Information Sharing Impact of Stochastic Diffusion Search on Population-Based Algorithms

Mohammad Majid Oudah al-Rifaie

Thesis submitted for the degree of Doctor of Philosophy

of the

University of London

Department of Computing, Goldsmiths College

January 2012

Declaration

I, Mohammad Majid al-Rifaie, declare that the work presented in this thesis is entirely my own.

Signature:

Date:

Abstract

This work introduces a generalised hybridisation strategy which utilises the information sharing mechanism deployed in Stochastic Diffusion Search when applied to a number of population-based algorithms, effectively merging this nature-inspired algorithm with some population-based algorithms. The results reported herein demonstrate that the hybrid algorithm, exploiting information-sharing within the population, improves the optimisation capability of some well-known optimising algorithms, including Particle Swarm Optimisation, Differential Evolution algorithm and Genetic Algorithm. This hybridisation strategy adds the information exchange mechanism of Stochastic Diffusion Search to any population-based algorithm without having to change the implementation of the algorithm used, making the integration process easy to adopt and evaluate. Additionally, in this work, Stochastic Diffusion Search has also been deployed as a global optimisation algorithm, and the optimisation capability of two newly introduced minimised variants of Particle Swarm algorithms is investigated.

Acknowledgements

Not one or two but many to thank for their direct and indirect contribution towards the crystallisation of this work, some of whom I have not seen for half a decade but sensed their encouraging presence throughout. My utmost love and gratitude goes to my parents in the Middle East and my siblings all around the world!

My greatest appreciation and thanks go to my supervisor, Prof. Mark Bishop, for all his generous support throughout these years. Although, while doing this course, life (politically and socially) has not been gentle, aiming to facilitate a premature convergence to local optima, his insight kept illuminating the global optimum, irrespective of whether or not this has been achieved by the author!

Further special thanks go to my second supervisor, Dr. Tim Blackwell, whose enthusiasm and frankness were essential factors in completing this work.

I would also like to thank my colleagues and the staff at the Computing Department, whose support I always had on my side.

At last but not least, I would like to name and thank all my *friends*, for which I would need double the size of this thesis. Without their support (e.g. Sacher Torte, pizzas, spices, cigars, authentic Middle Eastern meals, Greek tea, Sagrada Família tour at night, steak at the riverside of Bristol, traditional music travelling from overseas, their kind words, etc.), it would not have been feasible to reach the following dot.

Contents

List of Algorithms

List of Figures

List of Tables

List of Abbreviations

AA	Ant Algorithms
ACO	Ant Colony Optimisation
AI	Artificial Intelligence
AL	Artificial Life
API	Pachycondyla Apicalis Ant
CA	Cellular Automata
DAI	Distributed Artificial Intelligence
DE	Differential Evolution
EA	Evolutionary Algorithm
EC	Evolutionary Computation
EP	Evolutionary Programming
ES	Evolution Strategies
FE	Function Evaluation
GA	Genetic Algorithm
GO	Global Optimisation
GP	Genetic Programming
LSDS	Lattice SDS

MG Mining Game

PSO Particle Swarm Optimisation

PSO-BB Bare Bones PSO

PSO-BBJ1 Bare Bones with Jumps PSO Model 1

PSO-BBJ2 Bare Bones with Jumps PSO Model 2

PSO-CK Clerc-Kennedy PSO

SDS Stocahstic Diffusion Search

SI Swarm Intelligence

SS Search Space

TS Tabu Search

Chapter 1

Introduction

"Though this be madness, yet there is method in 't."
– William Shakespeare

This study originally intended to investigate the possible integration of two nature-inspired algorithms (i.e. Stochastic Diffusion Search and Particle Swarm Optimisation), which resulted in the development of a novel hybridisation strategy utilised for a larger variety of population-based algorithms.

The use of population-based algorithms for Global Optimisation (GO) is not uncommon within both commercial and academic fields, and their goal is to find better solutions for complex problems.

An 'everyday' example of optimisation is the process through which a decision is made on where to park a car; in this scenario, different parameters are likely to be considered and the best (optimal) choice might be made with regard to the following: the distance of the parking lot from the current location of the car, the suitability of the parking lot and the duration in which the car could be left parked.

In optimisation, candidate solutions are contrasted against each other with the intention of finding the optimal solution. Swarm intelligence and evolutionary algorithms are shown to be of significance in solving optimisation

problems. These algorithms are usually evaluated through commonly used benchmarks that are typically small in terms of their objective functions' computational costs [1, 2] (which is often not the case in many real-world applications). This justifies the initial motivation behind utilising Stochastic Diffusion Search, whose partial objective function evaluation technique alleviates the problem of having costly objective functions (see section 3.2.4 on page 57).

1.1 Objectives and Methodology

The core of this thesis seeks to investigate the possible integration strategies of Stochastic Diffusion Search (SDS) [3], with other population-based algorithms (e.g. Particle Swarm Optimisation (PSO) [4], Differential Evolution algorithm (DE) and Genetic Algorithm (GA)). As a result, a generalised hybridisation strategy is proposed, which introduces these population-based algorithms with another form of information exchange.

The work presented in this thesis investigates the following key research topics:

1. Deploying Stochastic Diffusion Search in the context of Global Optimisation
2. The effect of introducing restart mechanism in the context of two minimised variants of Particle Swarm Optimisation algorithm
3. The information-sharing impact of Stochastic Diffusion Search on other population-based algorithms and proposing a generalised hybridisation strategy for generic use with population-based algorithms

The first research topic addresses the deployment of Stochastic Diffusion Search as a Global Optimiser. Standard SDS has been used in discrete environments; therefore, in order to utilise the information sharing mechanism of SDS in the context of global optimisation, the algorithm is modified and

then run for a number of iterations, followed by a local search. The outcome of the experiments conducted on this topic demonstrates the optimisation outperformance resulted by using this approach.

Having presented PSO with different parameters and variations, Bare Bones PSO is explained as a minimised variant of standard PSO in order to investigate the second research topic. Two new minimised variants of PSO algorithms are then introduced followed by a set of experiments. The results demonstrate the positive effects of the restart mechanisms leading to the improvements in the optimisation capability of the conventional PSOs.

In order to address the third research topic, which is the main focus of this study, standard SDS is introduced alongside a few examples on how the algorithm functions. Then few population-based algorithms (PSO, GA and DE) are presented. After presenting these population-based algorithms, merging SDS with PSO is first investigated. Following the promising results of this integration, SDS is integrated with another population-based algorithm (DE). Afterwards, a generalised hybridisation strategy is introduced and a larger set of algorithms as well as a harder set of benchmarks are used to test this hybridisation strategy. The results achieved show the outperformance of the hybrid algorithms over their standard counterparts. It is also shown that using this strategy, SDS can be integrated with any population-based algorithm.

1.2 Chapter Overview

Chapter 2 provides a review of Artificial Intelligence (AI), Swarm Intelligence (SI) and optimisation. It begins with a brief account of AI (section 2.1), presenting two main schools of thought in the field (Connectionist and symbolic AI) and highlighting their pros and cons. An introduction to multi-agent systems which links AI to SI is accompanied by a background to swarm intelligence, communication in social insects and their methods of interaction (Section 2.2). A connection is made between the social behaviour of insects/animals and the swarm intelligence algorithms used in this study. Emphasis

will be placed on information exchange in swarm intelligence to demonstrate how information flow affects the behaviour of the swarm. This follows a brief discussion on optimisation, highlighting the relation between optimisation and search, the concept of global optimisation as well as evolutionary optimisation and its subcategories (Section 2.3).

The next chapter (Chapter 3) constitutes a review of Stochastic Diffusion Search (SDS). In this chapter, a social metaphor is used to describe the algorithm, and then the architecture of SDS is explained along with few examples on how SDS works. Different variations of the algorithm, including information-sharing strategies (recruitment or gossipping), are outlined, followed by a list of applications which have used SDS, both in the research community as well as in industry.

Chapter 4 presents few population-based optimisers. Section 4.1 presents Particle Swarm Optimisation (PSO) which has attracted many researchers due to its applicability as well as its simple structure and easy-to-implement nature. A description of the algorithm is followed by a discussion of different parameter changes and their effects on improving the performance of standard PSO algorithm. The significance of random-restart mechanism will be highlighted and the role of cooperative PSO in enhancing the algorithm will then be presented. Sections 4.2 and 4.3 present simple variants of Genetic Algorithm and Differential Evolution algorithm. In this chapter, PSO is explained in detail (more detailed than DE and GA), as it was originally studied with the intention of finding an integration strategy that could merge this algorithm with SDS (see Chapter 7).

Chapter 5 builds an initial set of experiments aiming to investigate a scenario where SDS is utilised as a global optimiser. In this chapter, the modified SDS algorithm (used for global optimisation) is run, followed by a local search. The modified algorithm is tested over a set of benchmarks and results which demonstrate improvement are reported.

Chapter 6.2 presents a minimised version of the PSO algorithm (i.e. Bare Bones PSO) and then introduces two variations of the Bare Bones PSO (Bare Bones with Jumps Models 1 & 2). This chapter comes to end by presenting a

set of experiments, comparing the performance of a number of PSO variants over a set of standard benchmarks, demonstrating the outperformance of the newly introduced algorithms – Bare-Bones with Jumps Models 1 & 2.

Chapter 7 and Chapter 8 are the focus of the study. In chapter 7, an initial investigation of the possibility of integrating SDS with PSO is explored. A similar approach is developed for integrating SDS with DE, followed by a set of experiments. Chapter 8 uses the ideas introduced in Chapter 7 on the possible integration strategies to propose a generalised hybridisation strategy that is applicable to any algorithm that is classified as population-based. The generalised hybridisation strategy is subsequently tested on a more recent set of benchmarks (other than those used in Chapter 7). This is followed by a discussion on the performance of the hybrid algorithms.

Finally, Chapter 9 provides a summary of the study as well as recommendations for future research. The appendices present a list of publications which were derived from or influenced by this work, as well as additional materials referred to in the report.

Chapter 2

Artificial Intelligence and Swarm Intelligence

"Painting is only a bridge linking the painter's mind with that of the viewer."
– Eugène Delacroix

This chapter presents a brief overview over Artificial Intelligence (AI), giving few definitions of the term followed by a background to two main schools of thought in the field (Connectionist and symbolic AI). Afterwards, an introduction to multi-agent approach to AI is given, linking AI to Swarm Intelligence (SI). Next, a background to swarm intelligence, communication in social insects and their methods of interaction is presented, followed by a brief discussion on the connection between the social behaviour of insects/animals and the swarm intelligence algorithms. This follows a brief discussion on optimisation, highlighting the relation between optimisation and search, the concept of global optimisation as well as evolutionary optimisation and its subcategories

2.1 Artificial Intelligence

For centuries, philosophers have been trying to formalise the human being as *Homo sapiens sapiens* – man the wise. They have been interested in the way these "wise creatures" can possibly draw valid conclusions as well as in the way knowledge leads to an action. By the same token, Artificial Intelligence (AI), a term coined in 1956 by John McCarthy,[1], has been interpreted in a variety of ways. Russell et al. [6] have suggested a categorisation of some of the definitions (systems that think or act like human, or systems that think or act rationally):

- Thinking like humans [7]: "The exciting new effort to make computers think ... machines with minds, in the full and literal sense."
- Acting like humans [8]: "The art of creating machines that perform functions that require intelligence when performed by people."
- Thinking rationally [9]: "The study of mental faculties through the use of computational models."
- Acting rationally [10]: "Computational Intelligence is the study of the design of intelligent agents."

Based on the above categorisation, the controversial Turing Test, devised by Alan Turing [11], proposed to "provide a satisfactory operational definition of intelligence" [6] to check whether a system acts in a human-like way by evaluating its responses to natural language text input. Typically this would involve: natural language processing, knowledge representation, automated reasoning, machine learning (as well as computer vision and robotics if passing the 'Total Turing Test' [2] is the goal).

As for thinking human-like, cognitive science offers means to investigate that premise. Whether a system thinks rationally or not is decided upon by means

[1] Although some might find this controversial, McCarthy in a c|net interview states: "I came up with the term". (see [5], p. 50))

[2] Ibid

of logic as laws of thought. For example, for a system to act rationally, a 'rational agent' approach maybe used (whereby rationality implies acting in a way that achieves the best result, or when there is no such possibility, the best possible result).

Early in the 19th century, William Paley had argued that creating complex adaptive systems requires intelligent designers (read Spector in [12]). However, the idea was challenged by Charles Darwin in 1859 by demonstrating that complex and adaptive systems can be created without an intelligent designer through the evolutionary processes. Still, Spector argues that most AI researchers "view AI as a set of design problems that human designers are expected to solve", whilst he also emphasises that they should be interested in *evolved* artificial intelligence.

The section begins by introducing the Turing Test; although perhaps controversial, the Turing Test has remained relevant six decades after its emergence (see [13] for a recent Kybernetes special issue on the Turing Test). Whether a system is genuinely 'intelligent' if it passes the test, remains debatable, but this question has not yet undermined the significance of the Turing Test and its role in the field of artificial intelligence.

2.1.1 Connectionist vs. Symbolic AI

The two major classical schools of thought in Artificial Intelligence are Symbolic AI (or representational AI) and Connectionism. While the former is thought to be more committed to a symbol level of representation (a state that combines syntactic and semantic structure [14]), the latter is considered to provide a closer account to the neural structure backed by different groups including those who believe it best to replace serial machines with massively parallel ones, the fans of neuroscience-leads-to-understanding-cognition club, psychologists who do not like the idea of seeing mind as a discrete system, etc.

According to Pollack [15], the term connectionism is used when neurally-inspired mechanisms are utilised to study computation and cognitive mod-

elling. Connectionism, which appeared in the attempt to mathematically model intelligence based on what was known about the architecture of brain, is usually seen as a rival to symbolic artificial intelligence [16].

In 1943, when Warren S. McCullogh and Walter Pitts proved the possibility of implementing any logical expression by using an appropriate structure of simplified neurons (in *A logical calculus of ideas immanent in nervous activity* [17]), they "formally" commenced research in the field of artificial neural networks [15]. As part of this research, they introduced the first sequential logic model of neuron [18]. Neurons were assumed to be binary with finite threshold where signals sent from one neuron can be excitatory (increasing firing rate) or inhibitory (decreasing firing rate).

Later, Hebb, in his 1949 *the Organisation of Behaviour* [19], was the first to add psychology, mainly based on Stimulus-Response, to the new field of neural networks. He is credited for the following statements [15]:

- memory is stored in connections
- learning takes place in synaptic modification

In his work [19], Hebb states:

> "Let us assume then that the persistence or repetition of a reverberatory activity (or "trace") tends to induce lasting cellular changes that add to its stability. The assumption can be precisely stated as follows: When an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased."

Hebbian learning is believed to be an important mechanism in the "tuning of neuronal connections during development and thereafter" [20]. This mechanism argues that simultaneous activation of cells increases the synaptic strength between these cells. As suggested by Doidge [21], this can be summarised in simple terms as "Cells that fire together, wire together".

Ashby, in his 1960 *Design for a Brain* [22], defined brain as an adaptive system for "*develop[ing] adaptation in its behaviour*" and considered adaptability as a key element in artificial systems.

In 1962, Rosenblatt presented the first simulatable and analysable neurally-inspired model of synaptic modification in *Principle of Neurodynamics* [23], which he called *a perceptron*. In the perceptron, instead of using neurons with fixed weights and threshold and absolute inhibition, some weights and threshold are variable and the inhibition is relative. The weights from the input (a "retina" of binary inputs) to the middle layer (feature detectors) are fixed and depend on the application. The adjustable weights are the ones between the middle layer to the response unit. Presenting procedures for adjusting these variable weights on different perceptron implementation, alongside proving the convergence of these procedures in finite time, has been Rosenblatt's key work [15].

Following what was seen as the too ambitious goals of researchers in the field, Minskey and Pepert, in their 1969 book, Perceptrons [24], took research in perceptrons and neurally-inspired modelling to a decade of hibernation, by highlighting its limitations as a general computational device; they showed that for a single-layer perceptron, it was impossible to learn an XOR function, which is an example showing the inability of single-layer perceptrons in learning linearly non-separable patterns; in a two dimensional space, two sets of points are linearly separable if they can be separated by one line (e.g. NOT, AND and OR functions are linearly separable). This principle is extendable to n-dimensional space. If no such line (or hyperplane, in case of n-dimensional space where $n > 2$) exists, the functions are linearly non-separable.

Within the field of AI in the 60s, according to Steels in [25], there was an emphasis on task specific rules of thumb (heuristics[3]) to take the problem solver at the vicinity of reasonable solutions as quickly as possible. This focus on heuristics, resulted in calling the field of AI 'Heuristic Search' for some

[3]In situations where greedy search (or exhaustive search) is impractical, heuristic methods are used to speed up the process of locating a satisfactory solution.

time.

In addition to the rule of heuristics, the significance of knowledge representation became clear towards the end of the 60s. Many ideas from logic poured into AI. The applicability of heuristics and knowledge proved to be important through the emergence of the first wave of expert systems, which are computer systems emulating the decision-making ability of human experts [26]; some examples of expert systems are the following:

- DENDRAL [27], which was introduced to investigate hypothesis formation for making new findings, for which a test is selected. The main of this test was to help organic chemists identify unknown organic molecules, by analysing their components and using knowledge of chemistry. DENDRAL project which started in 1965 is considered "one of the first large-scale programs to embody the strategy of using detailed, task-specific knowledge about a problem domain as a source of heuristics, and to seek generality through automating the acquisition of such knowledge." [28]

- MYCIN, which is developed in early 1970s to detect bacteria causing harmful infections (e.g. bacteremia and meningitis). This expert system recommends antibiotics, with the dosage specified to suit patient's body weight. The naming of MYCIN comes from the antibiotics themselves, as many of them have the suffix "-mycin" [29].

- PROSPECTOR [30], which is introduced in the 1970s, is proposed as a consultation system that assists geologists working in mineral exploration. This system attempts to represent the knowledge and reasoning processes of experts in the geological domain.

In the 1970s, research in the area of connectionist models did not appear widely in journals, until re-emergence in the 80s (when symbolic systems started showing their limits; the issue of parallel computers became more important and relevant, etc.).

Thereafter, research in two aspects of AI continued developing in parallel. Although through complex networks with hidden layers, neural networks made significant advances, symbolic AI did not lose its importance. Steels in [25][4] wrote:

> "... so far no adequate neural models have been proposed for language understanding, planning and areas in which symbolic AI excels."

From the early 80s, fundamental research decelerated and pragmatic research on applications took over. In 1982, Feldman and Ballard published *Connectionist Models and their Properties* [31] which gave connectionism a framework as a possible methodology for cognitive science and artificial intelligence. They justified their argument to adapt connectionism, rather than symbolic AI, relying on four points:

- First, the structure of the brain is different from that of computers.
- Second, the time issue:
  "The critical resource that is most obvious is time. Neurons whose basic computational speed is a few milliseconds must be made to account for complex behaviors which are carried out in a few hundred milliseconds. This means that entire complex behaviors are carried out in less than a hundred time steps. Current AI and simulation programs require millions of time steps."
- Third, studying connectionism helps give ideas on how to do parallel computing.
- Fourth, they believed that studying connectionism might lead to "better science" and that understanding many mechanisms behind intelligent behaviours (e.g. associative memory and the remarkable recovery ability of animals) is not possible under symbolic AI paradigm.

[4] pp. 23

The rebirth of connectionism attracted contributions from areas other than computer science and psychology (e.g. physics). Hopfield's significant contribution (the Hopfield net) was the introduction of a system for programming a model of associative memory by considering each memory as a local minimum for a global energy function [32]. The energy (E) of a network is calculated using the following formula:

$$E = -\tfrac{1}{2}\sum_{ij} w_{ij} s_i s_j + \sum_i \theta_i s_i$$

where w_{ij} is the weight of the connection between unit i and j; s_i is the state of i and $s_i \in \{0, 1\}$; θ_i is the threshold of unit i. In Hopfield networks, units are not reflexive (no connection to the self or $w_{ii} = 0$), and connections are symmetric: $\{\forall\ i, j;\ w_{ij} = w_{ji}\}$.

Additionally, back-propagation, another technique for learning in multi-layer artificial neural networks, was independently suggested by several researchers (e.g. Parker [33], Werbos [34] who developed it in his 1974 mathematics thesis and Yann Le Cun [35]). At the core of back-propagation technique is the use of a continuous activation function that allows weights to change slowly without resulting in major disturbances [15].

Connectionism is known for its contributions in facilitating massively parallel processing, machine learning and graceful degradation. In connectionism, in contrary to symbolic AI, a system is less likely to fail completely when one of its components fails, but continues to operate with reduced performance. However numerous limitations are also attributed to connectionism; among the major problems associated with it is the lack of representational adequacy (and especially the lack of compositionality [14]). An example is given in [15]:

> "... if the entire feature system is needed to represent a single element, then attempting to represent a structure involving those elements cannot be managed in the same system. For example, if all the features are needed to represent a *Nurse*, and all the features are needed to represent an *Elephant*, then the attempt to represent a *Nurse riding an elephant* will come out either as a *white elephant* or a rather *large nurse with four legs*."

Another major problem is the necessity to apply dramatic changes (to the weights) to the neural network in order to allow the system to deal with a similar problem with slightly different features or when re-sizing the scale of the problem (e.g. adding a city to the travelling salesman network [36] would require changing the configuration of the whole network).

Several limitations of symbolic AI against connectionism are discussed in [14] where the "lure of connectionism" is emphasised (e.g. computers/symbolic processings are too rapid compared to the neural speed; computers are rule governed; computers are sensitive to damage and noise and etc). These were seen as a few reasons behind the popularity of connectionism and its rise among psychologists and philosophers.

Although human cognition is or used to be the main measure to compare against machine intelligence, some researchers (e.g. Luc Steels in [25]) believe that this kind of comparison could only lead to disappointment due to the huge distance between the two at present.

However, amongst many fields in AI, the following probably benefited the most from progress made [18]:

- Formal representation techniques (logic, rules, frames, agents, causal networks, etc.)
- Treatment of uncertainty (Bayesian networks, fuzzy systems)
- Dealing with situations where there are more data than knowledge (Artificial Neural Networks)

2.1.2 Multi-Agent Systems

Despite the relative success of applications in both symbolic AI and connectionism towards the end of 80s, the body and environment as major causal forces in shaping intelligent behaviour started to be considered [37] and the necessity of having a smoother real-time behaviour by the *agent-environment*

interaction (instead of complicated calculations for motor control) became more vivid (see [25], pp.23).

In the early 90s, the idea of *emergent* behaviour (through interaction) and behaviour-based AI dominated AI laboratories [38] and building animal-like robots began once again. The new focus on the interaction between AI and biology led to the emergence of the new area of Artificial Life [39].

A multi-agent approach to AI was born in the 90s, when cooperation between agents became essential to have an *emerging* intelligence resulting from the interaction of a group of individuals [25]. It was time for sociologists and anthropologists to play their rules in helping AI with their models and social views on intelligence [40].

Looking at the historical perspective of AI research, the field's trajectory (see [12]) is observable from emphasising on complex mental faculties to focus on building complete, situated and embodied agents, which are more natural forms of intelligence.

Chaib-Draa *et al.* in their 1992 paper [41] argued that the inevitable existence of a number of agents in the real world makes a single agent approach insufficient. They mentioned four main reasons behind the importance of what is called Distributed Artificial Intelligence (DAI):

1. the need to deal with distributed knowledge for geographically remote applications like air-traffic control and cooperation between robots

2. extending man-machine cooperation, using a distributed resolution approach

3. bringing about a new perspective to knowledge representation and problem solving

4. changing our understanding of artificial intelligence [42] (the emphasis, however, was on modularity) as it may shed new light on the way cognitive science is perceived

As emphasised in [25], in today's densely interconnected world, there is an apparent global trend towards "collective phenomena in the information processing world"; considering the exponential growth of blogs, social networking websites, P2P sharing systems and wikis, it is clear that the traditional centralised top-down decision making and the definition of a universal ontology (defined by experts and imposed on users) are no longer desired approaches.

2.2 Swarm Intelligence

Swarm Intelligence (SI) which investigates collective intelligence, aims at modelling intelligence by looking at individuals in a social context and monitoring their interactions with one another as well as their interactions with the environment [43]. Natural examples of swarm intelligence that exhibit these forms of interaction include fish schooling, birds flocking, ant colonies in nesting and foraging, bacterial growth, animal herding, brood sorting by ants, etc.

Therefore, swarm intelligence can be characterised as the communications between agents as well as the communication of agents with the environment while expecting an emergent phenomenon (intelligence). In [44], communication between agents or social interaction is considered to result in a more human like intelligence:

> "Evaluating, comparing, and imitating one another, learning from experience and emulating the successful behaviors of others, people are able to adapt to complex environments through the discovery of relatively optimal patterns of attitudes, beliefs, and behaviors. Our species' predilection for a certain kind of social interaction has resulted in the development of the inherent intelligence of humans."

The story of the *blind men and the elephant* also suggests how social interaction can possibly lead to human intelligence. This famous tale set in verse

by John Godfrey Saxe [45] in the 19th century, characterises six blind men approaching an elephant. They ended up having six different ideas about the elephant, as each person experienced one aspect of the elephant's body: wall (elephant's side), spear (tusk), snake (trunk), tree (knee), fan (ear) and rope (tail). To read the whole tale, see Appendix B on page 188.

The moral of the story is that people build their beliefs based on incomplete beliefs derived from incomplete knowledge about the world [44]. If the blind men had been talking/listening to each other and exchanging information about what they were experiencing, they would have possibly come up with the conclusion that they were exploring the heterogeneous qualities that make up an elephant.

2.2.1 Swarm Intelligence in Nature

Communication – social interaction or information exchange – observed in social insects and animals is important in swarm intelligence. As stated in [44], in real social interactions, not just the syntactical information (i.e. contents) is exchanged between individuals but also semantic rules, tips and beliefs about how to process this information; in typical swarm intelligence algorithms, however, only the syntactical exchange of information is considered, without necessarily changing the thinking process (e.g. rules and beliefs) of the participants.

In the study of the interaction of social insects, two important elements are the individuals and the environment, which lead to two integration schemes: the first one is the way in which individuals self-interact and the second one is the interaction of the individuals with the environment [46] (stigmergy). Self-interaction between individuals is carried out through recruitment and it has been demonstrated that there are various recruitment strategies in ants [47] and honey bees [48, 49]. These recruitment strategies are used to attract other members of the society to gather around one or more desired areas, either for foraging purposes or for moving to a new nest site. In animals like fish or birds, self-interaction results in benefiting from discoveries and

previous experience of all other members of the school of flock during search for food ([50], p.209).

There are different forms of recruitment in social insects; it may take the form of local or global; one-to-one or one-to-many; and deploy stochastic or deterministic mechanisms. The nature of information exchange also varies in different environments and with different types of social insects and animals.

Sometimes the information exchange is more complex where, for example, it might carry data about the direction, suitability of the target and the distance; or sometimes the information sharing is simply a stimulation forcing a certain triggered action. What all these recruitment and information exchange strategies have in common is distributing useful information in their community [51].

Next, different forms of information exchange in some social insects and animals are discussed in further detail.

2.2.1.1 Communication in Ants and Bees

Chemical communication through pheromones forms the primary method of recruitment in ants. However in one species of ants, *Leptothorax acervorum*, where a 'tandem calling' mechanism (one-to-one communication) is used, the forager ant that finds the food location recruits a single ant upon its return to the nest, and therefore the location of the food is physically publicised [52]. In group recruitment, an ant convenes a larger number of ants, leading them to the food location. Laying the pheromone trail from the food source to the nest is of more advanced nature, in which the leading ant is not physically in contact with other ants. The most advanced form of ant recruitment is mass recruitment [53] in which the worker ants follow the pheromone trail, but individual ants add an amount of pheromones alongside their journey towards the food location. Therefore, the amount of pheromones plays an important role in the outflow attraction of the ants.

In another primitive ant species where nest replacement is studied [54], an ant with a better nest location, summons an ant with a poorer choice. In

this approach, Pachycondyla Apicalis Ant (API), ants are all called to the best nest found so far and subsequently start exploring the area again for a better nest location. Different types of recruitment and communication strategies induce different performances. Ants communicating through group recruitment are faster than tandem calling ants, and similarly, ants utilising mass recruitment are more efficient in their performances than the former recruitment strategies [53].

However, as mentioned in [55], the success of the ants in reaching the food they have been recruited to obtain varies from one species to another. In indirect or stigmergetic communication, the exchange of information is based on modifying the physical properties of the environment and its success lies in spatial and temporal attributes of mass recruitment and the positive feedback mechanism it employs. In this mode, which is based on using pheromone, short routes are loaded with more pheromone (because of the short time it takes the ants to travel [56]).

In honey bees, group recruitment is performed by means of waggle dances, in which the direction of the dance shows the location of the food source and the speed of the dance represents the distance to the target area. Each bee chooses one of the dancing bees as a guide to the food source.

2.2.1.2 Flocking, Schooling and Herding

There have been many efforts to formalise the movements of animals herding, fish schooling and birds flocking to (for instance) create computer simulations of these behaviours. Although birds are discrete units, their motions in general exhibit a fluid-like magnificently synchronised movement [57]. For the fish to school, they need to preserve two main requirements, staying close to the flock as well as avoiding collision with other fish [58]. Natural flocks do not get overloaded with new members joining; neither do they get unstable with a few members leaving [59]. Through observing flocks in nature, it seems that they have what is called *constant time algorithm*, which means birds, fish, animals can flock, school and herd respectively, irrespective of their populations [57].

Although, as stated in [60], a member of a flock does not seem to have full attention of every other members of the flock, at the same time, the awareness of each individual member of the flock has been categorised at three levels: self-awareness, awareness towards neighbours and awareness towards the flock.

2.2.2 Swarm Intelligence Algorithms

In recent years, studies of the behaviour of social insects and animals have suggested several new meta-heuristics for use in collective intelligence. This has given rise to a concomitant increasing interest in distributed computation through the interaction of simple agents in nature-inspired optimisation algorithms; among these are:

- Evolutionary Algorithms (EA) [61]: Genetic Algorithm (GA, which many believe to be the most popular type of evolutionary algorithms) [62, 63], Evolutionary Programming (EP, which was initially created to evolve finite state machine) and Evolution Strategies (ES, which originally aimed to solve difficult discrete and continuous parameter optimisation problems)

- Swarm intelligence algorithms: Ant Algorithms (AA, based on the idea of pheromone communication of ants) [64, 65], which were successfully applied to combinatorial optimisation problems [66] such as the travelling salesman problem[5] [67, 68, 64, 69] and the quadratic assignment problem[6] [70, 71], Particle Swarm Optimisation (PSO) [4], which was the result of an attempt to graphically simulate the choreography of

[5]Travelling salesman problem (TSP) is one of the combinatorial optimisation problems where candidate solutions are discrete or can be reduced to discrete ones. In this problem, given a number of cities and the distances between each pair, the goal is to find the shortest tour to pass by each city just once.

[6]In quadratic assignment problem (QAP), which is considered one of the fundamental combinatorial optimisation problems, given n nodes and n services, and having the distances between each pair of nodes as well as weights between each pair of services (e.g. communication loads), the goal is to allocate the services to different nodes in a way to minimise the product of the distances with the weights.

fish schooling or birds flying, and Stochastic Diffusion Search (SDS) [3], which is inspired by one species of ants, *Leptothorax acervorum*, etc.

Although some writers (e.g. [72, 22]) blur the difference between adaptation and intelligence by claiming that intelligence is actually the ability to adapt, other writers in the field of swarm intelligence (e.g. [44]) stress that an individual is not an isolated information processing entity.

Stochastic diffusion search and particle swarm optimisation algorithms, which function by interaction between agents, adopt the second view and share some characteristics and behaviours of swarms intelligence algorithms which can be best understood by observing the behaviours of social insects such as ants and bees in locating food sources and nest site locations or the behaviours of social animals like birds flocking and fish schooling.

According to Millonas [73], the basic principles of swarm intelligence are the following:

- Proximity: ability of the population to do simple space and time computation
- Quality: ability of the population to recognise and respond to quality factors in the environment
- Diverse response: the activity of the population should not be carried out along excessively narrow channels
- Stability: the population should not be over-sensitive to the changes in the environment
- Adaptability: the population should be able to change behaviour if it is computationally beneficial

As highlighted in [44], the last two principles are the opposite sides of the same coin.

Some of the swarm intelligence algorithms have been successfully deployed by the author in the field of computational creativity. In one such artwork by

the author, the swarm intelligence algorithms are assisted by a mechanism inspired from the behaviour of skeletal muscles activated by motor neurons [74] and in a different study, the mechanism of blood flow and blood vessels is used alongside the swarm intelligence algorithms [75]. In ongoing work and in an invited journal paper [76], we raise the question of whether integrating swarm intelligence algorithms (inspired by social systems in nature) could possibly lead to a novel way of producing 'artworks' and whether the swarms demonstrate computational creativity in a non-representational way.

2.3 Optimisation

2.3.1 Optimisation and Search

In swarm intelligence literature, search and optimisation are often used interchangeably. Nevertheless, the definition of search has been categorised in three broad types in [77]:

- In the first definition, search refers to finding a (target) model in a search space, and the goal of the algorithm is to find a match, or the closest match to the target in the search space. This is defined as *data search* and is considered a classical meaning of search in computer science [78].
- In the second type, the goal is finding a path (*path search*) and the list of the steps leading to a certain solution is what the search algorithm tries to achieve. In this type of search, paths do not exist explicitly but are rather created during the course of the search.
- In the third definition, *solution search*, the goal is to find a solution in a large problem space of candidate solutions. Similarly to the path search, where paths do not exist explicitly, the search space consists of candidate solutions which are not stored explicitly but are rather created and evaluated during the search process. However, in contrast

to the path search, the steps taken to find the solution are not the goal of the algorithm.

In optimisation, which is similar to the third definition of search, the model of the first definition is replaced with an objective or fitness function which is used to evaluate possible solutions. In both search and optimisation, the positions of the optima are not known in advance (even though the optima itself might be known a-priori). The task of the fitness function is to measure the proximity of the candidate solutions to the optima based on the criteria provided by each optimisation problem. The algorithm compares the output of the function to the output of the previously located candidate solutions and, in the case of a minimisation problem, the smaller the output the better the solution. Data search can be seen as a caste of optimisation if the objective function tests the equality of the candidate solution to the model.

