a particles velocity through social contribution. This means that a particles neighbourhood will never contain itself or multiples of the same particle. When particles are considered as vertices this means that there will be no loops or repeated edges in the graph. The graph can therefore be referred to as *simple*.

Graph edges can be either unidirectional or bidirectional [48]. In the case of bidirectional edges there is no distinction made between the direction of the connection between two vertices. Unidirectional edges must specify a source and destination vertex. Directed graphs, or digraphs, are those which contain unidirectional edges [50]. As the directionality of the graph is decided by the nature of its edges it is again important to consider the particle neighbourhoods to determine the directivity of the graph. Neighbourhoods are unique to individual particles in that there is no obligation for a particle to be in the neighbourhood

of one of its neighbours [9]. As a result it is necessary to correctly specify the source and destination of edges to indicate which particle is contained in the neighbourhood of which other. This means that connections between particles are unidirectional and thus a *digraph* is created.

Graphs can also be termed as weighted or unweighted [49]. Unweighted graphs are those in which there is no form of cost associated with edges. Weighted graphs assign costs to each edge. They can be useful when describing the desirability of traversing certain edges or the importance of specific edges [49]. Edge weights are particularly relevant in calculations relating to the traversal of graphs as they define path costs [47]. It has already been described how edges are used in graphs of PSO swarms to represent the transfer of the best fitness values to a given particle from its neighbours. These fitness values are decided among to act as the social contribution in the velocity update equation. The only factor in this decision is the actual fitness values provided, and has no relationship to the edge upon which it was received i.e. the particle from which it was received [16]. It can therefore be said that all edges are equally considered and that no connection is considered better or worse than any other. This implies that the connections between particles in PSO have no inherent weight associated. Therefore the graph is considered to be *unweighted*. Given connections are all of equal weight it is most obvious to assume a value of 1 for each edge.

In order for a graph to be considered *regular* it is required that every vertex have the same number of neighbours [47]. This property cannot be assumed but is found in common PSO variations. The *lBest* topology sees every particle connected to two others and the *gBest* topology sees every particle connected to all others in the swarm except itself [16]. Even PSO with dynamic topologies display regularities in neighbourhood size [17]. The GIDN PSO uses Equation 5.1 which simply depends on the current iteration to update the size of each neighbourhood simultaneously. Given a constant initial neighbour count it stands that despite the changing graph, updates will always see neighbourhood sizes equal for all particles in GIDN PSO.

Assuming the information presented above, it is most reasonable to model the

structure of the graph created by PSO topologies as a simple, directed, unweighted graph. The regularity of the graph will depend on the exact PSO implementation.

5.3.2 Applying graph theory measures

In order to better understand the operation of GIDN it is intuitive to want to gain an insight into its ability to manage the transition from exploration to exploitation over the course of the algorithms operation. It is known that the *gBest* topology converges quickly due to all particles being connected to one another, whereas the *lBest* topology reduces the speed of convergence through a reduced number of inter-particle connections [16]. It should be easily understood then that the speed at which information spreads throughout the network is governed by the number of connections between particles in the network [9]. A greater number of connections increases the speed of information transfer whereas fewer connections and hence more intermediary particles reduces information transfer speed. GIDN exploits this fact to gradually move from a small number of connections in the network at the beginning of the optimization to having connections between every pair of particles upon completion. The swarm therefore moves from exploration to exploitation due to an increase in information transfer speed as a result of an increasing number of network connections.

In order to numerically understand this feature of the algorithms operation it would be useful to plot the speed of information exchange within the network. Given a greater number of edges in the network increases the speed of information transfer it seems most likely that this characteristic will involve some measure of the number of edges in the network. The *average path length* parameter discussed in Section 2.3 is one such measure to describe information transport efficiency in a graph [18]. It is a single value measure for the entire graph that indicates the average over all particles of the mean of the shortest distances from a particle to all others. [49] The average path length l_G for a graph G is mathematically described as in [78]:

$$l_G = \frac{1}{N * (N-1)}) * \sum_{i \neq j} d(v_i, v_j) \tag{5.2}$$

where N is the number of vertices in the graph and $d(u_i, v_j)$ is the shortest path between u_i and v_j. As mentioned in Section 2.3, the shortest path between vertices may vary if edges between them have alternate weights. This will not be the case when applying the average path length to PSO topologies as particle connections are equivalent for the reasons previously described. Therefore the shortest path is analogous to the number of edges between particle pairs.

A *connected* graph is one in which there exists a path between every pair of vertices in the graph [49]. This characteristic should not be assumed and is intuitively less likely in a directed graph. It becomes even more unlikely as the number of vertices increases but the number of edges remains unchanged. If a vertex cannot be reached from another the distance between them is said to be undefined or infinite [79]. If a single infinite distance was added to the average path length calculation the result would always be infinity. Therefore, it was decided to exclude disconnected paths from the average path length calculation. Plots indicating the average number of disconnected paths will be included alongside those for the average path length when the potential for disconnected paths exists in a PSO variation.

The disconnected paths that can arise in graphs of PSO swarms does not create multiple separate swarms. Swarms graphs with disconnected paths contain particles that are in no way influenced by other particles in the same swarm. The non-influencing particles are unique to each particle in the swarm and may not exist for all particles. Separately, multi-swarm variations of PSO see multiple different swarms interact through attractors such as the repelling charged particles in multi-CPSO [10].

A somewhat unintuitive observation is that if a new connection is added between particles and an disconnected path is subsequently removed, the average path length may increase. This is due to the fact that additional paths that did not previously exist will now be added to the calculation which will result in a higher average path length. A reduction in subsequent path lengths elsewhere in the calculation could offset this increased cost.

5.4 Insights gained from applying the average path length parameter to GIDN

The *average path length* of the swarm was recalculated and plotted for every iteration of optimization during each run of the GIDN PSO algorithm. Given functions were run multiple times the values produced at each iteration were averaged over all runs to produce plots. Figure 5.2 shows the average path length plots for each of the first seven benchmark functions.

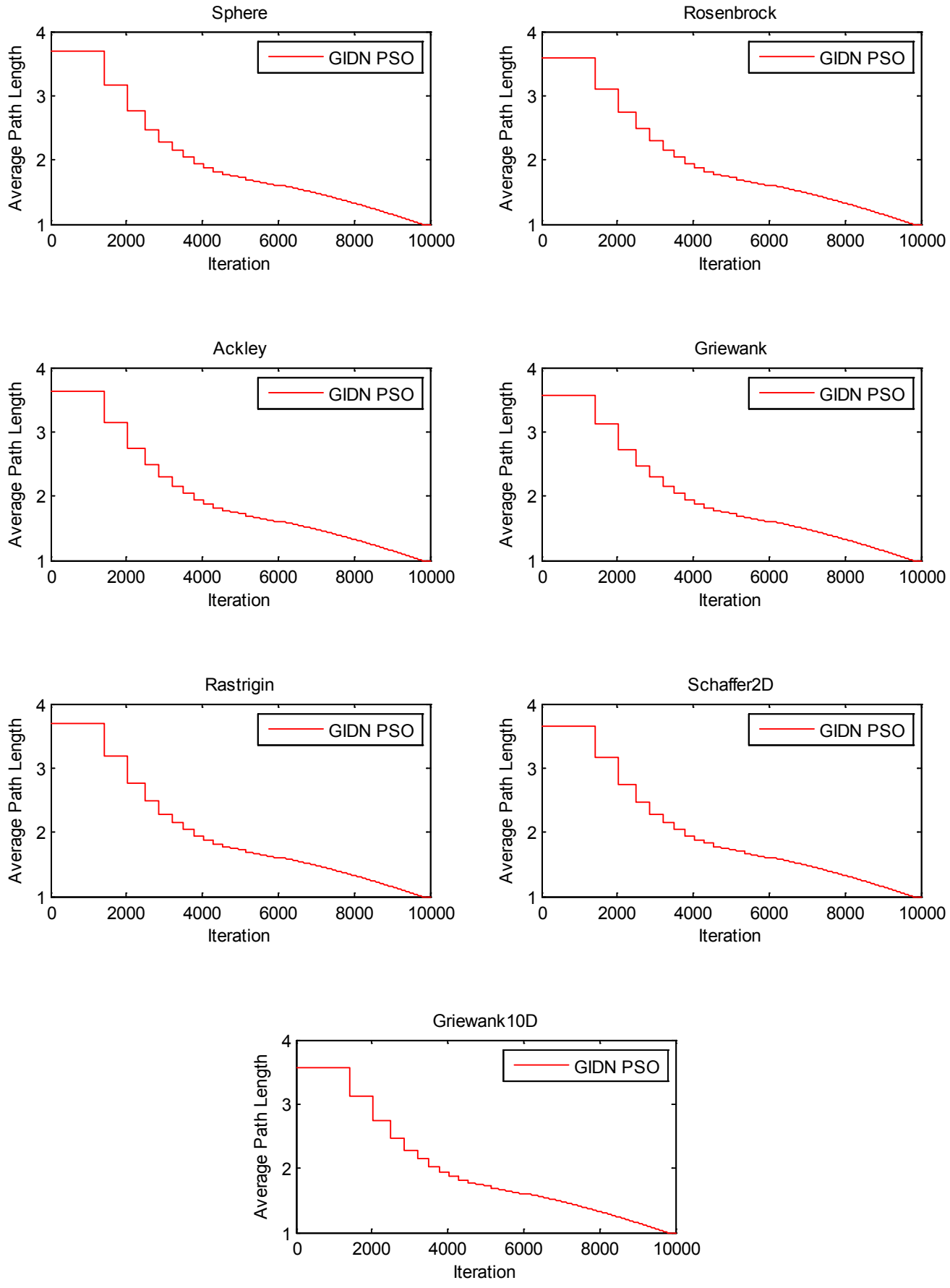


FIGURE 5.2: GIDN PSO - Average Path Length

It can be deduced from the graphs provided that the structure of the average path length plots remain relatively consistent over each function, despite the dynamic topology found in GIDN PSO. This is due to the fact that the formula derived to calculate the size of neighbourhoods is solely based on the iteration and is in no way dependent on the current function or instantaneous performance.
There are minor differences in the values for the average path length between each function and within individual runs. This is due to the random nature of particle selection when increasing neighbourhood size in GIDN, shown in Algorithm 2. The difference is minute enough with respect to the large decreases in the average path length that it is considered negligible.
It can also be seen that the path length never increases in any of the plots, which may occur in the situation described earlier when new connections are added that changes the parameters calculation. This is understandable based on the fact that multiple connections are added to every particle at once and thus the decrease in path length negates any increase from previously disconnected paths being added.

Analysing the graphs produced, it is of note that the average path length remains constant for a number of iterations. This is as a result of the flooring function used in the neighbourhood update equation (Equation 5.1). The number of iterations for which the path length remains constant decreases as the algorithm progresses. This is as a result of the γ value which is used to control how often the neighbourhood size increases.
It is also evident that the average path length decreases in steps, with the average path length dropping significantly during early topology update iterations. By retrospectively analysing the neighbourhood update equation and the pseudocode for GIDN it can be seen that the neighbourhood size is updated for every particle in the swarm at once. The neighbourhood update equation is irrespective of individual particles. That is to say that once it is determined that new particles should be added to the neighbourhood of any given particle the condition is true for all particles due to the fact that the decision is solely based on the current iteration. The result is that within a single iteration, the neighbourhood of every particle in the swarm is increased by the same amount of particles which results in a large decrease in the average path length of the graph.

The step sizes decrease as the algorithm progresses. This is an understandable consequence of an increasing number of connections in the network as the algorithm progresses. During the early operation of the algorithm, there are relatively few connections in the network. As a result, the addition of new connections results in large decreases of the average path length. At later stages in the algorithm, direct connections are being added between pairs of particles where there may already exist a connection through a small number, maybe even one other, intermediary particles. This results in less of a reduction in the average path length values for individual particles towards the latter stages of the algorithm which corresponds directly to smaller reductions in the average path length for the entire graph.
The combination of the constant path length for a finite number of iterations followed by the addition of one or more particles to every neighbourhood at once results in a step like plot for the average path length. Steps will be of decreasing size as the algorithm progresses.

Edges act as the means by which information spreads from a single vertex or particle throughout the graph [48]. This means that in a connected graph, it is guaranteed that information will eventually reach all other particles in the swarm given connectivity guarantees a path from any particle to all others [49]. The path length or the number of "steps" required controls the speed of this information transfer. As a result of the random addition of particles to neighbourhoods in GIDN there exists the possibility of disconnected paths between certain particles. The existence of disconnected paths would result in a disconnected graph being created. Particles are not be influenced by those with which they have no path. As more connections are added during the operation of GIDN, the number of disconnected paths will decrease to zero. Figure 5.3 contains the associated plots indicating the average number of disconnected paths for particles over the course of optimization.

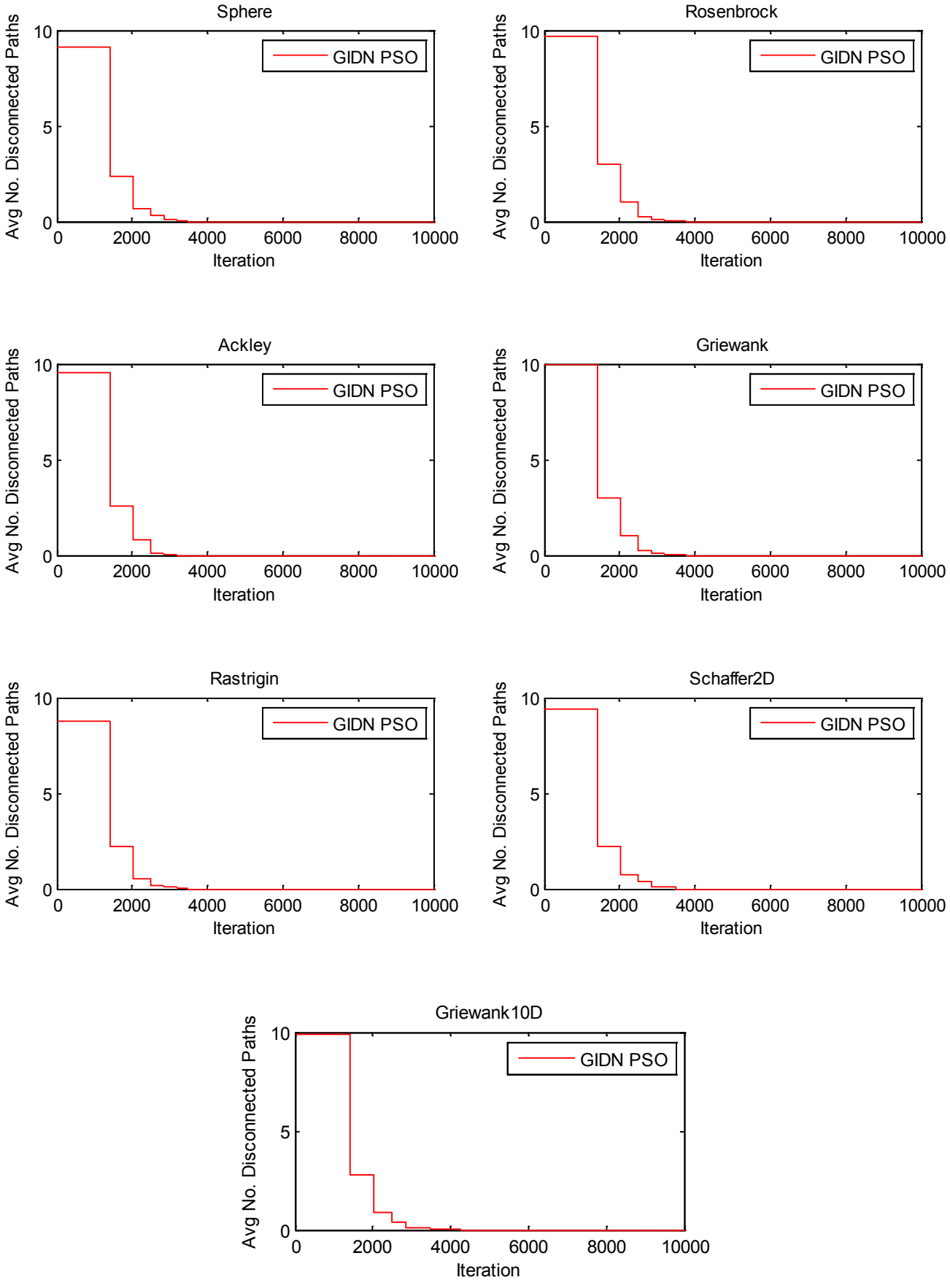


FIGURE 5.3: GIDN PSO - Average number of disconnected paths

It can be seen that there exists an initial average number of disconnected paths above 8 in each of the plots shown. This is reasonable given an initial particle count of $b = 2$ and a swarm size of 50 particles. As expected the number of disconnected paths decreases each time the average path length updates as both measures change in accordance with the topology updates. The number of disconnected paths decreases to zero well before the end of the optimization in the case of all functions shown.