2.3.2 Global Optimisation

Global Optimisation (GA) is concerned with locating the optimal solution within the entire search space and one of the main difficulties that global optimisers face, is the existence of local optima within the problem space.

According to [79], global optimisation techniques are categorised into four groups:

- Incomplete: This technique uses clever intuitive heuristics for searching without presenting safeguards if the search gets stuck in a local minimum.
- Asymptotically complete: This technique reaches a global minimum with certainty or at least with probability one with the assumption of allowing to run indefinitely long, without providing means to know when a global minimum has been found.
- Complete: This technique reaches a global minimum with certainty, with the assumption of having exact computations and indefinitely long

run time, and knows after a finite time that an approximate global minimum has been found (within prescribed tolerances).

- Rigorous: This technique reaches a global minimum with certainty and within given tolerances even in the presence of rounding errors, except in near-degenerate cases where the tolerances may be exceeded.

Most of the population-based algorithms which do not guarantee an optimal global solution (while capable of escaping a local minimum in some cases) are defined as *incomplete* global optimisers. However, solely searching parts of the search space and using the knowledge obtained to update the potential solutions based on their heuristic rules allows them to be faster than other methods.

2.3.3 Evolutionary Optimisation

Evolutionary optimisation is an application of the Evolutionary Computation technique (EC). Evolutionary algorithms are population-based and although derived from the idea of the survival of the fittest and natural selection, they can be further refined in the following three categories [80]:

- genetic algorithms (GA)[7]
- evolutionary programming (EP)
- evolution strategies (ES) .

Some of the similarities between these methods are listed below (for more details refer to Kennedy et al. [44] p.143):

- initialisation of the population
- using fitness function as a way to evaluate the quality of each member of the population

[7] with links to genetic programming (GP)

- deploying evolutionary operations (e.g. mutation, crossover and selection) in each generation
- producing the offspring population from the parent population

Each of the categories in evolutionary algorithms is briefly introduced in the next section.

Genetic Algorithms

Genetic Algorithms, introduced by John Holland [81, 62, 82] in the early 1970s, were originally proposed as a general model of adaptive processes, but, as mentioned by De Jong [83, 84], the largest application of the method is in the sphere of optimisation. As stated by [80], the same applies to the other aforementioned techniques.

As a population-based algorithm, a Genetic Algorithm starts with a set of solutions, each represented by a chromosome, and the number of solutions (population size) is fixed throughout each generation. During each generation, the fitness value of each chromosome is evaluated and the evolutionary operators (e.g. mutation, crossover and selection) are used to produce the population of the next generation (offspring). A simple genetic algorithm that is used in this work is presented in Section 4.2 on page 93.

Evolutionary Programming

Evolutionary programming, introduced by Fogel [85, 86], originally aimed to evolve finite state machine. One of the main differences between genetic algorithms and evolutionary programming is the lack of recombination (or crossover) in the later. The main operator in evolutionary programming is mutation which is applied randomly, using uniform probability distributions. Fogel [72] stated that evolutionary programming takes a fundamentally different approach to genetic algorithms:

> "The procedure abstracts evolution as a top-down process of adaptive behavior, rather than a bottom-up process of adaptive genetics. It is argued that this approach is more appropriate because natural selection does not act on individual components in isolation, but rather on the complete set of expressed behaviors of an organism in light of its interaction with its environment."

He considers that evolutionary programming implements "survival of the more skillful" rather than the "survival of the fittest" which is emphasised by genetic algorithm researchers.

Among other variants, the real-valued optimisers of the algorithm function by applying Gaussian mutations to solution vectors[8], whose performance could be enhanced by using a Cauchy-distributed mutation. This variant, Fast Evolutionary Programming (FEP) [87], uses the fatter tails of the Cauchy distribution which allows larger mutations to escape from local minima.

Evolution Strategies

Evolution strategies, introduced by Rechenberg [88, 89] and Schwefel [90, 91], originally intended to solve difficult discrete and continuous parameter optimisation problems. Although mutation plays a primary role in evolution strategy, recombination is used as a secondary update operator.

As stated in Kennedy *et al.* [44], evolution strategy is based upon the evolution of evolution:

> "If evolutionary programming is based on evolution, then, reasons Rechenberg [89], the field of evolution strategies is based upon the evolution of evolution. Since biological processes have been optimized by evolution, and evolution is a biological process, then evolution must have optimized itself."

Evolution strategy has two common selection mechanisms, namely (μ, λ) and $(\mu + \lambda)$:

[8] ibid.

- In (μ, λ), μ current individuals are used to generate λ offspring and the μ best ones among the λ generated offspring form the new population.
- In $(\mu + \lambda)$, μ current individuals are used to generate λ offspring; then combine μ parents with λ offspring and pick the μ best ones to form a new population. There is a similarity between this method and a form of elitism in GA where the best parent is kept for the next generation.

Differential Evolution (DE) algorithm, a global optimisation method, is similar to GA, but is usually classified as an evolution strategy algorithm. DE iterates through the evolutionary process of mutation, crossover and selection as explained in more detail in Section 4.3.

2.4 Summary

This chapter gives an overview on artificial intelligence and the ways it is viewed as well as an introduction to two classical schools of thought in artificial intelligence (symbolic AI and connectionism) and a brief historical account on their existence from late 40s. Next in the chapter, swarm intelligence, which aims at modelling intelligence by looking at individuals in a social context, is briefly discussed, followed by some examples of communication in social insects/animals and swarm intelligence algorithms. Finally, an introduction to optimisation is given, presenting different types and definitions for search and optimisation in the literature, followed by a summary of evolutionary optimisation algorithms (and its subcategories).

Chapter 3

Stochastic Diffusion Search

"All the world's a stage and all the men and women merely players; they have their exits and their entrances; and one man in his time plays many parts..."
– William Shakespeare

This chapter surveys SDS, a multi-agent global search and optimisation algorithm, which is based on simple interaction of agents. A high-level description of SDS in the form of a social metaphor is also presented, followed by a simple search example demonstrating the procedures through which SDS conducts the search. The architecture and development of SDS are then discussed in greater detail. In addition to analysing the behaviour of SDS and the possibility of embedding different interaction strategies, the novel way SDS deals with computational costly objective functions is investigated. The chapter concludes by discussing issues related to applications of SDS.

Stochastic Diffusion Search (SDS) [3] introduced a new probabilistic approach for solving best-fit pattern recognition and matching problems. As a multi-agent population-based global search and optimisation algorithm, SDS is a distributed mode of computation utilising interaction between simple agents [92].

Unlike many nature inspired search algorithms, SDS has a strong mathematical framework, which describes the behaviour of the algorithm by investigating its resource allocation [93], convergence to global optimum [94], robustness and minimal convergence criteria [95] and linear time complexity [96]. In order to introduce SDS, a social metaphor called *the Mining Game*[1] (MG [97]) is used.

3.1 The Mining Game

This metaphor provides a simple high-level description of the behaviour of agents in SDS, where a mountain range is divided into hills and each hill is divided into regions:

> A group of miners learn that there is gold to be found on the hills of a mountain range but have no information regarding its distribution. To maximise their collective wealth, the maximum number of miners should dig at the hill where the concentration of gold is highest; this information is not available a-priori. Thus the goal of the resource allocation process is to allocate the most miners to the hill which has the richest seams of gold. In order to solve this problem, the miners decide to employ a simple Stochastic Diffusion Search.
>
> At the start of the mining process each miner is randomly given a hill to mine (his hypothesis, h). Every day each miner mines at a randomly selected region on the hill. At the end of each day, the probability that a miner is happy is proportional to the amount of gold he has mined. Each evening the miners congregate and each miner who is not happy selects another miner at random for communication. If the chosen miner is happy, they share the

[1] The Mining Game simulator is available online at the following address. In this simulation, many of the practical aspects of SDS covered in this chapter can be explored: http://www.arcofbeing.com/miningame/

location of the gold and thus maintain it as their hypothesis, h; if not, the unhappy miner selects a new region to mine at random.

As this process is isomorphic to a probabilistic formulation of SDS, miners will naturally self-organise to congregate over hill(s) of the mountain with high concentration of gold.

Algorithm 3.1 The Mining Game

```
01: Initialisation phase
02: Allocate each miner (agent) to a random
03:   hill (hypothesis) to pick a region randomly
04:
05: Until (miners congregate over the highest
06:    concentration of gold)
07:
08:    Test phase
09:       Each miner evaluates the amount of gold
10:         they have mined (hypotheses evaluation)
11:       Miners are classified into happy (active)
12:         and unhappy (inactive) groups
13:
14:    Diffusion phase
15:       Unhappy miners consider a new hill by
16:         either communicating with another miner
17:         or,if the selected miner is also
18:         unhappy, there will be no information
19:         flow between the miners; instead the
20:         selecting miner must consider another
21:         hill (new hypothesis) at random
22: End
```

3.1.1 Refinements in the Metaphor

There are some refinements to the mining game analogy, which elaborate further more on the correlation between the metaphor and different implementations of the algorithm.

The happiness of the miners can be measured probabilistically or gold may be considered as an absolute unit. In both cases all the miners are either happy or unhappy at the end of each day; this represents standard SDS. SDS can be further refined through either of the following two assumptions:

1. Finite resources: the amount of gold is reduced each time a miner mines the area

2. Infinite resources: the imaginary situation of the existence of infinite amounts of gold

In the case of having finite resources, the analogy can be related to a real world experiment of robots looking for food to carry along to the nest [98]. Therefore the amount of food (or gold, in the mining analogy) is reduced after each discovery. In that experiment the following are investigated:

- an ant-like algorithm is used to avoid robots interfering with one another (knowledge about overall colony energy)
- considering individual variation in performing the task
- recruiting other robots when identifying a rich area

In this case, the goal is to identify the location of the resources throughout the search space. This is similar to conducting a search in a dynamically, agent-initiated changing environment where agents change their congregation from one area to another.

The second assumption has similarities with discrete function optimisation where values at certain points are evaluated. However further re-evaluation of the same points does not change their values as they remain constant.

The above is similar to an older metaphor of the Restaurant Game [51] used in the former SDS literature where each customer could choose a meal from a menu at a specific restaurant and it would be possible to try the same meal again in the restaurant[2]. In this case, the purpose of the algorithm is converging over the richest area rather than collecting the resources. Therefore, this mode can be considered as a caste of the finite resources mode.

[2]There is however a pitfall in this metaphor and therefore it is replaced with the Mining Game. The pitfall alongside the metaphor itself is illustrated in Appendix C on page 190.

3.2 SDS Architecture

An SDS algorithm commences the search or optimisation by initialising its population (e.g. miners, in the mining game metaphor) followed by the iteration of two phases (for the high-level SDS description see Algorithm 3.2 and also see Algorithm 3.1 for the test and diffusion phases in the mining game):

- the Test Phase (e.g. testing gold availability)
- and the Diffusion Phase (e.g. congregation and exchanging of information)

In the test phase, the objective function of SDS checks whether the agent is successful (happy) or not and it always returns a Boolean value. Later in the iteration, in the diffusion phase, if the objective function returns a positive result, the hypothesis (e.g. location of the hill, in the mining game) of the successful agent is diffused. Therefore, the information of potentially good solutions spreads throughout the entire population of agents.

In SDS, a function is not evaluated in full (in the same way that a miner in the mining game does not dig all the regions of a hill). This *partial evaluation* strategy of SDS helps escaping local minima (see Section 3.2.4 on page 57 for more detail) and helps to improve algorithm efficiency.

Next, SDS is illustrated using two search examples in details.

Algorithm 3.2 SDS Algorithm

```
01: Initialising agents()
02: While (stopping condition is not met)
03:    Testing hypotheses()
04:       Determining agents' activities (active/inactive)
05:    Diffusing hypotheses()
06:       Exchanging of information
07: End While
```

3.2.1 Search Example One

The search example here is defined in the form of a game, where the *respondent* selects, say, an animal without revealing it to others. Other participants (*questioners*) take turn to ask questions (one at a time) in order to figure out the selected animal.

In the initial phase, each questioner asks his/her question separately about the animal (hypothesis) they think of. Questioners are neither able to hear the questions of their peers nor the answers they are given. After each question, the respondent gives each questioner the answer in the form of Yes/No (see Table 3.1).

Table 3.1: Initialisation and Test Phases

Questioner	Question	Hypothesis	Activity
1	Does it climb trees?	Monkey?	No
2	Does it crawl?	Snake?	No
3	Does it fly?	Pigeon?	No

As described in the Mining Game, inactive questioners or agents (those who get 'No' as an answer) choose another questioner randomly to see if he/she is active. If the chosen questioner is active, it diffuses its *hypothesis* to the inactive one (see Table 3.2).

Table 3.2: Diffusion Phase 1

Questioner	Communicates with	Diffusion
1	3 (inactive)	No
2	1 (inactive)	No
3	2 (inactive)	No

If there is no active questioner, there will not be any diffusion of information (hypothesis) and each questioner puts forth another question (see Table 3.3).

The respondent gives his answer to the questioners and then communication between questioners (diffusion), which is illustrated in Table 3.4, takes place; questioner-2 randomly picks questioner-3 (which is active) and adopts its

Table 3.3: Test Phase 2

Questioner	Question	Hypothesis	Activity
1	Does it live in the sea?	Dolphin?	No
2	Has it got a trunk?	Elephant?	No
3	Does it live in deserts?	Camel?	Yes

hypothesis (Camel). Note that active questioners (here, questioner-3) do not pick another questioner.

Table 3.4: Diffusion Phase 2

Questioner	Communicates with	Diffusion
1	2 (inactive)	No
2	3 (active)	Yes
3	-	-

In the next phase, questioner-2 investigates its newly adopted hypothesis to see if it is a valid one (see Table 3.5) and questioner-1 (who was not able to communicate with an active questioner before) asks another question randomly; questioner-3 (who is active) re-checks the validity of his hypothesis by asking another question about it.

Table 3.5: Test Phase 3

Questioner	Question	Hypothesis	Activity
1	Does it live in jungles?	Tiger?	No
2	Has it got fur?	Camel?	Yes
3	Has it got a hump?	Camel?	Yes

As Table 3.6 shows, questioner-1 communicates with questioner-2 and adopts the same hypothesis and evaluates the hypothesis just adopted.

Questioner-2 and questioner-3 do not communicate as they have a hypothesis that they are happy with. They just evaluate another aspect of their hypothesis to make sure it is the optimal one (see Table 3.7)

As Table 3.7 shows, all the questioners are active now and they all converge to the same hypothesis, which is the correct animal in the respondent's mind.

Table 3.6: Diffusion Phase 3

Questioner	Communicates with	Diffusion
1	2 (active)	Yes
2	-	-
3	-	-

Table 3.7: Test Phase 4

Questioner	Question	Hypothesis	Activity
1	Does it resist thirst?	Camel?	Yes
2	Can it resist sand storms?	Camel?	Yes
3	Is it able to walk in sands?	Camel?	Yes

3.2.2 Search Example Two

In order to demonstrate the detailed process through which SDS functions, an example is presented which shows how to find a set of letters within a larger string of letters. The goal is to find a 3-letter model (Table 3.8) in a 16-letter search space (Table 3.9). In this example, there are four agents. For simplicity of exposition, a perfect match of the model exists in the Search Space (SS).

Table 3.8: Model

Index:	0	1	2
Model:	***S***	***I***	***B***

Table 3.9: Search Space

Index:	0	1	2	3	4	5	6	7
Search Space:	***X***	***Z***	***A***	***V***	***M***	***Z***	***S***	***I***
Index:	8	9	10	11	12	13	14	15
Search Space:	***B***	***V***	***G***	***O***	***L***	***B***	***E***	***H***

In this example, a hypothesis, which is a potential problem solution, identifies three adjacent letters in the search space (e.g. hypothesis '1' refers to Z-A-V, hypothesis '10' refers to G-O-L and etc).

In the first step, each agent initially picks a hypothesis randomly from the search space (see Table 3.10). Assume that:

- the first agent points to entry 12 of the search space and in order to partially evaluate this entry, it randomly picks one of the letters (e.g. the first one, L): | **L** | B | E |

- the second agent points to entry 9 and randomly picks the second letter (G): | V | **G** | O |

- the third agent refers to entry 2 in the search space and randomly picks the first letter (A): | **A** | V | M |

- the fourth agent goes entry 3 and randomly picks the third letter (Z): | V | M | **Z** |

Table 3.10: Initialisation and Iteration 1

Agent No:	1	2	3	4
Hypothesis position:	12 **L**-B-E	9 V-**G**-O	2 **A**-V-M	3 V-M-**Z**
Letter picked:	1^{st}	2^{nd}	1^{st}	3^{rd}
Status:	×	×	×	×

The letters picked are compared to the corresponding letters in the model, which is S-I-B (see Table 3.8).

In this case:

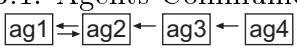
- The 1^{st} letter from the first agent (L) is compared against the 1^{st} letter from the model (S) and because they are not the same, the agent is set inactive.

- For the 2^{nd} agent, the second letter (G) is compared with the second letter from the model (I) and again because they are not the same, the agent is set inactive.
- For the third and fourth agents, letters 'A' and 'Z' are compared against 'S' and 'B' from the model. Since none of the letters correspond to the letters in the model, the status of the agents are set as inactive.

In the next step, as in the mining game, each inactive agent chooses another agent and adopts the same hypothesis if the selected agent is active. If the selected agent is inactive, the selecting agent generates a random hypothesis.

Assume that the first agent chooses the second one; since the second agent is inactive, the first agent must choose a new random hypothesis from the search space (e.g. 6). See Figure 3.1 for the communication between agents.

Figure 3.1: Agents Communication 1



The process is repeated for the other three agents. As the agents are inactive, they all choose new random hypotheses (see Table 3.11).

Table 3.11: Iteration 2

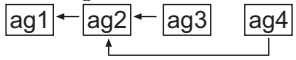
Agent No:	1	2	3	4
Hypothesis position:	6 S-**I**-B	10 G-O-**L**	0 **X**-Z-A	5 **Z**-S-I
Letter picked:	2^{nd}	3^{rd}	1^{st}	1^{st}
Status:	$\surd$	$\times$	$\times$	$\times$

In Table 3.11, the second, third and fourth agents do not refer to their corresponding letter in the model, therefore they become inactive. The first agent, with hypothesis '6', chooses the 2^{nd} letter (I) and compares it with the 2^{nd} letter of the model (I). Since the letters are the same, the agent becomes active.

At this stage, consider the following communication between the agents: (see Figure 3.2)

- the fourth agent randomly chooses the second one
- the third agent randomly chooses the second one
- the second agent randomly chooses the first one

Figure 3.2: Agents Communication 2



In this case, the third and fourth agents, which chose an inactive agent (the second agent), have to choose other random hypotheses each from the search space (e.g. agent three chooses hypothesis '1' which points to Z-A-V and agent four chooses hypothesis 4 which points to M-Z-S), but the second agent adopts the hypothesis of the first agent, which is active. As shown in Table 3.12:

- The first agent, with hypothesis '6', chooses the 3^{rd} letter (B) and compares it with the 3^{rd} letter of the model (B). Since the letters are the same, the agent remains active.
- The second agent, with hypothesis '6', chooses the 1^{st} letter (S) and compares it with the 1^{st} letter of the model (S). Since the letters are the same, the agent stays active.
- the third and fourth agents do not refer to their corresponding letter in the model, therefore they are set inactive.

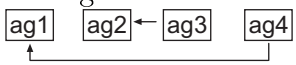
Table 3.12: Iteration 3

Agent No:	1	2	3	4
Hypothesis position:	6 S-I-**B**	6 **S**-I-B	1 Z-**A**-V	4 M-Z-**S**
Letter picked:	3^{rd}	1^{st}	2^{nd}	3^{rd}
Status:	$\surd$	$\surd$	$\times$	$\times$

Because the third and fourth agents are inactive, they try to contact other agents randomly. For instance (see Figure 3.3):

- agent three randomly chooses agent two
- agent four randomly chooses agent one

Figure 3.3: Agents Communication 3



Since agent three chose an active agent, it adopts its hypothesis (6). As for agent four, because it chose agent one, which is active too, it adopts its hypothesis (6). Table 3.13 shows:

- The first agent, with hypothesis '6', chooses the 1^{st} letter (S) and compares it with the 1^{st} letter of the model (S). Since the letters are the same, the agent remains active.
- The second agent, with hypothesis '6', chooses the 2^{nd} letter (I) and compares it with the 2^{nd} letter of the model (I). Since the letters are the same, the agent stays active.
- The third agent, with hypothesis '6', chooses the 3^{rd} letter (B) and compares it with the 3^{rd} letter of the model (B). Since the letters are the same, the agent becomes active.
- The fourth agent, with hypothesis '6', chooses the 1^{st} letter (S) and compares it with the 1^{st} letter of the model (S). Since the letters are the same, the agent is set active.

Table 3.13: Iteration 4

Agent No:	1	2	3	4
Hypothesis position:	6 **S**-I-B	6 S-**I**-B	6 S-I-**B**	6 **S**-I-B
Letter picked:	1^{st}	2^{nd}	3^{rd}	1^{st}
Status:	√	√	√	√

At this stage, the entire agent populations are active pointing to the location of the model inside the search space.

3.2.3 Initialisation and Termination

Although normally agents are uniformly distributed throughout the search space, if the search space is of a specific type, or knowledge exists about it *a priori*, it is possible to use a more intelligent (rather than random) startup by biasing the initialisation of the agents.

If there is a pre-defined pattern to find in the search space, the goal will be locating the best match or, if this does not exist, its best instantiation in the search space [94]. Similarly, in a situation which lacks a pre-defined pattern, the goal will be finding the best pattern in accord with the objective function.

In both cases, it is necessary to have a termination strategy. In one method[3], SDS terminates the process when a statistical equilibrium state is reached, which means that the threshold of the number of active agents is exceeded and the population maintains the same state for a specified number of iterations.

In [99], four broad types of halting criteria are introduced:

1. No stopping criterion, whereby the user interrupts the course of action of the search or optimisation and is usually preferred when dealing with *dynamically changing* problem spaces or when there is no predefined pattern to look for

2. Time-based criterion, in which passing a pre-set duration of time is the termination point of the algorithm

3. Activity-based criterion, which is a problem-dependent halting criterion and is the most prevalent form in the SDS algorithm. The termination of the process is decided upon through monitoring the overall activity of the agents (e.g. reaching a certain user defined activity level, reaching a stable population state after a sudden increase in their activities)

4. Cluster-based criterion that keeps tracks of the formation of stable clusters.

[3]Ibid

Introducing stopping criteria adds extra computations to what would otherwise be a distributed algorithm. As an alternative to the full-model cluster-based criteria, just a small proportion of the population can be checked on whether they point to the same hypothesis [99]. Increasing the size of the already monitored sample might also be considered afterwards.

Additionally, in order to reduce the computational complexity of the search, it is possible to run the termination procedure after every n iterations. The two most common termination strategies in SDS (introduced in [94]) are the following:

- Weak halting criterion is the ratio of the active agents to the total number of agents. In this criterion, cluster sizes are not the main concern.
- Strong halting criterion investigates the number of active agents that forms the largest cluster of agents all adopting the same hypothesis.

Therefore, the choice of the halting mechanism is based on whether to favour the active agents in the whole of the agent populations (weak halting mechanism), which is similar to the activity-based criterion, or to consider the largest cluster of active agents (strong halting mechanism), which is similar to the cluster-based criterion.

3.2.4 Partial Function Evaluation

One of the concerns associated with many optimisation algorithms (e.g. genetic algorithm [63], particle swarm optimisation [4], etc.) is the repetitive evaluation of a computationally expensive fitness function. In some applications, such as tracking a rapidly moving object, the repetitive function evaluation significantly increases the computational cost of the algorithm. Therefore, in addition to reducing the number of function evaluations, other measures should be taken in order to reduce the computations carried out during the evaluation of each possible solution as part of the optimisation or search processes.

The commonly used benchmarks for evaluating the performance of swarm intelligence algorithms are typically small in terms of their objective functions computational costs [1, 2], which is often not the case in real-world applications. Examples of costly evaluation functions are seismic data interpretation [2], selection of sites for the transmission infrastructure of wireless communication networks and radio wave propagation calculations of one site [100].

Costly functions have been investigated under different conditions [101] and the following two broad approaches have been proposed to reduce the cost of function evaluations:

- The first is to estimate the fitness by taking into account the fitness of the neighbouring elements, the former generations or the fitness of the same element through statistical techniques introduced in [102, 103].
- In the second approach, the costly fitness function is substituted with a cheaper, approximate fitness function.

When agents are about to converge, the original fitness function can be used for evaluation to check the validity of the convergence [101].

Many fitness functions are decomposable to components that can be evaluated separately. In partial evaluation of the fitness function in SDS, the evaluation of one or more of the components may provide partial information and means for guiding the optimisation. The role of partial evaluation process in dealing with noise is described in the following [51]:

> "Certain types of noise in the objective function may be completely absorbed in the probabilistic nature of the partial evaluation process, and do not influence the search performance of SDS: i.e., they have no effect on convergence times and stability of clusters. More formally, noise that introduces or increases variance in the evaluation of component functions f_i – without altering

> the averaged probabilities of the test score[4] – has no effect on the resource allocation process."

Dynamic Environments The application of partial function evaluation is of more significance when the problem space is dynamically changing and the evaluation process is of a more repetitive nature. Repeated (re)evaluations of fitness functions in many swarm intelligence algorithms necessitate having less costly fitness functions.

On the other hand, the diffusion mechanism tends to reduce the diversity in the population or the population homogeneity [51], which in turn leads to an inadequate subsequent reaction in a dynamically changing fitness function.

SDS aims at proposing a new solution (see Section 3.2.6) to the problem of population homogeneity by utilising an alternative method to balance the trade off between the *wide exploration* of all possible solutions in the problem space and the *detailed exploitation* of any possible smaller region which might be a candidate for holding the sought object.

3.2.5 Convergence

Convergence time is defined as the number of iterations needed before a stable population of active agents is formed.

An important factor which influences convergence is the ratio of the number of agents to the size of the solution space. In [104], it is proved that in a noiseless environment convergence always happens. In [94], it is proved that all agents become active when searching for a solution in a noiseless environment where a perfect match exists.

Additionally, noise, which does not alter the averaged probabilities of the test score but increases the variance in the evaluation of component functions, has no effect on the resource allocation process of SDS [51]. However, if the value

[4]Test score is the probability of producing active agents during the test phase, averaged over all component functions.

of test score changes as a result of noise presence, the resource allocation process may be influenced either:

- positively if the value of the test score increases
- or negatively if the value of the test score decreases

It is also proved that the population size and the test score determine the average cluster size as well as convergence times.

The approximately linear time complexity of SDS is analysed in [94] where two extreme cases in convergence time have been considered:

1. in the initial stages, some of the agents point to the correct position in the search space, which results in a shorter convergence time.
2. there is no agent pointing to the correct position for some time after the initialisation, which may lead to a longer process before the first agent locates a potentially correct location.

It has also been shown that convergence time in SDS is proportionately robust to the amount of noise in the search space.

Convergence to a global optimal solution in SDS is detailed in [96].

3.2.6 Resource Allocation and Stability

In addition to convergence time, steady-state resource allocation is one of the important factors in the performance criteria of SDS [105]. In order to measure the robustness of the algorithm, in the case of the presence of noise and imperfect matches, resource allocation is taken into account, which is defined as the average number of active states when the search shows steady-state behaviour. Although, resource allocation in standard SDS is *dynamic* and *self-regulatory*, there are certain issues to be investigated.

Local Exploitation and Global Exploration In standard SDS, there is no explicit mechanism to shift the balance from local exploitation (*detailed exploitation*) towards global exploration (*wide exploration*) of candidate solutions.

As observed in [106], a meta-heuristic approach tries to exploit self-similarity and regularities of the fitness function, which indicates that neighbouring solutions in the problem space have alike properties. Adding this mechanism to SDS may be helpful; one way of embedding this into the algorithm is to add a small random offset to the hypotheses before copying them to other agents during the diffusion phase, which is similar to mutation in evolutionary algorithms [51, 99]. The effect of this minor change in the hypotheses is to investigate nearby solutions, which generally serves as a hill-climbing mechanism improving the overall performance of the SDS and results in improved convergence time in solution spaces with self-similarity. Nevertheless, it also accelerates the identification of more optimal solutions in the vicinity of already found ones.

In dynamically changing environments, it is important to explore the solution space even after finding a suitable candidate solution, as once a good solution is detected, a large proportion of agents are attracted to it, thus limiting further exploration of the solution space. Therefore, the Context Sensitive and Context Free mechanisms (described in Section 3.3.4 and 3.3.5) are proposed to shift the balance of the search back to exploration.

A full account of Markov chain based analysis of the stochastic nature of standard SDS for resource allocation and the steady state probability distribution of the whole swarm is extensively discussed in [93]. More information about search behaviour and resource allocation can also be found in [107, 108].

In heuristic multi-agent systems, the possibility of agents losing the best solution results in destabilising or even non-convergence of the algorithm. Conversely, it is shown that the solutions found by SDS are exceptionally stable [109].

3.3 Variations in SDS and Recruitment Strategies

In SDS, similar to other optimisation algorithms, the goal is finding the best solution based on the criteria specified in the objective function. The collection of all candidate solutions (hypotheses) forms the search space and each point in the search space is represented by an objective value, from which the objective function is formed [51].

One of the issues related to SDS is the mechanism behind allocating resources to ensure that while potential areas of the problem space are exploited, exploration is not ignored. For this purpose, different recruitment methods, where one agent recruits another, are investigated:

Three recruitment strategies are proposed in [110]: active, passive and dual. These strategies are used in the Diffusion Phase of SDS. Each agent can be in either of the following states: It is *active* if the agent is successful in the Test Phase; an agent is *inactive* if it is not successful; it is *engaged* if it is involved in a communication with another agent.

The standard SDS algorithm [3] uses the passive recruitment mode, which will be described next followed by other recruitment modes.

3.3.1 Passive Recruitment Mode

In the *passive recruitment mode* (see Algorithm 3.3), if the agent is not active, another agent is randomly selected; if the randomly selected agent is active, the hypothesis of the active agent is communicated (or *diffused*) to the inactive one. Otherwise a new random hypothesis is generated for the inactive agent and there will be no flow of information.

Algorithm 3.3 Passive Recruitment Mode

```
01: For ag = 1 to No_of_agents
02:    If ( !ag.activity() )
03:       r_ag = pick a random agent()
04:       If ( r_ag.activity() )
05:          ag.setHypothesis( r_ag.getHypothesis() )
06:       Else
07:          ag.setHypothesis( randomHypothesis() )
08:       End If/Else
09:    End If
10: End For
```

3.3.2 Active Recruitment Mode

In the *active recruitment mode* (see Algorithm 3.4), active agents are in charge of communication with other agents. An active agent randomly selects another agent. If the randomly selected agent is neither active nor engaged in communication with another active agent, then the hypothesis of the active agent is communicated to the inactive one and the agent is flagged as engaged. The same process is repeated for the rest of the active agents. However, if an agent is neither active nor engaged, a new random hypothesis is generated for it.

Algorithm 3.4 Active Recruitment Mode

```
01: For ag = 1 to No_of_agents
02:    If ( ag.activity() )
03:       r_ag = pick a random agent()
04:       If ( !r_ag.activity() AND !r_ag.getEngaged() )
05:          r_ag.setHypothesis( ag.getHypothesis() )
06:          r_ag.setEngaged( true )
07:       End If
08:    End If
09: End For
10:
11: For ag = 1 to No_of_agents
12:    If ( !ag.activity() AND !ag.getEngaged() )
13:       ag.setHypothesis( randomHypothesis() )
14:    End If
15: End For
```

3.3.3 Dual Recruitment Mode

In *dual recruitment mode* (see Algorithm 3.5), both active and inactive agents randomly select other agents. When an agent is active, another agent is randomly selected. If the randomly selected agent is neither active nor engaged, then the hypothesis of the active agent is shared with the inactive one and the inactive agent is flagged as engaged. Also, if there is an agent which is neither active nor engaged, it selects another agent randomly. If the newly selected agent is active, there will be a flow of information from the active agent to the inactive one and the inactive agent is flagged as engaged. Nevertheless, if there remains an agent that is neither active nor engaged, a new random hypothesis is chosen for it.

Algorithm 3.5 Dual Recruitment Mode

```
01: For ag = 1 to No_of_agents
02:    If ( ag.activity() )
03:       r_ag = pick a random agent()
04:       If ( !r_ag.activity() AND !r_ag.getEngaged() )
05:          r_ag.setHypothesis( ag.getHypothesis() )
06:          r_ag.setEngaged( true )
07:       End If
08:    Else
09:       r_ag = pick a random agent()
10:       If ( r_ag.activity() AND !ag.getEngaged() )
11:          ag.setHypothesis( r_ag.getHypothesis() )
12:          ag.setEngaged( true )
13:       End If
14:    End If/Else
15: End For
16:
17: For ag = 1 to No_of_agents
18:    If ( !ag.activity() AND !ag.getEngaged() )
19:       ag.setHypothesis( randomHypothesis() )
20:    End If
21: End For
```

3.3.4 Context Sensitive Mechanism

Comparing the above-mentioned recruitment modes, it is theoretically determined in [110] that robustness and greediness decrease in the active recruitment mode. Conversely, these two properties are increased in dual recruitment strategy. Although, the greediness of dual recruitment mode results in decreasing the robustness of the algorithm, the use of *Context Sensitive Mechanism* limits this decrease [110, 93]. In other words, the use of context sensitive mechanism biases the search towards global exploration. In the context sensitive mechanism, if an active agent randomly chooses another active agent that maintains the same hypothesis, the selecting agent is set inactive and adopts a random hypothesis. This mechanism frees up some of the resources in order to have a wider exploration throughout the search space as well as preventing cluster size from overgrowing, while ensuring the formation of large clusters in case there exists a perfect match or good sub-optimal solutions (see Algorithm 3.6).