5.5 Summary

The aim of this chapter was to first illustrate the improved performance of dynamic topologies over their static counterparts. In order to statistically verify this improvement an implementation of the GIDN PSO was compared to the *gBest* and *lBest* static topologies over a range of 32 benchmark functions. The results are presented in Table 5.1. It was shown that the GIDN algorithm performed preferably on the range of functions.

Following this, clarity was sought to identify the reasons behind GIDN's improved performance and to describe them quantifiably. The application of the average path length to the algorithm placed a numerical value on the tendency of the swarm to move from exploration to exploitation by describing the increasing speed of information transfer within the network. The visual representation of the average path length revealed interesting insights into the means by which the the adjustment from exploration to exploitation occurred. It was shown that the random addition of particles and updating of neighbourhoods for every particle in the swarm at set intervals resulted in a stair like decrease of the average path length. The nature of GIDN also allows for disconnected paths which were shown to exist in the network, but that decline to zero as more connections are added during optimization.

Chapter 6

Investigation of the initial topology on the performance of GIDN

Owed to an enhanced understanding of the GIDN PSO algorithm gleaned during the previous chapter, the aim of this chapter is to investigate the impact of the initial topology on its operation. The initial topology is being investigated as GIDN was discovered to introduce a relatively high number of disconnected paths between particles in the swarm during early iterations of the algorithm. It is desirable to understand whether this feature positively or negatively impacts upon the exploration of the swarm, which occurs during the early stages of optimization when the average path length is highest.

6.1 Proposed alterations

Given the effect of the initial topology on the operation of the algorithm is to be examined it was decided to retain the standard implementation of GIDN described in Chapter 5, including all the same parameter values. This will act as a control to which alternatives can be compared. Most notably, the *initial neighbourhood particle count* previously chosen as 2 will remain the same during all variations proposed.

To discern the impact of the initially disconnected graph created by GIDN PSO, proposed variations will see the creation of connected graphs from the first iteration. By removing graph disconnectivity every particle is influenced by every other from the beginning of the optimization. The path length between particles will still determine the buffering effect on information shared between a pair of particles. Given an initially limited number of connections relative to the amount of particles in the swarm the path length between certain particles will be quite high but it is guaranteed never to be infinite when the graph is connected [49].

6.1.1 Sphere GIDN

The b parameter in GIDN topology update equation (Equation 5.1) dictates the number of initial neighbours for each particle in the swarm. As stated above this value will remain unchanged, leading each particle to have two neighbours at the start of the optimization. The first suggested variation on the original GIDN algorithm proposes to explicitly set both initial neighbourhood particles. In order to ensure connectivity, particles are to be arranged in a sphere topology. This structure guarantees a connection between every particle pair in the swarm, even if the connection created is distant in some cases. It is worth noting that by connecting particles in a cyclic manner to their direct neighbours the average path length will be close to its maximum possible value [49]. As the optimization proceeds and more particles are added to an individuals neighbourhood, connections will gradually be added between particles in the swarm until the topology resembles the *gBest* topology by the end of the optimization. This is exactly how GIDN PSO concludes with all particles connected to all others through no intermediate links. This variation is referred to as *Sphere GIDN* (SGIDN) based on the initial topology it implements.

6.1.2 Connected GIDN

A second variation seeks to ensure connectivity in the graph using the minimum number of edges possible. This is achieved by cyclically assigning the next particle in the swarm to the neighbourhood of the current particle, except in the case of the final particle whose neighbour is the first particle. As the initial

neighbourhood count is set to two, the second neighbour of each particle will be chosen at random during the first iteration of the optimization in accordance with Algorithm 2. This variation is referred to as *Connected GIDN* (CGIDN) as it starts by producing a connected graph. It is known that the chain topology created during the initial addition of a single particle to all neighbourhoods will produce the maximum average path length possible for a graph of that size [49]. The average path length will be reduced during the first iteration when a second particle is randomly added to the neighbourhood of each swarm member.

6.2 Variation evaluations

In order to compare the performance of these two new variations with respect to GIDN PSO, each variation is evaluated on the same 32 benchmark functions outlined in Chapter 4. The performance results can be seen in Table 6.1. The graphs of the average number of disconnected paths for these new variations are excluded as they will remain at zero during the entire operation of the algorithm given the basis for creating these variations was that the graphs produced would be connected and thus have no disconnected paths. The convergence graphs for the first 7 functions are included in Figure 6.1. The average path length plots for the SGIDN and CGIDN variations are shown in Figures 6.2 and 6.3 respectively.

One important note is that the average path length plots for the two new variations should not be compared with the corresponding plot for the GIDN algorithm. This is due to the disconnected nature of the graph created by GIDN PSO. When calculating the average path length, it was decided in Chapter 5 that disconnected paths should be excluded from the calculation (by assuming a distance of 0) which results in lower equivalent values. The plots could technically be compared once all disconnected path lengths have been removed from the GIDN swarm graph, which will occur after a certain update iteration before the end of the optimization.
Given both new variations contain no disconnected paths meaning their respective average path length plots can be suitably compared and contrasted.

TABLE 6.1: Initially connected variations vs GIDN

Function	GIDN Mean	GIDN std	SGIDN Mean	SGIDN std	CGIDN Mean	CGIDN std
Sphere	6.85E-128	4.49E-127	**5.84E-129**	2.58E-128	6.61E-128	3.27E-127
Rosenbrock	9.30E+00	2.01E+00	9.99E+00	2.71E+00	**9.03E+00**	1.08E+00
Ackley	**6.70E-15**	1.52E-15	7.05E-15	1.23E-15	6.98E-15	1.30E-15
Griewank	3.79E-03	7.26E-03	**0.00E+00**	0.00E+00	3.15E-03	5.53E-03
Rastrigin	**4.19E+01**	1.39E+01	6.04E+01	1.20E+01	4.29E+01	1.17E+01
Schaffer2D	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Griewank10D	**1.56E-02**	1.26E-02	1.61E-02	1.67E-02	2.09E-02	1.65E-02
f1	-4.50E+02	3.86E-14	-4.50E+02	4.25E-14	-4.50E+02	4.55E-14
f2	-4.50E+02	6.53E-12	**-4.50E+02**	1.40E-12	-4.50E+02	4.23E-12
f3	**7.54E+05**	4.03E+05	1.02E+06	4.05E+05	8.40E+05	4.34E+05
f4	**-4.46E+02**	5.77E+00	-4.40E+02	1.19E+01	-4.45E+02	7.35E+00
f5	4.17E+03	7.68E+02	4.87E+03	1.10E+03	**4.16E+03**	6.16E+02
f6	**4.08E+02**	2.67E+01	4.09E+02	2.70E+01	4.11E+02	4.21E+01
f7	-1.80E+02	1.23E-02	-1.80E+02	1.79E-02	**-1.80E+02**	1.47E-02
f8	-1.19E+02	6.00E-02	**-1.19E+02**	1.44E-01	-1.19E+02	6.54E-02
f9	**-2.80E+02**	1.59E+01	-2.48E+02	2.27E+01	-2.80E+02	1.54E+01
f10	-2.43E+02	3.34E+01	-2.21E+02	2.67E+01	**-2.57E+02**	1.99E+01
f11	**1.18E+02**	2.87E+00	1.20E+02	2.53E+00	1.18E+02	3.27E+00
f12	9.19E+03	1.05E+04	**5.97E+03**	6.21E+03	1.01E+04	9.53E+03
f13	**-1.26E+02**	7.69E-01	-1.24E+02	1.36E+00	-1.26E+02	1.05E+00
f14	**-2.88E+02**	4.24E-01	-2.87E+02	3.64E-01	-2.88E+02	4.97E-01
f15	4.75E+02	1.16E+02	4.56E+02	5.96E+01	**4.41E+02**	1.08E+02
f16	3.20E+02	1.14E+02	3.40E+02	9.57E+01	**3.18E+02**	1.09E+02
f17	3.44E+02	1.24E+02	3.39E+02	8.75E+01	**2.92E+02**	9.27E+01
f18	9.27E+02	2.53E+01	9.29E+02	3.65E+01	**9.20E+02**	3.30E+01
f19	9.30E+02	2.55E+01	9.26E+02	3.98E+01	**9.25E+02**	2.40E+01
f20	**9.28E+02**	2.53E+01	9.31E+02	3.31E+01	9.28E+02	2.70E+01
f21	9.91E+02	2.56E+02	**8.78E+02**	7.12E+01	9.28E+02	1.59E+02
f22	**1.31E+03**	2.58E+01	1.35E+03	4.50E+01	1.31E+03	2.23E+01
f23	9.82E+02	1.95E+02	**8.96E+02**	4.72E+00	9.84E+02	1.90E+02
f24	5.28E+02	2.48E+02	4.60E+02	1.59E-12	**4.60E+02**	1.55E-12
f25	4.77E+02	4.04E+01	4.73E+02	2.20E+00	**4.72E+02**	5.17E-01

Best	12		7		11	
Worst	8		16		5	

GIDN Statistically Better	7		0
GIDN Statistically Worse	4		2
GIDN Statistically Equal	21		30