Algorithm 3.6 Context Sensitive Mechanism

```
01: If ( ag.activity() )
02:     r_ag = pick a random agent()
03:     If ( r_ag.activity() AND
04:          ag.getHypothsis() == r_ag.getHypothsis() )
05:        ag.setActivity( false )
06:        ag.setHypotheis( randomHypothsis() )
07:     End If
08: End If
```

3.3.5 Context Free Mechanism

Context Free Mechanism is another recruitment mechanism, similar to context sensitive mechanism, where each active agent randomly chooses another agent. If the selected agent is active (irrespective of having the same hypothesis or not), the selecting agent becomes inactive and picks a new random

hypothesis. By the same token, this mechanism ensures that even if one or more good solutions exist, about half of the agents explore the problem space and investigate other possible solutions (see Algorithm 3.7).

Algorithm 3.7 *Context Free Mechanism*

```
01: If ( ag.activity() )
02:    r_ag = pick a random agent()
03:    If ( r_ag.activity() )
04:       ag.setActivity( false )
05:       ag.setHypotheis( randomHypothsis() )
06:    End If
07: End If
```

3.3.6 Synchronous and Asynchronous Update

In synchronous diffusion mode, the updates of all hypotheses occur simultaneously after each iteration of test and diffusion phases; whereas in asynchronous mode, each hypothesis is updated individually. Although, in the original SDS [3], the synchronous mode is used, it is possible to diffuse the hypotheses of successful agents synchronously or asynchronously.

As mentioned in [99], in many variants, the performance of an asynchronous process is approximately the same as the synchronous one, with each agent operating in its own time.

3.3.7 Composite Hypotheses

In standard SDS all hypotheses are homogeneous and thus have the same type. In this section, new variants of SDS are described where there are two *different* types of hypotheses working together. These SDS types are applied to solve parameter estimation problems, which is a more complicated search problem compared to pattern matching. In parameter estimation, outlier data (or random noise) is embedded in the data (or inlier data); the goal is to find parameter values that best describe the inlier data [111]. Data

Driven SDS [112] and coupled SDS [111], which have composite hypotheses, are both used to solve parameter estimation problems. In the estimation problem, similarly to other search problems, a cost function or objective function is required to measure how close the algorithm is to the inlier data or the model in the search space.

In parameter estimation, the objective function is optimised with respect to the estimated model parameters; that is why it is considered an optimisation problem [112].

3.3.7.1 Data Driven SDS

Data Driven SDS (DDSDS) is shown to outperform [112] Maximum Likelihood Estimator Sample Consensus (MLSESAC) which is a variant of RANdom SAmple Consensus (RANSAC), one of the most popular and robust estimators based on stochastic principles [113].

DDSDS contains a composite hypothesis: a manifold hypothesis, which maintains the minimum necessary dataset for describing a hypothesis; and a datum hypothesis, which represents the smallest building block of the hypothesis. If estimating a line is the problem, then the manifold hypothesis would consists of two points, which are sufficient to represent a line, and the datum hypothesis would be a single point that is randomly selected from the manifold hypothesis rather than the whole of the search space.

In the test phase, random datums are selected just from datum hypotheses that are associated with the agents. The probability of selecting a datum, which has no link with any agents is zero. This will dynamically constrain the selection to the data generated by the inlier distribution [112]. The distance of the agent's manifold hypothesis from the randomly selected datum is then evaluated to determine if the distance stays within the pre-set inlier threshold value. If this is the case, the agent's state becomes active.

In the diffusion phase, the active agent diffuses its manifold and datum hypotheses to the inactive agent. When an inactive agent is not involved in any information exchange, similarly to the initialisation phase, it chooses two

random data from the entire search space for the manifold hypothesis and the datum hypothesis is selected from one of the two elements of the manifold hypothesis.

3.3.7.2 Coupled SDS

In Coupled SDS (CSDS) two independent populations of agents are formed each maintaining different types of hypothesis, namely the manifold hypotheses and datum hypotheses. In contrast to DDSDS, datum hypotheses are selected randomly from the entire search space. The size of these two populations are not necessarily the same. They are randomly and independently initiated with data from the entire search space. In the test phase, the manifold hypothesis of one agent is compared with the datum hypothesis of another. Based on the distance threshold, if the datum matches the manifold, both agents become active. This evaluation is called composite hypothesis evaluation, which is more complicated than the synchronous evaluation in standard SDS, where there is just one population of agents. Therefore, in addition to the asynchronous test, two further synchronisation modes have been proposed:

- Master/Slave Synchronisation, where one of the populations is master and the other is slave. The master hypothesis randomly selects a hypothesis from the slave population for the test. In this mode, there will be m composite evaluation, where m is the size of the master population.

- Sequential Master Synchronisation is a variant of the master/slave mode, where populations take turn to be master. Each iteration has n composite evaluations, which is the sum of all agents in both manifold and datum populations.

The diffusion phase in CSDS is similar to the standard SDS for each population independently, where the information flow is allowed *within* each

population of agents and thus there is no information exchange between the manifold and datum population of agents [111].

It is empirically shown that DDSDS converges even when there are 50% more outliers while it also outperforms standard SDS in convergence time [112]. Both of these SDS variants have been proposed to improve the performance of the original SDS towards stable convergence in high noise estimation tasks.

3.4 Applications

There are several applications associated with SDS which have been successfully applied to diverse problems.

SDS was first introduced by a simple text searching algorithm in 1989 [114] demonstrating the use of partial function evaluation technique – by partially evaluating the text to find the model or the best match.

Subsequently, in 1992, tracking eyes was investigated in [104]. In this project, a hybrid stochastic search network was used to locate eye positions within grey scale images of human faces. It was shown that the network can accurately locate the eye features of all the subjects it has been trained with and it could reach over sixty percent success in locating eye features on subjects on which the system has not been explicitly trained with.

In 1995 project, similar to the two above, SDS was used in solving visual search tasks, such as object recognition (in this case, locating facial features[115]). The details of another visually related task for real time tracking of lips in video films was given in [115], where SDS uses a hybrid system of a set of n-tuple neurons [116].

Exploring a set of candidate positions to self-localise an autonomous wheelchair or robot in a complex busy environment through a number of cells was used in a method called Focused Stochastic Diffusion Network [117] in 1998; in this method, the space of possible positions was examined in parallel by a set of competitive cooperative cells to identify the most likely position of the robot or wheelchair in the environment.

In 1999, emergent characteristics of neuron functionality were also described using a new metaphor based on SDS utilising spiking neurons [118]; in the paper, it was also argued that the metaphor of conventional computational description of brain operation is too restrictive (limited to representing and processing knowledge of arity-zero; NESTER can represent and process knowledge of arity-n).

SDS was also used in wireless transformation networks, where the location of transmission infrastructure is of significance to keep the cost minimum while preserving adequate area coverage [100]. In this application, at 2002, given a set of candidate sites, a network should be designed so that at a certain number of reception points, the signal from at least one transmitter can be received.

In 2008 [119], SDS was used in feature tracking in the context of Atmospheric Motion Vectors derivation, as using template matching techniques, such as Euclidean distance or cross-correlation for tracking steps which are very expensive computationally.

In 2011, the SDS algorithm demonstrated a promising ability in identifying areas of metastasis from bone scintigraphy [120, 121].

Implementation on Hardware

SDS is inherently parallel in nature and the hardware implementation of the algorithm is feasible. Still, the fact that the original SDS model requires full inter-agent connectivity, where each agent is able to communicate directly with all others in the population, causes fundamental difficulty in the efficient implementation of the algorithm on parallel computer or dedicated hardware.

One of the solutions proposed in [105] was to limit the communication between the agents. Agents are considered to be spatially located in a lattice (SDS or LSDS) where each agent is only connected to the k-nearest neighbours.

Therefore, considering this form of SDS, agents just communicate with the

ones they are connected to. It was shown that a network with randomly connected agents (random graph), with a small number of long-range connections, performs similar to the standard SDS or ordered lattice with roughly the same number of connections[5]. The conclusion has been drawn that restricting the number of interconnectivity in random or small-world networks – which is a lattice with a small additional number of long-range connections – does not have a huge effect on the performance of SDS algorithm. Also, the rate of information spread is higher in random graphs and small-world networks than ordered lattices.

Analysing the number of connections and the connection topology leads to the following conclusion: it has been argued that when a high-dimensional, complex problem is considered, the time at which one of the agents becomes active (or 'time to hit' as defined in [114]), T_h, is bigger than the time required for the active agent to spread its successful hypothesis ('diffusion time') T_d [105]. Although random graphs have shorter T_d than regular lattices, they are harder to implement on parallel hardware, because the connections are not necessarily local or short. In ordered lattice SDS topology, which shows the performance of a fully interconnected standard SDS, adding random links decrease T_d exponentially.

T_d is considered to be an important factor, which not only affects convergence time, but is also seen as a parameter for resource allocation stability [122], as well as an indirect measure for robustness [105].

In another approach, the agent swarm can be divided into several sub-swarms. In this mode, each sub-swarm runs on a separate processor and they are fully connected while allowing just a low frequency of communication between swarms. This process is applied to the diffusion phase, during which agents communicate with each other.

[5]Ibid

Chapter 4

Population-Based Optimisers

"The used key is always bright."
– Benjamin Franklin

This chapter presents three well known population-based algorithms: Particle Swarm Optimisation (PSO), Genetic Algorithm (GA) and Differential Evolution algorithm (DE). A description of Particle Swarm Optimisation is followed by a discussion on different parameter changes and their effects on standard PSO algorithm. The section on PSO is more detailed than the other two algorithms in this chapter, as PSO was the primary algorithm investigated alongside SDS in order to explore their integration strategy (which is reported in Chapter 7 on page 122). The last two sections of this chapter give an overview on GA and DE algorithms.

4.1 Particle Swarm Optimisation

Particle Swarm Optimisation (PSO) is an evolutionary computation technique developed in 1995 by Kennedy and Eberhart [4, 123]. It came about as a result of an attempt to graphically simulate the choreography of fish

schooling or birds flying (e.g. pigeons, starlings, and shorebirds) in coordinated flocks that show strong synchronisation in turning, initiation of flights and landing, despite the fact that experimental research to find leaders in such flocks failed [124]. In particle swarms, although members of the swarm neither have knowledge about the global behaviour of the swarm nor global information about the environment, the local interactions of the swarms result in complex collective behaviour, such as flocking, herding, schooling, exploration and foraging behaviour [57, 125, 126, 127]. The boid simulation, developed in 1987 by Reynolds [57], visualises flocking as an emergent behaviour originated by the interaction of three simple rules:

- Collision avoidance: avoiding collision with neighbouring flock mates
- Velocity matching: matching the velocity of nearby mates
- Flock centring: attempting to stay close to neighbouring mates

Moreover, the socio-cognitive theory underpinning particle swarms or adaptive culture model is described in terms of three simple principles [128]:

- Evaluation, by which the closeness of particles to the optima is measured; in the simplest form, they are ranked as positive or negative.
- Comparison, where particles are compared with their neighbouring particles. Comparison, as it is described in [129], is a way to foster learning and change.
- Imitation, which is a way to learn from other members of the swarm to do things.

Combining the three principles of evaluation, comparison and imitation leads a simplified social being to adopt complex environmental challenges and to optimise hard problems as suggested by Kennedy et al. ([44], p. 284).

The origin of particle swarm optimisation goes back to Artificial Life (AL), social psychology, engineering and computer science. Although PSO lacks

operators such as mutation and crossover, it shares similarities with evolutionary computation (genetic algorithms, evolutionary programming, evolution strategies and genetic programming [130]). Here are some of these similarities:

- random initialisation of populations (potential solutions)
- updating generations to search for optima
- the use of the concept of fitness

However, one of the differences is that particles (possible solutions) in PSO are *flown* [4] through the problem space. At each iteration, each particle flies towards the weighted average of its former best position, which represents the best fitness value, as well as its neighbourhood's best position.

In evolutionary computation, current populations are transformed and the transformation is inspired by the neo-Darwinian view of evolution.

Darwin theory of evolution is mainly based on natural selection, but during Darwin's time, chromosomes were not known and Darwin's theory was not able to fully describe how variations arise and how they are passed on to the offspring. In the neo-Darwinian view of evolution (which includes Mendelian ideas of genetics from 1865), three main issues are presented [131]:

- the composition of chromosome is determined by the parents
- random mutation expands the diversity of species
- fitter individuals are more likely to survive to the next generation

Nowadays, as Kennedy et al. [131] believe, the Darwinian view of evolution is perhaps better described as the neo-Darwinian view.

Still, as Kauffman argues in 1993 and 1995 [132, 133], the following two issues are not fully described with the current theory:

- origin of life: considering the time frame of the earth, it's less likely to consider chance or mutation to be the origin of life

- complex life form: having complex life forms is highly improbable solely through mutation

On the contrary to evolutionary computation where transformation is inspired by the neo-Darwinian view of evolution, transformations in PSO come as a result of a simplified form of social behaviour of biological organisms. Both EC and PSO are inspired by natural phenomena [134].

One thing that distinguishes particle swarms from the evolutionary algorithms is that there is no selection in the form of replacement in particle swarms. There is of course selection in changing the older best with the new one whenever a better candidate solution is found, but the identity of the individual is preserved over time [135]. It can thus be said that in particle swarms an individual improves over time but is not replaced with their offspring, whilst improvements of the individual are not considered in evolutionary systems.

It is also apparent that there is a close relationship between particle swarms and Cellular Automata (CA), which is characterised by three main attributes [136]:

- individual cell updates are done in parallel
- each new cell's value depends only locally on the old values of the cell and its neighbours
- all cells are updated using the same rules.

Therefore, as stated in [44], particles can be conceptualised as cells in CA, whose states change in many dimensions simultaneously.

One of the main attractions of PSO, as a population-based global optimisation technique is its applicability to different problems (see Section 4.1.4 on page 92) whilst it remains simple to implement. Still, despite the simplicity of implementation of PSO algorithm and its increasing number of applications, little is known theoretically about how PSO achieves its results [137] (see Section 4.1.3 on page 86 for more details).

4.1.1 PSO Algorithm

Since its introduction in 1995, Particle Swarm Optimisation [4] has been expanded by different researchers. This section describes basic PSO and discusses some variations of the algorithm.

4.1.1.1 Standard PSO

A swarm in PSO algorithm comprises a number of particles and each particle represents a point in a multi-dimensional problem space. Particles in the swarm explore the problem space, searching for the optimal position, as defined by a fitness function. The position of each particle, x, is dependent on the particle's own experience and those of its neighbours. Each particle has a memory, containing the best position found so far during the course of the optimisation, which is called personal best (p). The best position found throughout the population – or in the neighbourhood – is called global best (p_g) (and local best (p_l) respectively).

The standard PSO algorithm defines the position of each particle by adding a velocity to the current position. Below is the equation for updating the velocity of each particle:

$$v_{id}^{t} = v_{id}^{t-1} + c_1 r_1 \left(p_{id} - x_{id}^{t-1}\right) + c_2 r_2 \left(p_{gd} - x_{id}^{t-1}\right) \tag{4.1}$$

$$x_{id}^{t} = v_{id}^{t} + x_{id}^{t-1} \tag{4.2}$$

where $\vec{v}_{id}^{t-1}$ is the velocity vector of particle i in dimension d at time step $t-1$; $c_{1,2}$ are the learning factors (also referred to as acceleration constants) for personal best and neighbourhood best respectively (they are constant and are usually set to 2); $r_{1,2}$ are random numbers adding stochasticity to the algorithm and they are drawn from a uniform distribution on the unit interval $U(0,1)$; $\vec{p}_{id}$ is the personal best position of particle x_i in dimension d; and $\vec{p}_{gd}$ is global best (or neighbourhood best).

Therefore, PSO optimisation is based on particles' individual experience and their social interaction with the particle swarms.

The influence of an individual particle is defined by means of $c_1 r_1 \left(p_{id} - x_{id}^{t-1}\right)$ which is the *cognitive component* (or "nostalgia" of the particle [4]) and the social influence in the optimisation is maintained through $c_2 r_2 \left(p_{gd} - x_{id}^{t-1}\right)$ which is the *social component*.

The high value of the cognitive component relative to the social one results in particles wandering through the search space; and the high value of the social component relative to the cognitive one results in a potentially pre-mature convergence of particles towards a local minimum[1].

Once the velocities of the particles are updated, their new positions are determined. Algorithm 4.1 summarises the behaviour of the PSO algorithm.

Algorithm 4.1 PSO Pseudo Code

```
01: Initialise particles
02:
03: While ( stopping condition is not met )
04:    For all particles
05:       Evaluate fitness value of the current particle
06:
07:       If ( current fitness < pbest )
08:          pbest = current fitness
09:       End If
10:
11:       If ( pbest < neighbourhood best )
12:          neighbourhood best = pbest
13:       End If
14:
15:       Update particle velocity
16:       Update particle position
17:    End For
18: End While
```

In local best PSO (lbest PSO), neighbourhoods are either formed by spatial similarity or particle indices. As stated in [134], neighbourhoods based on particle indices are preferred[2]. One reason is because computing spatial

[1] Ibid

[2] If particle indices are used, the left neighbour of i^{th} particle is $i-1$ and the right neighbour is $i+1$. Also the last particle is connected to the first one.

similarities of the particles is computationally expensive, as the Euclidean distance of the entire pairs of particles have to be calculated which would result in having a problem with the complexity of $O(n^2)$. Secondly, in spatially based neighbourhoods, information of the neighbourhood is restricted to the region where the neighbours exist, while in index based neighbourhoods, since neighbouring particles are not confined in a region, information is spread throughout the search space.

4.1.1.2 Stopping Condition

Different termination strategies have been used to stop the optimisation process in different problems. The following are some of the termination strategies in use:

- The maximum number of iterations (or function evaluation) is exceeded
- An acceptable solution is found
- No improvement is observed over a number of iterations. One way to measure improvement is to consider the objective function slope. The objective function slope is approximately zero, which is calculated through the following formula [138]:

$$f' = \frac{f\left(p_g^t\right) - f\left(p_g^{t-1}\right)}{f\left(p_g^t\right)} \tag{4.3}$$

  where function f returns the fitness value of particles. If $f' < \varepsilon$ for a number of iterations, the convergence criterion is considered to have been met. In other words, this method, monitors the improvement in p_g and if there is not enough improvement (based on the value of ε), the algorithm terminates.

- The normalised swarm radius is close to zero[3]

[3]Ibid

$$R_{norm} = \frac{R_{max}}{Diameter\,(S)} \tag{4.4}$$

where R_{max} (max radius) is the longest distance between a particle and the global best; and $Diameter\,(S)$ is the diameter of the initial swarm. When normalised radius or $R_{norm} < \epsilon$, the algorithm is terminated. However care should be given to (depending on the problem) neither choose a too large ϵ (as the algorithm might prematurely stop) nor a too small value (as the swarm may carry out excessive iterations to have a compact swarm, with all particles centred around the global best position).

4.1.1.3 Particles Initialisation

Particles are initialised within the boundaries of each dimension. If we assume that the boundaries of all the dimensions are the same, particles are then initialised based on the following equation:

$$X(i) = x_{min} + r_i(x_{max} - x_{min}) \tag{4.5}$$

where $X(i)$ is a particle position, x_{max} and x_{min} are upper and lower bounds respectively and r_i is a random number drawn from a uniform distribution on the unit interval $U(0, 1)$.

Although the velocity vector can be initialised in a similar way, it is usually initialised to zero. Since physical objects are initially stationary, if the velocity vector is not initialised to zero, physical analogy may be violated and thus initialisation should be done with care [134]. Particles' initial positions are used to initialise particles' personal best positions.

When the goal is to compare two or more different variants of the PSO algorithm, care should be taken not to initialise the particles nearby known optimal point(s) in the search space.

4.1.1.4 Interactivity and Diversity

PSO algorithm uses either global best or neighbourhood best (local best) position in the social component. Two of their main differences are discussed in terms of their convergence characteristics in [130, 139].

Since global best PSO has got a higher rate of interactivity than neighbourhood best PSO, it converges faster, but diversity is compromised.

Neighbourhood best PSO, which preserves more diversity than global best PSO, is able to cover a larger part of the problem space and, thus it is less likely to be trapped in local minima.

Particles in both global best and neighbourhood best PSO move towards the global best particle. In neighbourhood best PSO, this is possible because a particle can be a member in more than one neighbourhood; this allows information to be shared while it also facilitates the convergence of the swarm to an optimal point.

4.1.2 PSO Parameters and Variations

Similarly to other optimisation algorithms, PSO is influenced by its parameters, which in turn affect the balance between exploration and exploitation of the search space. Exploration is the ability of the algorithm to examine the search space as a whole, and exploitation is the ability of the algorithm to focus on a region where the possibility of finding an optimal solution is higher. The parameters and conditions influencing the behaviour of swarms in PSO are briefly discussed below.

4.1.2.1 Velocity Clamping

In order to control the exploration of particles, velocities are clamped to keep the particle swarm within the boundaries of the search space [130]. Therefore, if the velocity exceeds V_{max} (maximum allowable velocity), the new velocity is set to V_{max}. In addition to controlling exploration, V_{max} can also affect the

exploitation ability of the optimising algorithm. If V_{max} is large, exploration is facilitated, while for a smaller V_{max}, exploitation is emphasised. V_{max} is a fraction of the range of the problem space:

$$V_{max} = \alpha \left(x_{max} - x_{min} \right) \tag{4.6}$$

where $\alpha \in (0, 1]$ and, as a number of empirical studies suggests [140, 141], the optimal value of α is problem-dependent. The value of V_{max} is usually adjusted and, as Eberhart et al. [142] suggest, a better approach is to limit V_{max} to X_{max}, the dynamic range of the variable on each dimension.

4.1.2.2 Inertia Weight

Inertia weight, introduced in [143], aims at controlling the exploration and exploitation with less reliance on clamping velocities. Although inertia weight shows success in controlling global exploration and local exploitation, it still cannot completely keep the swarm in the boundaries of the search space.

By adding the inertia weight to the optimisation process, the new update equation in the standard particle swarm optimisation would be the following:

$$v_{id}^{t} = w v_{id}^{t-1} + c_1 r_1 \left(p_{id} - x_{id}^{t-1} \right) + c_2 r_2 \left(p_{gd} - x_{id}^{t-1} \right) \tag{4.7}$$

where w is the inertia weight whose optimal value is problem dependent, as suggested by Shi and Eberhart [141].

Another method introduced was defining w as a decreasing function of time (instead of a fixed constant), starting with a larger value and linearly decreasing over time. Some researchers [144] suggest the use of dynamic inertia weight in the terminal phase to increase convergence. Another study [145] recommended starting with a fixed value w, followed by a reduction in this parameter by the fraction $\alpha \in (0, 1)$ if no improved solution is found within h consecutive time steps.

4.1.2.3 Acceleration Coefficients

The acceleration coefficients (also called learning factors or trust parameters [134]) are c_1 and c_2 in the velocity update equation. They specify the confidence a particle should have in itself and its neighbours respectively. Small values of the acceleration coefficients results in the particles wandering away from good regions before returning to good regions again, while high values of acceleration coefficients induce more acceleration with swift movements towards or past potentially good regions.

4.1.2.4 Constriction Coefficient

This approach is similar to the use of inertia weight, where the goal is to balance exploration and exploitation by means of constricting velocities with a constant value χ, which is referred to as the constriction coefficient [146, 147]. The constriction factor is introduced in an attempt to mathematically analyse particle swarm optimisation. The following equation, which is known as Clerc-Kennedy (PSO-CK) update equation, represents this approach:

$$v_{id}^{t} = \chi \left(v_{id}^{t-1} + c_1 r_1 \left(p_{id} - x_{id}^{t-1} \right) + c_2 r_2 \left(p_{gd} - x_{id}^{t-1} \right) \right) \tag{4.8}$$

with

$$\chi = \frac{2k}{\mid 2 - \varphi - \sqrt{\varphi^2 - 4\varphi} \mid}, \varphi = c_1 + c_2 \tag{4.9}$$

and $\varphi \geq 4$ and $k \in [0, 1]$.

PSO has been empirically shown to outperform other optimisation techniques such as evolutionary algorithms over standard benchmarks (more details are reported in [148, 149, 150, 151, 152, 138]). As Bratton and Kennedy stated in [153], some of the parameters were empirically proved to be working well in general (e.g. $\chi = 0.72984$ when $\varphi = 4.1$ and $k = 1$). However, since these studies are based on a limited number of problem spaces, they should be interpreted with caution.

4.1.2.5 Velocity Models

Four different variations of PSO are discussed in [154], where the main difference lies in the velocity equation and the way the velocity vector is updated:

- Full Model
- Cognitive-Only Model
- Social-Only Model
- Selfless Model.

Full Model has already been introduced (see equations 4.7/4.1 and 4.8).

In Cognitive-Only Model, the social component is ignored, and there is a tendency to return to particle's previous best position, which can be psychologically assimilated to the willingness to return to the previously seen regions.

$$v_{id}^{t} = v_{id}^{t-1} + c_1 r_1 \left(p_{id} - x_{id}^{t-1}\right) \tag{4.10}$$

Kennedy[4] showed that this model is slightly more vulnerable than the original one, as particles tend to search locally around regions where they were first initialised and could be trapped in local minima. Poor performances of the model is reported in [155]. However, niching algorithms, where the goal is to locate multiple solutions, are shown to be among the promising areas for applying the cognitive-only model [149].

In Social-Only Model, as the name suggests, the cognitive component is removed from the velocity update equation:

$$v_{id}^{t} = v_{id}^{t-1} + c_2 r_2 \left(p_{gd} - x_{id}^{t-1}\right) \tag{4.11}$$

The best position of the neighbourhood is the focal point for the particles to be attracted to. It is empirically proven that Social-Only Model demonstrate faster convergence than the original and Cognitive-Only model [154, 155].

[4] Ibid

Based on this finding in [144], it is suggested to increase the social pressure at the cost of cognitive learning in the initial phase of the swarm search to enhance the migration of the particles to feasible regions.

The fourth model, the Selfless Model, is similar to the Social-Only Model except that a particle is not allowed to be the neighbourhood best position itself, because the neighbourhood best position must be chosen from a particle's neighbours. Although this model performs faster than the Social-Only Model in some instances, Carlisle and Dozier [155] report that it does not work well in dynamically changing environments.

4.1.2.6 Swarm Size

Increasing the number of particles increases the initial diversity of the swarm and decreases the iterations needed to find the optima, but the computational complexity of the optimisation is increased. Empirical studies from Brits et al. and van den Bergh and Engelbrecht [149, 156] had shown that having a swarm size of 10 to 30 is enough to lead the optimisation towards finding the optimal solutions.

However, Bratton and Kennedy [153] suggested a standard for PSO; 50 particles were used in the tests carried out there.

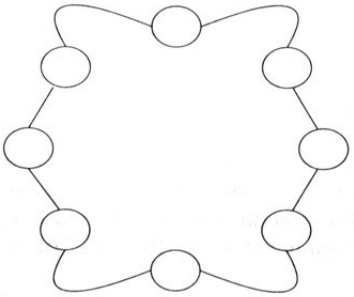
4.1.2.7 Network Topologies

Particle interaction plays an important role in the PSO optimisation process, and thus the structure of the social network has an influential impact on this interaction scheme. The following list shows some of the original topologies investigated [130, 157, 158, 159] (see Fig 4.1[5]):

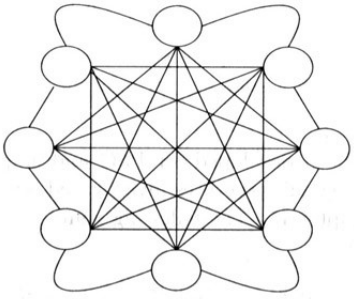
- ring (local)
- star (global)

[5] Taken from Engelbrecht [134]

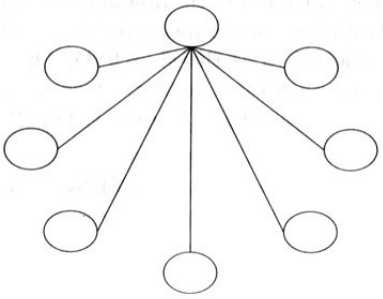
Figure 4.1: Network Topologies



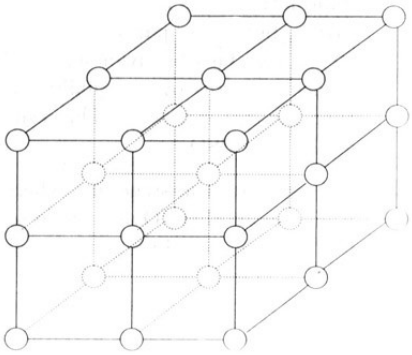
(a) Ring/Local



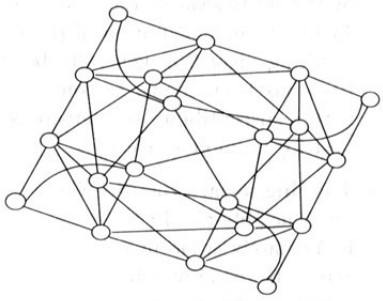
(b) Star/Global



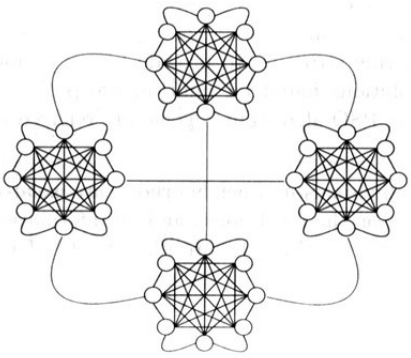
(c) Wheel



(d) Von Neumann



(e) Pyramid



(f) Four Clusters

- wheel
- Von Neumann
- pyramid
- four clusters

Considering the above mentioned network structures, there is no best topology for all problems. As discussed by Engelbrecht [134], while fully connected network topologies outperform less connected ones for unimodal problem space, multi-modal problem space are better optimised with less connected social networks.

4.1.2.8 Synchronous and Asynchronous Updates

In synchronous update, one snapshot of the search space is considered and the update for each particle is calculated but not applied. All the changes are applied at once after the completion of particles' updates. In asynchronous mode, changes are applied after each particle update.

In an experiment, Carlisle et al. [160] showed that the asynchronous mode is more useful in neighbourhood best PSO while the synchronous mode is more useful in global best PSO.

4.1.3 Understanding PSO

The collective influence of multiple particles and their stochastic elements make the theoretical analysis of PSO (shown to be sensitive to parameter changes) difficult. A number of theoretical studies have tried to understand the dynamics of PSO, mainly concentrating on particle trajectories [147, 161, 162, 138, 163], swarm equilibria and formal convergence to local optima proofs (see chapter 13 of Engelbrecht's Fundamentals of Computational Swarm Intelligence [134]).

Ozcan and Mohan, in their first theoretical studies of particle trajectory [161, 162], concluded that the metaphor of particles *flying* through the search space (introduced by Kennedy and Eberhart [4]) is better changed to *surfing* the search space on periodic sinusoidal waves, where an optimum is searched by randomly *catching* another wave through manipulating its frequency and amplitude. However, limiting velocity by means of V_{max} helps particles to "jump" onto another wave. In the first study (in [161]) the PSO algorithm without inertia weight is used; in their subsequent work [162], a more general system (deploying inertia weight) is used for the analysis.

In another analysis [147], Clerc and Kennedy, tried to understand swarms' behaviour and the result of the their work was the introduction of constriction coefficient version of PSO, which keeps velocity within the allowed bounds without the necessity of using velocity clamping. Equation 4.8 shows the equation of the velocity update using the constriction coefficient (χ).

In addition to various attempts to understand PSO algorithm, different variations of PSO algorithm (e.g. Bare Bones PSO in section 6.1 on page 110) are used to gain a better understanding of the performance of the algorithm.

4.1.3.1 Random-Restart PSO Algorithms

As stated previously, one of the problem associated with the standard form of PSO is the early stagnation of particles away from global optima or sometimes even not local optima. In order to prevent this premature stagnation, some methods are used to "shake" the particles or to induce some random displacements of particle swarms. The main purpose for adding this randomness is to increase diversity in order to explore a larger part of the problem space. However the unstructured injection of chaos into the swarm may cause the particles never to converge. The idea of random-restart in PSO was first introduced by Kennedy [4] as *craziness*. Among the aspects considered in the restart mechanism are:

- the parameters which are randomised

- the time at which restart is scheduled
- the way it is applied
- the particles or parameters which are affected in the process
- whether to preserve the memory of the particles through keeping their personal best values

Removing the memory of the particles prevents them from returning to their former positions and the diversity of the swarm can be adjusted through the random initialisation of the position vectors and/or the velocity vectors [134]. However, keeping personal bests of the particles results in having less diversity than when particles memory is cleared.

Another factor to consider is the frequency at which re-initialisation is applied. If re-initialisation happens late, particles may have already converged while they could possibly find better solution(s) if an earlier initialisation is induced. Although this situation is not a problem, it wastes the computational time as no improvement is incurred for a number of iterations [134]. If, in contrast, re-initialisation happens too early, particles are not given enough time to explore the region they are in before being displaced.

There are different approaches for re-initialisation. Fixed-interval re-initialisation is one approach. In [164], the velocities of all particles are reinitialised at a predefined interval and in [165], if a particle does not show improvements for a predefined number of iterations, its position and velocity is re-initialised. This approach, may however, not allow improvement.

In the probabilistic approach, re-initialisation is defined by means of a probability as described in Xie et al. [166] through the use of two random numbers, c_v, c_l, for re-initialising velocities and locations respectively; or, as Schutte et al. [144] suggest, through craziness, which is described as "crazy birds temporarily departing from the flocks with random direction and magnitude".

In [167, 168, 165], again re-initialisation is based on convergence where particles are re-initialised when there is no improvement over time. Following

this approach, some convergence tests have been suggested to control the re-initialisation [138]. Self-organised criticality (SOC) is used to specify the time when particles are re-initialised [169, 170]. The criticality represents the closeness of particles to each other. If they are closer than a threshold distance, ϵ, their criticality is increased by one. Therefore having a higher criticality rate for the whole of the swarm indicates the swarm is more uniform. When the global criticality reaches the specified limit, the swarm is re-initialised to ensure diversity. In the approaches introduced, the re-initialisation phase either re-allocates the particle randomly or pushes it a little further along the same direction it was moving in.

Deciding which particle to initialise is another issue to consider. However, former techniques can be used to decide upon the choice of particles to be re-initialised (e.g. using probability, SOC or etc.)

Random re-initialisation of particles (through position or velocity), can be managed within the allowed boundary constraints.

4.1.3.2 Cooperative Particle Swarm Optimiser

The notion of cooperation which has been used in many heuristic search methods has also been applied in PSOs. Here more than one search modules run on the search space, exchanging information and aiming at exploring the problem space more efficiently.

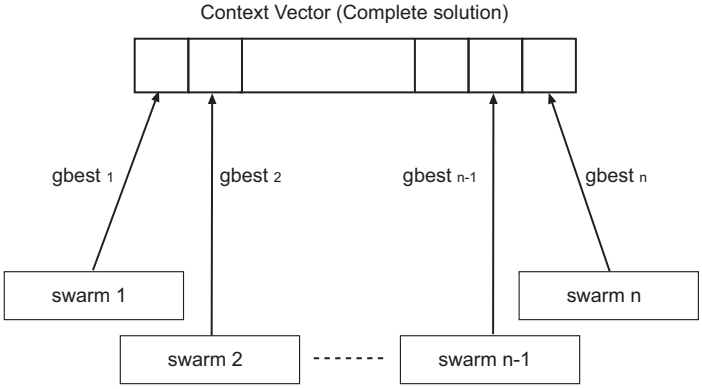
Ant colony optimisation (ACO) [171, 172], tabu search (TS) [173, 174], genetic algorithm [175, 176], stochastic diffusion search [3] and particle swarm optimisation [177, 178] are among the heuristic search methods that have investigated the use of the cooperative approach.

In a Cooperative Particle Swarm Optimiser (CPSO) [177], multiple swarms run in parallel mode while sharing information to explore the search space. As discussed in [179], there are a number of cooperative PSO algorithms:

- Standard Cooperative PSO (CPSO S) is based on partitioning the space into sub-spaces (see Fig 4.2). Therefore, instead of having one swarm

of s particles trying to optimise an n-dimensional vector, n swarms of 1-dimensional vectors are used, each optimising one component out of n. Since the function evaluating the particles still needs to be n-dimensional (rather than 1D), a context vector is constructed by concatenating the global best of each of the n swarms [177]. In order to update the fitness value of all particles in swarm j, all $n-1$ components in the context vector are kept constant (using the global bests of the other $n-1$ swarms), but allowing the j component to be updated by each particle in swarm j in turn. This model was originally proposed for the genetic algorithm [180].

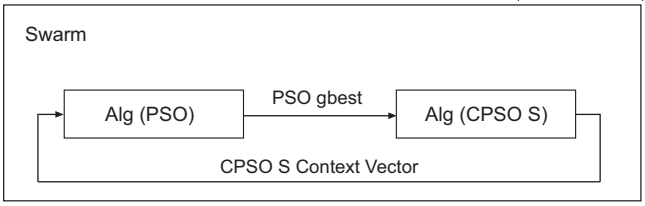
Figure 4.2: Standard Cooperative PSO (CPSO S)



- Hybrid Cooperative PSO (CPSO H) [177, 156] uses two consequent phases, with the first using CPSO S mode for one iteration while in the second phase, the standard PSO is run for one iteration and so on (see Fig 4.3). This mode helps utilising the fast convergence of CPSO S and benefiting PSO in escaping local minima.

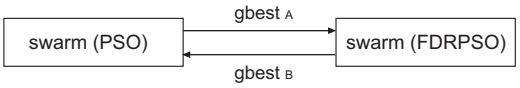
- Concurrent PSO (CONPSO) [178] is another type of cooperative PSO where two swarm optimisers run in parallel and frequently exchange their global bests to be compared in order to follow the best one (see Fig 4.4). In this mode, one swarm uses the standard PSO algorithm and

Figure 4.3: Hybrid Cooperative PSO (CPSO H)



the other applies the Fitness-to-Distance Ratio PSO (FDRPSO) [181]. This mode improves the performance of both of the PSO algorithms.

Figure 4.4: Concurrent PSO (CONPSO)

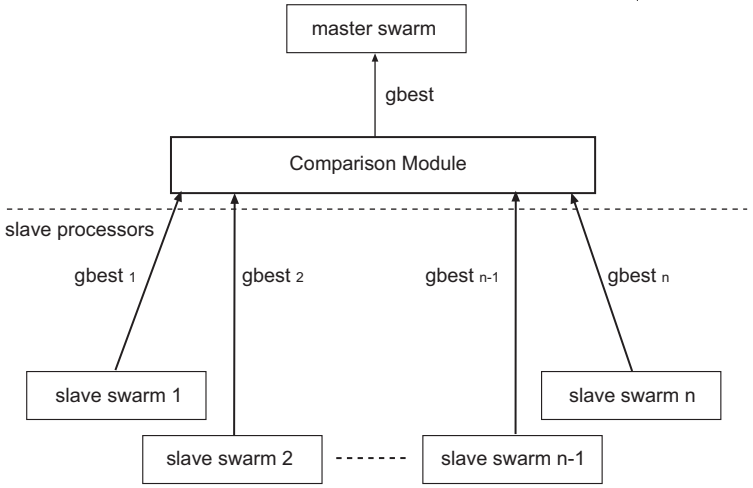


- Hierarchal Cooperative PSO [182] is based on having two swarms, one using CONPSO and the other using CPSO S models, both searching the problem space in parallel. This model (with multiple restarts in some cases) has been shown to outperform CPSO S, CPSO H and CONPSO models.
- Multi-population cooperative PSO (MCPSO)[183] is a master/slave approach, where the best of each slave swarm, which works in parallel with other slaves, reports back to the master swarm (see Fig 4.5). This value is integrated in the velocity update equation of the master swarm as a third component.

If swarms are not static and particles are allowed to move from one swarm to another during the optimisation, another type of PSO, nichePSO [149] which is a dynamic multi-swarm approach, is used.

- In Guaranteed Convergence PSO (GCPSO) [138], sub-swarms are created from the main original swarm, and it is possible for sub-swarms to attract other particles or to merge if they intersect.

Figure 4.5: Multi-population cooperative PSO (MCPSO)



- Another dynamic multi-swarm approach is presented in [184], where each sub-swarm utilises the local best neighbourhood approach and after a pre-specified number of iterations, particles are randomly associated to another sub-swarm, carrying their information along.

- In [185], swarms, which evolve in parallel, are compared after each iteration. In cases where the attractors of two swarms are close enough to each other, the swarm with less suitable attractors, re-initialises all its particles positions and velocities. Although the number of swarms can change, the number of particles and the number of allowed function evaluations are constant throughout the optimisation process. The optimal number of swarms is relevant to the number of optima.

4.1.4 Applications

Despite being relatively new, PSO has been applied to a diverse set of problems. This section briefly outlines some PSO applications (a more comprehensive set is reported in [186, 187]):

- Antennas design: the design of broadband antenna [101, 188], multi-band antennas for automotive rescue systems [189], near-field antenna

measurements [190], etc.

- Biological, medical, and pharmaceutical applications: human tremor analysis and cancer classification [191], identification of transcription factor binding sites in DNA, biometrics [192], etc.
- Control: power plants and systems control [193, 194], etc.
- Distribution networks: network reconfiguration and expansion [195], etc.
- Image and video: microwave imaging [196, 197], image registration [198], etc.
- Neural networks: neural network control for nonlinear processes [199], design of recurrent neural networks [200], etc.
- Signal processing: speech coding [201], etc.

PSO has also been used in electronics and electromagnetics (e.g. FPGA-based temperature control [202]), scheduling (e.g. flow shop scheduling [203]) and robotics (e.g. voice control of robots [204]).

PSO has recently been utilised (along with SDS) by the author for visualisation in [75, 74, 76] where computational creativity in the context of swarm intelligence is discussed.

4.2 Genetic Algorithm

This section gives a brief account to a simple variant of Genetic Algorithm (GA), which is used in Chapters 7 and 8.

The Genetic Algorithm is probably the most famous EA. This appendix introduces a simple real-valued GA which has previously shown to work well on real-world problems [205, 206]. The GA works in the following way: the individuals are first randomly initialised and their fitness is evaluated through

an objective function. Afterwards, in a iterative process, each individual has a probability of being exposed to recombination or mutation (or both). These probabilities are p_c and p_m respectively. The recombination operator used is arithmetic crossover and the mutation operator used is Cauchy mutation using an annealing scheme. At the end, in order to comb out the least fit individual, tournament selection [80] is often utilised.

The reason behind using Cauchy mutation operator vs. the well-known Gaussian mutation operator is the thick trails of the Cauchy distribution that allows it to generate considerable changes, more frequently, compared to the Gaussian distribution. The Cauchy distribution is defined by:

$$C(x, \alpha, \beta) = \frac{1}{\beta\pi\left(1 + \left(\frac{x-\alpha}{\beta}\right)^2\right)} \tag{4.12}$$

where $\alpha \leq 0$, $\beta > 0$, $-\infty < x < \infty$ (α and β are parameters that affect the mean and spread of the distribution). As specified in [206], all of the solution parameters are subject to mutation and the variance is scaled with $0.1 \times$ the range of the specific parameter in question.

In order to decrease the value of β as a function of the elapsed number of generations t, an annealing scheme was applied (α was set to 0):

$$\beta(t) = \frac{1}{1+t} \tag{4.13}$$

As for the arithmetic crossover, the offspring is generated as a weighted mean of each gene of the two parents:

$$\text{offspring}_i = r \times \text{parent1}_i + (1 - r) \times \text{parent2}_i \tag{4.14}$$

where offspring_i is the i'th gene of the offspring, and parent1_i and parent2_i refer to the i'th gene of the two parents, respectively. The weight r is drawn from a uniform distribution on the unit interval $U(0, 1)$.

In the experiments reported in Chapters 7 on page 122 and 8 on page 140, the probability of crossover and mutation of the individuals is set to $p_c = 0.7$

and $p_m = 0.9$ respectively. The tournament size of the tournament selection is set to two, and elitism with an elite size of one is deployed to maintain the best found solution in the population.

4.3 Differential Evolution Algorithm

This section briefly describes Differential Evolution (DE) for use in Chapters 7 and 8.

Differential Evolution, one of the most successful evolutionary algorithms (EAs), is a simple global numerical optimiser over continuous search spaces which was first introduced by Storn and Price [207, 208].

DE is a population based stochastic algorithm, proposed to search for an optimum value in the feasible solution space. The parameter vectors of the population are defined as follows:

$$x_i^g = \left[x_{i,1}^g, x_{i,2}^g, ..., x_{i,D}^g\right], i = 1, 2, ..., NP \tag{4.15}$$

where g is the current generation, D is the dimension of the problem space and NP is the population size. In the first generation, (when $g = 0$), the i^{th} vector's j^{th} component could be initialised as:

$$x_{i,j}^0 = x_{min,j} + r\left(x_{max,j} - x_{min,j}\right) \tag{4.16}$$

where r is a random number drawn from a uniform distribution on the unit interval $U(0, 1)$, and x_{min}, x_{max} are the lower and upper bounds of the j^{th} dimension, respectively. The evolutionary process (mutation, crossover and selection) starts after the initialisation of the population.

Mutation

At each generation g, the mutation operation is applied to each member of the population x_i^g (target vector) resulting in the corresponding vector v_i^g

(mutant vector). Among the five most frequently used mutation approaches are the following:

- DE/rand/1

$$v_i^g = x_{r_1}^g + F\left(x_{r_2}^g - x_{r_3}^g\right) \tag{4.17}$$

- DE/target-to-best/1

$$v_i^g = x_i^g + F\left(x_{best}^g - x_i^g\right) + F\left(x_{r_1}^g - x_{r_2}^g\right) \tag{4.18}$$

- DE/best/1

$$v_i^g = x_{best}^g + F\left(x_{r_1}^g - x_{r_2}^g\right) \tag{4.19}$$

- DE/best/2

$$v_i^g = x_{best}^g + F\left(x_{r_1}^g - x_{r_2}^g\right) + F\left(x_{r_2}^g - x_{r_3}^g\right) \tag{4.20}$$

- DE/rand/2

$$v_i^g = x_{r_1}^g + F\left(x_{r_2}^g - x_{r_3}^g\right) + F\left(x_{r_4}^g - x_{r_5}^g\right) \tag{4.21}$$

where r_1, r_2, r_3, r_4 are different from i and are distinct random integers drawn from the range $[1, NP]$. In generation g, the vector with the best fitness value is x_{best}^g; and F (which is set to 0.5) is a positive control parameter for constricting the difference vectors.

Crossover

Crossover operation improves population diversity through exchanging some components of v_i^g (mutant vector) with x_i^g (target vector) to generate u_i^g (trial vector). This process is led as follows:

$$u_{i,j}^{g} = \begin{cases} v_{i,j}^{g}, & \text{if } r \leq CR \text{ or } j = r_d \\ \\ x_{i,j}^{g}, & \text{otherwise} \end{cases} \tag{4.22}$$

where r is a uniformly distributed random number drawn from the unit interval $U(0,1)$, r_d is a randomly generated integer from the range $[1, D]$; this value guarantees that at least one component of the trial vector is different from the target vector. The value of CR (set to 0.5), which is another control parameter, specifies the level of inheritance from v_i^g (mutant vector).

Selection

The selection operation decides whether x_i^g (target vector) or u_i^g (trial vector) would be able to pass to the next generation $(g+1)$. In case of a minimisation problem, the vector with a smaller fitness value is admitted to the next generation:

$$x_i^{g+1} = \begin{cases} u_i^g, & \text{if } f(u_i^g) \leq f(x_i^g) \\ \\ x_i^g, & \text{otherwise} \end{cases} \tag{4.23}$$

where $f(x)$ is the fitness function.

Algorithm 4.2 summarises the behaviour of the DE algorithm.

DE, like other evolutionary algorithms, suffers from premature convergence where the populations lose their diversity too early and get trapped in local optima, therefore performing poorly on problems with high dimension and many local optima.

However, DE is known to be relatively good in comparison to other Evolutionary Algorithms (EAs) and Particle Swarm Optimisation (PSO) at avoiding premature convergence. In order to further reduce the risk of premature convergence in DE and to preserve population diversity, several methods have been proposed, among which are: multi-population approaches

Algorithm 4.2 DE Pseudo Code

```
01: Initialise population
02:
03: For ( generation = 1 to n )
04:    For ( agent = 1 to NP )
05:       Mutation : Generate Mutant Vector
06:       Crossover: Generate Trial  Vector
07:       Selection: Generate Target Vector
08:    End For
09:
10:    Find agent with best fitness value
11: End For
```

[209, 210, 211, 212, 213]; providing extra knowledge about the problem space [214, 215]; information storage about previously explored areas [216, 217]; utilising adapting and control parameters to ensure population diversity [218].

4.4 Summary

This chapter gives a background on Particle Swarm Optimisation and a brief description of Genetic Algorithm and Differential Evolution algorithm. The presented algorithms have many similarities such as the initialisation of the population and the use of fitness function as a way to evaluate the quality of each member of the population. The aim of this chapter is to introduce few algorithms as population-based optimisers and later (in Chapters 7 on page 122 and 8 on page 140) deploy them for integration with SDS algorithm.

Chapter 5

SDS as Global Optimiser

"Coming together is a beginning, staying together is progress, and working together is success."
– Henry Ford

This chapter builds an initial set of experiments aiming to investigate a scenario where SDS is utilised as a global optimiser; in these experiments, DE (see Section 4.3 on page 95) provides local search on convergence. The performance of DE is compared with the coupled SDS-DE algorithm and the results show the outperformance of the coupled algorithm over the classical DE algorithm.

5.1 The Coupled Algorithm

In the experiments reported, the optimisation process is initialised by n number of function evaluations (FEs) performed within the SDS test-diffusion cycle in order to allocate the resources (agents) to the promising areas of the search space and subsequently pass on the agents' positions to a Differential Evolution (DE) algorithm to resume the optimisation process. Hence, SDS is utilised as a global optimiser with DE providing local search on convergence.

The goal of this process is to verify whether the information diffusion and random restart mechanisms deployed in SDS may on their own improve DE behaviour. These are the results that are primarily reported here.

In this new architecture, a standard set of benchmarks is used to evaluate the performance of the coupled algorithm. The recruitment mechanism deployed in the diffusion phase of SDS is used to allocate resources after partially evaluating the search space.

Each DE agent has three vectors (target, mutant and trial vectors); and each SDS agent has one hypothesis and one status. In the experiment reported here (coupled algorithm), as stated before, SDS test-diffusion cycle is run for n FEs and then DE commences with the optimisation, taking its target vectors from SDS agents' positions.

The behaviour of the coupled algorithm in its simplest form is presented in Algorithm 5.1 on page 108.

5.2 Test and Diffusion Phases in the Coupled Algorithm

During the test-phase of a standard stochastic diffusion search, each agent has to partially evaluate its hypothesis. In the context of the coupled SDS-DE algorithm, in order to determine the activity of each agent, a simple test is used (as illustrated in Algorithm 5.1); the test-phase is simply conducted by comparing the fitness of each agent against that of a random one. If the selecting agent has a better fitness value, it will become active; otherwise it will be flagged inactive.

In the Diffusion Phase, each inactive agent picks another agent randomly. If the selected agent is active, the selected agent communicates its hypothesis to the inactive one; if the selected agent is also inactive, the selecting agent generates a new hypothesis at random from the search space.

As outlined in the pseudo-code of the coupled algorithm (see Algorithm 5.1),

after the initial n function evaluations (during which SDS's test-diffusion cycles iterate), DE algorithm should run[1].

The next section outlines the experiment setup and the results follow.

5.3 Experiments

In this section the performance of one variation of DE algorithm (DE/best/1) is contrasted against the coupled SDS-DE algorithm (sDE). The measures used to determine the quality of each algorithms are accuracy and reliability (see Section 5.3.1 for definitions).

5.3.1 Performance Measures

Three different performance measures [134] are used in the experiments conducted in this thesis. These performance measures are accuracy, reliability and efficiency.

Accuracy of the swarms is defined by the quality of the best position in terms of its closeness to the optimum position. If knowledge about the optimum position is known *a priori* (which is the case here), the following would define the accuracy:

$$\text{Accuracy} = \left| f\left(p_g^t\right) - f\left(x_{opt}\right) \right| \tag{5.1}$$

where p_g^t is the best position at time t and x_{opt} is the position of the known optimum solution.

If no information exists about the optimum solution, the fitness of the best position will be the accuracy of the swarm.

[1] We believe similar techniques can be applied to other swarm intelligence and evolutionary algorithms. For example, in [219], SDS is adopted for continuous global optimisation, using four benchmarks (each with different required accuracies and different maximum number of FEs allowed). In that experiment, SPSO [220] is utilised providing local search on convergence.

Another measure used is reliability which is the percentage of trials where swarms converge with a specified accuracy; this is defined by:

$$\text{Reliability} = \frac{n^{'}}{n} \times 100 \tag{5.2}$$

where n is the total number of trials in the experiment and $n^{'}$ is the number of successful trials.

Finally, efficiency is the number of iterations or objective function evaluations needed to converge with a specified accuracy (i.e. 10^{-8}):

$$\text{Efficiency} = \frac{1}{n} \sum_{i=0}^{n} FEs \tag{5.3}$$

where n is the total number of trials and FEs is the number of function evaluations before convergence.

5.3.2 Experiment Setup

The algorithms are tested over a number of standard benchmarking functions, preserving different dimensionality and modality (see Tables 5.1 and 5.2 for more information on the benchmarks used).

The first two functions (Sphere/Parabola and Schwefel 1.2) have a single minimum and are unimodal functions; Generalised Rosenbrock for dimension D, where $D > 3$, is multimodal; Generalised Schwefel 2.6, Generalized Rastrigin, Ackley, Generalized Griewank, Penalised Function P8 and Penalised Function P16 are complex high-dimensional multi-modal problems with many local minima and a single global optimum; Six-hump Camel-back, Goldstein-Price, Shekel 5, 7 and 10 are lower-dimensional multi-modal problems with fewer local minima. Goldstein-Price, Shekel 5, 7 and 10 have one global optimum and Six-hump Camel-back has two global optima symmetric about the origin.

The experiments are conducted with a population of 100 agents. The halting criterion for this experiment is when $300,000$ FEs is reached. There are 30

Table 5.1: Benchmark Functions Equations

Function	**Equation**
Sphere/Parabola	$f_1 = \sum_{i=1}^{D} x_i^2$
Schwefel 1.2	$f_2 = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$
Generalised Rosenbrock	$f_3 = \sum_{i=1}^{D-1} \left\{ 100 \left(x_{i+1} - x_i^2 \right)^2 + (x_i - 1)^2 \right\}$
Generalised Schwefel 2.6	$f_4 = -\sum_{i=1}^{D} x_i \sin\left(\sqrt{x_i}\right)$
Generalised Rastrigin	$f_5 = \sum_{i=1}^{D} \left\{ x_i^2 - 10\cos\left(2\pi x_i\right) + 10 \right\}$
Ackley	$f_6 = -20\exp\left\{ -0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2} \right\} - \exp\left\{ \frac{1}{D}\sum_{i=1}^{D} \cos\left(2\pi x_i\right) \right\} + 20 + e$
Generalised Griewank	$f_7 = \frac{1}{4000}\sum_{i=1}^{D} x_i^2 - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$
Penalized Function P8	$f_8 = \frac{\pi}{D}\left\{ 10\sin^2\left(\pi y_1\right) + \sum_{i=1}^{D-1} (y_i - 1)^2 \left\{ 1 + 10\sin^2\left(\pi y_{i+1}\right) \right\} + (y_D - 1)^2 \right\} + \sum_{i=1}^{D} \mu\left(x_i, 10, 100, 4\right)$ $y_i = 1 + \frac{1}{4}\left(x_i + 1\right)$ $\mu\left(x_i, a, k, m\right) = \begin{cases} k\left(x_i - a\right)^m & x_i > a \\ 0 & -a \le x_i \le a \\ k\left(-x_i - a\right)^m & x_i < -a \end{cases}$
Penalized Function P16	$f_9 = 0.1\left\{ \sin^2\left(3\pi x_1\right) + \sum_{i=1}^{D-1} (x_i - 1)^2 \left\{ 1 + \sin^2\left(3\pi x_{i+1}\right) \right\} + (x_D - 1)^2 \times \left\{ 1 + \sin^2\left(2\pi x_D\right) \right\} \right\} + \sum_{i=1}^{D} \mu\left(x_i, 5, 100, 4\right)$
Six-hump Camel-back	$f_{10} = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1 x_2 - 4x_2^2 + 4x_2^4$
Goldstein-Price	$f_{11} = \left\{ 1 + (x_1 + x_2 + 1)^2 \left(19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1 x_2 + 3x_2^2 \right) \right\} \times \left\{ 30 + (2x_1 - 3x_2)^2 \left(18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1 x_2 + 27x_2^2 \right) \right\}$
Shekel 5	$f_{12} = -\sum_{i=1}^{5} \left\{ \sum_{j=1}^{4} (x_j - a_{ij})^2 + c_i \right\}^{-1}$
Shekel 7	$f_{13} = -\sum_{i=1}^{7} \left\{ \sum_{j=1}^{4} (x_j - a_{ij})^2 + c_i \right\}^{-1}$
Shekel 10	$f_{14} = -\sum_{i=1}^{10} \left\{ \sum_{j=1}^{4} (x_j - a_{ij})^2 + c_i \right\}^{-1}$

independent runs for each benchmark function and the results are averaged over these independent trials.

The stopping condition for decreasing the error vectors is reaching $80,000$ FEs. DE is run after $100,000$ FEs until the termination criterion which is $300,000$ FEs. These values were selected merely to provide a brief initial exploration of the behaviour of the new coupled algorithm; no claim is made for their optimality.

5.3.3 Results

Table 5.3 shows the performance of the coupled algorithm (sDE) alongside DE algorithm. For each benchmark and algorithm, the above mentioned

Table 5.2: Benchmark Functions Details

	Function	**D**	**Feasible Bounds**	**Optimum**	**Initialisation**
f_1	Sphere/Parabola	30	$(-100, 100)^D$	0.0^D	$(50, 100)^D$
f_2	Schwefel 1.2	30	$(-100, 100)^D$	0.0^D	$(50, 100)^D$
f_3	Generalized Rosenbrock	30	$(-30, 30)^D$	1.0^D	$(15, 30)^D$
f_4	Generalized Schwefel 2.6	30	$(-500, 500)^D$	420.9687^D	$(250, 500)^D$
f_5	Generalized Rastrigin	30	$(-5.12, 5.12)^D$	0.0^D	$(2.56, 5.12)^D$
f_6	Ackley	30	$(-32, 32)^D$	0.0^D	$(16, 32)^D$
f_7	Generalized Griewank	30	$(-600, 600)^D$	0.0^D	$(300, 600)^D$
f_8	Penalized Function P8	30	$(-50, 50)^D$	-1.0^D	$(25, 50)^D$
f_9	Penalized Function P16	30	$(-50, 50)^D$	1.0^D	$(25, 50)^D$
f_{10}	Six-hump Camel-back	2	$(-5, 5)^D$	$(-0.0898, 0.7126)$, $(0.0898, -0.7126)$	$(2.5, 5)^D$
f_{11}	Goldstein-Price	2	$(-2, 2)^D$	$(0, -1)$	$(1, 2)^D$
f_{12}	Shekel 5	4	$(0, 10)^D$	4.0^D	$(7.5, 10)^D$
f_{13}	Shekel 7	4	$(0, 10)^D$	4.0^D	$(7.5, 10)^D$
f_{14}	Shekel 10	4	$(0, 10)^D$	4.0^D	$(7.5, 10)^D$

table and Figure 5.1 illustrate the accuracy measure.

As Tables 5.3 and 5.4 and show, over all benchmarks, other than f_7, DE algorithm does not significantly outperform the coupled algorithm. On the other hand, in most cases ($f_{5,6}$ and f_{8-14}), the coupled algorithm significantly outperforms the classical DE algorithm.

The Diffusion Phase of SDS algorithm is modified (see Algorithm 5.2) to investigate the SDS-led random restart effect caused by randomising a selection of agent hypotheses (effectively instantiating the population with SDS-led random-restarts). In other words, after the SDS test-phase, the hypothesis of each inactive agent is randomised.

As shown in Figure 5.1 and Tables 5.3 and 5.4, although information sharing plays an important role in the performance of the coupled algorithm, the significance of the SDS-led restart mechanism (in randomly restarting some of the agents) in improving the performance of the algorithm cannot be discarded.

In some cases ($f_{4,5,7}$), the SDS restart mechanism ($sReDE$) alone, which is facilitated by the test-phase of the SDS algorithm, demonstrates a signifi-

Table 5.3: Accuracy Details

Accuracy ± Standard Error is shown with two decimal places after 30 trials of 300,000 FEs. For each benchmark, the best algorithm(s) which is **significantly** better (see Table 5.4) than the others is highlighted. In cases where more than one algorithm is highlighted in a row, the highlighted algorithms do not significantly outperform each other.

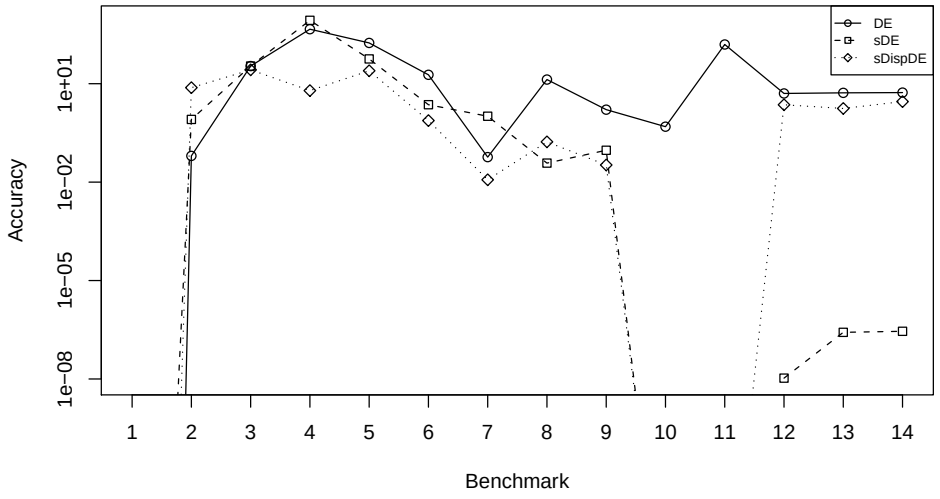
	DE	**sDE** ***SDS-DE***	**sReDE**
f_1	2.80E-78±2.65E-78	1.35E-37±1.06E-37	3.36E-54±2.01E-54
f_2	**6.31E-02±1.55E-02**	**8.15E-01±2.00E-01**	7.58E+00±1.55E+00
f_3	3.45E+01±8.04E+00	3.45E+01±4.52E+00	2.65E+01±4.08E+00
f_4	4.59E+02±1.31E+02	8.55E+02±2.44E+02	6.17E+00±1.10E+00
f_5	1.75E+02±8.18E+00	5.69E+01±1.80E+00	**2.48E+01±1.26E+00**
f_6	1.87E+01±8.84E-01	**2.29E+00±6.48E-02**	**7.52E-01±1.30E-01**
f_7	**5.79E-02±1.77E-02**	1.02E+00±4.68E-01	**1.18E-02±2.99E-03**
f_8	1.34E+01±2.94E+00	**3.80E-02±2.20E-02**	**1.69E-01±8.07E-02**
f_9	1.62E+00±3.56E-01	**9.36E-02±2.50E-02**	**3.33E-02±1.48E-02**
f_{10}	4.90E-01±7.42E-02	**1.04E-16±2.06E-17**	**1.18E-16±2.06E-17**
f_{11}	1.57E+02±4.21E+01	**0.00E+00±0.00E+00**	**5.92E-17±2.80E-17**
f_{12}	5.05E+00±7.38E-17	**1.06E-08±2.37E-09**	2.28E+00±4.90E-01
f_{13}	5.27E+00±0.00E+00	**2.64E-07±4.22E-08**	1.76E+00±4.63E-01
f_{14}	5.36E+00±9.99E-17	**2.84E-07±5.17E-08**	2.85E+00±5.37E-01

cantly better performance compared to the coupled algorithm. However, in several cases, the coupled algorithm outperforms the modified one: $f_{2,8}$ and f_{10-14}, out of which f_2 and f_{12-14} are performing significantly better.

Table 5.3 indicates that among the highlighted algorithms, out of 14 benchmarks, sDE exhibits the best performance (as it is among the most significant) in 9 cases; $sReDE$ and DE are among the best in 7 and 2 cases, respectively.

The results demonstrate the importance of coupling the SDS-led restart and the information sharing mechanisms which are both deployed in the SDS algorithm.

Figure 5.1: SDS as Global Optimiser; Accuracy Plot



Algorithm 5.2 Modified Algorithm – SDS Restart coupled with DE (*sReDE*)

```
01: // DIFFUSION PHASE
02: For ag = 1 to No_of_agents
03:    If ( !ag.activity() )
04:       ag.setHypo( randomHypo() )
05:    Else
06:       ag.setHypo(Gaussian(ag.getHypo(),aErrorV))
07:    End If
08: End For
```

5.4 Summary

This chapter presents the use of SDS as a Global Optimiser, with DE providing local search on convergence. The performance of DE is compared with the coupled SDS-DE algorithm and the results show the outperformance of the coupled algorithm over DE. This highlights the impact of the information sharing and SDS-led restart mechanisms deployed in SDS on the optimisation process.

Table 5.4: TukeyHSD Test Results for Accuracy

Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right algorithm is significantly better than the one, on the left.

	DE - sDE	**DE - sReDE**	**sDE - sReDE**
f_1	–	–	–
f_2	–	X–o	X–o
f_3	–	–	–
f_4	–	–	o–X
f_5	o–X	o–X	o–X
f_6	o–X	o–X	–
f_7	X–o	–	o–X
f_8	o–X	o–X	–
f_9	o–X	o–X	–
f_{10}	o–X	o–X	–
f_{11}	o–X	o–X	–
f_{12}	o–X	o–X	X–o
f_{13}	o–X	o–X	X–o
f_{14}	o–X	o–X	X–o

Algorithm 5.1 Coupled Algorithm

```
01: Initialise Agents
02:
03: x = initialInactiveErrorVector (e.g. 4)
04: y = initialActiveErrorVector   (e.g. 1)
05: // x > y
06:
07: n = SDS_FE_Allowed
08:
09: //SDS cycle
10: While ( FE <= n )
11: {
12:    // Decreasing the error vector over time
13:    If ( FE < stoppingErrV_DecreasePoint )
14:       iErrorV = x - (x*FE) / stoppingErrV_DecPoint
15:       aErrorV = y - (y*FE) / stoppingErrV_DecPoint
16:    End If
17:    // stoppingErrV_DecPoint < SDS_FE_Allowed
18:
19:    // TEST PHASE
20:    For ag = 1 to NP
21:       r_ag = pick-random-agent()
22:       If ( F(ag) < F(r_ag) )
23:          ag.setActivity (true)
24:       Else
25:          ag.setActivity (false)
26:       End If
27:    End For
28:
29:    // DIFFUSION PHASE
30:    For ag = 1 to NP
31:       If ( !ag.activity() )
32:          r_ag = pick-random-agent()
33:          If ( r_ag.activity() )
34:             ag.setHypo(
35:                Gaussian(r_ag.getHypo(),iErrorV))
36:          Else
37:             ag.setHypo( randomHypo() )
38:          End If
39:       End If
40:       Else
41:             ag.setHypo(Gaussian(ag.getHypo(),aErrorV))
42:    End for
43: }
44:
45: // DE
46: While ( FE < FE_Allowed )
47:    For ( Agent = 1 to NP )
48:       Mutation : generate mutant vector
49:       Crossover: generate trial vector
50:       Selection: generate target vector
51:    End For
52:    Find Agent with best fitness value
53: End For
```

Chapter 6

Bare Bones with Jumps PSO

"Our life is frittered away by detail ... Simplify, simplify."
– Henry Thoreau

Despite the simplicity of the update formula in Particle Swarm Algorithm (see Section 4.1 on page 72), the presence of many moving parts makes different aspects of the algorithm hard to understand (e.g. the effects of various parameters on the trajectory of the particle, particles' oscillation around constantly changing centres, the effects of swarm topology on its performance, etc.). In an attempt by Kennedy [135], a modified algorithm (Bare Bones PSO) is proposed where the velocity formula is eliminated from the update equation, aiming to understand some of these questions and identify the similarity it has with other stochastic population-based optimisers. In this chapter, after briefly explaining Bare Bones PSO, two new variants (Bare Bones with Jumps PSO Models 1 & 2) are introduced and their impact on improving the optimisation capability of conventional PSOs is investigated.