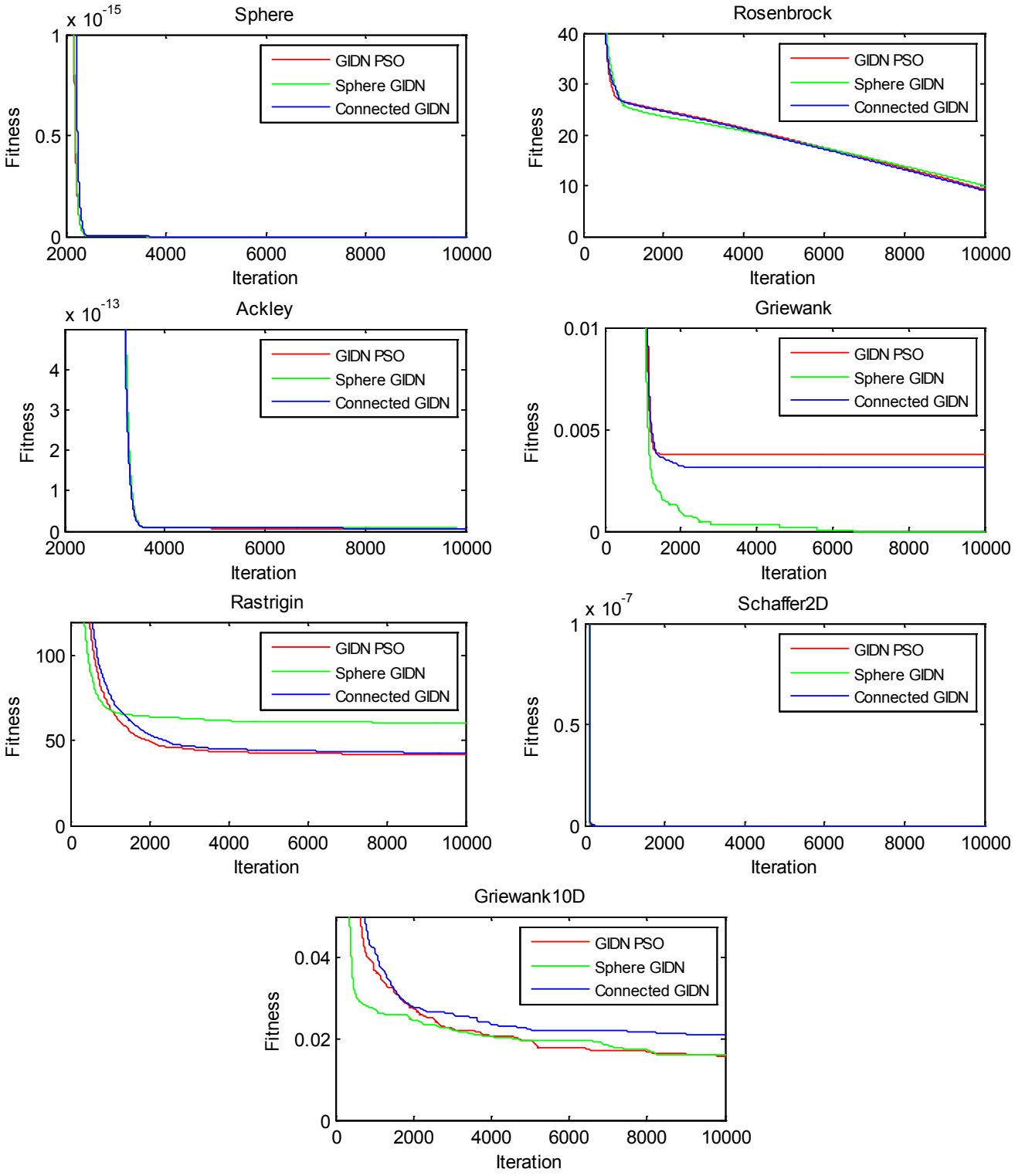


FIGURE 6.1: GIDN initial topology variations - Convergence

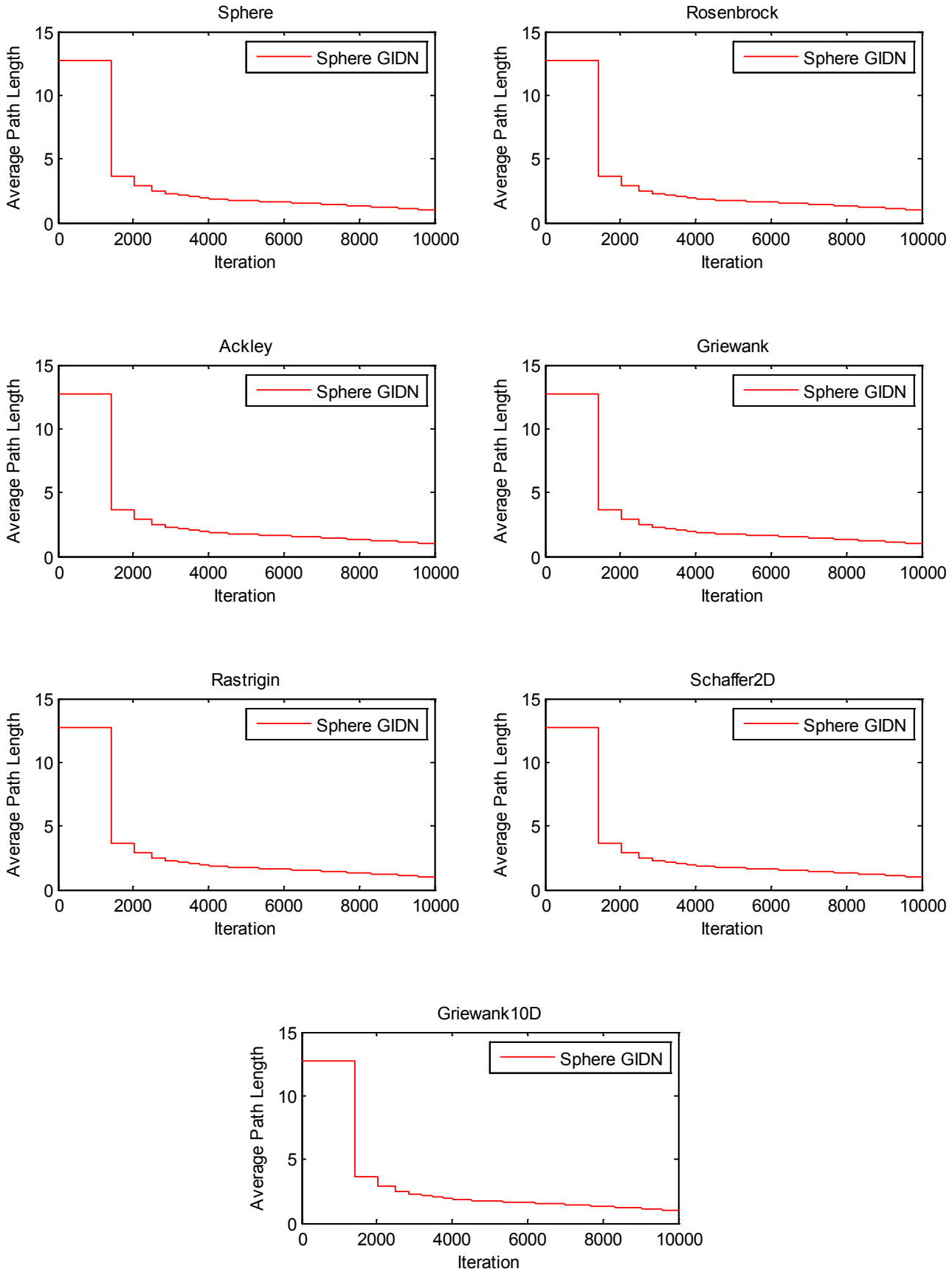


FIGURE 6.2: Sphere GIDN - Average Path Length

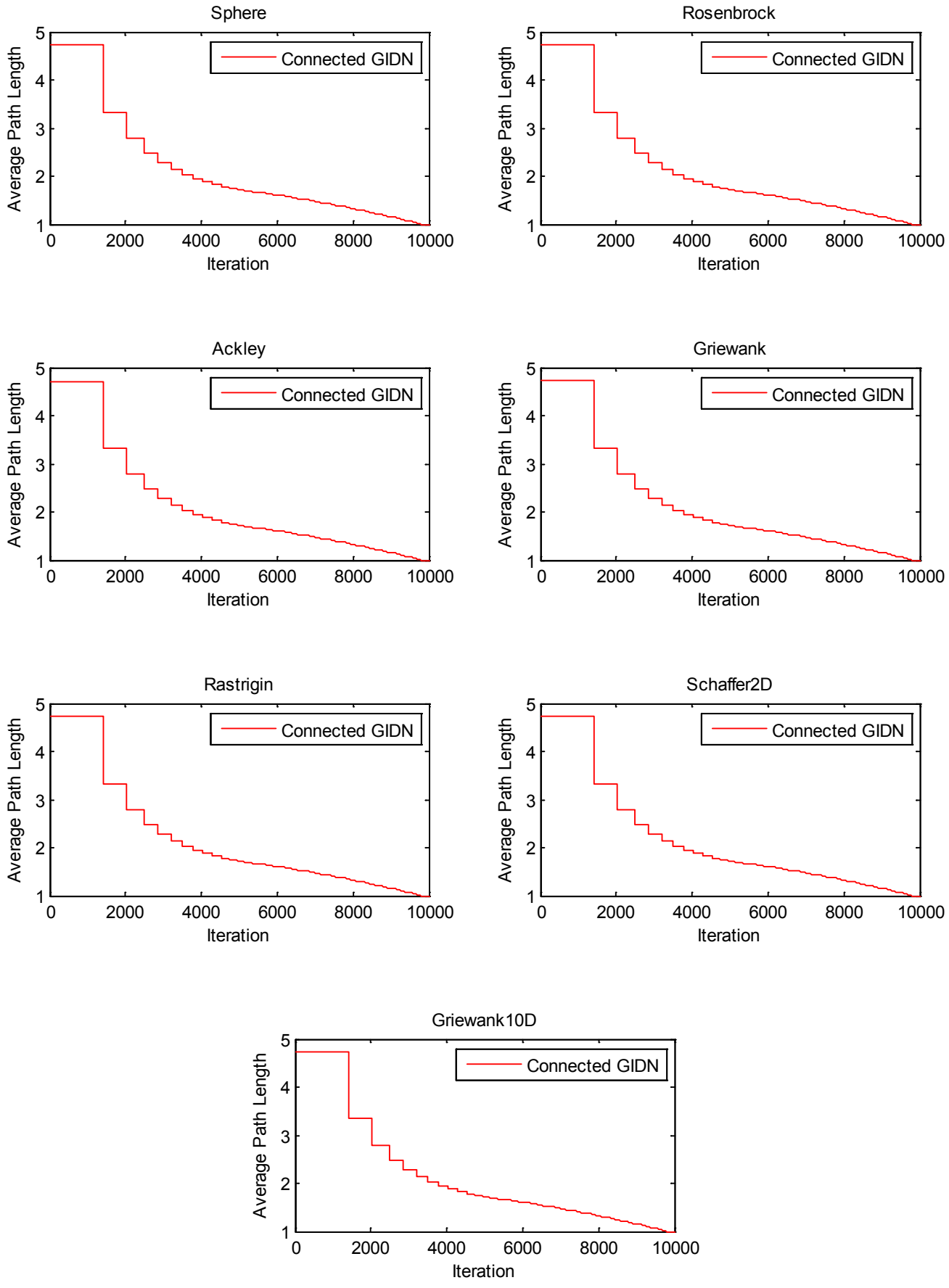


FIGURE 6.3: Connected GIDN - Average Path Length

Considering all three variations, the Sphere GIDN performed the best on the least amount of functions and worst on the most amount of functions. It was the worst performing algorithm on 16 of the 32 benchmark functions and performed statistically worse than the original GIDN PSO on 7 of these functions. It only statistically improved the performance of GIDN on 4 functions.
The Connected GIDN and GIDN PSO algorithms performed very similarly, producing the best results on 11 and 12 of the functions respectively. Two functions were deemed statistically different between in CGIDN and GIDN PSO algorithms. The Connected GIDN variation performed better on both functions which were two of the later hybrid composition functions.

6.2.1 Sphere GIDN Discussion

On first observation, the most striking result of the Sphere GIDN algorithm comes from analysis of the average path length graphs produced. It can be seen that the initial value is very high across all functions. The initial sphere topology created means that every run of the function will produce the same initial average path length value of 12.755. As previously mentioned this approximates the highest average path length possible when each particle has two neighbours [49]. Given the nature of Equation 5.1 which determines the neighbourhood update iterations, the average path length value remains constant at this value for a significant period of the total optimization. As a result of this high initial average path length value the stepped decreases at earlier iterations are sizable. The large decreasing steps are caused by a pair of contributing features:

- As previously described, the step like decreases in average path length are as a result of each particles neighbourhood size being increased at the same specific iterations.
- The reason for such a large decrease in the average path length from the initial value is as a result of random particles being added to neighbourhoods. Intuitively, the addition of random connections results in a lower average than a structured addition of connections where the average path length is intentionally kept high. Therefore when adding random connections to the

sphere topology graph the average path length decreases from its maximum value when each particle has two values to a non optimal value when each particle has three connections. It is known that random graphs are good approximations of the lower bound for the average path length when every particle has the same number of neighbours [49]. The random addition of particles to swarm neighbourhoods thus results in a close to minimum average path length value. The transition from the maximum average path length possible to a near minimum value understandably leads to a greater initial step size than in normal GIDN.

It was discussed how the number of connections in a graph relates to the speed of information transfer within it. It has been shown that the average path length parameter relates to the tendency of the swarm to explore or converge. It can be understood then that a rapid decrease in the average path length from an initially high average path length to a low value has the effect of moving the swarm from a high degree of exploration towards a focused exploitation after the first topology update.

The result is that the Sphere GIDN performs statistically worse on 7 of the benchmark functions, and better on only 4. Given it seems the initial sphere topology negatively influences the performance of GIDN it is reasonable to hypothesise that the disconnected nature of the GIDN graph could be the reason for its improvement in performance, and that the lack of disconnected paths in SGIDN is hampering the optimization process. It could also be that the instant descent from exploration to exploitation could be contributing to a decrease in performance. The third research question addressed in Chapter 7 seeks to understand the effect of a more gradual transition from exploration to exploitation.

6.2.2 Connected GIDN Discussion

The Connected GIDN variation should be analysed in an effort to understand whether the degradation in performance of the Sphere GIDN is an effect of the high average path length or a lack of disconnection in the graph. From the graphs provided in Figure 6.3 it can be seen that the initial average path length of the

CGIDN variation begins much lower than that of SGIDN. Swarm particles are initially connected in a chain like fashion to guarantee the graphs connectedness [49]. The second particle in each initial neighbourhood is randomly assigned. This has the effect of immediately lowering the average path length. The initial value will vary slightly in each run due to the element of randomness in particle selection but usually results in a value just below 5. This is less than half that of SGIDN. From Table 6.1 it can be seen that CGIDN performs very similarly to GIDN PSO, only improving on two functions, with all other functions producing statistically equivalent results.

As a consequence of these results, the suggestion that the disconnected particles in a swarm are the reason GIDN performs so well seems to be untrue. Given both the standard GIDN algorithm and the CGIDN variation perform well compared to SGIDN it stands to reason that it is a lower initial average path length rather than the disconnected graph that results in improved performance. Given the nature of the GIDN algorithm sees the initial topology sustained for a defined number of iterations it is reasonable to assume that the high average path length in SGIDN is causing too much exploration amongst the swarm for too long a period at the beginning of the optimization. This results in less exploitation time at later stages and hence lower results.

This result does not disregard the performance of the original implementation of GIDN. Instead it recognises that the disconnected particles in the graph simply act as one means of reducing the initial average path length. It seems to be equally valid, even slightly favourable, to have a fully connected graph of particles when implementing GIDN, so long as the average path length is kept low to begin with. It requires extra, if trivial, implementation effort to produce a connected graph of particles without much benefit to the performance across the benchmark functions. For this reason it can be said that the standard GIDN algorithm with random initialization of connections between particles produces a desirable initial average path length with the least amount of programming effort. Should improved performance be required the CGIDN may yield better results than the classic GIDN algorithm, but this decision is problem and application specific.

6.3 Structured GIDN

Arising from the insight into the operation of the Sphere GIDN variation it could be argued that it performs worse due to a sharp transition from exploration to exploitation. This is evident given a large decline in the average path length at early stages of the algorithms operation.
A third variation was implemented that aims to confirm that it is indeed the high initial average path length that explains its bad performance and not the quick transition from exploration to exploitation.

There are two requirements that must be met in order to act as a valid test are:

- The algorithm must begin with a similarly high average path length to SGIDN.
- The algorithm must decrease the average path length in a similar step like fashion to SGIDN, but with reduced step sizes, particularly during early stages.

In order to achieve a high initial average path length the Structured GIDN variation will use the same topology initialization as in SGIDN; that is to connect particles in a sphere like fashion. The reason for the large reduction in the average path length during update iterations is that particles are added to neighbourhoods randomly which produces a close to minimum average path length [49]. To reduce the size of the steps in the average path length plot it was decided to try to maintain the maximum average path length possible when each particle had to have a given number of neighbours. The number of neighbours in each neighbourhood at any given iteration is obviously defined by Equation 5.1, originally provided as part of GIDN PSO. Watts [49] described how the maximum average path length is achieved when particles have one neighbour and are connected in a chain-like manner with each particle connected to its nearest neighbour. He noted that this does not hold true as neighbourhood size increases, but that by connecting graph vertices to the next-nearest unconnected vertex a close to maximum average path length can be produced for any number of neighbours in a neighbourhood. For example, when each particle is required to have two neighbours the average path length is a maximum when every particle is connected

to the two neighbours directly either side of it. If the number of neighbours required increases to four, the direct neighbours of every particles direct neighbours should be included in its neighbourhood to approximate the maximum average path length. A particles neighbourhood should still not contain itself, even if it is one of the particles suggested to be included in the neighbourhood next.

The general idea behind this variation is that the particles with which a given particle has the greatest distance should remain unconnected for as long as possible. This is achieved by adding the closest particles a given particle does not already have as its neighbour to its neighbourhood. Given a particle has neighbours on both sides, it was decided that the algorithm should only increase the neighbourhood size when multiples of two particles are to be added. The

pseudocode for the particle addition in Structured GIDN is provided:

Algorithm 3: Structured GIDN

Step 1: Calculate neighbourhood size for iteration using Equation 5.1;

Step 2: For each particle:

```
for p=i to N do
    if | H_t^+(p_i) | >| H_{t-1}^+(p_i) | then
        let numParticlesToAdd = | H_{t-1}^+(p_i) | − | H_t^+(p_i) |
        if ((numParticlesToAdd % 2) == 0) then
            let numAddEitherSide = numParticlesToAdd/2
            for j=0 to numAddEitherSide do
                let clockwiseIndex = neighbourhood size of p_i + 1
                if clockwiseIndex >SWARM_SIZE then
                    clockwiseIndex = clockwiseIndex - SWARM_SIZE
                end
                Add particle at clockwiseIndex to neighbourhood of p_i
                let antiClockwiseIndex = -(neighbourhood size of p_i + 1)
                if antiClockwiseIndex <0 then
                    antiClockwiseIndex = SWARM_SIZE + antiClockwiseIndex
                end
                Add particle at antiClockwiseIndex to neighbourhood of p_i
            end
        end
    end
end
```

An implementation of the Structured GIDN variation was tested on the same 32 benchmark functions as before. Table 6.2 shows its performance alongside that of GIDN PSO and SGIDN. The average path length plots for the first 7 functions of the Structured GIDN algorithm are included in Figure 6.4.