6.1 Bare Bones PSO

It is known that particles converge to a weighted average between their personal best and global (or neighbourhood) best positions [221, 222], but in order to understand the behaviour of particles, Kennedy [135] proposed a modified algorithm without the velocity formula in the update equation. Here is the update formula for the Bare Bones PSO (PSO-BB) algorithm:

$$x_{id} = g + \sigma_{id} N(0,1) \tag{6.1}$$
$$g = \frac{1}{2}(p_{id} + p_{gd}) \tag{6.2}$$
$$\sigma_{id} = |p_{id} - p_{gd}| \tag{6.3}$$

where $N(0,1)$ is the Gaussian distribution between 0 and 1. This update equation is used when the probability test – through generating a random number – is passed (e.g. $U(0,1) < 0.5$). Otherwise $x_{id} = p_{id}$. See Algorithm 6.1.

Algorithm 6.1 Bare Bones PSO (PSO-BB)

```
r = random number from U(0,1)
if r < 0.5
    x_id = p_id
else
    x_id = g + σ_id N(0,1)
```

Other variations of Bare Bones PSO are presented in Section 6.2 and their performance is contrasted against each other.

6.2 Bare Bones with Jumps PSO

In a similar attempt to bare bones PSO, Blackwell[1] also removed the velocity formula from the update process of the optimising algorithm and introduced

[1] The work is still in progress.

the following formula for what he called Bare Bones with Jumps PSO Model 1 (PSO-BBJ1):

$$x_{id} = g_i + \sigma_{id} N(0,1) \tag{6.4}$$

$$\sigma_{id} = \alpha \left| p_{i-1\,d} - p_{i+1\,d} \right| \text{ –for ring topology} \tag{6.5}$$

$$\sigma_{id} = \alpha \left| g_i - p_{id} \right| \text{ –for star topology} \tag{6.6}$$

where g_i is the neighbourhood best of particle i; α is an arbitrary number (theoretically shown to perform better when it is between 0.7 and 0.8) and $N(0,1)$ is the Gaussian distribution between 0 and 1.

In the re-initialisation mechanism which uses a randomly generated number, if $U(0,1) < 0.01$, the particle is re-initialised within its range, $U(-X_d, X_d)$. See Algorithm 6.2.

Algorithm 6.2 Bare Bones with Jumps PSO 1 (PSO-BBJ1)

```
r = random number from U(0,1)
if r < 0.01
    x_id = U(-X_d, X_d)
else
    x_id = g_i + σ_id N(0,1)
```

This method outperforms standard PSO and shows significant improvement in finding the optima (see Section 6.3.2).

In another experiment, a slightly different version – Bare Bones with Jumps PSO Model 2 (PSO-BBJ2) – has been proposed, with changes in the re-initialisation process as well as the update equation. In this algorithm (see Algorithm 6.3), re-initialisation is triggered if $U(0,1) < 0.001$. Otherwise the update equation is called using a newly defined Ω_{id}:

$$x_{id} = g_i + \Omega_{id} N(0,1)$$
$$\Omega_{id} = \alpha |g_i - x_{id}| \tag{6.7}$$

where g_i is the neighbourhood best of particle i; α is an arbitrary number which is set to 0.7 in this experiment, and $N(0,1)$ is the Gaussian distribution between 0 and 1.

Algorithm 6.3 Bare Bones with Jumps PSO 2 (PSO-BBJ2)

```
r = random number from U(0,1)
if r < 0.001
    x_id = U(-X_d, X_d)
else
    x_id = g_i + Ω_id N(0,1)
```

This algorithm is empirically shown (see Section 6.3) to outperform the standard PSO algorithm as well as the previously discussed PSO-BBJ1 algorithm in most cases. The experiment setup is presented in the following section accompanied by the results and the relevant statistical analysis.

6.3 Experiments

In this section, a number of experiments are carried out and the performance of two variations of PSO algorithms (PSO-BBJ 1 & 2) as well as Bare Bones PSO (PSO-BB) and standard PSO (PSO-CK) are contrasted. The measures used to determine the quality of each algorithms are accuracy, efficiency and reliability (see Section 5.3.1 on page 101 for definitions).

6.3.1 Experiment Setup

These algorithms are tested over a number of benchmarking functions from Jones et al. [223] and De Jong [224] test suite, preserving different dimen-

sionality and modality (see Tables 5.1 on page 103 and 5.2 on page 104).

In order not to initialise the particles on or near a region in the search space known to have the global optimum, *region scaling* technique is used [225], which makes sure particles are initialised at a corner of the search space where there are no optimal solutions.

The experiments are conducted with a population of 50 particles in global and local neighbourhoods independently. However, the halting criterion for this experiment is either to reach the optima (with distances less than 10^{-8}) or to exceed the $300,000$ function evaluations (FEs). There are 30 independent runs for each benchmarking function and results are averaged over these independent trials.

6.3.2 Results

In this experiment two types of neighbourhoods (global and local) are used and the algorithms (PSO-CK, PSO-BB, PSO-BBJ1 and PSO-BBJ2) are tested in both neighbourhoods.

The results are shown in the following tables and figures:

- Global neighbourhood:
  - Table 6.1 on page 116(a) reflects the accuracy of each algorithm over each function and reliability of each algorithms averaged over all benchmarks in global neighbourhood. Table 6.1(b) highlights any significant difference in the accuracy of the algorithms over each function.
  - Table 6.2 on page 117(a) shows the efficiency of each algorithm over each benchmark. Table 6.2(b) underlines any existing significant difference between any two algorithms over the benchmarks in the global neighbourhood.
  - Figure 6.1 on page 118 shows the plots for the accuracy and efficiency measures

- Local neighbourhood:
  - Table 6.3 on page 119 displays the results using the same measures (accuracy & reliability) as Tables 6.1 but in the local neighbourhood topology
  - Table 6.4 on page 120 displays the results using the same measure (efficiency) as Table 6.2 but in a local neighbourhood topology
  - Figure 6.2 on page 121 shows the plots for the accuracy and efficiency measures

Observing the reliability of the algorithms both in global and local neighbourhoods (see the last rows of Tables 6.1(a) and 6.3(a)), shows that on average PSO-BB is the least reliable algorithm (this finding does not come as a surprise as PSO-BB was proposed for understanding PSO rather than being deployed for optimisation problems; the result of this experiment confirms this view empirically). Among other algorithms, PSO-BBJ2 shows the most reliable performance in both local and global neighbourhood.

PSO-CK and PSO-BBJ1 show contradicting results in different neighbourhoods: PSO-BBJ1 is more reliable than PSO-CK in the global neighbourhood, but less reliable in the local neighbourhood.

In terms of the accuracy of the algorithms in the global neighbourhood (see Table 6.1(b)), PSO-BB shows significantly worse accuracy. When there exists convergence, in most cases, PSO-BBJ1 and PSO-BBJ2 outperform PSO-CK significantly. Over all benchmarks, PSO-BBJ1 and PSO-BBJ2 do not outperform each other significantly (except in one case, f_{11}).

As for the efficiency of the algorithms in the global neighbourhood (see Table 6.2), when there exists a significant difference PSO-BBJ2 outperform all algorithms over all benchmarks significantly. The second best algorithm is PSO-BBJ1.

In the local neighbourhood (see Table 6.3), compared to other algorithms, PSO-BB and PSO-BBJ1, are significantly worse in terms of accuracy. When

functions with convergence are considered, PSO-BBJ2 outperform other algorithms, but PSO-CK shows better accuracy over all benchmarks.

In terms of efficiency in the local neighbourhood (see Table 6.4), in functions with successful convergence, PSO-BBJ1 is the least efficient and PSO-BBJ2 is the most efficient algorithms. As for PSO-CK, it is outperformed by PSO-BB in all significant cases, except in f_{11}.

6.4 Summary

This chapter briefly describes Bare Bones PSO which was proposed to provide better understanding of the behaviour of particle swarm algorithms. Although this algorithm does not intend to enhance the optimisation capability of standard PSO, the new variations proposed in this chapter (Bare Bones with Jumps PSO 1 & 2) offer promising results. The results are investigated using three measures (i.e. accuracy, efficiency and reliability).

In brief, in terms of accuracy, although PSO-CK demonstrates a better performance when all benchmarks are considered, the accuracy of PSO-BBJ2 compared to other algorithms is significantly better when benchmarks with successful convergence are considered.

Additionally, PSO-BBJ2 is empirically shown to be *both* the most efficient and the most reliable algorithm in both local and global neighbourhoods. PSO-BBJ2 shows better reliability in global vs. local neighbourhood, which is not always the expectation (see section 8.5 on page 149 for some criticisms on the use of global neighbourhood in PSO).

A theoretical analysis is required (which is an ongoing process) to better understand how such results are obtained in these variants of minimal PSO algorithm.

Table 6.1: Accuracy Details; Global Neighbourhood

(a) Accuracy± Standard Error is shown with two decimal places after 30 trials of 300,000 function evaluations. Total number of convergence of each algorithm over each benchmark is shown in brackets after the accuracy and standard error. Total number of convergence of each algorithm over the benchmarks can be found in the last row.

	PSO-CK	**PSO-BB**	**PSO-BBJ1**	**PSO-BBJ2**
f_1	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)
f_2	**0.0 ±0.0** (30)	6.34E+03±4.69E+02 (0)	8.51E-04±7.86E-04 (26)	**0.0 ±0.0** (30)
f_3	9.14E+00±3.18E+00 (0)	5.86E+01±1.80E+01 (0)	1.08E+01±4.47E+00 (0)	1.28E-06±6.09E-07 (7)
f_4	3.60E+03±8.50E+01 (0)	3.46E+03±2.29E+01 (0)	8.32E-02±1.43E-02 (0)	**0.0 ±0.0** (30)
f_5	6.33E+01±2.57E+00 (0)	1.59E+02±4.93E+00 (0)	9.93E-03±3.37E-03 (0)	**0.0 ±0.0** (30)
f_6	1.17E+00±1.95E-01 (10)	1.92E+01±8.43E-02 (0)	2.07E-05±1.69E-05 (20)	**0.0 ±0.0** (30)
f_7	2.88E-02±6.13E-03 (7)	9.40E-02±3.39E-02 (4)	4.42E-02±7.18E-03 (3)	3.37E-02±6.43E-03 (7)
f_8	6.22E-02±2.03E-02 (19)	4.16E+00±1.36E+00 (8)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)
f_9	3.00E-02±1.44E-02 (24)	4.13E+00±3.23E+00 (15)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)
f_{10}	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)
f_{11}	**0.0 ±0.0** (30)	4.86E+01±7.37E+00 (12)	1.89E+01±6.36E+00 (23)	4.32E+01±7.50 (14)
f_{12}	1.85E+00±4.97E-01 (0)	5.05E+00±0.00E+00 (0)	5.05E+00±7.38E-17 (0)	5.05E+00±1.13E-16 (0)
f_{13}	2.39E+00±5.95E-01 (0)	5.27E+00±3.01E-17 (0)	5.35E+00±7.92E-02 (0)	5.27E+00±8.52E-17 (0)
f_{14}	1.11E+00±4.68E-01 (0)	5.36E+00±6.02E-17 (0)	5.36E+00±9.03E-17 (0)	5.36E+00±9.52E-17 (0)
$\sum$	(180)	(99)	(198)	(268)
	42.68%	23.57%	47.14%	63.81%

(b) Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right one is significantly better than the left algorithm.

	BBJ1-BB	**BBJ2-BB**	**CK-BB**	**BBJ2-BBJ1**	**CK-BBJ1**	**CK-BBJ2**
f_1	–	–	–	–	–	–
f_2	X – o	X – o	X – o	–	–	–
f_3	X – o	X – o	X – o	–	–	–
f_4	X – o	X – o	–	–	o – X	o – X
f_5	X – o	X – o	X – o	–	o – X	o – X
f_6	X – o	X – o	X – o	–	o – X	o – X
f_7	–	–	–	–	–	–
f_8	X – o	X – o	X – o	–	–	–
f_9	–	–	–	–	–	–
f_{10}	–	–	–	–	–	–
f_{11}	X – o	–	X – o	o – X	–	X – o
f_{12}	–	–	X – o	–	X – o	X – o
f_{13}	–	–	X – o	–	X – o	X – o
f_{14}	–	–	X – o	–	X – o	X – o

Table 6.2: Efficiency Details; Global Neighbourhood

(a) Mean FEs ± Standard Error is shown with two decimal places after 30 trials of 300,000 function evaluations.

	PSO-CK	**PSO-BB**	**PSO-BBJ1**	**PSO-BBJ2**
f_1	23224±194	12262±164	13270±148	22685±119
f_2	–	160358±2920	89637±575	191064±1290
f_3	–	–	–	276020±7039
f_4	–	–	–	63399±3805
f_5	–	124701±12900	124701±12900	54825±3182
f_6	–	41811±870	37004±318	47486±2226
f_7	22786±259	11518±136	13807±335	24006±259
f_8	44735±567	20194±1701	15013±285	33627±744
f_9	49228±1309	39656±3719	18855±981	31147±720
f_{10}	1458±17	516±4	551±5	3515±37
f_{11}	5876±397	61199±11951	663±10	3929±39
f_{12}	–	–	–	–
f_{13}	–	–	–	–
f_{14}	–	–	–	–

(b) Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right one is significantly better than the left algorithm.

	BBJ1-BB	**BBJ2-BB**	**CK-BB**	**BBJ2-BBJ1**	**CK-BBJ1**	**CK-BBJ2**
f_1	X – o	X – o	–	–	o – X	o – X
f_2	NP	NP	NP	X – o	o – X	o – X
f_3	NP	NP	NP	NP	NP	NP
f_4	NP	NP	NP	NP	NP	NP
f_5	NP	NP	NP	X – o	NP	NP
f_6	NP	NP	NP	X – o	o – X	o – X
f_7	X – o	X – o	–	–	o – X	o – X
f_8	X – o	X – o	–	–	o – X	o – X
f_9	–	X – o	–	X – o	–	–
f_{10}	X – o	X – o	o – X	–	o – X	o – X
f_{11}	o – X	–	–	X – o	X – o	–
f_{12}	NP	NP	NP	NP	NP	NP
f_{13}	NP	NP	NP	NP	NP	NP
f_{14}	NP	NP	NP	NP	NP	NP

Figure 6.1: PSO Bare Bones Variants; Global Neighbourhood Plots

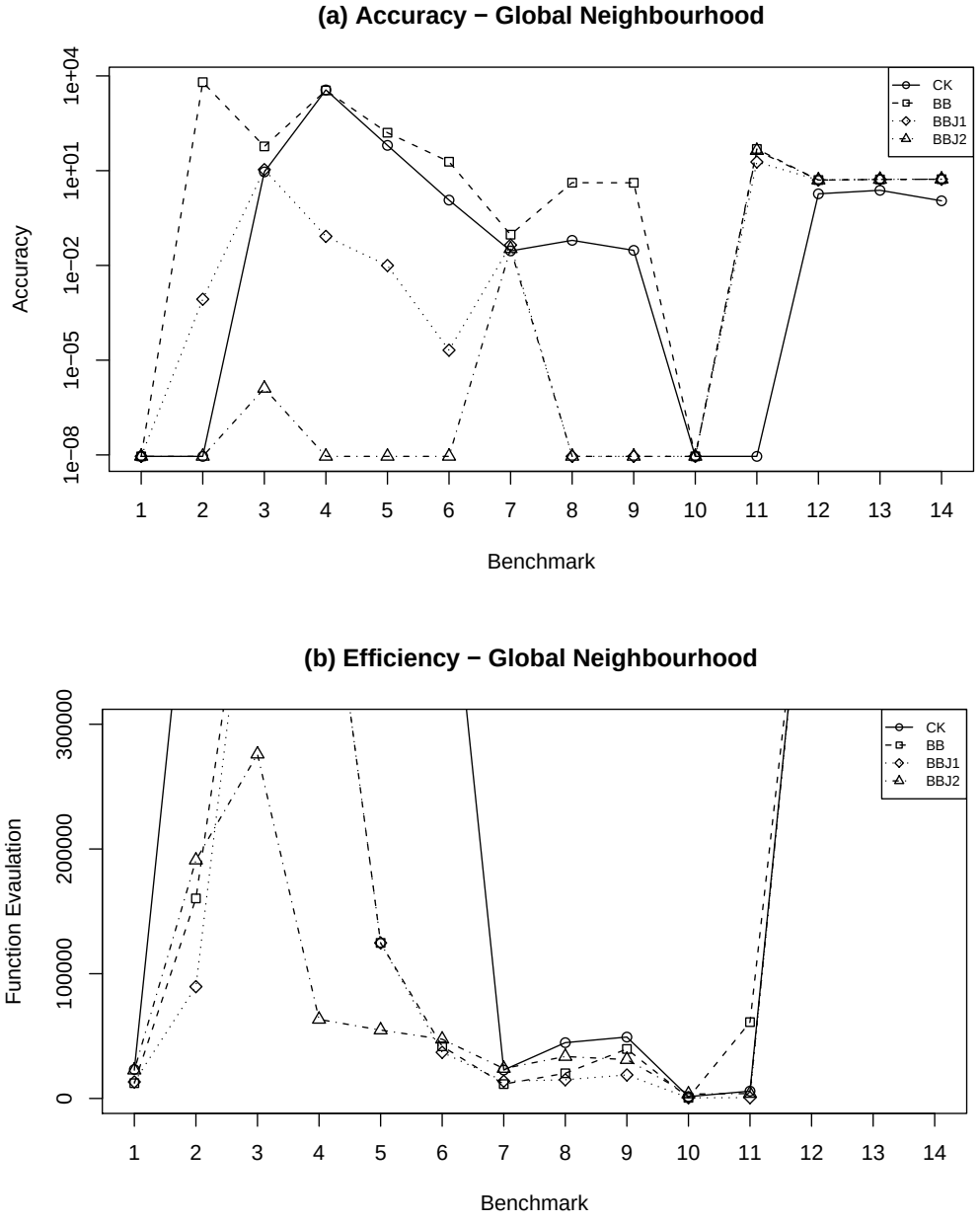


Table 6.3: Accuracy Details; Local Neighbourhood

(a) Accuracy ± Standard Error is shown with two decimal places after 30 trials of 300,000 function evaluations. Total number of convergence of each algorithm over each benchmark is shown in brackets after the accuracy and standard error. Total number of convergence of each algorithm over the benchmarks can be found in the last row.

	PSO-CK	**PSO-BB**	**PSO-BBJ1**	**PSO-BBJ2**
f_1	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)
f_2	7.84E-02±1.09E-02 (0)	9.66E+01±8.68E+00 (0)	3.93E+02±4.38E+01 (0)	1.87E-01±3.02E-02 (0)
f_3	1.33E+01±3.73E+00 (0)	1.27E+01±5.50E-01 (0)	2.88E+01±3.20E+00 (0)	2.59E+01±5.73E+00 (0)
f_4	4.14E+03±7.11E+01 (0)	3.26E+03±3.10E+01 (0)	1.92E+03±6.89E+01 (0)	**0.0 ±0.0** (30)
f_5	5.87E+01±1.88E+00 (0)	2.46E+01±3.04E+00 (0)	9.22E+01±4.47E+00 (0)	**0.0 ±0.0** (30)
f_6	**0.0 ±0.0** (30)	1.96E+01±2.24E-02 (0)	1.89E-06±1.55E-06 (26)	**0.0 ±0.0** (30)
f_7	1.07E-03±6.10E-04 (27)	1.41E-05±1.04E-05 (21)	2.48E-04±2.46E-04 (26)	1.19E-02±2.96E-03 (12)
f_8	**0.0 ±0.0** (30)	2.76E-02±1.92E-02 (28)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)
f_9	**0.0 ±0.0** (30)	5.27E-02±5.27E-02 (29)	3.62E-07±2.84E-07 (28)	**0.0 ±0.0** (30)
f_{10}	**0.0 ±0.0** (30)	8.16E-02±4.55E-02 (27)	**0.0 ±0.0** (30)	**0.0 ±0.0** (30)
f_{11}	**0.0 ±0.0** (30)	7.92E+01±2.71E+01 (10)	1.27E-05±1.27E-05 (29)	2.79E+01±7.03E+00 (19)
f_{12}	3.70E-06±1.27E-07 (1)	5.05E+00±0.00E+00 (0)	5.05E+00±0.00E+00 (0)	5.05E+00±4.26E-17 (0)
f_{13}	1.22E-04±0.00E+00 (0)	5.27E+00±0.00E+00 (0)	5.10E+00±1.76E-01 (0)	5.27E+00±0.00E+00 (0)
f_{14}	1.26E-04±1.12E-16 (0)	5.36E+00±5.22E-17 (0)	5.18E+00±1.79E-01 (0)	5.36E+00±1.09E-16 (0)
$\sum$	(208)	(145)	(199)	(241)
	49.52%	24.52%	47.38%	57.38%

(b) Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right one is significantly better than the left algorithm.

	BBJ1-BB	**BBJ2-BB**	**CK-BB**	**BBJ2-BBJ1**	**CK-BBJ1**	**CK-BBJ2**
f_1	–	–	–	–	–	–
f_2	o – X	X – o	X – o	X – o	X – o	–
f_3	o – X	–	–	–	X – o	–
f_4	X – o	X – o	o – X	X – o	o – X	o – X
f_5	o – X	X – o	o – X	X – o	X – o	o – X
f_6	X – o	X – o	X – o	–	–	–
f_7	–	o – X	–	o – X	–	X – o
f_8	–	–	–	–	–	–
f_9	–	–	–	–	–	–
f_{10}	–	–	–	–	–	–
f_{11}	X – o	–	X – o	–	–	–
f_{12}	–	–	X – o	–	X – o	X – o
f_{13}	–	–	X – o	–	X – o	X – o
f_{14}	–	–	X – o	–	X – o	X – o

Table 6.4: Efficiency Details; Local Neighbourhood

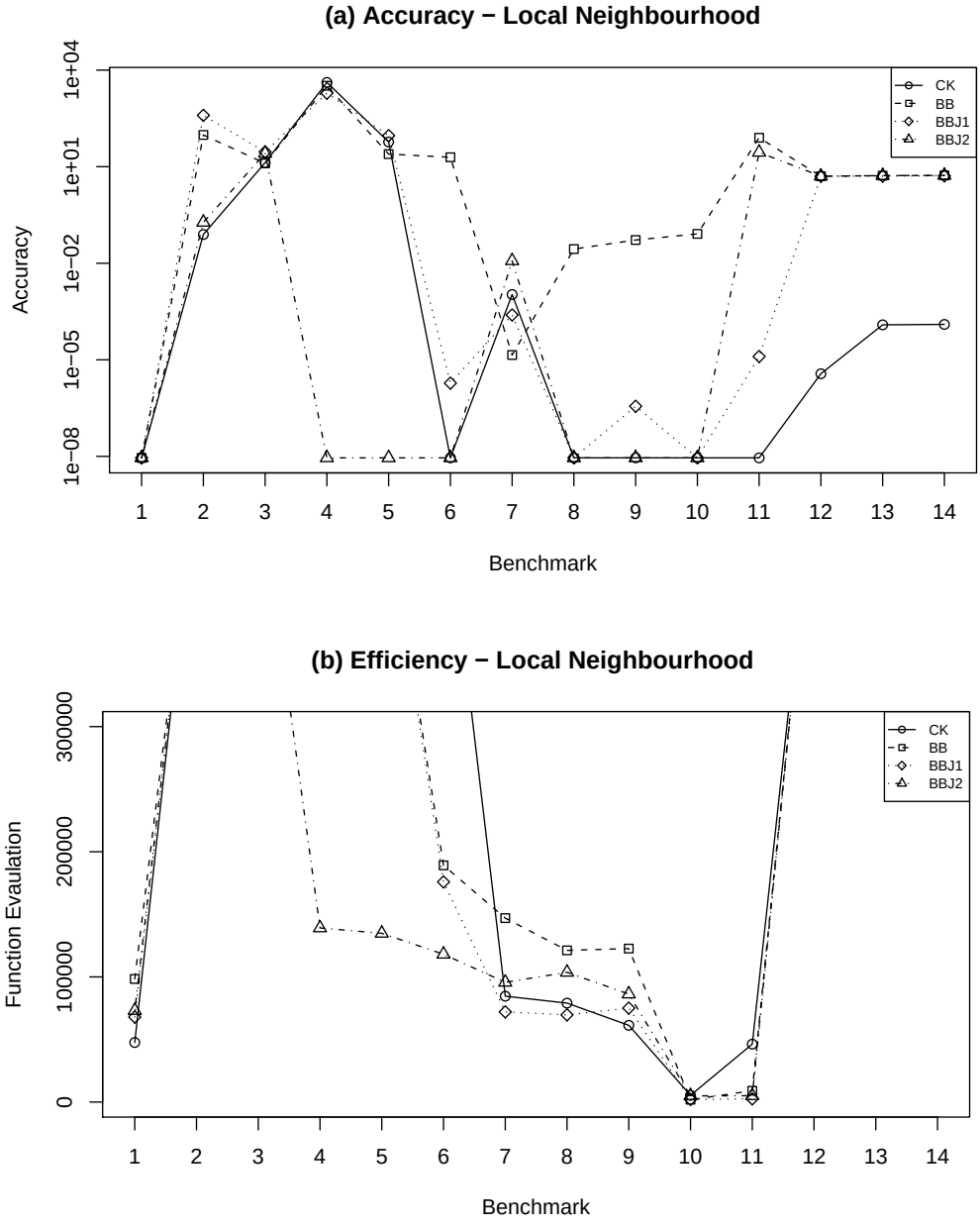
(a) Mean FEs ± Standard Error is shown with two decimal places after 30 trials of 300,000 function evaluations.

	PSO-CK	**PSO-BB**	**PSO-BBJ1**	**PSO-BBJ2**
f_1	47589±97	98383±327	67968±213	73090±196
f_2	–	–	–	–
f_3	–	–	–	–
f_4	–	–	–	139118±3975
f_5	–	–	–	134816±2801
f_6	–	189139±4687	175902±944	118098±389
f_7	84612±4962	146979±4494	72048±332	95680±4051
f_8	79067±765	121186±1035	69658±489	103658±1287
f_9	61328±374	122631±853	75080±392	86281±480
f_{10}	5389±100	1891±31	2161±161	4935±53
f_{11}	46300±2012	9030±2367	2536±75	5063±51
f_{12}	–	–	–	8895±0
f_{13}	–	–	–	–
f_{14}	–	–	–	–

(b) Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right one is significantly better than the left algorithm.

	BBJ1-BB	**BBJ2-BB**	**CK-BB**	**BBJ2-BBJ1**	**CK-BBJ1**	**CK-BBJ2**
f_1	o – X	o – X	o – X	X – o	X – o	o – X
f_2	NP	NP	NP	NP	NP	NP
f_3	NP	NP	NP	NP	NP	NP
f_4	NP	NP	NP	NP	NP	NP
f_5	NP	NP	NP	NP	NP	NP
f_6	NP	NP	NP	X – o	X – o	X – o
f_7	o – X	–	–	X – o	X – o	-
f_8	o – X	X – o	o – X	X – o	X – o	o – X
f_9	o – X	o – X	o – X	X – o	X – o	o – X
f_{10}	X – o	X – o	–	–	o – X	o – X
f_{11}	X – o	X – o	X – o	–	–	–
f_{12}	NP	NP	NP	NP	NP	NP
f_{13}	NP	NP	NP	NP	NP	NP
f_{14}	NP	NP	NP	NP	NP	NP

Figure 6.2: PSO Bare Bones Variants; Local Neighbourhood Plots



Chapter 7

Merging SDS with PSO and DE

"Not everything that counts can be counted, and not everything that can be counted counts."
– Albert Einstein

This chapter explores the first attempts on the integration of SDS with PSO, and SDS with DE, with the intention of utilising the information sharing mechanism in SDS. This chapter reports the outcome of the research [226, 227, 228] which applies the resource allocation mechanism deployed in Stochastic Diffusion Search to the Particle Swarm Optimiser and Differential Evolution metaheuristics for the first time, effectively merging a nature inspired swarm intelligence algorithm (SDS) with other swarm intelligence algorithms, PSO and DE independently. The results reported herein suggest that the hybrid algorithm, exploiting information sharing between particles/agents, has the potential to improve the optimisation capability of conventional PSOs and classical DE.

The experiments conducted in this chapter seek to investigate whether the information diffusion mechanism deployed in SDS may on its own improve PSO or DE behaviour. It is these results that are primarily reported.

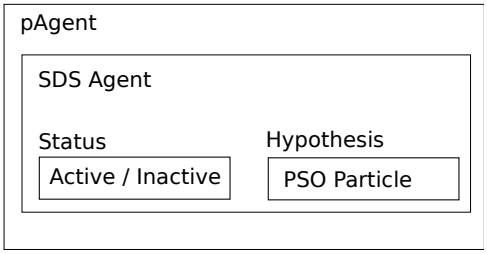
In this new architecture, a standard set of benchmarks are used to evaluate the performance of the hybrid algorithm. The resource allocation (or recruitment) sides of SDS are used to assist allocating resources (e.g. particles of the swarm, members of the DE population, etc.) after partially evaluating the search space.

7.1 Merging SDS with PSO

In the hybridised algorithm of SDS and PSO, each PSO particle has a current position, a memory (personal best position) and a velocity; each SDS agent, on the other hand, has a hypothesis and a status.

In the experiment reported here, every PSO particle is an SDS agent too – together termed *pAgents*. Within each pAgent, the SDS hypothesis is defined by the PSO particle, and an additional Boolean variable (status) determines whether the pAgent is active or inactive (see Figure 7.1).

Figure 7.1: Architecture of pAgent



The behaviour of the hybrid algorithm in its simplest form is presented in Algorithm 7.1 on page 131.

In the context of the hybrid algorithm, there are a number of different tests that could be performed in order to determine the activity of each pAgent. A very simple test is illustrated in Algorithm 7.1. Here, the test-phase is simply conducted by comparing the fitness of each pAgent's particle's personal best against that of a random pAgent; if the selecting pAgent has a better fitness value, it will become active, otherwise it will be flagged inactive. On

average, this mechanism will ensure that 50% of pAgents remain active from one iteration to another. In standard SDS such high average activity would not be useful as it entails most agents will continue to exploit their current hypothesis rather than explore the search space, however in the hybrid algorithm the randomised subsequent behaviour of each pAgent offsets this effect.

As outlined in the pseudo-code of the hybrid algorithm (see Algorithm 7.1), after each n number of PSO function evaluations, one full SDS cycle is executed. A full SDS cycle includes:

- one Test Phase which decides about the status of each pAgent, one after another
- one Diffusion Phase which shares information according to the algorithm presented

The hybrid algorithm is called $SDSnPSO$, where n refers to the number of PSO function evaluations before an SDS cycle should run.

7.1.1 Experiments

In this section, a number of experiments are carried out and the performance of PSO is contrasted against the hybrid algorithm, $SDSnPSO$. The measures used to determine the quality of each algorithms are accuracy, efficiency and reliability (see Section 5.3.1 on page 101 for definitions).

7.1.1.1 Experiment Setup

The algorithms are tested over a number of benchmarking functions from Jones et al. [223] and De Jong [224] test suite, preserving different dimensionality and modality (see Tables 5.1 on page 103 and 5.2 on page 104, where benchmark function equations, feasible bounds, the number of dimensions in which the benchmarks are used in the experiments, the optimum of each

function which is known *a priori* and also the boundaries where particles are first initialised are presented).

In order not to initialise the particles on or near a region in the search space known to have the global optimum, the *region scaling* technique is used [225] which makes sure particles are initialised at a corner of the search space where there are no optimal solutions.

The experiments are conducted with a population of 50 particles in the global neighbourhood. The halting criterion for this experiment is either to reach the optima (with distances less than 10^{-8}) or to exceed the $300,000$ function evaluations (FEs). There are 30 independent runs for each benchmark function and the results are averaged over these independent trials.

In this section, $SDSnPSO$ is presented with few variations of parameter, n, (the number of PSO function evaluation before an SDS cycle is performed), $n = 1000$, 3000, and $30,000$. These values were selected merely to provide a brief initial exploration of the behaviour of the new hybrid algorithm over three relatively widely separated parameter values; no claim is made for their optimality.

7.1.1.2 Results

Table 7.1 on page 132 and Figure 7.2 on page 133 illustrate the performance of the various hybrid algorithms alongside PSO-CK. For each benchmark and each algorithm, the table shows the accuracy, efficiency and reliability.

Although the focus of this study is not finding the best n for $SDSnPSO$ (for this set of benchmarks), $n = 3000$ shows better results compared to other variants. The results table suggests that over-running the SDS cycle (e.g. when $n = 1000$) might move the swarm away from convergence. On the other hand, reducing information sharing (e.g. when $n = 30,000$) appears to reduce the positive effect that SDS has on the overall behaviour of the swarm.

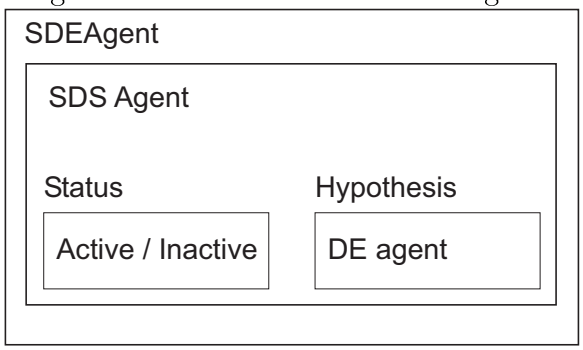
As Table 7.1 shows (for statistical details, see Tables 7.3 on page 137 and 7.4 on page 138), there is a trade-off between the reliability and the efficiency

measures of $SDSnPSO$ and PSO. Adding SDS, decreases the efficiency, but increases the reliability. This can be viewed in f_{1-2} and f_{6-9}. In terms of the total number of convergences, $SDSnPSO$ ($n = 3000$), outperforms PSO (a more detailed comparison and statistical analysis of the results are presented in section 7.3).

7.2 Merging SDS with DE

In the hybridised algorithm of SDS and DE, each DE agent has three vectors (target, mutant and trial vectors); and each SDS agent has one hypothesis and one status. In the experiment reported here (hybrid algorithm), every member of DE population is an SDS agent too – together termed *SDEAgents*. In SDEAgents, each SDS hypothesis is defined by a DE agent, and an additional Boolean variable (status) determining whether the SDEAgent is active or inactive (see Figure 7.3). The behaviour of the hybrid algorithm in its simplest form is presented in Algorithm 7.2 on page 134.

Figure 7.3: Architecture of SDEAgent



Similarly to the hybrid SDS-PSO algorithm, in the context of the hybrid SDS-DE algorithm, there are many different tests that could be performed in order to determine the activity of each SDEAgent. A simple test is illustrated in Algorithm 7.2. Here, the test-phase is simply conducted by comparing the fitness of each SDEAgent's target vector against that of a random SDEAgent;

if the selecting SDEAgent has a better fitness value, it will become active, otherwise it will be flagged inactive.

As outlined in the pseudo-code of the hybrid algorithm (see Algorithm 7.2), after each n generations, one full SDS cycle is executed. The hybrid algorithm is called $SDSnDE$, where n refers to the number of generations before an SDS cycle should run.

7.2.1 Experiments

In this section, a number of experiments are carried out and the performance of one variation of DE algorithm (DE/best/1), which is described in 4.3 on page 95, is contrasted against the hybrid algorithm, $SDSnDE$. The measures used to determine the quality of each algorithms are accuracy and reliability (see Section 5.3.1 on page 101 for definitions).

7.2.1.1 Experiment Setup

This experiment uses the same benchmarks introduced earlier (see Tables 5.1 on page 103 and 5.2 on page 104). In order not to initialise the DE agents on or near a region in the search space known to have the global optimum, the *region scaling* technique is used here too.

The experiments are conducted with a population of 100 agents. The halting criterion for this experiment is when the number of generations reaches $2,000$. There are 30 independent runs for each benchmark function and the results are averaged over these independent trials.

In this section, $SDSnDE$ is presented with few variations of parameter, n (the number of generations before an SDS cycle is performed): $n = 5$, 50, and 200. These values were selected merely to provide a brief initial exploration of the behaviour of the new hybrid algorithm over three relatively widely separated parameter values; no claim is made for their optimality.

7.2.1.2 Results

Table 7.2 on page 135 and Figure 7.4 on page 136 show the accuracy performance of the various hybrid algorithms alongside the DE algorithm.

Similarly to the previous experiment, the focus of this experiment is not finding the best n for $SDSnDE$ (for this set of benchmarks), but rather investigate the effect of the SDS algorithm on the performance of the DE algorithm.

As Table 7.5 on page 139 shows, over all benchmarks other than f_2 in $(DE-H5)$, the DE algorithm does not significantly outperform any of the hybrid algorithms SDSnDE ($n = 5, 50, 200$). On the other hand, in most cases (e.g. f_{3-6}, f_8 and f_{10-14}), the hybrid algorithms outperform the classical DE algorithm significantly.

As detailed in Table 7.2, in f_{1-3}, f_{11}, the performance of H5, which has the highest rate of information exchange, is weaker than the other hybrid algorithms with lower rate information sharing. This implies that the performance of some problems might be negatively affected by excessive information exchange (e.g. in f_1, $F_{H5} > F_{H50} > F_{H200}$, where F is the fitness value).

However, in another set of problems, a higher rate of information exchange (more communication between the agents) results in better outcomes (e.g. f_{4-6}, f_{8-9}, f_{12-14}). More specifically, in f_{4-6} and f_{12-14} less communication between the agents corresponds to a poorer performance of the hybrid algorithms ($F_{H5} < F_{H50} < F_{H200}$).

This demonstrates the importance of deploying the right frequency of communication and information exchange, depending on the problem.

7.3 Discussion

To further analyse the role of SDS in the hybrid algorithms, the Diffusion Phase of the SDS algorithm is modified (see Algorithm 7.3) to investigate

the SDS-led random restart effect caused by randomising a selection of agent hypotheses after a number of function evaluations / iterations. In other words, after the SDS test-phase, the hypothesis of each inactive hybrid agent is randomised.

Algorithm 7.3 Modified Hybrid Algorithm

```
01 // DIFFUSION PHASE
02 For ag = 1 to No_of_agents
03   If ( !ag.activity() )
04     ag.setHypo( randomHypo() )
05   End If
06 End For
```

7.3.1 Modified SDSnPSO Algorithm

The performance of the modified hybrid algorithm can be contrasted against PSO using the three performance measures (accuracy, efficiency and reliability) defined in Section 6.3.1 on page 112. TukeyHSD test is used for accuracy and efficiency measures (see Tables 7.3 on page 137 and 7.4 on page 138).

In terms of accuracy, Tables 7.1 and 7.3 illustrate that no algorithm outperforms all benchmarks. However, Table 7.4 shows that in the case of successful convergence, whenever there is a significant difference between any pair of the algorithms, the efficiency of H3M is significantly worse than PSO-CK and the hybrid algorithms (H1, H3 & H30); and as the last row of Table 7.1 proves, the control algorithm is less reliable than PSO and the hybrid algorithms (H1, H3 & H30). As the efficiency and reliability of the control experiment (see the last column in Table 7.1) is worse than that of the hybrid algorithm, we can conclude that the SDS information sharing mechanism must play an essential role in improving the performance of the hybrid algorithm.

7.3.2 Modified SDSnDE Algorithm

Similarly to SDSnPSO, in order to further analyse the role of SDS in the SDSnDE hybrid algorithm, the diffusion phase of SDS algorithm is modified

(see Algorithm 7.3) to investigate the SDS-led restart effect, where after the SDS test-phase, the hypothesis of each inactive SDEAgent is randomised.

As detailed in Table 7.2 on page 135, although information sharing plays an important role in the performance of the hybrid DE algorithm, the significance of SDS-led random restart (in randomly restarting some of the agents) in improving the performance of DE algorithm cannot be discarded.

In few cases ($f_{3,4,8}$), solely the restart (H50M), which is facilitated by the test-phase of the SDS algorithm, demonstrates a slightly better performance compared to the hybrid algorithm (see Table 7.2). However, in the majority of cases, the hybrid algorithms outperform the modified algorithm: $f_{1,2}, f_{5-7}, f_9, f_{12-14}$, out of which f_9 and f_{12-14} are performing significantly better (see Table 7.5 on page 139). Also it is shown that the algorithm with modified diffusion phase is shown to be less reliable than its corresponding hybrid algorithm.

The results show the importance of coupling the SDS-led restart mechanism (dispensation mechanism) and the communication of agents which are both deployed in SDS algorithm.

7.4 Summary

This chapter presented an overview on the potential of merging PSO with SDS, and DE with SDS. Here, SDS is primarily used as an efficient resource allocation mechanism responsible for facilitating communication between PSO particles or DE agents.

Results reported in this chapter have demonstrated that the hybrid algorithms outperform the performance of standard PSO and (one variation of) classical DE architectures, even when applied to problems with low-cost fitness function evaluations (the benchmarks presented).

The next chapter uses the findings detailed herein to introduce a generalised hybridisation strategy for using SDS-led information sharing mechanism in population-based algorithms.

Algorithm 7.1 Hybrid Algorithm SDSnPSO

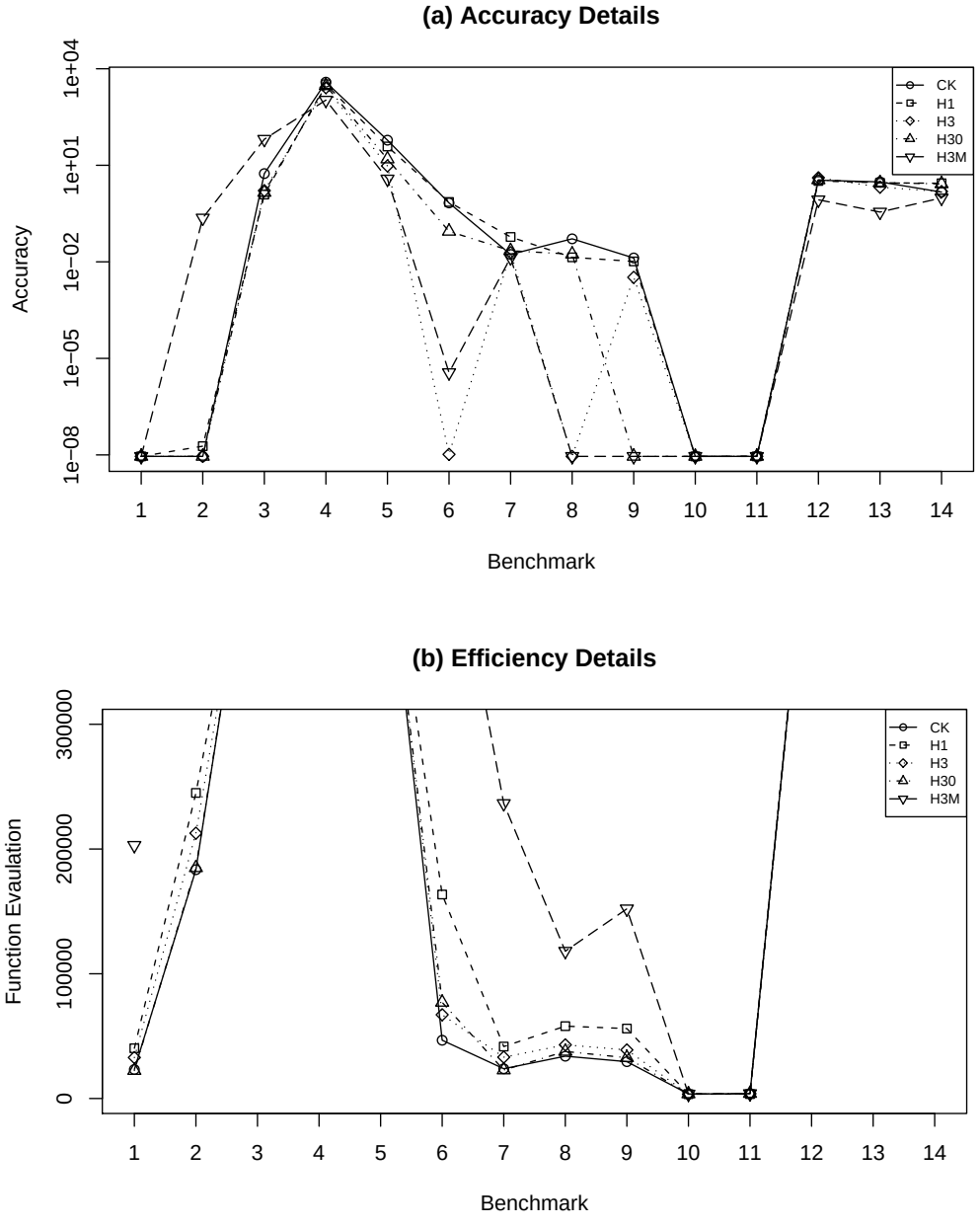
```
01: Initialise pAgents
02:
03: While ( stopping condition is not met )
04:    For all pAgents
05:       Evaluate fitness value of each particle
06:
07:       If ( evaluation counter % n == 0 )
08:       // START SDS
09:       // TEST PHASE
10:          For ag = 1 to No_of_pAgents
11:             r_ag = pick-random-pAgent()
12:             If ( ag.pbestFitness() <=
13:               r_ag.pbestFitness() )
14:                ag.setActivity (true)
15:             Else
16:                ag.setActivity (false)
17:             End If
18:          End For
19:
20:         // DIFFUSION PHASE
21:         For ag = 1 to No_of_pAgents
22:            If ( !ag.activity() )
23:              r_ag = pick-random-pAgent()
24:              If ( r_ag.activity() )
25:                 ag.setHypo( r_ag.getHypo() )*
26:              Else
27:                 ag.setHypo( randomHypo() )
28:              End If
29:         End For
30:       End If
31:       // END SDS
32:
33:       If (current fitness < pbest)
34:          pbest = current fitness
35:       If (pbest < neighbourhood best)
36:          neighbourhood best = pbest
37:       Update particle velocity
38:       Update particle position
39:    End
40: End

* In setHypo() and getHypo(), Hypo refers to
the pAgent's hypothesis (PSO particle).
```

Table 7.1: Accuracy and Efficiency Details

Accuracy ± Standard Error is shown with two decimal places after 30 trials of 300,000 function evaluations. Mean FEs ± Standard Error of successful trials are also shown in the second row of each benchmark alongside with the reliability of the algorithm; when there is no convergence, this row is removed. Total number of convergence of each algorithm over the benchmarks can be found in the last row.

	PSO-CK	**H1: SDSnPSO** $n = 1{,}000$	**H3: SDSnPSO** $n = 3{,}000$	**H30: SDSnPSO** $n = 30{,}000$	**H3M: Modified H3** $n = 3{,}000$
f_1	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**
	23273 ±321 (100%)	40265 ±1006 (100%)	32842 ±736 (100%)	22386 ±265 (100%)	202963 ±4929 (100%)
f_2	**0.0 ±0.0**	1.87E-08 ±8.90E-09	**0.0 ±0.0**	**0.0 ±0.0**	2.30E-01 ±2.07E-02
	183450 ±1655 (100%)	244969 ±4571 (93.33%)	212703 ±3172 (100%)	184962 ±2013 (100%)	-
f_3	5.53E+00 ±8.15E-01	1.23E+00 ±3.69E-01	1.63E+00 ±3.88E-01	1.43E+00 ±3.48E-01	6.45E+01 ±9.99E+00
f_4	3.83E+03 ±9.35E+01	3.30E+03 ±1.20E+02	2.59E+03 ±8.25E+01	2.96E+03 ±1.27E+02	1.06E+03 ±3.81E+01
f_5	6.04E+01 ±3.47E+00	3.84E+01 ±2.82E+00	9.55E+00 ±9.49E-01	1.56E+01 ±1.02E+00	3.76E+00 ±3.25E-01
f_6	6.78E-01 ±1.42E-01	7.24E-01 ±1.57E-01	1.04E-08 ±6.61E-10	8.86E-02 ±6.21E-02	3.61E-06 ±3.96E-07
	46728 ±3408 (53.33%)	163539 ±9356 (53.33%)	67155 ±1989 (96.67%)	77059 ±5350 (93.33%)	-
f_7	1.70E-02 ±2.90E-03	5.92E-02 ±1.34E-02	1.93E-02 ±3.59E-03	2.21E-02 ±3.61E-03	1.35E-02 ±2.45E-03
	23865 ±713 (26.67%)	41641 ±2924 (16.67%)	33131 ±1118 (33.33%)	22818 ±507 (26.67%)	236479 ±11177 (26.67%)
f_8	5.19E-02 ±3.06E-02	1.38E-02 ±6.54E-03	**0.0 ±0.0**	1.73E-02 ±1.41E-02	0.0 ±0.0
	33934 ±1803 (83.33%)	58005 ±2436 (86.67%)	43019 ±1338 (100%)	37734 ±1626 (93.33%)	118090 ±2866 (100%)
f_9	1.32E-02 ±6.24E-03	1.03E-02 ±5.72E-03	3.30E-03 ±3.30E-03	**0.0 ±0.0**	**0.0 ±0.0**
	29543 ±1495 (86.67%)	56071 ±2795 (90%)	38946 ±1319 (96.67%)	32684 ±2808 (100%)	152050 ±2951 (100%)
f_{10}	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**
	3607 ±61 (100%)	3470 ±81 (100%)	3498 ±65 (100%)	3661 ±74 (100%)	3551 ±79 (100%)
f_{11}	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**	**0.0 ±0.0**
	3880 ±63 (100%)	3534 ±57 (100%)	3832 ±78 (100%)	3984 ±79 (100%)	3921 ±94 (100%)
f_{12}	3.44E+00 ±5.44E-01	3.26E+00 ±6.25E-01	4.10E+00 ±6.06E-01	3.52E+00 ±5.96E-01	8.47E-01 ±3.52E-01
f_{13}	3.05E+00 ±6.18E-01	2.86E+00 ±6.22E-01	2.09E+00 ±5.61E-01	2.87E+00 ±6.21E-01	3.52E-01 ±2.44E-01
f_{14}	1.47E+00 ±5.52E-01	2.76E+00 ±6.42E-01	1.53E+00 ±5.70E-01	2.65E+00 ±6.57E-01	9.70E-01 ±4.09E-01
$\sum$	(195)	(193)	(218)	(214)	(158)
	46.43%	45.95%	51.90%	50.95%	37.62%

Figure 7.2: SDSnPSO Accuracy and Efficiency Plots



Algorithm 7.2 Hybrid Algorithm, SDSnDE

```
01: Initialise SDEAgents
02:
03: For ( generation = 1 to generationsAllowed )
04:
05:    For ( SDEAgent = 1 to NP )
06:       Mutation : generate mutant vector
07:       Crossover: generate trial vector
08:       Selection: generate target vector for next generation
09:    End For
10:
11:    If ( generation counter MOD n == 0 )
12:    // START SDS
13:    // TEST PHASE
14:       For ag = 1 to NP
15:          r_ag = pick-random-SDEAgent()
16:          If (ag.targetVecFitness()<r_ag.targetVecFitness())
17:             ag.setActivity (true)
18:          Else
19:             ag.setActivity (false)
20:          End If
21:       End For
22:
23:       // DIFFUSION PHASE
24:       For ag = 1 to No_of_SDEAgents
25:
26:          If ( !ag.activity() )
27:             r_ag = pick-random-SDEAgent()
28:             If ( r_ag.activity() )
29:                ag.setHypo( r_ag.getHypo() )*
30:             Else
31:                ag.setHypo( randomHypo() )
32:             End If
33:          End If
34:
35:       End For
36:    End If
37:    // END SDS
38:
39:    Find SDEAgent with best fitness value
40:
41:    End For

* In setHypo() and getHypo(), Hypo refers to
the SDEAgent's hypothesis.
```

Table 7.2: Accuracy and Reliability Details

Accuracy ± Standard Error is shown with two decimal places after 30 trials of 2,000 generations. Total number of convergence of each algorithm over each benchmark is shown in brackets after the accuracy and standard error. The reliability of each algorithm over all the benchmarks is given in the last row of the table. For each benchmark, algorithms which are **significantly** better (see Table 7.5) than the others are highlighted. Note that the highlighted algorithms do not significantly outperform each another.

	DE	H5: SDSnDE $n = 5$	H50: SDSnDE $n = 50$	H200: SDSnDE $n = 200$	H50M: Modified H50 $n = 50$
f_1	1.06E-107±7.92E-108 *(30)*	5.29E-10±4.72E-10 *(28)*	5.52E-92±4.03E-92 *(30)*	4.70E-104±3.11E-104 *(30)*	2.03E-85±1.61E-85 *(30)*
f_2	1.20E-03±2.60E-04 *(0)*	1.21E+01±1.88E+00 *(0)*	2.55E-05±7.27E-06 *(0)*	1.48E-04±3.86E-05 *(0)*	8.58E-04±2.42E-04 *(0)*
f_3	3.66E+01±8.23E+00 *(0)*	4.40E+01±6.46E+00 *(0)*	**1.71E+00±5.36E-01** *(0)*	**3.87E+00±2.29E+00** *(0)*	**1.26E+00±3.22E-01** *(0)*
f_4	5.00E+02±1.23E+02 *(0)*	**3.02E-02±8.28E-03** *(0)*	**4.83E-01±4.37E-01** *(0)*	**6.23E-01±2.39E-01** *(0)*	**2.59E-02±9.26E-03** *(0)*
f_5	1.61E+02±8.49E+00 *(0)*	**2.67E-01±8.15E-02** *(2)*	**1.34E+01±7.49E+00** *(0)*	**2.79E+01±1.74E+00** *(0)*	**2.41E+01±1.00E+01** *(9)*
f_6	1.45E+01±1.34E+00 *(0)*	**2.36E-06±1.10E-06** *(0)*	**1.02E-01±7.00E-02** *(17)*	**3.23E-01±1.11E-01** *(19)*	**1.45E-01±1.34E-01** *(21)*
f_7	5.26E-02±1.05E-02 *(1)*	3.85E-02±1.43E-02 *(6)*	1.99E-02±4.40E-03 *(5)*	2.82E-02±6.76E-03 *(4)*	7.42E-02±5.50E-02 *(2)*
f_8	1.31E+01±3.07E+00 *(3)*	**5.66E-12±3.11E-12** *(30)*	**1.96E-02±1.28E-02** (24)	**1.05E-02±5.77E-03** *(25)*	**7.00E-03±4.86E-03** *(28)*
f_9	3.24E+00±2.41E+00 *(8)*	1.51E-10±9.08E-11 *(29)*	5.27E-01±3.68E-01 *(19)*	1.03E-02±5.72E-03 *(26)*	3.50E+01±1.73E+01 *(23)*
f_{10}	1.90E-01±6.41E-02 *(23)*	**2.48E-04±2.34E-04** *(28)*	**4.44E-17±1.65E-17** *(30)*	**5.92E-17±1.82E-17** *(30)*	**4.44E-17±1.65E-17** *(30)*
f_{11}	2.55E+02±5.97E+01 *(1)*	**1.13E-08±1.13E-08** *(29)*	**0.00E+00±0.00E+00** *(30)*	**2.96E-17±2.96E-17** *(30)*	**0.00E+00±0.00E+00** *(30)*
f_{12}	5.05E+00±6.73E-17 *(0)*	**1.25E+00±4.77E-01** *(24)*	3.02E+00±5.43E-01 *(14)*	3.37E+00±5.31E-01 *(7)*	4.80E+00±2.52E-01 *(2)*
f_{13}	5.27E+00±0.00E+00 *(0)*	**7.03E-01±3.33E-01** *(23)*	**1.28E+00±4.33E-01** *(11)*	3.78E+00±5.56E-01 *(0)*	4.83E+00±3.09E-01 *(1)*
f_{14}	5.36E+00±6.02E-17 *(0)*	**3.57E-01±2.48E-01** *(27)*	**5.81E-01±3.26E-01** *(13)*	4.19E+00±4.86E-01 *(0)*	4.82E+00±2.99E-01 *(0)*
Σ	66	226	193	171	176
	15.71%	53.81%	45.95%	40.71%	41.90%

Figure 7.4: SDSnDE Accuracy Plot

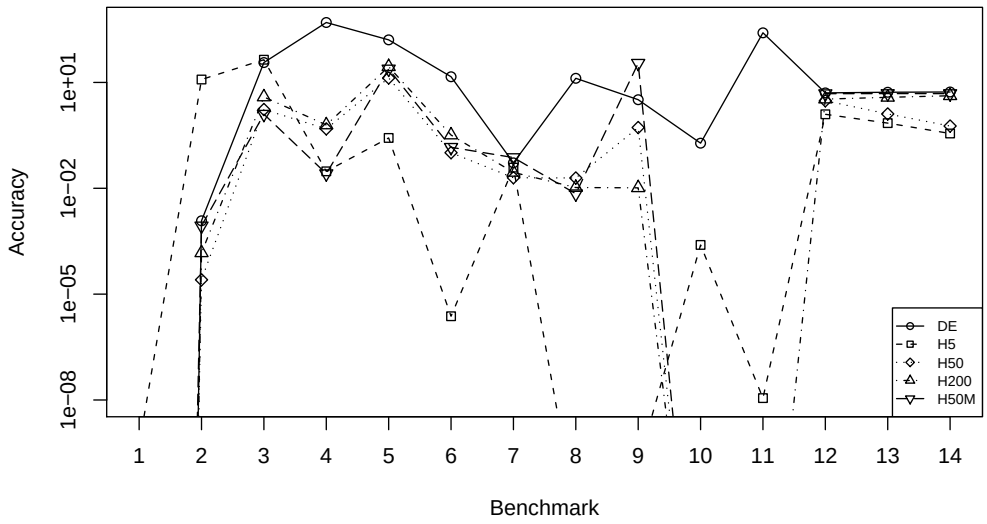


Table 7.3: TukeyHSD Test Results for Accuracy

Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right algorithm is significantly better than the one, on the left. Benchmarks with no significance between algorithms are removed.

	H1-CK	**H3-CK**	**H30-CK**	**H3M-CK**	**H3-H1**	**H30-H1**	**H3M-H1**	**H30-H3**	**H3M-H3**	**H3M-H30**
f_1	-	-	-	o–X	-	-	-	-	-	-
f_2	-	-	-	o–X	-	-	o–X	-	o–X	o–X
f_3	-	-	-	o–X	-	-	x	-	o–X	o–X
f_4	X–o	X–o	X–o	X–o	X–o	-	X–o	-	X–o	X–o
f_5	X–o	X–o	X–o	X–o	X–o	X–o	X–o	-	-	X–o
f_6	-	X–o	X–o	X–o	X–o	X–o	X–o	-	-	-
f_7	o–X	-	-	-	X–o	X–o	X–o	-	-	-
f_{12}	-	-	-	X–o	-	-	X–o	-	X–o	X–o
f_{13}	-	-	-	X–o	-	-	X–o	-	-	X–o

Table 7.4: TukeyHSD Test Results for Efficiency

Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right algorithm is significantly better than the one, on the left. Benchmarks with no convergence are removed.

	H1-CK	**H3-CK**	**H30-CK**	**H3M-CK**	**H3-H1**	**H30-H1**	**H3M-H1**	**H30-H3**	**H3M-H3**	**H3M-H30**
f_1	o–X	o–X	-	o–X	-	X–o	o–X	X–o	o–X	o–X
f_2	o–X	o–X	-		X–o	X–o		X–o		
f_6	o–X	o–X	o–X		X–o	X–o		-		
f_7	-	-	-	o–X	-	-	o–X	-	o–X	o–X
f_8	o–X	o–X	-	o–X	X–o	X–o	o–X	-	o–X	o–X
f_9	o–X	-	-	o–X	X–o	X–o	o–X	-	o–X	o–X
f_{10}	-	-	-	-	-	-	-	-	-	-
f_{11}	x	-	-	-	o–X	o–X	o–X	-	-	-

Table 7.5: TukeyHSD Test Results for Accuracy

Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right algorithm is significantly better than the one, on the left.

	DE-H5	**DE-H50**	**DE-H200**	**DE-H50M**	**H5-H50**	**H5-H200**	**H5-H50M**	**H50-H200**	**H50-H50M**	**H200-H50M**
f_1	–	–	–	–	–	–	–	–	–	–
f_2	X–o	–	–	–	o–X	o–X	o–X	–	–	–
f_3	–	o–X	o–X	o–X	o–X	o–X	o–X	–	–	–
f_4	o–X	o–X	o–X	o–X	–	–	–	–	–	–
f_5	o–X	o–X	o–X	o–X	–	–	–	–	–	–
f_6	o–X	o–X	o–X	o–X	–	–	–	–	–	–
f_7	–	–	–	–	–	–	–	–	–	–
f_8	o–X	o–X	o–X	o–X	–	–	–	–	–	–
f_9	–	–	–	X–o	–	–	X–o	–	X–o	X–o
f_{10}	o–X	o–X	o–X	o–X	–	–	–	–	–	–
f_{11}	o–X	o–X	o–X	o–X	–	–	–	–	–	–
f_{12}	o–X	o–X	o–X	–	X–o	X–o	X–o	–	X–o	–
f_{13}	o–X	o–X	o–X	–	–	X–o	X–o	X–o	X–o	–
f_{14}	o–X	o–X	–	–	–	X–o	X–o	X–o	X–o	–

Chapter 8

Generalised Hybridisation Strategy

"We can only see a short distance ahead, but
we can see plenty there that needs to be done."
– Alan Turing

This chapter uses the ideas introduced in previous chapter (see Chapter 7 on page 122) to propose a generalised hybridisation strategy that is applicable to any population-based optimiser. The generalised hybridisation strategy is subsequently tested on a harder and more recent set of benchmarks (other than those used in the previous chapter) and a larger set of algorithms. The results of the experiments are followed by a discussion on the performance of the hybrid algorithms.

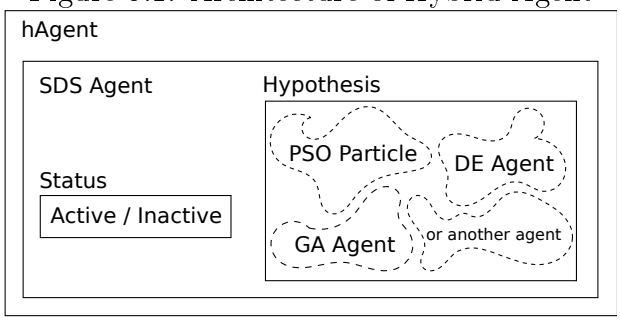
8.1 Hybridisation Strategy

The initial motivating thesis justifying the hybridisation of SDS and the population-based algorithms (via what we call *Hybridisation Strategy*) is the

partial function evaluation deployed in SDS, which may mitigate the high computational overheads entailed when deploying these algorithms onto a problem with a costly (and decomposable) fitness function. However, before commenting on and exploring this area – which remains an ongoing research – an initial set of experiments (with cheap fitness functions) are conducted, aiming to investigate if the information diffusion mechanism deployed in SDS may on its own improve the behaviour of population-based algorithms (for more details about these algorithms see Sections 4.1 on page 72, 4.3 on page 95 and 4.2 on page 93).

As mentioned earlier (see Chapter 3 on page 44), each SDS agent has a hypothesis and a status. In the hybrid algorithms, every member of the SI and EA population is an SDS agent too – together termed hybrid agents (*hAgents*). More specifically, in the hybrid algorithms, SDS hypotheses contain the solution vectors of the member of the populations, and an additional Boolean variable (status) determines whether the hAgent is active or inactive (see Figure 8.1).

Figure 8.1: Architecture of Hybrid Agent



The behaviour of the hybrid algorithm in its simplest form is presented in Algorithm 8.1 on page 152.

The hybridisation strategy was first investigated on a standard PSO algorithm (see Section 7.1 or [226]) and a variant of classical DE (see Section 7.2 or [227]) in order to investigate the information sharing mechanism of SDS on the performance of the PSO particles and DE agents. The positive effects of the hybridisation strategy on the behaviour of PSO and DE, provided the

motivation to examine the strategy on a new set of swarm intelligence and evolutionary algorithms and a different set of benchmarks.

Looking at Algorithm 8.1, lines $1-7$ and $34-36$ represent the code related to the population-based algorithms; and lines $9-32$ represent the SDS part of the strategy. Once the SDS part is implemented, it can be applied to the population-based algorithms.

Since the Generalised Hybridisation Strategy does not have an impact on the implementation aspect of the algorithms intended to use this strategy, the integration process is smooth, allowing researchers to focus on analysing the final results; in other words, once the Generalised Hybridisation Strategy is implemented and "plugged into" one algorithm, the same implementation can be plugged into other algorithms.

8.2 Test and Diffusion Phases in the Hybrid Algorithms

In the test-phase of a standard stochastic diffusion search, each agent partially evaluates its hypothesis. The guiding heuristic is that hypotheses that are promising are maintained and those that appear unpromising are discarded. In the context of the hybrid algorithms presented here, different tests could be conducted in order to determine the activity of each hAgent. One test is illustrated in Algorithm 8.1 on page 152 (see Lines: 12-19). Here, the Test Phase is simply performed by comparing the fitness of each hAgent against that of a random one (excluding itself); if the selecting hAgent has a better fitness value, it will become active, otherwise it is flagged inactive.

In the Diffusion Phase (see Algorithm 8.1, Lines: 22-30), each inactive hAgent picks another hAgent randomly. If the selected hAgent is active, it communicates its hypothesis to the inactive one; if the selected hAgent is inactive too, the selecting hAgent generates a new hypothesis at random from the search space.

As outlined in the pseudo-code of the hybrid algorithm (see Algorithm 8.1), after each n generations, one full SDS cycle is executed.

8.3 Experiments

In this section, a number of experiments are carried out and the performance of a standard PSO [153], one variation of DE (DE/best/1) and one simple type of real-valued GA algorithms are contrasted against their hybrid algorithm counterparts (SDSnPSO, SDSnDE, SDSnGA). The aim in evaluating a number of population-based algorithms is to demonstrate the generality of the hybridisation strategy. The measure used to determine the quality of each algorithms is accuracy (see Section 5.3.1 on page 101 for the definition).

8.3.1 Experiment Setup

In order to examine the Generalised Hybridisation Strategy, the experiments use a set of test functions that were designed for the Special Session on Real Parameter Optimization organised as part of the 2005 IEEE Congress on Evolutionary Computation (CEC 2005), reported in [229], where a complete description of these benchmarks are provided:

- Unimodal Functions (5):
  - F_1: Shifted Sphere Function
  - F_2: Shifted Schwefel's Problem 1.2
  - F_3: Shifted Rotated High Conditioned Elliptic Function
  - F_4: Shifted Schwefel's Problem 1.2 with Noise in Fitness
  - F_5: Schwefel's Problem 2.6 with Global Optimum on Bounds
- Multimodal Functions[1] (9):

[1] Hybrid Composition Functions are not used in this work.

- Basic Functions (7):
  - F_6: Shifted Rosenbrock's Function
  - F_7: Shifted Rotated Griewank's Function without Bounds
  - F_8: Shifted Rotated Ackley's Function with Global Optimum on Bounds
  - F_9: Shifted Rastrigin's Function
  - F_{10}: Shifted Rotated Rastrigin's Function
  - F_{11}: Shifted Rotated Weierstrass Function
  - F_{12}: Schwefel's Problem 2.13
- Expanded Functions (2):
  - F_{13}: Expanded Extended Griewank's plus Rosenbrock's Function (F_8F_2)
  - F_{14}: Shifted Rotated Expanded Scaffer's F_6

All benchmarks have been shifted in order to ensure there are no optima in the centre of the search space.

The experiments are conducted with the generic population size of 50, 100 and 100 for PSO particles, DE and GA agents respectively; the halting criterion for this experiment is exceeding $300,000$ function evaluations (FEs) and with regards to PSO, in addition to the mentioned criterion, if the algorithm reaches the optima (with distances less than 10^{-8}), it terminates; the dimensionality of the problems is 30 [153, 206].

In [226, 227], the impact of using different n values, after which the SDS cycle begins, is reported, showing the effect of having frequent or less frequent SDS cycles on the performance of the hybrid algorithms. In this work, one n value ($n = 50$) is used, aiming to provide a general yet clear idea on the performance of the hybrid algorithms, which use the Generalised Hybridisation Strategy: SDSnPSO, SDSnDE, SDSnGA; no claims were made for the optimality of the frequency rate used; sRePSO, sReDE and sReGA represent algorithms that just use the **s**ds-led **Re**start mechanism, without information sharing (for this mode, the diffusion phase is modified. See Algorithm 7.3 on page 129).