Table 6.2: Structured GIDN vs GIDN

Function	Structured GIDN Mean	Structured GIDN std	GIDN Mean	GIDN std	SGIDN Mean	SGIDN std
Sphere	**2.22E-143**	8.68E-143	6.85E-128	4.49E-127	5.84E-129	2.58E-128
Rosenbrock	**7.46E+00**	1.20E+00	9.30E+00	2.01E+00	9.99E+00	2.71E+00
Ackley	7.27E-15	9.64E-16	**6.70E-15**	1.52E-15	7.05E-15	1.23E-15
Griewank	2.95E-04	2.07E-03	3.79E-03	7.26E-03	**0.00E+00**	0.00E+00
Rastrigin	5.81E+01	9.61E+00	**4.19E+01**	1.39E+01	6.04E+01	1.20E+01
Schaffer2D	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Griewank10D	2.17E-02	1.76E-02	**1.56E-02**	1.26E-02	1.61E-02	1.67E-02
f1	-4.50E+02	5.45E-14	-4.50E+02	3.86E-14	-4.50E+02	4.25E-14
f2	-4.50E+02	1.80E-12	-4.50E+02	6.53E-12	-4.50E+02	1.40E-12
f3	7.98E+05	3.28E+05	**7.54E+05**	4.03E+05	1.02E+06	4.05E+05
f4	-2.98E+02	2.46E+02	**-4.46E+02**	5.77E+00	-4.40E+02	1.19E+01
f5	4.88E+03	1.02E+03	**4.17E+03**	7.68E+02	4.87E+03	1.10E+03
f6	**4.06E+02**	3.20E+01	4.08E+02	2.67E+01	4.09E+02	2.70E+01
f7	**-1.80E+02**	5.85E-03	-1.80E+02	1.23E-02	-1.80E+02	1.79E-02
f8	-1.19E+02	7.22E-02	-1.19E+02	6.00E-02	**-1.19E+02**	1.44E-01
f9	-2.38E+02	2.07E+01	**-2.80E+02**	1.59E+01	-2.48E+02	2.27E+01
f10	-2.18E+02	2.39E+01	**-2.43E+02**	3.34E+01	-2.21E+02	2.67E+01
f11	1.20E+02	2.03E+00	**1.18E+02**	2.87E+00	1.20E+02	2.53E+00
f12	**4.57E+03**	3.86E+03	9.19E+03	1.05E+04	5.97E+03	6.21E+03
f13	-1.24E+02	1.55E+00	**-1.26E+02**	7.69E-01	-1.24E+02	1.36E+00
f14	-2.87E+02	4.02E-01	**-2.88E+02**	4.24E-01	-2.87E+02	3.64E-01
f15	**4.48E+02**	5.72E+01	4.75E+02	1.16E+02	4.56E+02	5.96E+01
f16	3.33E+02	9.01E+01	**3.20E+02**	1.14E+02	3.40E+02	9.57E+01
f17	3.60E+02	7.13E+01	3.44E+02	1.24E+02	**3.39E+02**	8.75E+01
f18	9.32E+02	2.57E+01	**9.27E+02**	2.53E+01	9.29E+02	3.65E+01
f19	9.27E+02	3.54E+01	9.30E+02	2.55E+01	**9.26E+02**	3.98E+01
f20	**9.19E+02**	4.44E+01	9.28E+02	2.53E+01	9.31E+02	3.31E+01
f21	8.79E+02	1.01E+02	9.91E+02	2.56E+02	**8.78E+02**	7.12E+01
f22	1.35E+03	2.87E+01	**1.31E+03**	2.58E+01	1.35E+03	4.50E+01
f23	8.96E+02	7.36E+00	9.82E+02	1.95E+02	**8.96E+02**	4.72E+00
f24	4.60E+02	9.16E-13	5.28E+02	2.48E+02	**4.60E+02**	1.59E-12
f25	4.75E+02	1.19E+00	4.77E+02	4.04E+01	**4.73E+02**	2.20E+00

Best	7		14		8	
Worst	9		11		10	

Structured GIDN Statistically Better			8		6	
Structured GIDN Statistically Worse			8		2	
Structured GIDN Statistically Equal			16		24	

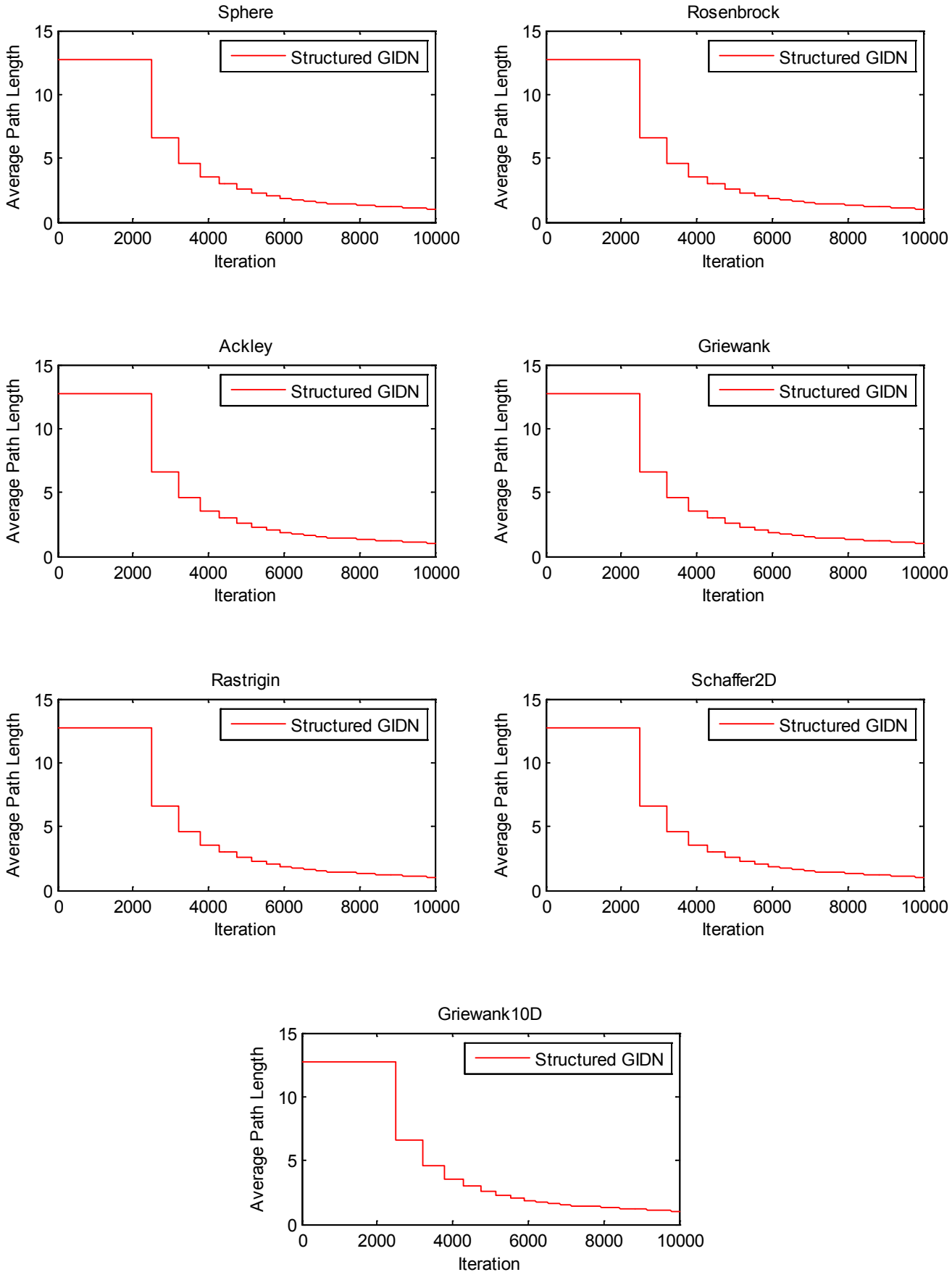


FIGURE 6.4: Structured GIDN - Average Path Length

The Structured GIDN algorithm performed statistically better than the GIDN on 8 functions and worse on 8. Despite not improving the performance of the GIDN PSO considerably, the Structured GIDN approach showed improvements over the SGIDN algorithm. It managed to statistically perform better on 6 functions and worse on only 2. It is still the GIDN implementation that performed best most often.

6.3.1 Structured GIDN Discussion

It is interesting to note that the Structured GIDN algorithm performs statistically worse on 5 of the last 17 functions compared to GIDN PSO, and better on 3. It shows almost identical performance to SGIDN on the same functions, with Structured GIDN only worsening on a single function. This suggests that GIDN PSO performs much better than either the Structured GIDN or Sphere GIDN on the later, more complex functions. Given both new variations have similarly high initial average path length values it seems reasonable to assume that this is the reason for their poor performance on later functions. It could be argued that the initial path length is so high that it causes particles to aimlessly wander the landscapes during early iterations. This phenomena could be analogous to the *lBest* which sees particles aimlessly wandering due to the inhibition of communication among the swarm members [9]. The speed of information transfer within the network may be too slow to provide any meaningful benefit to the updating of particle velocities through the social contribution factor.
Perhaps it is the case that there exists a point when enough exploration has occurred. Given the high initial average path lengths in both new variations it could be the case that time is wasted exploring when the best regions have been identified and instead the swarm should focus on exploiting. Whether the amount of exploration required is dependent on time (a certain number of iterations), a certain initial average path length or a function of both remains unclear and further research would require to be able to fully understand these consequences.

The Structured GIDN and SGIDN variations differ when analysed in terms of decreases in the average path length values during optimization. Structured GIDN

sees the average path length value decrease much later in the optimization process when compared to SGIDN which sees an instant drop from exploration to exploitation during the first neighbourhood topology update. Given the Structured GIDN variation performs statistically better on 6 functions and only worse on 2, it seems fair to say that the quick transition from exploration to exploitation negatively affects the performance in SGIDN. Chapter 7 addresses the effect of a much more gradual transition from exploration to exploitation in detail.

It is clear that the Structured GIDN algorithm performs better than the SGIDN algorithm when compared across all functions. When comparing GIDN PSO against the Structured GIDN algorithm on the first 15 functions not discussed above, it is the Structured GIDN that performs better on 5 functions and worse on 3. This suggests that the performance is improved on the simpler functions when the Structured GIDN is used versus GIDN PSO. Overall both algorithms perform better and worse on an equal number of functions. The consequences of the 'No Free Lunch' theorem [68], where it is said that no algorithm can perform the best on all types of functions, seem applicable.

The removal of the flaw in SGIDN sees the Structured GIDN algorithm produce a more gradual transition from exploration to exploitation. The high initial average path length and subsequent increase in the amount of exploration conducted now provides a different balance between exploration and exploitation compared to GIDN PSO. The negative trade-off is that there is less time for the swarm to perform exploitation. Differences in algorithm properties may make the Structured GIDN more applicable to certain types of problems than GIDN PSO, or vice versa.

6.4 Summary

The research question addressed by this chapter sought to uncover the effect of the initial topology on the performance of various PSO implementations with dynamic topologies.
It had been discovered in Chapter 5 that the GIDN PSO algorithm introduced

disconnected paths between particles which lasted for a period during the optimization. The aim of this chapter was to discover whether this unique feature was the reason for the superior performance of GIDN. The first variation implemented initialized the swarm with a sphere topology. This ensured the removal of all disconnected path lengths between particles. This variation performed worse than the original GIDN algorithm (in part due to a sharp transition between exploration and exploitation).

By connecting particles in a sphere topology the minimum average path length possible when every particle has two connections was achieved. This was an unintended consequence of creating a connected graph representing the swarms topology. To remove this feature another variation was implemented that connected particles to their next neighbour to create a chain. This lead to the lowest possible average path length where every particle was connected and each particle had one neighbour. The chosen parameters meant that a second particle was randomly added to each neighbourhood during the first iteration of the algorithm. The random addition of particles lead to a reduction in initial average path length compared to SGIDN. Thus, the connectivity of the graph was still guaranteed but the average path length was much reduced. The Connected GIDN variation performed almost identically to GIDN PSO, only improving statistically on a single function.

Finally, it was worth investigating whether a large decrease in the average path length of the Sphere GIDN variation after the initial network topology update contributed to its negative performance as the swarm moved for high exploration to exploitation instantly at the first network update. This new variation known as Structured GIDN sought to maintain the maximum possible average path length while ensuring that each neighbourhood contained the same number of neighbours at any point during optimization as is the case in GIDN PSO. The results showed performance improvements over SGIDN. This lead to the conclusion that the quick transition from exploration to exploitation in SGIDN hindered its performance. Structured GIDN performed similarly compared to GIDN PSO. Structured GIDN seems to perform better on earlier functions thanks to an increase in exploration but worse on later functions due to a resultant lack of exploitation. The converse is true of GIDN PSO.

From all of this, it can be said that the lower average path length in the GIDN

PSO is sufficient to produce good results but that further research should focus on the effect of a gradual decline in average path length from a higher initial level of exploration.

Chapter 7

Topological updates informed by the average path length

During both previous chapters it has been shown that changes in the the average path length plots correspond to significant differences in the operation of PSO variants with dynamic topologies. It is also known that the average path length is an indicator of the speed of information transfer within a swarm and therefore describes the exploration versus exploitation tendencies of the swarm [18][9]. Consequently, this chapter focuses on implementing PSO variants in which the topology is determined based on the average path length at each iteration. Adjusting the topology to match a desired average path length is an attempt to better control the exploration and exploitation of the swarm during optimization. It is to be investigated whether this results in statistically significant performance improvements.

7.1 Flaws Identified

Analysis of the average path length plot for the swarm topology of GIDN revealed that a step like graph was produced, for the reasons explained in Section 5.4. Given the decision to update neighbourhood size is solely based on the current iteration it stands to reason that the condition is true for all particles at the same time. This means that the neighbourhood of each particle will be expanded

simultaneously. The result is that many new edges are added to the graph at the same time which sees the average path length reduce by large amounts. A stepped pattern is produced by the fact that the update equation is only satisfied after a set number of iterations. Steps decrease in magnitude as the algorithm progresses due to an increasing number of connections within the swarm.

Chapter 6 previously dealt with the presence of disconnected paths between particles in the GIDN algorithm. The aim of this chapter is to attempt to control the average path length parameter to produce a more robust variation of GIDN. By removing the stepped nature from the average path length plot it is hoped that a more gradual transition between exploration and exploitation can be produced during optimization.

7.2 Proposed alterations

Proposed variations aim to produce smooth shifts from exploration to exploitation over the course of optimization rather than operating at discrete average path length values for sustained periods. The GIDN PSO algorithm will be used as a basis for the development of both variations proposed. Guiding the operation of new variants using the average path length parameter instead of an iterative approach means that the nature of the algorithm changes from simply describing the speed of information transfer within the network to influencing it. It is hoped that by controlling this parameter an improved performance can be achieved.

7.2.1 Linear GIDN

The first variation produces a linear descent of the average path length from the initial value to the minimum possible value of 1. The equation to produce the desired, linearly decreasing average path length is:

$$apl_t = apl_i - ((apl_i - 1) * \frac{t}{max_t}) \tag{7.1}$$

Where apl_t is the desired average path length for the current iteration, apl_i is the initial average path length after adding the initial particles to all neighbourhoods, t is the current iteration and t_{max} is the maximum number of iterations. Given the average path length produced will be linear in nature over the course of the optimization this variation was termed *Linear GIDN* (LGIDN).

The algorithm begins operation in the exact same manner as GIDN PSO by initializing particles with a random position and velocity. The number of particles defined by the *initial neighbourhood particle count* are randomly added to the neighbourhood of each particle as before. The average path length of the network before optimization is then stored to be used in the calculation of the desired average path length apl_t during each iteration of the optimization. The use of Equation 7.1 replaces the need for the network update equation found in GIDN (Equation 5.1). On each iteration, the desired average path length was compared to the actual path length. If the actual value was greater than the desired value the average path length must be decreased by increasing the number of connections in the swarm i.e. increase the size of a particle neighbourhoods. In order to add to the neighbourhood of each particle in a consistent manner, particles are added to neighbourhoods in a round-robin fashion. A counter was setup that incremented each time a particles neighbourhood was updated (or reset to zero if the final particles neighbourhood was updated) to indicate the next particle to have its neighbourhood increased. The counter was used to check if the current particle was the next to be updated. If it was, then a random particle was added to its neighbourhood, the average path length was recomputed and the counter index was updated. It is worth noting that this is the first variation in which graph regularity does not hold. As described in Chapter 2 it is required that each particle have the same number of neighbours for a graph to be considered regular [47]. Given the use of the round-robin system certain particles may have more neighbours than others at different points during the algorithms operation. Some but not all of the graphs created during optimization will be regular.
By adding a single particle to the neighbourhood of one particle at a time the average path length could be controlled with a very fine level of granularity. The

pseudocode to manage the implementation of this variation is provided below:

Algorithm 4: PSO Linear GIDN

Step 1: Randomly initialize particles positions and velocities

Step 2: Setup initial particle neighbourhoods randomly

Step 3: Calculate the initial average path length before optimization begins

Step 4: For every particle p on each iteration:

```
if apl_actual >= apl_t then
    if p == particles[currentIndex] then
        Add a random particle to the neighbourhood of particle p
        Recaculate the average path length
        if currentIndex == SWARM_SIZE then
            currentIndex = 0
        else
            currentIndex++
        end
    end
end
```

Step 5: Evaluate each particles fitness according to the fitness function

Step 6: Update particle's best position and neighbourhood best position if either respective value has been improved upon

Step 7: Update each particle's position and velocity

Step 8: Check for termination condition: If not return to **Step 3**, otherwise output best position

The average path length graphs produced for the first 7 benchmark functions are included in Figure 7.1 below. It should be noted that the plots are not perfectly linear. This is acceptable given the average path length is changed by adding new connections randomly to the network. This has the effect of reducing the average path length by an amount that may be more than the desired amount. From the plots produced it can be seen that the difference is minute and can be considered negligible.

The plots of the average number of disconnected paths at each iteration of the algorithm for the first 7 benchmark functions are also included in Figure 7.2.

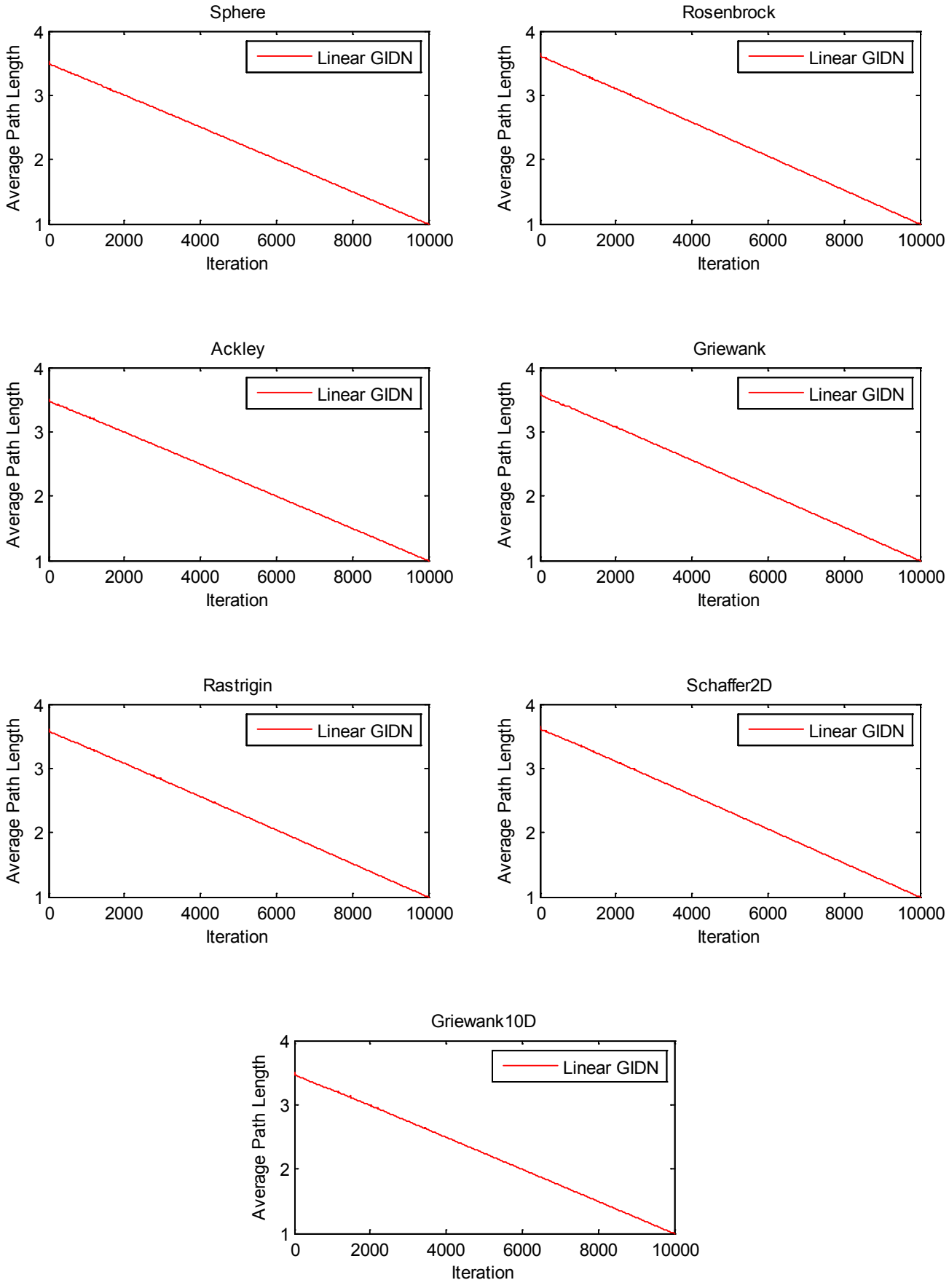


FIGURE 7.1: Linear GIDN - Average Path Length

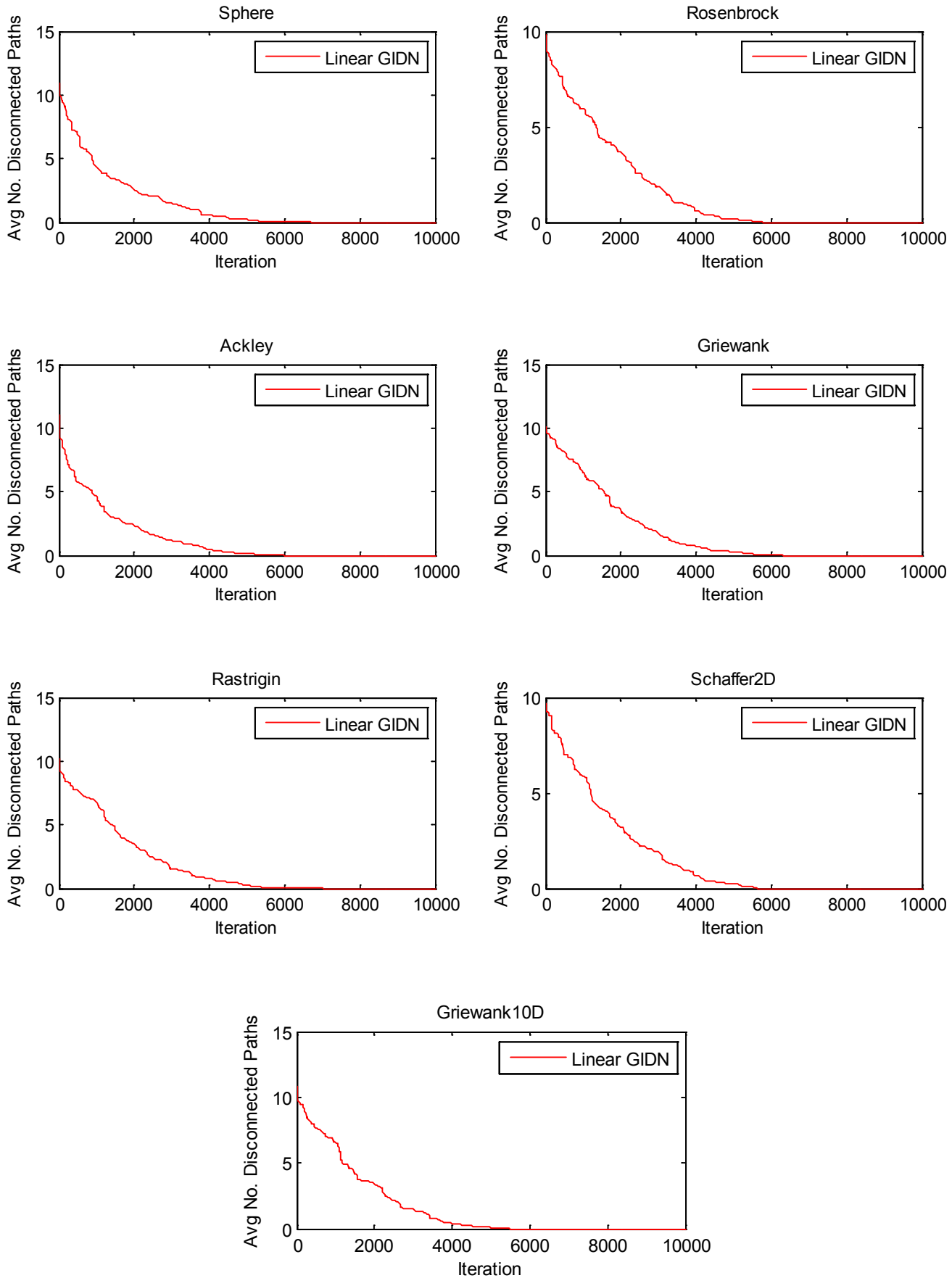


FIGURE 7.2: Linear GIDN - Average number of disconnected paths

7.2.2 Gradual GIDN

A second variation attempted to produce average path length plots that resembe that of the original GIDN algorithm but with its stepped nature smoothed out.

The Gradual GIDN (GGIDN) algorithm operates in much the same way as the original GIDN function but increases the size of the neighbourhoods for each particle at various times to avoid the increase of every neighbourhood at once. It relies on the fact that two important factors in the GIDN algorithm can be predetermined:

- The iteration at which the next neighbourhood update will occur.
- The increase in the size of each neighbourhood at update iterations.

Given this information, a function can be written that determines the next iteration at which a topology update will occur. This iteration should be stored along with the increased neighbourhood size.

Algorithm 5: Next Update Iteration and Neighbourhood Size Function

Step 1: Let currentSize = neighbourhoodSizeForIteration(currentIteration)

Step 2: Let nextIncreaseIteration = currentIteration

Step 3: Let nextSize = neighbourhoodSizeForIteration(nextIncreaseIteration)

Step 3: Repeat until the neighbourhood size increases:

while *currentSize == nextSize* **do**
    nextNeighbourhoodIncreaseIteration++
    nextSize = neighbourhoodSizeForIteration(nextIncreaseIteration)
end

The *neighbourhoodSizeForIteration* function is an implementation of Equation 5.1 that is adapted to allow the iteration t be be specified as an argument.

Using the knowledge of particles' current and future neighbourhood states, a more gradual transition in average path length over multiple iterations can be achieved.

Firstly, the number of iterations until the next neighbourhood update and the difference in the neighbourhood size can be divided to determine the number of

iterations until the neighbourhood of every particle should be increased by one particle. This could be used to decrease the size of the steps in the average path length plot of the original GIDN implementation but would not remove them entirely as every neighbourhood is still being increased at once, if only by one particle.

Further granularity in average path length decreases can be achieved if the neighbourhood of every particle is not increased at the same time after a given number of iterations. This requires that the neighbourhoods of individual particles be updated on subsequent iterations. Dividing the number of iterations until each neighbourhood should add one particle by the size of the swarm reveals the number of iterations until a single particle should be added to a single neighbourhood. This value is used as a counter in that every time the current iteration reaches a multiple of this value, the neighbourhood of the next particle in the cycle is increased. The cycling of particles works exactly the same as in the case of the Linear GIDN variant; the next particle to be updated is the one that directly follows it in the list of particles, unless it is the final particle in which case the first particle is chosen. As was the case due to the use of the counter in LGIDN, neighbourhoods will of different sizes during optimization and hence the graphs created by the swarm during optimization are not assumed to be *regular*.

The counter value only remains constant until the current iteration reaches the *nextNeighbourhoodIncreaseIteration*, at which point it is recomputed based on the next neighbourhood increase iteration. This reduces the steps in the average path length plot to a series of connected linear lines with non zero slopes.

The pseudocode relating to the above describe network update procedure is:

Algorithm 6: Next Update Iteration and Neighbourhood Size Function

Step 1: Let nextNeighbourhoodIncreaseIteration = currentIteration

Step 2: For each iteration:

if *(iteration == nextNeighbourhoodIncreaseIteration)* **then**

Step 2.1: Determine next neighbourhood update iteration and size using Algorithm 5

Step 2.2: Let numIter = (nextNeighbourhoodIncreaseIteration - currentIteration)

Step 2.3: Let neighbourhoodSizeDifference = (nextSize - currentSize)

Step 2.4: Let numIterNeighbourhoodIncreaseByOne = (numIter / neighbourhoodSizeDifference)

Step 2.5: Let numItersBetweenUpdates = round(numIterNeighbourhoodIncreaseByOne / SWARM_SIZE)

Step 2.6: Let iterationSpacingCounter = currentIteration

end

Step 3: For each particle p:

if $(p == particles[currentIndex])$ *&&*
$(iterationSpacingCounter == iteration)$ **then**

Add a random (still unconnected) particle to the neighbourhood of p

iterationSpacingCounter += numItersBetweenUpdates

if *currentIndex == SWARM_SIZE* **then**

currentIndex = 0

else

currentIndex++

end

end

The average path length graphs produced for the first 7 benchmark functions are documented in Figure 7.3.

It should be noted that the average path length increases during the early stages of most functions shown. As discussed in Chapter 5, disconnected paths are not included in average path length calculations. Given this variation is based on GIDN PSO a number of disconnected path lengths will exist in the network at the commencement of optimization. In the GIDN algorithm, increases were not seen in the average path length value as the addition of multiple particles to all

neighbourhoods at once reduced the average path length sufficiently to offset any increase in calculations as a result of previously disconnected paths being added to the computation. Given the gradual addition of particles in GGIDN there exists situations in early iterations where the increased computation due to the removal of disconnected path lengths does not outweigh the reduction in average path length elsewhere when the number of network connections is increased and thus the average path length actually increases. The actual disconnected path length plots for the first 7 functions are included in Figure 7.4.