There are 30 independent runs for each benchmark function and the results are averaged over these independent trials.

8.3.2 Results

Tables 8.1(a), 8.2(a) and 8.3(a) on pages 153, 154, 155 respectively) show the performance of PSO, DE and GA algorithms alongside their hybrid counterparts integrated with SDS.

In these tables, over each benchmark, algorithms which are significantly better (see Tables 8.1(b), 8.2(b), 8.3(b)) than the one(s) with the least accuracy, are highlighted.

Figure 8.2 on page 156 shows the accuracy plots of each one of the above mentioned set of algorithms.

Although the value of n, which determines the frequency of running SDS cycles is important, but the emphasis in this chapter is rather on investigating the general effect of the hybridisation strategy on the performance of these population-based algorithms.

Below, the hybrid algorithms are contrasted against their vanilla counterparts and in Section 8.4, their performance without the information sharing mechanism is discussed.

As Table 8.1(b) shows, over all benchmarks, other than f_7, PSO does not significantly outperform the hybrid algorithm (SDSnPSO). On the other hand, in the rest of the cases with a significant difference (e.g. f_2, f_4, $f_{8,9}$ and f_{11-14}), the hybrid algorithm outperforms the standard PSO algorithm significantly. Table 8.1(a) also confirms that whenever there is a significant difference in the performance of the standard and hybrid algorithm, the hybrid algorithm is (among) the best in most cases (SDSnPSO column has the most highlighted records).

Table 8.2(b) demonstrates that whenever there is a significant difference between classical DE and the hybrid algorithms (SDSnDE), the hybrid algorithm significantly outperforms the classical one ($f_{2,3}$, f_7 and f_{9-14}). Table

8.2(a) also confirms that the hybrid algorithm has the highest record of being (among) the best algorithm(s).

Table 8.3(b) shows that the hybrid algorithm demonstrates 71% outperformance when there exists a significant difference between the hybrid algorithm (SDSnGA) and the vanilla real-valued GA (e.g. in f_2, $f_{4,5}$, f_7, f_{14}). Although the hybrid algorithm does not overwhelmingly outperform the vanilla GA, it holds its place as the algorithm with the highest number of best performances (see Table 8.3(a)).

As stated above, the results from Tables 8.1, 8.2, 8.3 show the effects of generalised hybridisation strategy in the hybrid algorithms using the information sharing mechanism deployed in SDS.

8.4 Discussion

The resource allocation process underlying standard SDS offers three closely coupled mechanisms to the algorithm's search component to speed its convergence to global optima.

- the first component is 'efficient, non-greedy information sharing' instantiated via positive feedback of potentially good hypotheses between agents.
- the second component is the SDS-led random-restart deployed as part of the diffusion phase.
- the third component which is not used explicitly in this work is random 'partial hypothesis evaluation', whereby a complex, computationally expensive objective function is broken down into 'k independent partial-functions', each of which, when evaluated, offers partial information on the absolute quality of current algorithm search parameters. It is this mechanism of iterated selection of a *random* partial function that ensures a standard SDS does not prematurely converge on local minimum. In current tests, this component is not explicitly exploited.

The resource allocation and restart components of SDS in the hybrid algorithm are executed in the Diffusion Phase, where information is shared (diffused) among hAgents (see Algorithm 3.3 on page 63). The analysis of the performance of the hybrid algorithm (see results above) demonstrates that adding the SDS resource allocation and SDS-led restart mechanisms to the swarm intelligence and evolutionary algorithms used in this work improves the overall performance of the algorithm (i.e. it enhances algorithm accuracy, as defined herein).

To further analyse the role of SDS in the hybrid algorithms, the Diffusion Phase of the SDS algorithm is modified (see Algorithm 7.3 on page 129) investigating the restart effect caused by randomising a selection of agent hypotheses after a number of iterations (effectively instantiating the swarm intelligence and evolutionary algorithms with SDS-led random-restarts). In other words, after the SDS test-phase, the hypothesis of each inactive hAgent is randomised.

As detailed in Tables 8.1 on page 153, 8.2 on page 154, 8.3 on page 155, although information sharing plays an important role in the performance of the hybrid algorithms, the significance of SDS-led restart mechanism (in randomly restarting some of the agents) in improving the performance of the algorithms cannot be discarded.

Other than f_{11} in Table 8.1(b) where sRePSO (using restart-only mechanism) significantly outperforms sPSO (the hybrid algorithm), in the rest of the experiments (be it PSO or DE and GA), when there exists a significant difference, the hybrid algorithms significantly outperform the restart-only hybrid algorithms.

Although the algorithms with restart-only mechanism are generally outperformed by the hybrid algorithms, they still compete with the vanilla swarm intelligence and evolutionary algorithms. For instance, in PSO, out of 10 significant differences, sRePSO significantly outperforms PSO in 8 cases (80%); in DE, the outperformance is marginal, where sReDE outperforms in 57% of the cases; in GA however, sReGA is outperformed by GA which performs 70% better. The results show the importance of coupling the restart mecha-

nism and the communication of agents which are both deployed in the SDS algorithm.

The third SDS component feature, which is currently only implicitly exploited by the hybrid algorithm, is 'randomised partial hypothesis evaluation'. In the Mining Game (see Section 3.1 on page 45), "At the start of the mining process each miner maintains a [randomly allocated] hypothesis - their current belief of 'best hill' to mine"; and each miner mines one small randomly selected area of this hill rather than the entirety of it (i.e. revealing a partial estimate of the gold content of the entire hill); following this approach, each miner forms a partial view of the gold content of their hill hypothesis (which is merely part of the overall mountain range: the entire search space).

In typical optimisation algorithms, the search process iterates the evaluation of one point in the n-dimensional search space (iterating an objective function evaluation). In swarm intelligence and evolutionary algorithms' population, in addition to this information, each agent has implicit partial knowledge about the search space (from its former experience and/or other agents).

In PSO, this implicit partial knowledge is derived from the fact that each particle has implicit knowledge of a discrete sub-space (or $dSubS$) comprising the historical evidence implicit in the prior [m] objective-function evaluations it has performed. Thus, since the memory of each particle maintains the best point found so far, each particle, covering its dSubS, has partial knowledge of the full search space [226].

In DE or GA, this implicit partial knowledge (coming from other agents) is derived from the mutation, crossover and selection mechanisms. Therefore, as long as each agent finds its current position by using this implicit knowledge, it should have a partial knowledge of the full problem space [227]. In the hybrid algorithms each hAgent maintains a fitness value which is the best objective function value it has currently found, based on its exploration of the search space so far. Thus constituted, each hAgent's target vector (in case of the DE algorithm, or personal best in case of the PSO) defines a 'partial view' of the entire search space (via the partial interaction it has with the rest of the

population (e.g. through mutation, crossover and selection). Hence, when the fitness values of two hAgents are compared in the test-phase of the hybrid algorithms, two partial views of the entire search space are contrasted. This is analogous to the 'test' process of the Mining Game as, in both processes, agents become active or inactive contingent upon the agent's evaluation of a randomised partial view of the entire search space.

In both the Mining Game and the new hybrid algorithms, the notion of partial-function evaluation differs importantly from that traditionally deployed in a simple discrete partial function SDS, where, for a given set of parameter values (the agent hypothesis) a complex objective function is broken into m components, only one randomly selected of which will be evaluated and the subsequent agent-activity is based on this. Clearly, as this process merely evaluates $1/m$ of the total number of computations required for the full hypothesis evaluation, it concomitantly offers a potentially significant performance increase. Whereas in the hybrid algorithms, the objective function is evaluated in-toto, using a given set of parameter values (the agent's hypothesis) and the subsequent agent-activity is based on this. In the former case, the agent exploits knowledge of the partial objective function and in the process gains a potential partial-function performance dividend; in the latter the agent merely exploits partial knowledge of the search space without the concomitant explicit partial-function performance increase.

Ongoing work on computationally more complex benchmark problems, seeks to exploit this 'partial-function dividend' with the hybrid algorithms; if successful, this offers further, potentially significant, performance improvements for the new hybrid algorithms.

8.5 Observations

One of the differences between the hybrid algorithm using PSO in this chapter and the former one is visible in the test-phase. In the previous chapter, if a particle's pbest is better than or equal to another randomly selected particle,

the selecting particle becomes inactive; however, in this chapter, the test-phase also dictates that if the selecting hAgent's solution vector is better than (but not equal to) a randomly selected hAgent, it is flagged active. This strategy is adopted in order to allow more particles to be inactive and possibly explore the search space rather than their current hypotheses.

Additionally, during the test-phase, the selecting hAgent should select another hAgent randomly from all other hAgents, excluding itself, a criterion which was not in place in Section 7.1 on page 123 for the hybrid algorithms using PSO (SDSnPSO).

This chapter uses local neighbourhood PSO to respond to criticism that might arise due to the use of global neighbourhood PSO (deployed in the former chapter and [226]). Global neighbourhood PSO is not favoured because of premature convergence as stated by one of the inventors of PSO, James Kennedy [220]:

> "it might be time to mount a sword-swinging crusade against any use of the gbest topology. How did this happen? It is the worst way to do it, completely unnecessary. I will not tolerate anybody who uses that topology complaining about "premature convergence"."

In brief, the following distinguishes the experiments run in this chapter from the ones in the previous chapter:

- A generalised hybridisation strategy is designed to consider any population-based algorithm
- Standard local neighbourhood is used for PSO, which is more preferred than global neighbourhood [220]
- Different frequencies of running SDS cycles are abstracted
- During communication, both in test and diffusion phases, hAgents pick any other hAgents randomly, excluding the self

- All algorithms use the same test mechanism to determine their status
- A more recent and harder set of benchmarks is used (i.e. CEC'05).

8.6 Summary

This chapter has presented a strategy for the integration of population-based algorithms (e.g. swarm intelligence and evolutionary algorithms) with Stochastic Diffusion Search (SDS), using the Generalised Hybridisation Strategy which benefits from the deployed SDS-led resource allocation and restart mechanisms.

It is shown that facilitating communication between population-based agents is the responsibility of SDS via its resource allocation mechanism. Additionally, an initial discussion of the similarity between the hypothesis test employed in the hybrid algorithms and the test-phase in SDS algorithm was presented.

This chapter has demonstrated that the hybrid algorithms, even when applied to problems with low-cost fitness function evaluations (the benchmarks presented), outperform the performance of few population-based algorithms.

Algorithm 8.1 Generalised Hybridisation Strategy – Hybrid Algorithm

```
01: Initialise hAgents
02:
03: For ( generation = 1 to generationsAllowed )
04:
05:    For ( hAgent = 1 to NP )
06:       Run one iteration of population-based algorithms
07:    End For
08:
09:    If ( generation counter % n == 0 )
10:    // START SDS
11:       // TEST PHASE
12:       For ag = 1 to NP
13:          r_ag = pick-random-hAgent()
14:          If ( ag.fitness() <  r_ag.fitness() )
15:             ag.setActivity (true)
16:          Else
17:             ag.setActivity (false)
18:          End If
19:       End For
20:
21:       // DIFFUSION PHASE
22:       For ag = 1 to No_of_hAgents
23:          If ( !ag.activity() )
24:             r_ag = pick-random-hAgent()
25:             If ( r_ag.activity() )
26:                ag.setHypo( r_ag.getHypo() )*
27:             Else
28:                ag.setHypo( randomHypo() )**
29:             End If
30:       End For
31:    End If
32:    // END SDS
33:
34:    Find hAgent with best fitness value
35:
36: End For

*  In setHypo() and getHypo(), 'Hypo' refers to
the hAgent's hypothesis.
** 'randomHypo()' uses the entire search space to
reinitialise the agent.
```

Table 8.1: Generalised Hybridisation Strategy on PSO

(a) Accuracy ± Standard Error is shown with two decimal places. For each benchmark, algorithms which are **significantly** better (see Table 8.1(b)) than the one with the least accuracy, are highlighted. Note that the highlighted algorithms do not significantly outperform one another.

	PSO	**sPSO** ***SDSnPSO***	**sRePSO**
f_1	9.52E-10±7.90E-12	9.61E-10±7.43E-12	9.60E-10±1.46E-11
f_2	1.64E-01±2.67E-02	**3.42E-03±5.67E-04**	**5.63E-02±1.29E-02**
f_3	1.50E+06±1.04E+05	1.40E+06±1.24E+05	1.62E+06±1.01E+05
f_4	7.37E+03±3.91E+02	**7.53E+02±6.32E+01**	1.71E+03±1.39E+02
f_5	5.63E+03±1.89E+02	5.06E+03±1.89E+02	4.94E+03±1.54E+02
f_6	2.27E+01±5.64E+00	4.28E+01±1.06E+01	1.38E+02±3.43E+01
f_7	**9.31E-03±1.27E-03**	2.46E-02±3.34E-03	2.79E-02±4.09E-03
f_8	2.09E+01±1.20E-02	**2.01E+01±9.00E-03**	2.09E+01±9.43E-03
f_9	9.24E+01±3.73E+00	**1.17E+01±8.30E-01**	**1.59E+01±7.86E-01**
f_{10}	1.16E+02±4.11E+00	1.05E+02±5.18E+00	9.08E+01±3.63E+00
f_{11}	3.00E+01±4.31E-01	2.50E+01±6.93E-01	**2.11E+01±4.18E-01**
f_{12}	9.42E+03±1.18E+03	**4.89E+03±8.58E+02**	1.19E+04±1.67E+03
f_{13}	4.72E+00±1.68E-01	**2.93E+00±1.08E-01**	**2.66E+00±1.12E-01**
f_{14}	1.26E+01±6.18E-02	**1.22E+01±7.35E-02**	**1.21E+01±5.87E-02**

(b) Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right one is significantly better than the left algorithm.

	PSO-sPSO	**PSO-sRePSO**	**sPSO-sRePSO**
f_1	–	–	–
f_2	o – X	o – X	–
f_3	–	–	–
f_4	o – X	o – X	X – o
f_5	–	o – X	–
f_6	–	X – o	X – o
f_7	X – o	X – o	–
f_8	o – X	–	X – o
f_9	o – X	o – X	–
f_{10}	–	o – X	–
f_{11}	o – X	o – X	o – X
f_{12}	o – X	–	X – o
f_{13}	o – X	o – X	–
f_{14}	o – X	o – X	–

Table 8.2: Generalised Hybridisation Strategy on DE

(a) Accuracy ± Standard Error is shown with two decimal places. For each benchmark, algorithms which are **significantly** better (see Table 8.2(b)) than the one with the least accuracy, are highlighted. Note that the highlighted algorithms do not significantly outperform one another.

	DE	**sDE** ***SDSnDE***	**sReDE**
f_1	1.57E-13±2.09E-14	5.68E-14±0.00E+00	9.50E+03±5.36E+03
f_2	1.39E-01±4.12E-02	4.55E-03±1.04E-03	8.09E-02±2.13E-02
f_3	1.58E+07±1.78E+06	**2.59E+06±2.06E+05**	**2.15E+06±2.08E+05**
f_4	**7.78E-01±1.33E-01**	**3.04E-01±5.30E-02**	2.48E+00±4.04E-01
f_5	1.94E+03±1.62E+02	2.15E+03±1.37E+02	2.48E+03±1.40E+02
f_6	4.96E+01±2.01E+01	1.30E+01±3.72E+00	1.21E+01±3.00E+00
f_7	5.40E-01±8.11E-02	**1.54E-01±3.49E-02**	**1.67E-02±2.79E-03**
f_8	2.10E+01±9.01E-03	2.10E+01±6.45E-03	2.10E+01±9.36E-03
f_9	2.84E+01±1.34E+00	**4.88E+00±4.68E-01**	4.61E+01±9.35E+00
f_{10}	1.88E+02±3.79E+00	**6.25E+01±3.59E+00**	1.05E+02±1.23E+01
f_{11}	3.85E+01±1.16E+00	**2.62E+01±1.28E+00**	3.85E+01±1.04E+00
f_{12}	**6.74E+05±1.30E+04**	**5.25E+04±4.20E+03**	7.46E+05±9.66E+03
f_{13}	8.52E+00±5.33E-01	**1.98E+00±8.00E-02**	**2.06E+00±7.09E-02**
f_{14}	1.35E+01±3.45E-02	**1.28E+01±7.22E-02**	1.34E+01±4.78E-02

(b) Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right one is significantly better than the left algorithm.

	DE-sDE	**DE-sReDE**	**sDE-sReDE**
f_1	–	–	–
f_2	o – X	–	–
f_3	o – X	o – X	–
f_4	–	X – o	X – o
f_5	–	X – o	–
f_6	–	–	–
f_7	o – X	o – X	–
f_8	–	–	–
f_9	o – X	–	X – o
f_{10}	o – X	o – X	X – o
f_{11}	o – X	–	X – o
f_{12}	o – X	X – o	X – o
f_{13}	o – X	o – X	–
f_{14}	o – X	–	X – o

Table 8.3: Generalised Hybridisation Strategy on GA

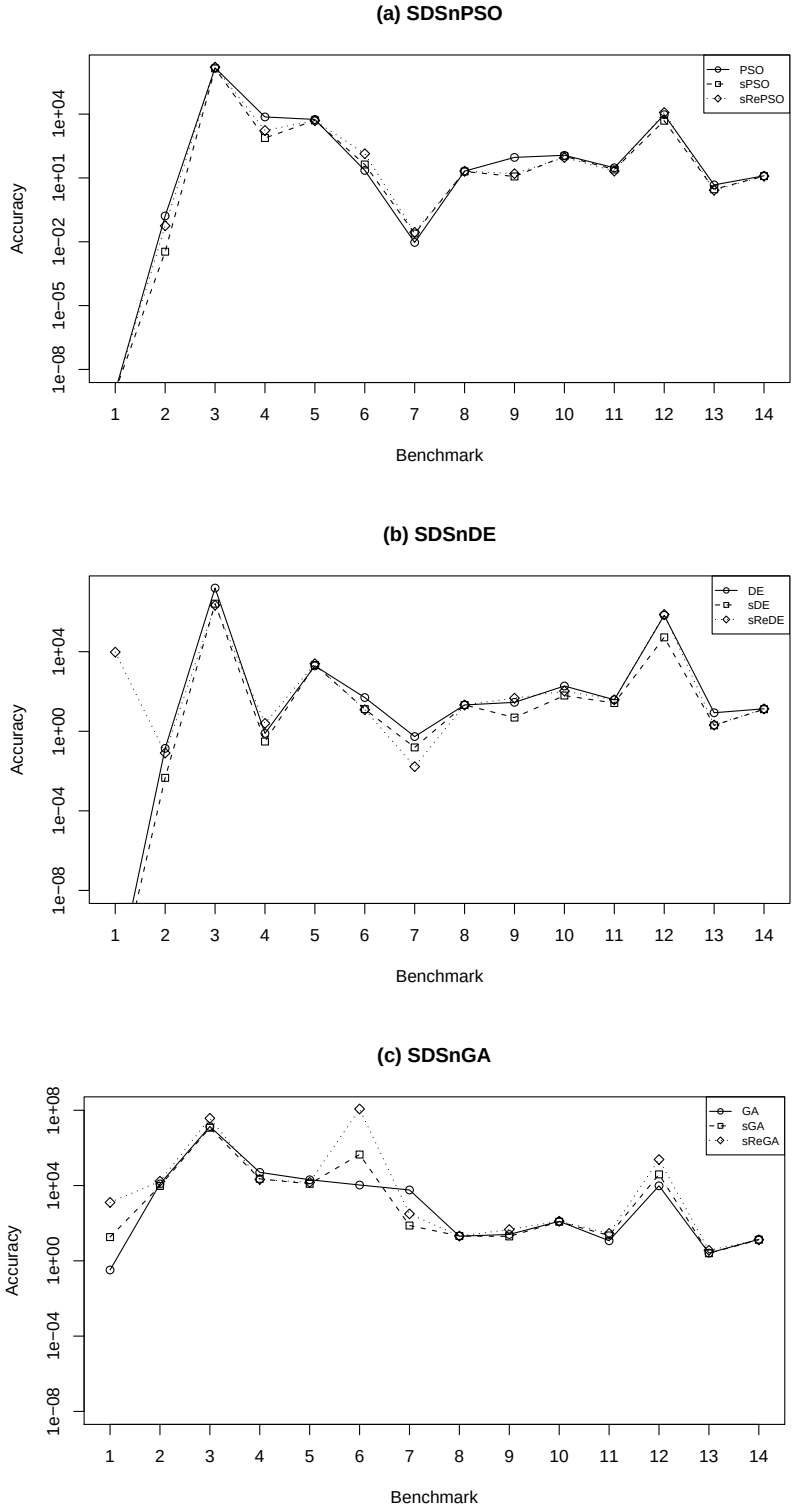
(a) Accuracy ± Standard Error is shown with two decimal places. For each benchmark, algorithms which are **significantly** better (see Table 8.3(b)) than the one with the least accuracy, are highlighted. Note that the highlighted algorithms do not significantly outperform one another.

	GA	**sGA** ***SDSnGA***	**sReGA**
f_1	**3.27E-01±3.27E-01**	**1.83E+01±9.09E+00**	1.27E+03±2.77E+02
f_2	1.22E+04±5.75E+02	**9.50E+03±5.16E+02**	1.69E+04±7.05E+02
f_3	**1.37E+07±1.10E+06**	**1.17E+07±9.76E+05**	3.75E+07±2.96E+06
f_4	5.02E+04±2.10E+03	**2.27E+04±8.08E+02**	**2.11E+04±4.33E+02**
f_5	2.03E+04±5.23E+02	**1.23E+04±2.36E+02**	1.49E+04±3.90E+02
f_6	**1.07E+04±9.81E+03**	**4.45E+05±3.11E+05**	1.17E+08±2.64E+07
f_7	5.83E+03±1.21E+02	**7.49E+01±4.39E+00**	3.13E+02±1.50E+01
f_8	**2.04E+01±2.25E-02**	2.09E+01±1.18E-02	2.09E+01±1.65E-02
f_9	**2.52E+01±1.48E+00**	**1.98E+01±1.33E+00**	4.60E+01±2.77E+00
f_{10}	1.26E+02±5.94E+00	1.19E+02±4.04E+00	1.28E+02±5.35E+00
f_{11}	**1.17E+01±5.11E-01**	2.24E+01±9.18E-01	2.87E+01±1.07E+00
f_{12}	**9.72E+03±1.30E+03**	**3.92E+04±6.37E+03**	2.41E+05±1.88E+04
f_{13}	**2.59E+00±8.48E-02**	**2.52E+00±9.39E-02**	3.62E+00±2.31E-01
f_{14}	1.38E+01±6.22E-02	**1.31E+01±6.68E-02**	**1.31E+01±5.42E-02**

(b) Based on TukeyHSD Test, if the difference between each pair of algorithms is significant, the pairs are marked. X–o shows that the left algorithm is significantly better than the right one; and o–X shows that the right one is significantly better than the left algorithm.

	GA-sGA	**GA-sReGA**	**sGA-sReGA**
f_1	–	X – o	X – o
f_2	o – X	X – o	X – o
f_3	–	X – o	X – o
f_4	o – X	o – X	–
f_5	o – X	o – X	X – o
f_6	–	X – o	X – o
f_7	o – X	o – X	X – o
f_8	X – o	X – o	–
f_9	–	X – o	X – o
f_{10}	–	–	–
f_{11}	X – o	X – o	X – o
f_{12}	–	X – o	X – o
f_{13}	–	X – o	X – o
f_{14}	o – X	o – X	–

Figure 8.2: Generalised Hybridisation Strategy Plot



Chapter 9

Conclusions and Future Work

"When you call me that, smile."
– Owen Wister

This chapter concludes the thesis by providing a summary of the study and making recommendations for future research.

9.1 Summary

In this study, after giving a brief background on artificial intelligence, swarm intelligence, optimisation and search, Stochastic Diffusion Search (SDS) is presented in Chapter 3 on page 44, followed by an introduction to three well known population-based optimisers: Particle Swarm Optimisation (PSO), Genetic Algorithm (GA) and Differential Evolution algorithm (DE) in Chapter 4 on page 72.

Following the literature review, SDS algorithm has been further investigated as a global optimiser, with differential evolution algorithm (DE) providing local search on convergence. The performance of DE is compared with the coupled SDS-DE algorithm and the results show the outperformance of the

coupled algorithm over classical DE. This highlights the impact of the information sharing and SDS-led restart mechanisms deployed in SDS on the optimisation process.

Next, after presenting Bare Bones PSO, which was initially proposed to provide better understanding of the behaviour of particle swarm algorithms, two new minimised variants of standard PSO are introduced – Bare Bones with Jumps PSO Models 1 & 2 (PSO-BBJ 1 & 2) – and their performance is compared against standard PSO and Bare Bones PSO, using three different measures: accuracy, efficiency and reliability, which are defined in section 5.3.1 on page 101. In terms of accuracy, although standard PSO demonstrates a better performance when all benchmarks are considered, the accuracy of PSO-BBJ2 compared to other algorithms is significantly better when benchmarks with successful convergence are considered. Additionally, PSO-BBJ2 is empirically shown to be *both* the most efficient and the most reliable algorithm in both local and global neighbourhoods. PSO-BBJ2 shows better reliability in global vs. local neighbourhood.

This study mainly focuses on the information sharing impact of SDS on swarm intelligence and evolutionary algorithms. It investigates whether the information diffusion mechanism deployed in SDS may on its own improve the behaviour of population-based algorithms.

This research initially aimed at investigating the integration of SDS with PSO, utilising the resource allocation mechanism of SDS to facilitate communication between PSO particles. The promising results of this integration provided the motivation to extend the scope of the research and apply the generalised SDS-led resource allocation mechanism (using the Generalised Hybridisation Strategy) to other population-based algorithms.

The resulting hybridisation strategy, utilises the information sharing mechanism in SDS for exchanging information between the members of population-based optimisers. The performance of the hybridisation strategy is investigated on more than one set of state-of-the-art benchmarks and applied to a number of population-based algorithms (e.g. PSO, DE and GA) with promising results: Out of 25 cases where the difference between the hybrid

algorithms and their classical counterparts were significant, the hybrid algorithms outperformed in 22 cases, demonstrating 88% outperformance.

The resource allocation process underlying standard SDS offers three closely coupled mechanisms to the algorithm's search component to speed its convergence to global optima: efficient, non-greedy information sharing (instantiated via positive feedback of potentially good hypotheses between agents); SDS-led random-restart which is deployed as part of the diffusion phase; and random 'partial hypothesis evaluation', which is not used explicitly in this work.

The resource allocation and restart components of SDS in the hybrid algorithm are executed in the Diffusion Phase, where information is diffused among the hybrid agents.

In order to verify the role of information sharing mechanism, a number of control experiments, which lack this mechanism, were conducted. The performance of the control algorithms, which only deploy the SDS-led random restart mechanism, are examined and the performance of the control algorithms that are majorly outperformed by the hybrid algorithms, demonstrates the positive impact of using the information exchange strategy deployed in SDS (out of 21 cases where the difference between the hybrid algorithms and their control counterparts were significant, the hybrid algorithms outperformed in 20 cases, demonstrating 95% outperformance). Thus, the hybrid algorithms, using the Generalised Hybridisation Strategy outperform the population-based algorithms used.

The easy-to-implement structure of the Generalised Hybridisation Strategy allows researchers to adopt the strategy and run experiments on the population-based algorithms of their choice by simply adding a few lines of code.

Other than the potential contribution of this hybridisation strategy to the concept of information exchange within the fields of population-based optimisers, and the significantly promising results it delivers, the applicability of the Generalised Hybridisation Strategy (in theory) to any population-based algorithm, makes it an attractive research topic to pursue further in future.

9.2 Future Work

The time spent on this piece of work proved to the author that there are far more questions emerging than answers; many potential directions can be pursued from here, some of which are summarised:

- Adopting vs. generating hypothesis: investigating the impact of adopting a random hypothesis (from the population of already existing hAgents) during the diffusion phase, instead of generating a random hypothesis (by randomly choosing a position within the search space)

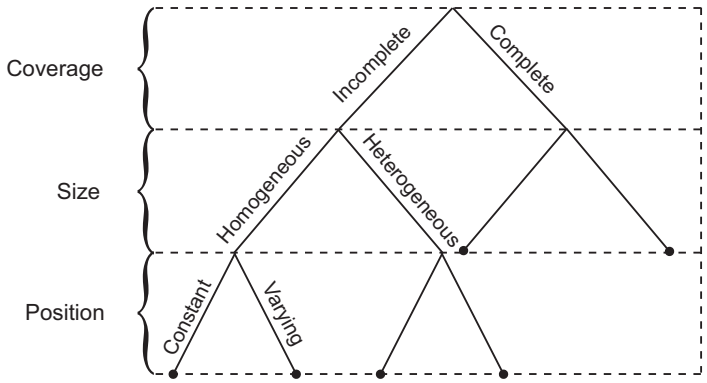
- Atomic vs. Full SDS Cycle:
  In the presented hybridisation strategy, after a certain number of generations the SDS cycle is run, which means that SDS goes through the test phase and considers all agents, then the diffusion phase considers the entire population of agents. This full cycle of test-diffusion phase is called Full SDS Cycle. Instead of iterating through Full SDS Cycles, running 'Atomic SDS Cycle' remains a future research. In this arrangement, after each n FEs, two agents are randomly picked for communication; before the communication phase, their status should be decided. One method for labelling them active or inactive is to pick another agent for each one of the existing agents; if the selected agent has a better fitness value, the selecting agent is labelled inactive, otherwise active. Once their status is determined, the same principle of SDS diffusion is applicable. While using the same principle of diffusion in SDS, this approach allows exploring a different method of exchanging information.

- Analysing the performance of SDS as a continuous optimiser using different benchmarks, and investigating the quality of the solution found before and after running the local search

- Further investigation of the behaviour of Bare Bones with Jumps algorithms

- Observing other forms of information exchange and their effects on the optimisation performance of population-based algorithms
- Analysing the behaviour of active and inactive hybrid agents separately at different stages of the optimisation process to better understand the underlying activities of the population in the hybrid algorithms
- Investigating the multi-swarm approaches and using SDS for resource allocation among the swarms. The following criteria might be considered: partitioning the search space (with partitions having fixed or varying positions; and homogeneous or heterogeneous sizes); allocating particles to swarms (same or different number of particles for each partition); search space coverage where all parts of the search space is covered either uniquely by one swarm or at least by one swarm; or random parts of the search are covered). See Figure 9.1
- Investigating the performance of the hybrid algorithms on some real-world problems; further theoretical work seeks to develop the core ideas presented in this work on problems with significantly more computationally expensive (and decomposable) objective functions

Figure 9.1: Possible Multi-Swarm Approaches



Bibliography

[1] J. Digalakis and K. Margaritis, "An experimental study of benchmarking functions for evolutionary algorithms," *International Journal*, vol. 79, pp. 403–416, 2002.

[2] D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias, "Evaluating evolutionary algorithms," *Artificial Intelligence*, vol. 85, no. 1-2, pp. 245–276, 1996.

[3] J. Bishop, "Stochastic searching networks," (London, UK), pp. 329–331, Proc. 1st IEE Conf. on Artificial Neural Networks, 1989.

[4] J. Kennedy and R. C. Eberhart, "Particle swarm optimization," in *Proceedings of the IEEE International Conference on Neural Networks*, vol. IV, (Piscataway, NJ), pp. 1942–1948, IEEE Service Center, 1995.

[5] D. Crevier, *AI: the tumultuous history of the search for artificial intelligence.* Basic Books, Inc., 1993.

[6] S. J. Russell and P. Norvig, *Artificial intelligence : a modern approach.* Prentice Hall series in artificial intelligence, Upper Saddle River, N.J.: Prentice Hall, 2nd ed., 2003.

[7] J. E. CHaugeland, *Artificial Intelligence: The Very Idea.* Cambridge, Massachusetts: MIT Press, 1985.

[8] R. Kurzweil, *The Age of Intelligent Machines.* Cambridge, Massachusetts: MIT Press, 1990.

[9] E. Charniak and D. McDermott, *Introduction to Artificial Intelligence.* Reading, Massachusetts: Addison-Wesley, 1985.

[10] D. Poole, A. K. Mackworth, and R. Goebel, *Computational intelligence: A logical approach.* Oxford, UK: Oxford University Press, 1998.

[11] A. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, pp. 433–460, 1950.

[12] L. Spector, "Evolution of artificial intelligence," *Artificial Intelligence*, vol. 170, no. 18, pp. 1251–1253, 2006.

[13] Kybernetes, *The Turing Test.* Emerald Group Publishing Limited, 2010.

[14] J. Fodor and A. Pylyshyn, "Connectionism and cognitive architecture: A critical analysis," *Cognition*, vol. 28, pp. 3–71, 1988.

[15] J. Pollack, "Connectionism: past, present, and future," *Artificial Intelligence Review*, vol. 3, no. 1, pp. 3–20, 1989.

[16] H. Dreyfus and S. Dreyfus, "Making a mind versus modeling the brain: Artificial intelligence back at a branchpoint," *Daedalus*, vol. 117, no. 1, pp. 15–43, 1988.

[17] W. S. McCullah and W. Pitts, "A logical calculus of ideas immanent in nervous activity," *Bulletin of Mathematical Biophysics*, vol. 5, pp. 115–133, 1943.

[18] J. Mira, "Symbols versus connections: 50 years of artificial intelligence," *Neurocomputing*, vol. 71, no. 4-6, pp. 671–680, 2008.

[19] D. Hebb, "The organisation of behaviour," 1949.

[20] R. Kempter, W. Gerstner, and J. Van Hemmen, "Hebbian learning and spiking neurons," *Physical Review E*, vol. 59, no. 4, p. 4498, 1999.

[21] N. Doidge, *The brain that changes itself: Stories of personal triumph from the frontiers of brain science.* Penguin Group USA, 2007.

[22] W. Ashby, *Design for a Brain.* Chapman and Hall London, 1960.

[23] F. Rosenblatt, *Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms.* Spartan Books, Washington, DC, 1962.

[24] M. Minsky and S. Papert, "Perceptrons," 1969.

[25] M. Lungarella, F. Iida, J. Bongard, and R. Pfeifer, *50 years of artificial intelligence: essays dedicated to the 50th anniversary of artificial intelligence.* Springer, 2007.