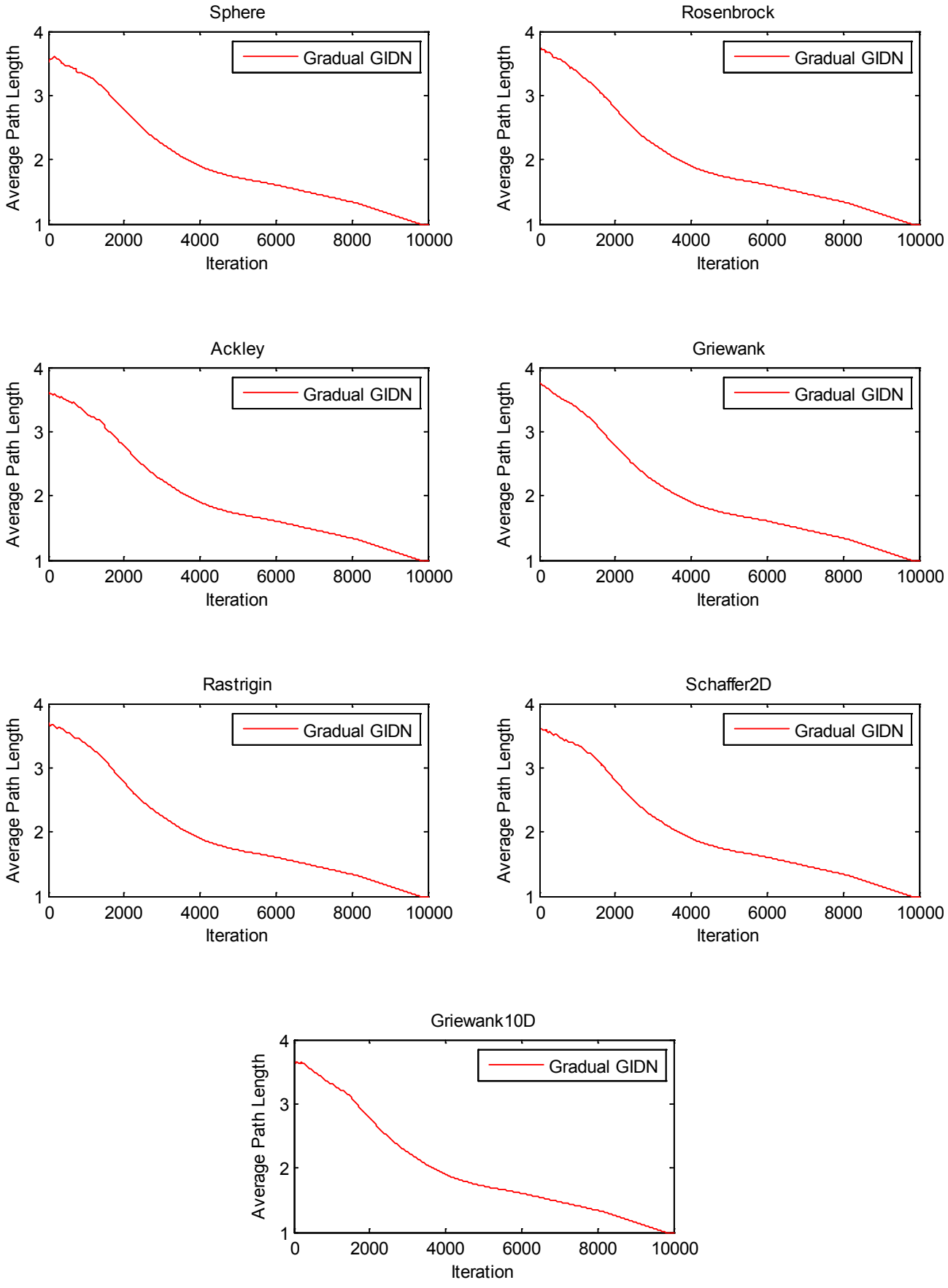


FIGURE 7.3: Gradual GIDN - Average Path Length

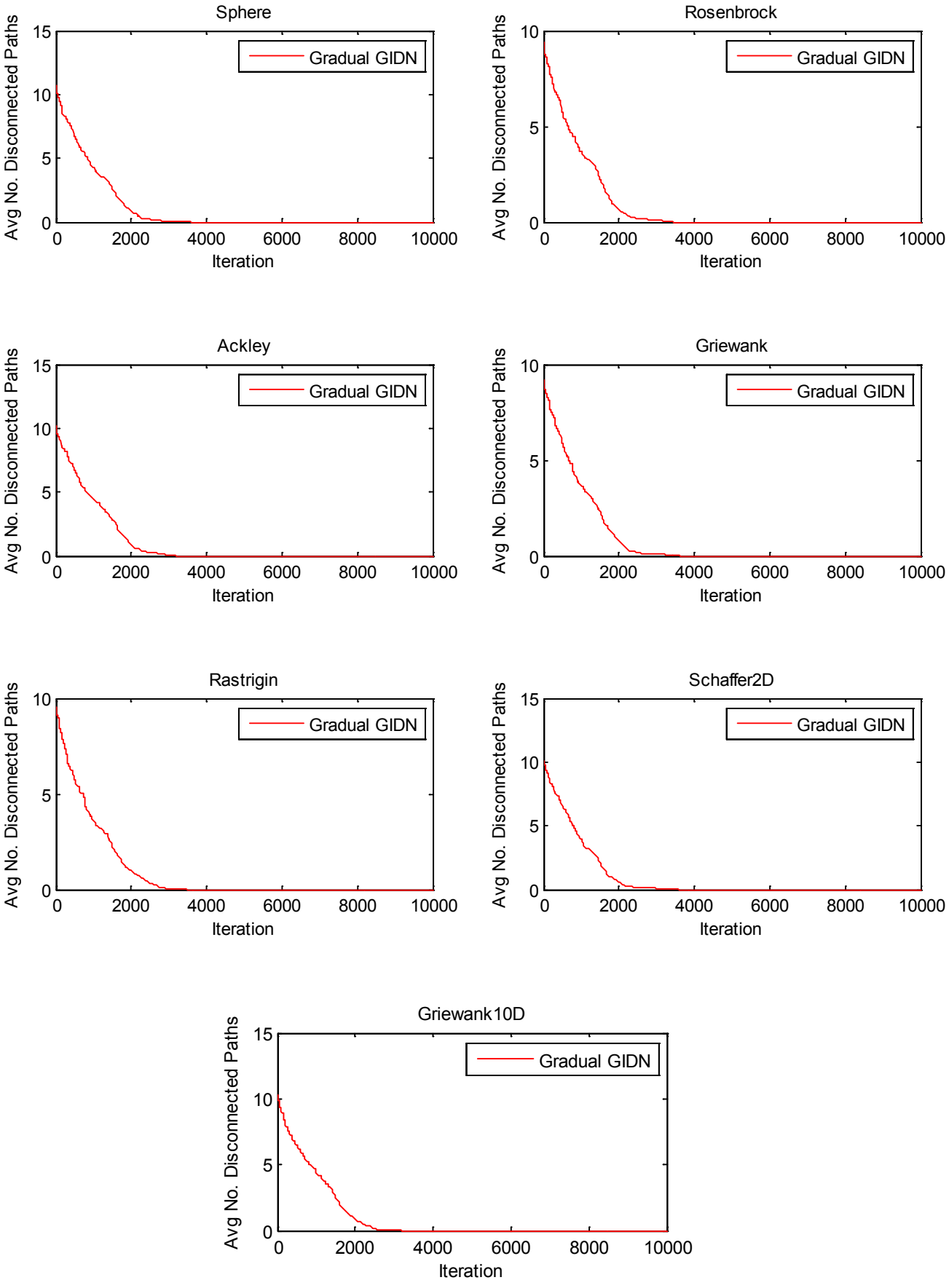


FIGURE 7.4: Gradual GIDN - Average number of disconnected paths

7.3 Variation Evaluation

The two new variations are compared against the original GIDN implementation by measuring optimization performance on the same set of 32 benchmark functions as before. The Wilcoxon test is applied to statistically analyse and compare the results to check for significance. The performance results are detailed in Table 7.1. The convergence graphs for both proposed variations are also plotted against the original GIDN PSO in Figure 7.5.

TABLE 7.1: Controlled APL GIDN variants vs GIDN PSO

Function	GIDN Mean	GIDN std	GGIDN Mean	GGIDN std	LGIDN Mean	LGIDN std
Sphere	6.85E-128	4.49E-127	**8.91E-137**	5.38E-136	9.68E-115	4.09E-114
Rosenbrock	9.30E+00	2.01E+00	**8.31E+00**	1.48E+00	9.81E+00	2.46E+00
Ackley	**6.70E-15**	1.52E-15	7.27E-15	9.64E-16	6.77E-15	1.47E-15
Griewank	3.79E-03	7.26E-03	**3.44E-03**	9.23E-03	5.47E-03	8.34E-03
Rastrigin	4.19E+01	1.39E+01	**3.91E+01**	9.97E+00	4.13E+01	1.24E+01
Schaffer2D	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
Griewank10D	**1.56E-02**	1.26E-02	2.73E-02	2.20E-02	1.78E-02	1.39E-02
f1	-4.50E+02	3.86E-14	-4.50E+02	4.18E-14	-4.50E+02	4.18E-14
f2	-4.50E+02	6.53E-12	**-4.50E+02**	5.62E-13	-4.50E+02	1.37E-09
f3	7.54E+05	4.03E+05	**7.20E+05**	3.09E+05	7.48E+05	3.37E+05
f4	**-4.46E+02**	5.77E+00	-4.41E+02	1.38E+01	-4.30E+02	1.94E+01
f5	**4.17E+03**	7.68E+02	4.29E+03	1.08E+03	4.19E+03	8.76E+02
f6	4.08E+02	2.67E+01	**3.99E+02**	1.16E+01	4.09E+02	2.89E+01
f7	-1.80E+02	1.23E-02	**-1.80E+02**	1.16E-02	-1.80E+02	1.63E-02
f8	-1.19E+02	6.00E-02	-1.19E+02	7.65E-02	**-1.19E+02**	8.69E-02
f9	**-2.80E+02**	1.59E+01	-2.75E+02	1.95E+01	-2.75E+02	1.41E+01
f10	-2.43E+02	3.34E+01	-2.47E+02	2.13E+01	**-2.50E+02**	2.47E+01
f11	**1.18E+02**	2.87E+00	1.18E+02	3.22E+00	1.18E+02	3.32E+00
f12	9.19E+03	1.05E+04	9.04E+03	1.10E+04	**8.84E+03**	9.29E+03
f13	-1.26E+02	7.69E-01	-1.26E+02	1.17E+00	**-1.26E+02**	1.13E+00
f14	-2.88E+02	4.24E-01	**-2.88E+02**	4.94E-01	-2.87E+02	3.30E-01
f15	4.75E+02	1.16E+02	**4.53E+02**	1.04E+02	4.53E+02	1.17E+02
f16	**3.20E+02**	1.14E+02	3.95E+02	1.43E+02	3.65E+02	1.46E+02
f17	3.44E+02	1.24E+02	**3.29E+02**	1.42E+02	3.51E+02	1.30E+02
f18	9.27E+02	2.53E+01	**9.21E+02**	3.79E+01	9.29E+02	2.51E+01
f19	9.30E+02	2.55E+01	**9.26E+02**	3.25E+01	9.26E+02	3.15E+01
f20	9.28E+02	2.53E+01	**9.19E+02**	3.68E+01	9.26E+02	2.49E+01
f21	9.91E+02	2.56E+02	1.03E+03	2.66E+02	**9.87E+02**	2.49E+02
f22	**1.31E+03**	2.58E+01	1.31E+03	2.87E+01	1.31E+03	3.47E+01
f23	9.82E+02	1.95E+02	9.91E+02	1.98E+02	**9.39E+02**	1.22E+02
f24	5.28E+02	2.48E+02	**4.72E+02**	5.88E+01	4.81E+02	1.45E+02
f25	4.77E+02	4.04E+01	**4.72E+02**	5.13E-01	4.78E+02	4.03E+01

Best	8		16		6	
Worst	9		9		12	

GIDN Statistically Better			1		2	
GIDN Statistically Worse			3		2	
GIDN Statistically Equal			28		28	

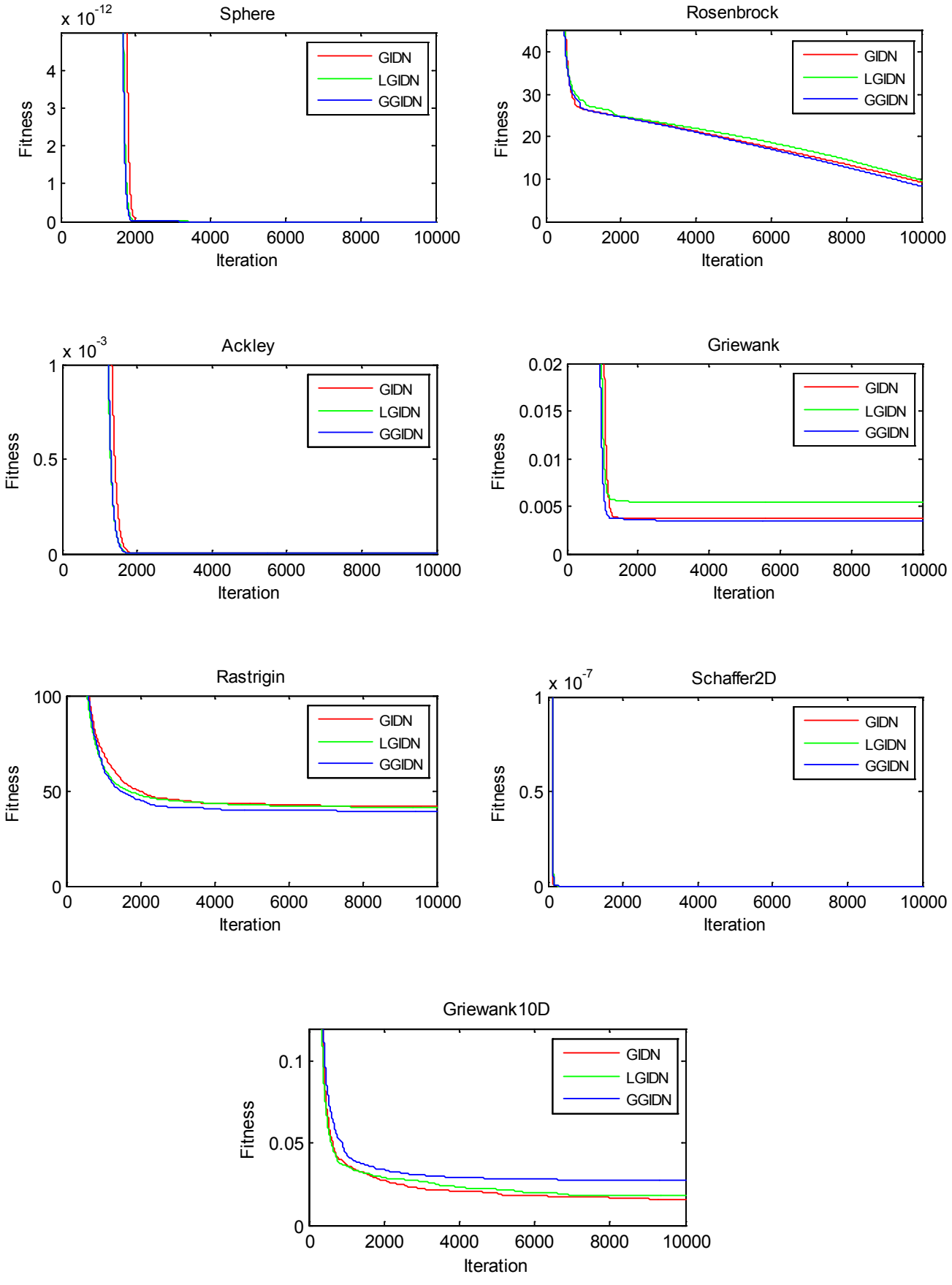


FIGURE 7.5: Controlled APL GIDN variations - Convergence

Linear GIDN can be considered the worst performing of all three variations. It performed best on 6 functions and worse on 12 which makes it the best performing least often and worst performing most often. The Linear GIDN performed similarly to GIDN PSO when compared statistically as it was worse on 2 of the later functions and better on 2 of the earlier functions.

Considering all three algorithms, the newly created GGIDN algorithm is the strongest as it performs the best most often and worst least often. It is the best on 16 of the functions and the worst on only 9. The GGIDN algorithm performs better on 18 functions and worse on 11 functions compared to the original GIDN PSO in isolation. When analysed statistically, GGIDN performs better on 3 and worse on 1 function compared to GIDN PSO. Notably, GGIDN performs best on 7 of the more complex Hybrid Composition functions whereas GIDN only performs best on 4 of these functions. This could imply that the GGIDN is better able to avoid local optima in more complex multimodal functions.

7.4 Variation Discussion

From the results presented it can be seen that the variations implemented affect the optimization performance across the 32 benchmark functions.
The Linear GIDN algorithm is statistically better on 2 and worse on 2 functions compared to the original GIDN algorithm. This implies that the alteration to produce a linear decrease in the average path length leads to it being slightly more inclined towards exploration, as it is better on more complex multimodal problems, but sacrifices performance on two simpler unimodal functions as a result of less exploitation time.
The Gradual GIDN is more interesting for the purpose of analysis as it attempts to replicate the operation of GIDN but without the stepped decreases in the average path length plot. This means that the new variation follows a similar path on the average path length plot as GIDN but does so in a gradual manner. Statistically, this change increases the performance across three functions and only worsens on a single unimodal function when the Wilcoxon test is performed between it and GIDN PSO. GGIDN performed better than GIDN on 18 of the 32 benchmark functions and equal on 3. More interestingly, GGIDN performed the

best on seven of the eleven Hybrid Composition functions compared to GIDN, which only performed the best on four. These later functions are much more difficult to optimize given they are comprised of a number of different functions [75]. This implies that the increased control over the average path length parameter in the Gradual GIDN variation produces a much more robust algorithm than the previously suggested GIDN PSO.

Also worth examining are the graphs for the average number of disconnected paths during LGIDN and GGIDN, shown in Figure 7.2 and 7.4 respectively. Both plots are much more gradual in nature than the equivalent stepped plot produced by GIDN, shown in Figure 5.3. This is as a direct result of the more gradual addition of particles to neighbourhoods and very much resembles the changes seen in the average path length plots. Despite its altered shape, the disconnected path length plot for GGIDN reaches a 0 value at a similar iteration number to GIDN PSO, around 4000 iterations. This is due to the GGIDN algorithm using the same underlying network update equation (Equation 5.1). The fact that particles are added gradually does not greatly affect the point at which all disconnected paths are removed.
The plot for the average number of disconnected paths in LGIDN is different to both other variations when analysed with respect to the iteration at which the metric reaches a zero value. It occurs at a much later stage, usually just below 6000 iterations. The LGIDN algorithm also maintains a higher average path length at this later stage in the algorithms operation compared to both other variations. Thus, as would be expected, the higher average number of disconnected paths is correlated with a higher average path length at the later stages of optimization. This suggests that the LGIDN variation is exploring more at a later later stage which could be contributing to its negative performance.

7.5 Summary

The aim of this chapter was to investigate what effect the addition of particles to produce a certain average path length plot could have on the optimization capabilities of the swarm. Two variations were implemented that attempted to remove the stepped pattern first noticed in Chapter 5 from the average path

length plot of GIDN PSO. This was an attempt to achieve a more gradual transition from exploration to exploitation.

The first variation altered the original GIDN PSO algorithm to perform with a linearly decreasing average path length from the initial value to the minimum average path length possible. The second variation sought to produce a similar pattern to that created by the original GIDN implementation without the stepped decreases in average path length. This was achieved by creating stretches of linear descents with non-zero slopes between predetermined points on the original GIDN PSO average path length plots. The changes made in these variations are examples of very simple developments to existing algorithms that begin to control dynamic parameters in the network rather than simply observing them. The results achieved were presented and compared in Table 7.1. It was shown that the alteration of the average path length characteristic can produce positive and negative effects on the optimization capabilities of the swarm. Most notably, the Gradual GIDN algorithm produced better results than GIDN PSO on a majority of functions. This proved that the gradual transition from exploration to exploitation over the course of optimization can produce improved optimization results.

Chapter 8

Conclusions

This final chapter will review the research carried out during this thesis with a particular focus on its relevance to the research questions posed in Chapter 1. The applicability of the work along with an overview of potential future work will also be detailed.

8.1 Summary of Work

The main aim of this work was to investigate the application of graph theory to PSO with dynamic topologies. The intent was that a better insight into their improved operation [17] would lead to the discovery of further areas of improvement. It was found that the application of average path length to the swarm topology was an effective means of describing the spread of information with the network [18]. This was used as a direct corollary for the swarms tendency to perform early exploration followed by later exploitation [9]. The performance of existing PSO variants with dynamic topologies were altered in a variety of ways to observe the effect on performance. An attempt was made to control the average path length over the course of the optimization with the intent of improving performance.

A simulator was designed and developed in Java for the purposes of experimentation. It allowed for numerous PSO variations to be easily implemented and included a set of 32 benchmark functions [75] which were to be optimized.

Initially, a PSO variation with a dynamic topology, GIDN PSO [17], was evaluated on the benchmark functions alongside two common static topology implementations, *gBest*[16] and *lBest* [2]. The results were compared using the Wilicoxon signed rank test which revealed the benefits of dynamic topologies over their static counterparts to be statistically significant. The application of graph theory to the network created by the swarm topology was then explained. Through this it was discovered that the average path length plot in GIDN decreases in a step like manner. This discovery became the foundation for later research.

It was also discovered that the random addition of initial particles to neighbourhoods in GIDN PSO resulted in a number of disconnected paths between particles. This had the equivalent effect of reducing the speed of information transfer in the swarm as some particles had no means of communicating with others, even through intermediary particles. In order to determine whether this contributed to the favourable performance of GIDN PSO two new variations were created. Given two initial particles, the *Sphere GIDN* (SGIDN) created an initial sphere topology and the *Connected GIDN* (CGIDN) linked particles in a circular manner and randomly assigned the second particle. Both of these variations had no disconnected paths between particles because of their initial topologies guaranteed connectivity [49]. Over the 32 benchmark functions neither performed favourably. The SGIDN performed much worse than the original GIDN PSO. It is believed that this is as a result of the initial average path length being too high. The maximum average path length when every particle has two neighbours is achieved using a sphere topology [49]. The sustained nature of an initially high average path length seems to cause the swarm to spend too much time exploring which forfeits exploitation. This was supported by the fact that CGIDN had a much lower initial average path length compared to SGIDN and performed very similarly to the original GIDN PSO. Consequently, given the CGIDN variation produced a connected graph it can be extrapolated that disconnected paths in the original GIDN variation are not the exclusive reason for its good performance, but rather one means of reducing the amount of information shared among particles in the swarm.

The final research question sought to control the average path length during optimization to improve upon the performance of GIDN PSO. Through earlier

experimentation during this research it was found that the average path length plot for GIDN PSO decreased in a stepped manner. This is as a result of certain network topologies being sustained for a number of iterations followed by an increase in the size of every particles neighbourhood simultaneously. Two variations were designed to reduce the average path length gradually over the course of optimization. The *Linear GIDN* (LGIDN) saw a linear decrease from the initial average path length value to the minimum possible average path length value of 1. The *Gradual GIDN* (GGIDN) variant created short linear paths between desired values calculated using the predetermined drop points on the average path length plot of the original GIDN variation. Results favoured the new variation over GIDN PSO on 18 of the 32 benchmark functions. This proves that controlling the average path length to produce a more gradual descent from exploration to exploitation can lead to improved results.
The insights gained through the application of graph theory to PSO with dynamic topologies ultimately lead to improved performance over state of the art PSO variants [17].

8.2 Research Questions

Three research questions to be answered were posed in Chapter 1 of this work. From the experiments conducted and analysis applied these questions have consequently been answered:

1. The key reason PSO variations with dynamic topologies perform better than those with static topologies was determined as the ability to balance early exploration and later convergence during optimization [9]. Graph theory was successfully applied to the GIDN PSO through the use of the average path length parameter to effectively describe this transition.

2. Investigations into the effect of the initial topology on the performance of PSO with dynamic topologies found that a high average path length can lead to an overage of exploration in the early stages of the algorithm which worsens overall performance.

3. It was also proven that control over swarm neighbourhoods during optimization can produce a desired average path length plot. Certain plots can be created to augment the performance of PSO.

The research presented in this thesis proves that the application of the average path length parameter provides a quantifiable insight into the operation of PSO with dynamic topologies. Through this, the reasons for an improved performance of PSO variations with dynamic topologies are highlighted. By analysing the average path length plots created by different variants with dynamic topologies, the reasons for alternate performance are now better understood.
Controlling the average path length parameter also led to an improvement of the GIDN PSO algorithm, most notable on the more complex multimodal functions. There is an extra computational cost required to recalculate the average path length across the entire swarm after every neighbourhood update. This cost becomes less relevant as the problem being optimized becomes more demanding to evaluate. Therefore the trade-off between increased time to convergence due to added computational effort may be negligible when considered against the more difficult functions on which the proposed variations performed best. The improvement achieved is problem specific and its requirement is largely application dependent.

8.3 State of the Art

One of the factors most fundamental to the favourable performance of PSO is the unique interactions between particles in the swarm [3]. These interactions are decided by the swarm topology. Given a desire to transition from early exploration to later exploitation [9] it is desirable for the swarm to have the ability to facilitate this over the course of optimization.
Dynamic topologies are thus a very promising area of research within PSO [17]. In order to be able to understand the differences in performance of variants with dynamic topologies, it is necessary to be able to describe and thus compare them. This work provides a much deeper insight into the operation of dynamic topologies. Applying these new methods revealed numerically the exact reasons

for good performance in GIDN, which up until now has been somewhat unknown. The performance of the state of the art GIDN was improved upon using the data obtained.

8.4 Impact

As described in Chapter 2, heuristic optimization algorithms are necessary to solve complex problems with to large a search space to be searched fully. They are crucial to providing a reasonable alternative to an exact solution [19]. These types of problems exist in a wide range of real world applications such as imaging [12], portfolio management [13], power generation [14] and neural network training [15].

It has been proved by earlier research that the use of dynamic topologies can provide vast improvements in performance over their static counterparts [17]. Until now the complexity or operation of these variants has not been properly understood or depicted. This work introduces a novel means of explaining some of the underlying operation of dynamic topologies. The use of graph theory adds quantifiable measures to the key factors in the operation of dynamic topologies, thus providing a means of comparison beyond simple convergence values.

8.5 Reflection

The primary goal of this research was to apply graph theory to better understand the operation of PSO with dynamic topologies, by providing a deeper insight into the tendencies of swarms to move from early exploration to later convergence.

This has been achieved by analysing the average path length of the swarm graph over the course of the optimization. It has long been known that the topology dictates whether the swarm is exploring or converging [16]. Changing the topology over time allows the ratio of exploration to exploitation to be adjusted appropriately over time [17]. This work provided a quantifiable insight into this ratio, thus explaining one of the most fundamental attributes of PSO with dynamic topologies.

Interesting results were produced owing to this new insight. It was found that

the initial topology of the swarm has an effect on its performance and that too high an initial average path length can result in too much initial exploration. It also showed that by gradually decreasing the average path length, rather than using a stepped function as in GIDN the performance could be improved upon, particularly on the more complex benchmark functions.

8.6 Future Work

This thesis introduces a novel means of analysing dynamic topologies to better understand the transition of a swarm from exploration to exploitation. Its application to PSO with dynamic topologies revealed numerous insights into the characteristics that affect performance. It is recommended that the average path length parameter plot be applied when analysing all dynamic topologies to act as a descriptive means of comparison beyond simple convergence values for variations of this type. It is hoped that further research will reveal the most suitable values to achieve the optimal balance between exploration and exploitation.
Graph theory is a broad area of study that is only briefly explored in this work. There exists many other measures that could be applied to describe PSO swarm graphs. The application of these measures in further research may uncover traits relating to the operation of PSO which could be used to enhance its performance, as is done in this work.

Regarding initial topologies, it seems that the number of connections and initial structure greatly impact the performance of PSO. This is just one example indicating how the performance of PSO is heavily dependent on the choice of parameters. Further comprehensive research into the initial topology may indicate the optimal values based on insights gleaned through this work.

This paper mainly focuses on the application of graph theory with the intent of better understanding the operation of PSO. Only briefly is an attempt made to improve upon existing implementations of dynamic topologies. It would be hoped that further research will be conducted in an attempt to produce better results using the insights gained from this thesis. This may include but is certainly not limited to changing the initial topology or shape of the average path length plot.

Furthermore, all variations created based on controlling the average path length plots attempt to match a predefined structure in a static manner. Benefits could be achieved by controlling the spread of information dynamically based on the instantaneous performance of the swarm.

It would also be of interest to see the Gradual GIDN variation applied to a real world optimization problem. Given its improved performance on the more complex functions it is predicted that it would perform equally positively on a broad range of real-world problems.

Bibliography

[1] S. Xu and Y. Rahmat-Samii. Boundary conditions in particle swarm optimization revisited. *IEEE Transactions on Antennas and Propagation*, 55 (3):760–765, March 2007. ISSN 0018-926X. doi: 10.1109/TAP.2007.891562.

[2] D. Bratton and J. Kennedy. Defining a standard for particle swarm optimization. In *2007 IEEE Swarm Intelligence Symposium*, pages 120–127, April 2007. doi: 10.1109/SIS.2007.368035.

[3] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4, Nov 1995. doi: 10.1109/ICNN.1995.488968.

[4] A. Engelbrecht. Particle swarm optimization: Velocity initialization. In *2012 IEEE Congress on Evolutionary Computation*, pages 1–8, June 2012. doi: 10.1109/CEC.2012.6256112.

[5] Y. Shi and R. C. Eberhart. Empirical study of particle swarm optimization. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, page 1950 Vol. 3, 1999. doi: 10.1109/CEC.1999.785511.

[6] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, Feb 2002. ISSN 1089-778X. doi: 10.1109/4235.985692.

[7] S. Helwig and R. Wanka. *Theoretical Analysis of Initial Particle Swarm Behavior*, pages 889–898. Springer Berlin Heidelberg, Berlin, Heidelberg,

2008. ISBN 978-3-540-87700-4. doi: 10.1007/978-3-540-87700-4_88. URL http://dx.doi.org/10.1007/978-3-540-87700-4_88.

[8] M. A. Montes de Oca and T. Stützle. Convergence behavior of the fully informed particle swarm optimization algorithm. In *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pages 71–78. ACM, 2008.

[9] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on*, volume 2, pages 1671–1676, 2002. doi: 10.1109/CEC.2002.1004493.

[10] T. Blackwell and J. Branke. Multi-swarm optimization in dynamic environments. In *Applications of Evolutionary Computing, EvoWorkshops 2004: EvoBIO, EvoCOMNET, EvoHOT, EvoIASP, EvoMUSART, and EvoSTOC, Coimbra, Portugal, April 5-7, 2004, Proceedings*, pages 489–500, 2004. doi: 10.1007/978-3-540-24653-4_50. URL http://dx.doi.org/10.1007/978-3-540-24653-4_50.

[11] K. Mason and E. Howley. *Avoidance Strategies in Particle Swarm Optimisation*, pages 3–15. Springer International Publishing, Cham, 2015. ISBN 978-3-319-19824-8. doi: 10.1007/978-3-319-19824-8_1. URL http://dx.doi.org/10.1007/978-3-319-19824-8_1.

[12] E. Marami and A. Tefas. Using particle swarm optimization for scaling and rotation invariant face detection. In *IEEE Congress on Evolutionary Computation*, pages 1–7, July 2010. doi: 10.1109/CEC.2010.5586159.

[13] S. G. Reid, K. M. Malan, and A. P. Engelbrecht. Carry trade portfolio optimization using particle swarm optimization. In *2014 IEEE Congress on Evolutionary Computation (CEC)*, pages 3051–3058, July 2014. doi: 10.1109/CEC.2014.6900497.

[14] M. A. Abido. Multiobjective particle swarm for environmental/economic dispatch problem. In *2007 International Power Engineering Conference (IPEC 2007)*, pages 1385–1390, Dec 2007.

[15] S. M. Elsayed, R. A. Sarker, and E. Mezura-Montes. Particle swarm optimizer for constrained optimization. In *2013 IEEE Congress on Evolutionary Computation*, pages 2703–2711, June 2013. doi: 10.1109/CEC.2013.6557896.

[16] R. Eberhart and J. Kennedy. A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on*, pages 39–43, Oct 1995. doi: 10.1109/MHS.1995.494215.

[17] H. Liu, E. Howely, and J. Duggan. Particle swarm optimisation with gradually increasing directed neighbourhoods. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation*, GECCO '11, pages 29–36, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0557-0. doi: 10.1145/2001576.2001582. URL `http://doi.acm.org/10.1145/2001576.2001582`.

[18] C. Yen, Mi. Yeh, and M. Chen. An efficient approach to updating closeness centrality and average path length in dynamic networks. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on*, pages 867–876. IEEE, 2013.

[19] X. Yang. *Nature-inspired metaheuristic algorithms*. Luniver press, 2010.

[20] E. Biondi and P. C. Palermo. *A heuristic approach to combinatorial optimization problems*, pages 460–470. Springer Berlin Heidelberg, Berlin, Heidelberg, 1973. ISBN 978-3-540-37903-4. doi: 10.1007/3-540-06583-0_45. URL `http://dx.doi.org/10.1007/3-540-06583-0_45`.

[21] G. T. Pulido and Coello A. C. Coello. A constraint-handling mechanism for particle swarm optimization. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1396–1403 Vol.2, June 2004. doi: 10.1109/CEC.2004.1331060.

[22] J. Sun, V. Palade, X. J. Wu, W. Fang, and Z. Wang. Solving the power economic dispatch problem with generator constraints by random drift particle swarm optimization. *IEEE Transactions on Industrial Informatics*, 10 (1):222–232, Feb 2014. ISSN 1551-3203.

[23] U. Paquet and A. P. Engelbrecht. A new particle swarm optimiser for linearly constrained optimisation. In *Evolutionary Computation, 2003. CEC '03. The 2003 Congress on*, volume 1, pages 227–233 Vol.1, Dec 2003. doi: 10.1109/CEC.2003.1299579.

[24] M. Setayesh, M. Zhang, and M. Johnston. Edge detection using constrained discrete particle swarm optimisation in noisy images. In *2011 IEEE Congress of Evolutionary Computation (CEC)*, pages 246–253, June 2011. doi: 10.1109/CEC.2011.5949625.

[25] B. A. Garro, Humberto Sossa, and Roberto A. Vázquez. *Evolving Neural Networks: A Comparison between Differential Evolution and Particle Swarm Optimization*, pages 447–454. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-21515-5. doi: 10.1007/978-3-642-21515-5_53. URL `http://dx.doi.org/10.1007/978-3-642-21515-5_53`.

[26] Z. Michalewicz and M. Schoenauer. Evolutionary algorithms for constrained parameter optimization problems. *Evol. Comput.*, 4(1):1–32, March 1996. ISSN 1063-6560. doi: 10.1162/evco.1996.4.1.1. URL `http://dx.doi.org/10.1162/evco.1996.4.1.1`.

[27] B. Xue, M. Zhang, and W. N. Browne. Multi-objective particle swarm optimisation (pso) for feature selection. In *Proceedings of the 14th Annual Conference on Genetic and Evolutionary Computation*, GECCO '12, pages 81–88, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1177-9. doi: 10.1145/2330163.2330175. URL `http://doi.acm.org/10.1145/2330163.2330175`.

[28] Z. Zhan and J. Zhang. An parallel particle swarm optimization approach for multiobjective optimization problems. In *Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation*, GECCO '10, pages 81–82, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0072-8. doi: 10.1145/1830483.1830497. URL `http://doi.acm.org/10.1145/1830483.1830497`.

[29] M. R. AlRashidi and M. E. El-hawary. Emission-economic dispatch using a novel constraint handling particle swarm optimization strategy. In *2006 Canadian Conference on Electrical and Computer Engineering*, pages 664–669, May 2006. doi: 10.1109/CCECE.2006.277592.

[30] Z. Bo and C. Yi-jia. Multiple objective particle swarm optimization technique for economic load dispatch. *Journal of Zhejiang University-SCIENCE A*, 6(5):420–427, 2005. ISSN 1862-1775. doi: 10.1007/BF02839410. URL `http://dx.doi.org/10.1007/BF02839410`.

[31] M Reyes-Sierra and C.A. Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, 2(3):287–308, 2006.

[32] C. A. Coello Coello, G. T. Pulido, and M. S. Lechuga. Handling multiple objectives with particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, 8(3):256–279, June 2004. ISSN 1089-778X. doi: 10.1109/TEVC.2004.826067.

[33] J. Barrera and C. A Coello Coello. A review of particle swarm optimization methods used for multimodal optimization. In *Innovations in swarm intelligence*, pages 9–37. Springer, 2009.

[34] J. Kennedy and W. M. Spears. Matching algorithms to problems: An experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC*, pages 78–83, 1998.

[35] G. Lapizco-Encinas, C. Kingsford, and J. Reggia. Particle swarm optimization for multimodal combinatorial problems and its application to protein design. In *IEEE Congress on Evolutionary Computation*, pages 1–8, July 2010. doi: 10.1109/CEC.2010.5586157.

[36] J. Fliege and R. Werner. Robust multiobjective optimization amp; applications in portfolio optimization. *European Journal of Operational Research*, 234(2):422 – 433, 2014. ISSN 0377-2217. doi: http://dx.doi.org/10.1016/j.ejor.2013.10.028. URL `http://www.sciencedirect.com/science/`

article/pii/S0377221713008515. 60 years following Harry Markowitzs contribution to portfolio theory and operations research.

[37] P. B. Mullen, C. K. Monson, K. D. Seppi, and S. C. Warnick. Particle swarm optimization in dynamic pricing. In *2006 IEEE International Conference on Evolutionary Computation*, pages 1232–1239, 2006. doi: 10.1109/CEC.2006.1688450.

[38] Y. Marinakis and M. Marinaki. Combinatorial expanding neighborhood topology particle swarm optimization for the vehicle routing problem with stochastic demands. In *Proceedings of the 15th Annual Conference on Genetic and Evolutionary Computation*, GECCO '13, pages 49–56, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1963-8. doi: 10.1145/2463372.2463375. URL http://doi.acm.org/10.1145/2463372.2463375.

[39] L. Steels. When are robots intelligent autonomous agents? *Robotics and Autonomous Systems*, 15(1):3 – 9, 1995. ISSN 0921-8890. doi: http://dx.doi.org/10.1016/0921-8890(95)00011-4. URL http://www.sciencedirect.com/science/article/pii/0921889095000114.

[40] C. A. Coello Coello. *An Introduction to Evolutionary Algorithms and Their Applications*, pages 425–442. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31674-9. doi: 10.1007/11533962_39. URL http://dx.doi.org/10.1007/11533962_39.

[41] A. P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons, 2006. ISBN 0470091916.

[42] C. Peng, M. Anbo, and Z. Chunhua. Particle swarm optimization in multi-agent system for the intelligent generation of test papers. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 2158–2162, June 2008. doi: 10.1109/CEC.2008.4631085.

[43] D. Karaboga and Bahriye Basturk. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

doi: 10.1007/s10898-007-9149-x. URL http://dx.doi.org/10.1007/s10898-007-9149-x.

[44] R. C. Eberhart and Y. Shi. *Comparison between genetic algorithms and particle swarm optimization*, pages 611–616. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-68515-9. doi: 10.1007/BFb0040812. URL http://dx.doi.org/10.1007/BFb0040812.

[45] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4 (2):65–85, 1994. ISSN 1573-1375. doi: 10.1007/BF00175354. URL http://dx.doi.org/10.1007/BF00175354.

[46] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, Feb 1996. ISSN 1083-4419. doi: 10.1109/3477.484436.

[47] R. J. Wilson and J. J. Watkins. *Graphs: an introductory approach: a first course in discrete mathematics*. Wiley New York, 1990.

[48] F. Harary. *Graph Theory*. Addison-Wesley, Reading, MA, 1969.

[49] D. J. Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 1999.

[50] J. Bang-Jensen and G. Gutin. Theory, algorithms and applications. 2007.

[51] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. *The ubiquity of chaos*, pages 233–238, 1990.

[52] M. M. Millonas. Swarms, phase transitions, and collective intelligence. Technical report, Los Alamos National Lab., NM (United States), 1992.

[53] Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pages 69–73, May 1998. doi: 10.1109/ICEC.1998.699146.

[54] A. El-Gallad, M. El-Hawary, A. Sallam, and A. Kalas. Enhancing the particle swarm optimizer via proper parameters selection. In *Electrical and Computer Engineering, 2002. IEEE CCECE 2002. Canadian Conference on*, volume 2, pages 792–797 vol.2, 2002. doi: 10.1109/CCECE.2002.1013043.

[55] Z. Li-ping, Z. Huan-jun, and H. Shang-xu. Optimal choice of parameters for particle swarm optimization. *Journal of Zhejiang University-SCIENCE A*, 6(6):528–534, 2005. ISSN 1862-1775.

[56] R.E. Bellman. *Dynamic Programming*. Dover Books on Computer Science Series. Dover Publications, 2003. ISBN 9780486428093. URL `https://books.google.ie/books?id=fyVtp3EMxasC`.

[57] T. Hendtlass. Particle swarm optimisation and high dimensional problem spaces. In *2009 IEEE Congress on Evolutionary Computation*, pages 1988–1994, May 2009. doi: 10.1109/CEC.2009.4983184.

[58] A. P. Engelbrecht. Roaming behavior of unconstrained particles. In *2013 BRICS Congress on Computational Intelligence and 11th Brazilian Congress on Computational Intelligence*, pages 104–111, Sept 2013. doi: 10.1109/BRICS-CCI-CBIC.2013.28.

[59] S. Helwig and R. Wanka. Particle swarm optimization in high-dimensional bounded search spaces. In *2007 IEEE Swarm Intelligence Symposium*, pages 198–205, April 2007. doi: 10.1109/SIS.2007.368046.

[60] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, page 1938 Vol. 3, 1999. doi: 10.1109/CEC.1999.785509.

[61] A. E. Mu noz Zavala, A. Hernández Aguirre, and E. R. Villa Diharce. The singly-linked ring topology for the particle swarm optimization algorithm. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation*, GECCO '09, pages 65–72, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-325-9. doi: 10.1145/1569901.1569911. URL `http://doi.acm.org/10.1145/1569901.1569911`.

[62] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8 (3):204–210, June 2004. ISSN 1089-778X. doi: 10.1109/TEVC.2004.826074.

[63] I. Broderick and E. Howley. *Particle Swarm Optimisation with Enhanced Memory Particles*, pages 254–261. Springer International Publishing, Cham, 2014. ISBN 978-3-319-09952-1. doi: 10.1007/978-3-319-09952-1_24. URL `http://dx.doi.org/10.1007/978-3-319-09952-1_24`.

[64] C. Yang and D. Simon. A new particle swarm optimization technique. In *18th International Conference on Systems Engineering (ICSEng'05)*, pages 164–169, Aug 2005. doi: 10.1109/ICSENG.2005.9.

[65] T. Jayabarathi, R. T. Kolipakula, M. V. Krishna, and A. Yazdani. Application and comparison of pso, its variants and hde techniques to emission/economic dispatch. *Arabian Journal for Science and Engineering*, 39 (2):967–976, 2014. ISSN 2191-4281.

[66] C. K. Chow and S. Y. Yuen. A non-revisiting particle swarm optimization. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 1879–1885, June 2008. doi: 10.1109/CEC.2008.4631045.

[67] T. M. Blackwell and P. J. Bentley. Dynamic search with charged swarms. In *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, New York, USA, 9-13 July 2002*, pages 19–26, 2002.

[68] David H Wolpert and William G Macready. No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82, 1997.

[69] S. Lahmiri and M. Boukadoum. An evaluation of particle swarm optimization techniques in segmentation of biomedical images. In *Proceedings of the Companion Publication of the 2014 Annual Conference on Genetic and Evolutionary Computation*, GECCO Comp '14, pages 1313–1320, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2881-4. doi: 10.1145/2598394.2609855. URL `http://doi.acm.org/10.1145/2598394.2609855`.

[70] S. Wang. *Artificial Neural Network*, pages 81–100. Springer US, Boston, MA, 2003. ISBN 978-1-4615-0377-4. doi: 10.1007/978-1-4615-0377-4_5. URL http://dx.doi.org/10.1007/978-1-4615-0377-4_5.

[71] H. Markowitz. Portfolio selection. *Journal of Finance*, 7(1):77 – 91. ISSN 00221082.

[72] J. Wang, W. Chen, J. Zhang, and Y. Lin. A dimension-decreasing particle swarm optimization method for portfolio optimization. In *Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO Companion '15, pages 1515–1516, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3488-4. doi: 10.1145/2739482.2764652. URL http://doi.acm.org/10.1145/2739482.2764652.

[73] M. Basu. Dynamic economic emission dispatch using nondominated sorting genetic algorithm-ii. *International Journal of Electrical Power Energy Systems*, 30(2):140 – 149, 2008. ISSN 0142-0615. doi: http://dx.doi.org/10.1016/j.ijepes.2007.06.009. URL http://www.sciencedirect.com/science/article/pii/S0142061507000877.

[74] P. J. Angeline. *Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences*, pages 601–610. Springer Berlin Heidelberg, Berlin, Heidelberg, 1998. ISBN 978-3-540-68515-9. doi: 10.1007/BFb0040811. URL http://dx.doi.org/10.1007/BFb0040811.

[75] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. 2005.

[76] R. L. Haupt and S. E. Haupt. *Practical genetic algorithms*. John Wiley & Sons, 2004.

[77] F. Wilcoxon. Individual comparisons by ranking methods. In *Breakthroughs in Statistics*, pages 196–202. Springer, 1992.

[78] L. Gulyás, G. Horváth, T. Cséri, and G. Kampis. An estimation of the shortest and largest average path length in graphs of given density. *arXiv preprint arXiv:1101.2549*, 2011.

[79] G. Chartrand, L. Lesniak, and P. Zhang. *Graphs & digraphs*, volume 39. CRC Press, 2010.