[26] P. Jackson, *Introduction to expert systems.* Addison-Wesley Longman Publishing Co., Inc., 1990.

[27] R. Lindsay, B. Buchanan, E. Feigenbaum, J. Lederberg, P. Singapore, and S. Toronto, "Applications of artificial intelligence for organic chemistry: The dendral project," *Structure*, vol. 2, no. 2.7, pp. 2–8, 1980.

[28] R. Lindsay, B. Buchanan, E. Feigenbaum, and J. Lederberg, "Dendral: a case study of the first expert system for scientific hypothesis formation," *Artificial Intelligence*, vol. 61, no. 2, pp. 209–261, 1993.

[29] E. Shortliffe and B. Buchanan, *Rule-based expert systems: the MYCIN experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, 1984.

[30] R. Duda, J. Gaschnig, and P. Hart, "Model design in the prospector consultant system for mineral exploration," *Expert systems in the microelectronic age*, pp. 153–167, 1979.

[31] J. Feldman and B. A, "Connectionist models and their properties," *Cognitive Science*, vol. 6, pp. 205–254, 1982.

[32] J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities," *Proceedings of the national academy of sciences*, vol. 79, no. 8, p. 2554, 1982.

[33] D. Parker, "Learning-logic (TR-47)," *Center for Computational Research in Economics and Management Science, MU, Cambridge, Massachusetts*, 1985.

[34] P. Werbos, *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. Harvard University, 1974.

[35] Y. Le Cun, "A learning procedure for asymmetric threshold networks," *Proc. Cognitiva*, vol. 85, pp. 599–604, 1985.

[36] J. Hopfield and D. Tank, "'Neural' computation of decisions in optimization problems," *Biological cybernetics*, vol. 52, no. 3, pp. 141–152, 1985.

[37] L. Steels and R. Brooks, *The artificial life route to artificial intelligence: Building embodied, situated agents*. Lawrence Erlbaum Associates, 1995.

[38] R. Pfeifer, J. Bongard, and S. Grand, *How the body shapes the way we think: a new view of intelligence*. The MIT Press, 2007.

[39] C. Langton, *Artificial life: An overview*. The MIT Press, 1998.

[40] M. Wooldridge, "An introduction to multiagent systems. 2002," *West Sussex, England: John Wiley and Sons Ltd*, vol. 348, 2002.

[41] B. Chaib-Draa, B. Moulin, R. Mandiau, and P. Millot, "Trends in distributed artificial intelligence," *Artificial Intelligence Review*, vol. 6, no. 1, pp. 35–66, 1992.

[42] N. Nilsson, "Two heads are better than one," *Sigart Newsletter*, vol. 73, p. 43, 1980.

[43] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm intelligence: from natural to artificial systems*. Oxford University Press, USA, 1999.

[44] J. F. Kennedy, R. C. Eberhart, and Y. Shi, *Swarm intelligence*. San Francisco ; London: Morgan Kaufmann Publishers, 2001.

[45] J. G. Saxe, D. Lathen, and B. Chief, "The Blind Man and the Elephant," *The Poems of John Godfrey Saxe*, 1882.

[46] E. Bonabeau, M. Dorigo, and G. Theraulaz, "Inspiration for optimization from social insect behaviour," *Nature*, vol. 406, p. 3942, 2000.

[47] B. Holldobler and E. O. Wilson, *The Ants*. Springer-Verlag, 1990.

[48] L. J. Goodman and R. C. Fisher, *The Behaviour and Physiology of Bees*. Oxon, UK: CAB International, 1991.

[49] T. D. Seeley, *The Wisdom of the Hive*. Harvard University Press, 1995.

[50] E. Wilson, *Sociobiology: The new synthesis*. Belknap Press, 1975.

[51] K. de Meyer, S. Nasuto, and J. Bishop, "Stochastic diffusion optimisation: the application of partial function evaluation and stochastic recruitment in swarm intelligence optimisation," *Springer Verlag*, vol. 2, Chapter 12 in Abraham, A. and Grosam, C. and Ramos, V. (eds), "Swarm intelligence and data mining", 2006.

[52] M. Moglich, U. Maschwitz, and B. Holldobler, "Tandem calling: A new kind of signal in ant communication," *Science*, vol. 186, no. 4168, pp. 1046–1047, 1974.

[53] R. Chadab and C. Rettenmeyer, "Mass recruitment by army ants," *Science*, vol. 188, pp. 1124–1125, 1975.

[54] N. Monmarche, G. Venturini, and M. Slimane, "On how pachycondyla apicalis ants suggest a new search algorithm," *Future Generation Computer Systems*, vol. 16, no. 9, pp. 937–946, 2000.

[55] J. Deneubourg, J. Pasteels, and J. Verhaeghe, "Probabilistic behaviour in ants: a strategy of errors?," in *Journal of Theoretical Biology*, vol. 105, pp. 259–271, Elsevier, 1983.

[56] H. Fan, Z. Hua, J. Li, and D. Yuan, "Solving a shortest path problem by ant algorithm," in *Machine Learning and Cybernetics, 2004. Proceedings of 2004 International Conference on*, vol. 5, pp. 3174–3177 vol.5, 2004.

[57] C. W. Reynolds, "Flocks, herds, and schools: A distributed behavioral model," *Computer Graphics*, vol. 21, no. 4, pp. 25–34, 1987.

[58] E. Shaw, "Fish in schools," *Natural History*, vol. 84, no. 8, pp. 40–46, 1975.

[59] V. Scheffer, "Spires of Form: Glimpses of Evolution," 1983.

[60] B. Partridge, "The structure and function of fish schools," *Scientific American*, vol. 246, no. 6, pp. 114–123, 1982.

[61] T. Back, *Evolutionary Algorithms in Theory and Practice*. New York: Oxford University Press, 1996.

[62] J. H. Holland, "Adaptation in natural and artificial systems," *Ann Arbor, MI, University of Michigan press*, 1975.

[63] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[64] M. Dorigo, *Optimization, learning and natural algorithms*. PhD thesis, Milano: Politecnico di Italy, 1992.

[65] M. Dorigo, V. Maniezzo, and A. Colorni, "Positive feedback as a search strategy," *Dipartimento di Elettronica e Informatica, Politecnico di*, 1991.

[66] M. Dorigo, G. D. Caro, and L. M. Gambardella, "Ant algorithms for discrete optimization," *Artificial Life*, vol. 5, no. 2, pp. 137–172, 1999.

[67] A. Colorni, M. Dorigo, V. Maniezzo, *et al.*, "Distributed optimization by ant colonies," in *Proceedings of the first European conference on artificial life*, vol. 142, pp. 134–142, 1991.

[68] A. Colorni, M. Dorigo, and V. Maniezzo, "An investigation of some properties of an ant algorithm," in *Proceedings of the Parallel Problem Solving from Nature Conference (PPSN 92)*, pp. 509–520, Elsevier Publishing, 1992.

[69] M. Dorigo, V. Maniezzo, and A. Colorni, "Ant system: optimization by a colony of cooperating agents," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 26, no. 1, pp. 29–41, 1996.

[70] V. Maniezzo and A. Colorni, "The ant system applied to the quadratic assignment problem," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 11, no. 5, pp. 769–778, 1999.

[71] L. Gambardella, E. Taillard, and M. Dorigo, "Ant colonies for the qap," *IDSIA, Lugano, Switzerland, Tech. Rep. IDSIA*, pp. 97–4, 1997.

[72] D. B. Fogel, *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*. Piscataway, NJ: IEEE Press, 1995.

[73] M. M. Millonas, "Swarms," *Phase Transitions, and Collective Intelligence. In Proceedings of Artificial Life III*, 1994.

[74] M. M. al-Rifaie, M. Bishop, and A. Aber, "Creative or not? birds and ants draw with muscles," in *AISB 2011: Computing and Philosophy*, (University of York, York, U.K.), pp. 23–30, 2011. ISBN: 978-1-908187-03-1.

[75] M. M. al-Rifaie, A. Aber, and M. Bishop, "Cooperation of nature and physiologically inspired mechanisms in visualisation," in *Biologically-Inspired Computing for the Arts: Scientific Data through Graphics* (A. Ursyn, ed.), IGI Global, United States, 2012. ISBN13: 9781466609426, ISBN10: 1466609427.

[76] M. M. al-Rifaie, M. Bishop, and S. Caines, "Creativity and autonomy in swarm intelligence systems," in *Cognitive Computation: Computational Creativity, Intelligence and Autonomy* (M. Bishop and Y. Erden, eds.), Springer, 2012. DOI: 10.1007_s12559-012-9130-y.

[77] M. Mitchell, *An introduction to genetic algorithms, 1996.* MIT press.

[78] D. E. Knuth, *The art of computer programming. Vol. 3, Sorting and Searching.* Addison-Wesley Reading, MA, 1973.

[79] A. Neumaier, "Complete search in continuous global optimization and constraint satisfaction," *Acta Numerica*, vol. 13, no. 1, pp. 271–369, 2004.

[80] T. Back, D. B. Fogel, and Z. Michalewicz, *Handbook of evolutionary computation.* IOP Publishing Ltd., 1997.

[81] J. Holland, "Outline for a logical theory of adaptive systems," *Journal of the ACM (JACM)*, vol. 9, no. 3, pp. 297–314, 1962.

[82] J. Holland and J. Reitman, "Cognitive systems based on adaptive algorithms," *ACM SIGART Bulletin*, no. 63, pp. 49–49, 1977.

[83] K. De Jong, "Are genetic algorithms function optimizers?," *Parallel problem solving from nature*, vol. 2, pp. 3–14, 1992.

[84] K. De Jong, "Genetic algorithms are not function optimizers," *Foundations of genetic algorithms*, vol. 2, pp. 5–17, 1993.

[85] L. Fogel, "Autonomous automata," *Industrial Research*, vol. 4, no. 2, pp. 14–19, 1962.

[86] L. Fogel, *On the organization of intellect.* PhD thesis, UCLA- Engineering, 1964.

[87] X. Yao and Y. Liu, "Fast evolutionary programming," in *Proceedings of the Fifth Annual Conference on Evolutionary Programming*, pp. 451–460, Citeseer, 1996.

[88] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Fromman-Holzboog.* Stuttgart. German, 1973.

[89] I. Rechenberg, *Evolutionsstrategie'94, volume 1 of Werkstatt Bionik und Evolutionstechnik*. Frommann-Holzboog, Stuttgart, 1994.

[90] H. Schwefel, *Evolutionsstrategie und numerische Optimierung, Technische Universität Berlin, Fachbereich Verfahrenstechnik, Dr.-Ing.* PhD thesis, Dissertation, 1975.

[91] H. Schwefel, "Evolution and optimum seeking," in *Sixth-Generation Computer Technology Series*, Wiley, New York, 1995.

[92] K. de Meyer, J. M. Bishop, and S. J. Nasuto, "Stochastic diffusion: Using recruitment for search," *Evolvability and interaction: evolutionary substrates of communication, signalling, and perception in the dynamics of social complexity (ed. P. McOwan, K. Dautenhahn & CL Nehaniv) Technical Report*, vol. 393, pp. 60–65, 2003.

[93] S. J. Nasuto, *Resource Allocation Analysis of the Stochastic Diffusion Search*. PhD thesis, University of Reading, Reading, UK, 1999.

[94] S. J. Nasuto and J. M. Bishop, "Convergence analysis of stochastic diffusion search," *Parallel Algorithms and Applications*, vol. 14(2), 1999.

[95] D. R. Myatt, J. M. Bishop, and S. J. Nasuto, "Minimum stable convergence criteria for stochastic diffusion search," *Electronics Letters*, vol. 40, no. 2, pp. 112–113, 2004.

[96] S. J. Nasuto, J. M. Bishop, and S. Lauria, "Time complexity of stochastic diffusion search," *Neural Computation*, vol. NC98, 1998.

[97] M. M. al-Rifaie and M. Bishop, "The mining game: a brief introduction to the stochastic diffusion search metaheuristic," *The Society for the Study of Artificial Intelligence and the Simulation of Behaviour Quarterly (AISBQ)*, vol. 130, 2010.

[98] M. J. Krieger, J. B. Billeter, and L. Keller, "Ant-like task allocation and recruitment in cooperative robots.," *Nature*, vol. 406, no. 6799, pp. 992–5, 2000.

[99] K. de Meyer, *Foundations of Stochastic Diffusion Search.* PhD thesis, PhD thesis, University of Reading, Reading, UK, 2003.

[100] R. Whitaker and S. Hurley, "An agent based approach to site selection for wireless networks," in *1st IEE Conf. on Artificial Neural Networks*, (Madrid Spain), ACM Press Proc ACM Symposium on Applied Computing, 2002.

[101] Y. Jin, "A comprehensive survey of fitness approximation in evolutionary computation," *In: Soft Computing*, vol. 9, pp. 3–12, 2005.

[102] J. Branke, C. Schmidt, and H. Schmeck, "Efficient fitness estimation in noisy environments," *In Spector, L., ed.: Genetic and Evolutionary Computation Conference, Morgan Kaufmann*, 2001.

[103] M. A. el Beltagy and A. J. Keane, "Evolutionary optimization for computationally expensive problems using gaussian processes," in *Proc. Int. Conf. on Artificial Intelligence'01*, pp. 708–714, CSREA Press, 2001.

[104] J. Bishop and P. Torr, "The stochastic search network," in *Neural Networks for Images, Speech and Natural Language*, (Chapman & Hall, New York), pp. 370–387, 1992.

[105] K. de Meyer, M. Bishop, and S. Nasuto, "Small world effects in lattice stochastic diffusion search," in *Proc. ICANN 2002*, (Madrid, Spain), pp. 147–152, Lecture Notes in Computer Science, 2415, 2002.

[106] S. Christensen and F. Oppacher, "What can we learn from no free lunch? a first attempt to characterize the concept of a searchable function," in *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1219–1226, 2001.

[107] S. J. Nasuto and M. J. Bishop, "Steady state resource allocation analysis of the stochastic diffusion search," *Arxiv preprint cs/0202007*, 2002.

[108] K. de Meyer, "Explorations in stochastic diffusion search: Soft- and hardware implementations of biologically inspired spiking neuron

stochastic diffusion networks," Tech. Rep. KDM/JMB/2000/1, University of Reading, 2000.

[109] S. Nasuto and M. Bishop, "Stabilizing swarm intelligence search via positive feedback resource allocation," in *Nature Inspired Cooperative Strategies for Optimization (NICSO)*, Springer, 2007.

[110] D. Myatt, S. Nasuto, and J. Bishop, "Alternative recruitment strategies for stochastic diffusion search," *Artificial Life X, Bloomington USA*, 2006.

[111] J. Bishop, "Coupled stochastic diffusion processes," in *Proc. School Conference for Annual Research Projects (SCARP), Reading, UK*, pp. 185–187, 2003.

[112] D. Myatt and J. Bishop, "Data driven stochastic diffusion networks for robust high-dimensionality manifold estimation - more fun than you can shake a hyperplane at," in *Proc. School Conference for Annual Research Projects (SCARP), Reading, UK*, 2003.

[113] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[114] J. Bishop, *Anarchic Techniques for Pattern Classification*. PhD thesis, University of Reading, Reading, UK, 1989.

[115] E. Grech-Cini, *Locating Facial Features*. PhD thesis, University of Reading, Reading, UK, 1995.

[116] I. Aleksander and T. Stonham, "Computers and digital techniques 2(1)," *Lect. Notes Art. Int 1562*, pp. 29–40, 1979.

[117] P. Beattie and J. Bishop, "Self-localisation in the senario autonomous wheelchair," *Journal of Intellingent and Robotic Systems*, vol. 22, pp. 255–267, 1998.

[118] S. J. Nasuto, K. Dautenhahn, and J. Bishop, "Communication as an emergent methaphor for neuronal operation," *Lect. Notes Art. Int 1562*, pp. 365–380, 1999.

[119] A. Hernandez-Carrascal and S. Nasuto, "A SWARM INTELLIGENCE METHOD FOR FEATURE TRACKING IN AMV DERIVATION," *Ninth International Wind Workshop*, 2008.

[120] M. M. al-Rifaie, A. Aber, and R. Raisys, "Swarming robots and possible medical applications," in *International Society for the Electronic Arts (ISEA 2011)*, (Istanbul, Turkey), 2011.

[121] M. M. al-Rifaie, A. Aber, and M. Bishop, "Swarms search for cancerous lesions: Artificial intelligence use for accurate identification of bone metastasis on bone scans," *The European Federation of National Associations of Orthopaedics and Traumatology (EFORT), 13th EFORT Congress, Berlin, Germany*, 2012.

[122] K. de Meyer, "Explorations in stochastic diffusion search: Soft- and hardware implementations of biologically inspired spiking neuron stochastic diffusion networks," tech. rep., Technical Report KDM/JMB/2000, 2000.

[123] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Proceedings of the sixth international symposium on micro machine and human science*, vol. 43, New York, NY, USA: IEEE, 1995.

[124] F. Heppner and U. Grenander, "A stochastic nonlinear model for coordinated bird flocks.," *American Association for the Advancement of Science, Washington, DC(USA).*, 1990.

[125] M. Mataric, *Interaction and Intelligent Behavior*. PhD thesis, Department of Electrical, Electronics and Computer Engineering, MIT, USA, 1994.

[126] O. B. Bayazit, J.-M. Lien, and N. M. Amato, "Roadmap-based flocking for complex environments," in *PG '02: Proceedings of the 10th Pacific*

Conference on Computer Graphics and Applications, (Washington, DC, USA), p. 104, IEEE Computer Society, 2002.

[127] C. H. Janson, "Experimental evidence for spatial memory in foraging wild capuchin monkeys,cebus apella," *Animal Behaviour*, vol. 55, pp. 1229–1243, 1998.

[128] J. Kennedy, "Thinking is social: Experiments with the adaptive culture model," *Journal of Conflict Resolution*, vol. 42, pp. 56–76, 1998.

[129] L. Festinger, "A theory of social comparison processes," *Human relations*, vol. 7, no. 2, pp. 117–140, 1954.

[130] R. C. Eberhart, P. K. Simpson, and R. Dobbins, *Computational Intelligence PC Tools*. Boston: Academic Press, 1996.

[131] J. Kennedy, R. C. Eberhart, and Y. Shi, "Swarm intelligence," *San Francisco: Morgan Kauffman*, 2001.

[132] S. A. Kauffman, *The Origins of Order: Self-organization and Selection in Evolution (1993)*. New York: Oxford University Press, 1993.

[133] S. A. Kauffman, *At home in the universe: The search for the laws of self-organization and complexity*. New York: Oxford University Press, 1995.

[134] A. P. Engelbrecht, *Fundamentals of Computational Swarm Intelligence*. Wiley, 2006.

[135] J. Kennedy, "Bare bones particle swarms," in *Proceedings of Swarm Intelligence Symposium, 2003 (SIS'03)*, pp. 80–87, IEEE, 2003.

[136] R. Rucker, *Seek!* New York: Four Walls Eight Windows, 1999.

[137] T. Blackwell and D. Bratton, "Examination of particle tails," in *Journal of Artificial Evolution and Applications*, Hindawi Publishing Corporation, 2008.

[138] F. V. den Bergh, *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria, South Africa, 2002.

[139] T. Krink, J. S. Vesterstrom, and J. Riget, "Particle swarm optimisation with spatial particle extension," in *Proceedings of the Evolutionary Computation on 2002. CEC '02.*, (Washington, DC, USA), pp. 1474–1479, IEEE Computer Society, 2002.

[140] M. Omran, A. Salman, and A. P. Engelbrecht, "Image classification using particle swarm optimization," in *Proceedings of the 4th Asia-pacific Conference on Simulated Evolution and Learning*, pp. 370–374, 2002.

[141] Y. Shi and R. C. Eberhart, "Parameter selection in particle swarm optimization," *Lecture notes in computer science*, pp. 591–600, 1998.

[142] Eberhart and Y. Shi, "Particle swarm optimization: developments, applications and resources," in *Proceedings of the 2001 Congress on Evolutionary Computation, 2001.*, vol. 1, pp. 81–86 vol. 1, 2001.

[143] Y. Shi and R. C. Eberhart, "A modified particle swarm optimizer," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, (Piscataway, NJ), pp. 69–73, IEEE Press, 1998.

[144] J. F. Schutte and A. A. Groenwold, "Sizing design of truss structures using particle swarms," *Structural and Multidisciplinary Optimization*, vol. 25, pp. 261–269, October 2003.

[145] P. C. Fourie and A. A. Groenwold, "The particle swarm optimization algorithm in size and shape optimization," *Structural and Multidisciplinary Optimization*, vol. 23, no. 4, pp. 259–267, 2002.

[146] M. Clerc, "The swarm and the queen: towards a deterministic and adaptive particle swarm optimization," in *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99.*, vol. 3, p. 1957 Vol. 3, 1999.

[147] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in amultidimensional complex space," *Evolutionary Computation, IEEE Transactions on*, vol. 6, no. 1, pp. 58–73, 2002.

[148] J. C. Anthony, *Applying the Particle Swarm Optimizer to Non-Stationary Environments*. PhD thesis, Auburn University, 2002.

[149] R. Brits, A. P. Engelbrecht, and F. van den Bergh, "A niching particle swarm optimizer," *Proceedings of the 4th Asia-Pacific Conference on Simulated Evolution and Learning (SEAL'02)*, vol. 2, pp. 692–696, 2002.

[150] R. Brits, "Niching strategies for particle swarm optimization," *Master's thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa*, 2002.

[151] R. J. W. Hodgson, "Partical swarm optimization applied to the atomic cluster optimization problem," in *GECCO 2002*, pp. 68–73, 2002.

[152] A. P. Engelbrecht and A. Ismail, "Training product unit neural networks," *Stability and Control: Theory and Applications*, vol. 2, no. 1-2, pp. 59–74, 1999.

[153] D. Bratton and J. Kennedy, "Defining a standard for particle swarm optimization," in *Proc of the Swarm Intelligence Symposium*, (Honolulu, Hawaii, USA), pp. 120–127, IEEE, 2007.

[154] J. Kennedy, "The particle swarm: social adaptation of knowledge," *Evolutionary Computation, 1997., IEEE International Conference on*, pp. 303–308, 1997.

[155] A. Carlisle and G. Dozier, "Adapting particle swarm optimization to dynamic environments," *International Conference on Artificial Intelligence*, vol. 1, pp. 429–434, 2000.

[156] F. van den Bergh and A. P. Engelbrecht, "Effects of swarm size on cooperative particle swarm optimisers," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, 2001.

[157] J. Kennedy, "Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance," in *In: Proceedings of the 1999, Congress of Evolutionary Computation*, vol. 3, pp. 1931–1938, IEEE Press, 1999.

[158] J. Kennedy and R. Mendes, "Population structure and particle swarm performance," in *Proceedings of the 2002 Congress of the Evolutionary Computation on 2002. CEC '02.*, (Washington, DC, USA), pp. 1671–1676, IEEE Computer Society, 2002.

[159] R. Mendes, P. Cortez, M. Rocha, and J. Neves, "Particle swarms for feedforward neural network training," in *Proceedings of the 2002 Congress of the Evolutionary Computation on 2002. CEC '02*, vol. 2, (Honolulu, Havai, USA), pp. 1895–1899, International Joint Conference On Neural Networks (IJCNN), 2002.

[160] A. Carlisle and G. Dozier, "An off-the-shelf pso," *Proceedings of the Workshop on Particle Swarm Optimization*, vol. 1, pp. 1–6, 2001.

[161] E. Ozcan and C. K. Mohan, "Analysis of a simple particle swarm optimization system," *Intelligent Engineering Systems Through Artificial Neural Networks*, vol. 8, pp. 253–258, 1998.

[162] E. Ozcan and C. K. Mohan, "Particle swarm optimization: surfing the waves," *Proceedings of the 1999 Congress on Evolutionary Computation, 1999. CEC 99.*, vol. 3, 1999.

[163] F. van den Bergh and E. A. P., "A study of particle swarm optimization particle trajectories," *Information Sciences*, vol. 176, no. 8, pp. 937–971, 2006.

[164] X. F. Xie, W. J. Zhang, and Z. L. Yang, "Hybrid particle swarm optimizer with mass extinction," *IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions*, vol. 2, 2002.

[165] X. F. Xie, W. J. Zhang, and Z. L. Yang, "Adaptive particle swarm optimization on individual level," *Signal Processing, 2002 6th International Conference on*, vol. 2, 2002.

[166] X. F. Xie, W. J. Zhang, and Z. L. Yang, "A dissipative particle swarm optimization," *Arxiv preprint cs.NE/0505065*, 2005.

[167] A. GIUNTA, V. BALABANOV, D. HAIM, B. GROSSMAN, W. MASON, L. WATSON, and R. HAFTKA, "Multidisciplinary optimisation of a supersonic transport using design of experiments theory and response surface modelling," *Aeronautical Journal*, vol. 101, no. 1008, pp. 347–356, 1997.

[168] G. Venter and J. Sobieszczanski-Sobieski, "Particle swarm optimization," *American Institute of Aeronautics and Astronautics (AIAA)*, vol. 41, no. 8, pp. 1583–1589, 2003.

[169] M. Lovbjerg, "Improving particle swarm optimization by hybridization of stochastic search heuristics and self-organized criticality," 2002.

[170] M. Lovbjerg and T. Krink, "Extending particle swarm optimisers with self-organized criticality," *Proceedings of the 2002 Congress on Evolutionary Computation, 2002. CEC'02.*, vol. 2, 2002.

[171] M. Middendorf, F. Reischle, and H. Schmeck, "Information exchange in multi colony ant algorithms," *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 645–652, 2000.

[172] M. Middendorf, F. Reischle, and H. Schmeck, "Multi colony ant algorithms," *Journal of Heuristics*, vol. 8, no. 3, pp. 305–320, 2002.

[173] T. G. Crainic, M. Toulouse, and M. Gendreau, "Synchronous tabu search parallelization strategies for multicommodity location-allocation with balancing requirements," *OR Spectrum*, vol. 17, no. 2, pp. 113–123, 1995.

[174] T. G. Crainic and M. Gendreau, "Cooperative parallel tabu search for capacitated network design," *Journal of Heuristics*, vol. 8, no. 6, pp. 601–627, 2002.

[175] E. Cantu-Paz, "A survey of parallel genetic algorithms," *Calculateurs Paralleles, Reseaux et Systems Repartis*, vol. 10, no. 2, pp. 141–171, 1998.

[176] M. Nowostawski and R. Poli, "Parallel genetic algorithm taxonomy," in *Knowledge-Based Intelligent Information Engineering Systems, 1999. Third International Conference*, pp. 88–92, 1999.

[177] F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 3, pp. 225–239, 2004.

[178] S. Baskar and P. N. Suganthan, "A novel concurrent particle swarm optimization," in *Congress on Evolutionary Computation, 2004. CEC2004.*, vol. 1, 2004.

[179] M. el Abd and M. S. Kamel, "A taxonomy of cooperative particle swarm optimizers," *International Journal of Computational Intelligence Research*, vol. 4, no. 2, pp. 137–144, 2008.

[180] M. A. Potter and K. A. D. Jong, "A cooperative coevolutionary approach to function optimization," *LECTURE NOTES IN COMPUTER SCIENCE*, pp. 249–249, 1994.

[181] T. Peram, K. Veeramachaneni, and C. K. Mohan, "Fitness-distance-ratio based particle swarm optimization," in *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pp. 174–181, 2003.

[182] M. el Abd and M. Kamel, "A hierarchal cooperative particle swarm optimizer," in *In Proc. IEEE swarm intelligence symposium*, 2006.

[183] B. Niu, Y. Zhu, and X. He, "Multi-population cooperative particle swarm optimization," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 3630, pp. 874–883, 2005.

[184] J. J. Liang and P. N. Suganthan, "Dynamic multi-swarm particle swarm optimizer," in *Swarm Intelligence Symposium, 2005. SIS 2005. Proceedings 2005 IEEE*, pp. 124–129, 2005.

[185] T. Blackwell and J. Branke, "Multi-swarm optimization in dynamic environments," *LECTURE NOTES IN COMPUTER SCIENCE*, vol. 3005, pp. 489–500, 2004.

[186] R. Poli, "Analysis of the publications on the applications of particle swarm optimisation," *Journal of Artificial Evolution and Applications*, vol. 2008, p. 3, 2008.

[187] R. Poli, "An analysis of publications on particle swarm optimization applications," *Essex, UK: Department of Computer Science, University of Essex*, 2007.

[188] Y. Kim, S. Keely, J. Ghosh, and H. Ling, "Application of artificial neural networks to broadband antenna design based on a parametric frequency model," *Antennas and Propagation, IEEE Transactions on*, vol. 55, no. 3, pp. 669–674, 2007.

[189] R. Azaro, F. De Natale, M. Donelli, A. Massa, and E. Zeni, "Optimized design of a multifunction/multiband antenna for automotive rescue systems," *Antennas and Propagation, IEEE Transactions on*, vol. 54, no. 2, pp. 392–400, 2006.

[190] J. Perez and J. Basterrechea, "Comparison of different heuristic optimization methods for near-field antenna measurements," *Antennas and Propagation, IEEE Transactions on*, vol. 55, no. 3, pp. 549–555, 2007.

[191] S. Selvan, C. Xavier, N. Karssemeijer, J. Sequeira, R. Cherian, and B. Dhala, "Parameter estimation in stochastic mammogram model by heuristic optimization techniques," *Information Technology in Biomedicine, IEEE Transactions on*, vol. 10, no. 4, pp. 685–695, 2006.

[192] K. Veeramachaneni, L. Osadciw, and P. Varshney, "An adaptive multimodal biometric management algorithm," *Systems, Man, and Cy-*

bernetics, Part C: Applications and Reviews, IEEE Transactions on, vol. 35, no. 3, pp. 344–356, 2005.

[193] J. Heo, K. Lee, and R. Garduno-Ramirez, "Multiobjective control of power plants using particle swarm optimization techniques," *Energy Conversion, IEEE Transactions on*, vol. 21, no. 2, pp. 552–561, 2006.

[194] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, and Y. Nakanishi, "A particle swarm optimization for reactive power and voltage control considering voltage security assessment," *Power Systems, IEEE Transactions on*, vol. 15, no. 4, pp. 1232–1239, 2000.

[195] S. Kannan, S. Slochanal, and N. Padhy, "Application and comparison of metaheuristic techniques to generation expansion planning problem," *Power Systems, IEEE Transactions on*, vol. 20, no. 1, pp. 466–475, 2005.

[196] M. Donelli and A. Massa, "Computational approach based on a particle swarm optimizer for microwave imaging of two-dimensional dielectric scatterers," *Microwave Theory and Techniques, IEEE Transactions on*, vol. 53, no. 5, pp. 1761–1776, 2005.

[197] T. Huang and A. Mohan, "A microparticle swarm optimizer for the reconstruction of microwave images," *Antennas and Propagation, IEEE Transactions on*, vol. 55, no. 3, pp. 568–576, 2007.

[198] M. Wachowiak, R. Smolíková, Y. Zheng, J. Zurada, and A. Elmaghraby, "An approach to multimodal biomedical image registration utilizing particle swarm optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 8, no. 3, pp. 289–301, 2004.

[199] Y. Song, Z. Chen, and Z. Yuan, "New chaotic pso-based neural network predictive control for nonlinear process," *Neural Networks, IEEE Transactions on*, vol. 18, no. 2, pp. 595–601, 2007.

[200] C. Juang, "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design," *Systems, Man, and Cybernetics,*

Part B: Cybernetics, IEEE Transactions on, vol. 34, no. 2, pp. 997–1006, 2004.

[201] G. Venayagamoorthy and W. Zha, "Comparison of nonuniform optimal quantizer designs for speech coding with adaptive critics and particle swarm," *Industry Applications, IEEE Transactions on*, vol. 43, no. 1, pp. 238–244, 2007.

[202] C. Juang and C. Hsu, "Temperature control by chip-implemented adaptive recurrent fuzzy controller designed by evolutionary algorithm," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 52, no. 11, pp. 2376–2384, 2005.

[203] B. Liu, L. Wang, and Y. Jin, "An effective pso-based memetic algorithm for flow shop scheduling," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, no. 1, pp. 18–27, 2007.

[204] A. Chatterjee, K. Pulasinghe, K. Watanabe, and K. Izumi, "A particle-swarm-optimized fuzzy-neural network for voice-controlled robot systems," *Industrial Electronics, IEEE Transactions on*, vol. 52, no. 6, pp. 1478–1489, 2005.

[205] R. Thomsen, "Flexible ligand docking using evolutionary algorithms: investigating the effects of variation operators and local search hybrids," *Biosystems*, vol. 72, no. 1-2, pp. 57–73, 2003.

[206] J. Vesterstrom and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," in *Evolutionary Computation, 2004. CEC2004. Congress on*, vol. 2, pp. 1980–1987, 2004.

[207] R. Storn and K. Price, "Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces," 1995. TR-95-012, [online]. Available: http://www.icsi.berkeley.edu/ storn/litera.html.

[208] R. Storn and K. Price, "Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces," *J. Global Optim.*, vol. 11, pp. 341–359, 1997.

[209] D. Tasoulis, N. Pavlidis, V. Plagianakos, and M. Vrahatis, "Parallel differential evolution," in *Congress on Evolutionary Computation CEC2004.*, vol. 2, pp. 2023–2029, IEEE, 2004.

[210] K. Kozlov and A. Samsonov, "New migration scheme for parallel differential evolution," in *Proceedings of the international conference on bioinformatics of genome regulation and structure*, pp. 141–144, 2006.

[211] M. Tasgetiren and P. Suganthan, "A multi-populated differential evolution algorithm for solving constrained optimization problem," in *IEEE Congress on Evolutionary Computation CEC2006.*, pp. 33–40, IEEE, 2006.

[212] R. Mendes and A. Mohais, "DynDE: a differential evolution for dynamic optimization problems," in *The 2005 IEEE Congress on Evolutionary Computation CEC2005.*, vol. 3, pp. 2808–2815, IEEE, 2005.

[213] J. Brest, A. Zamuda, B. Boskovic, M. Maucec, and V. Zumer, "Dynamic optimization using self-adaptive differential evolution," in *IEEE Congress on Evolutionary Computation, 2009. CEC'09.*, pp. 415–422, IEEE, 2009.

[214] T. Smuc, "Improving convergence properties of the differential evolution algorithm," in *Proceedings of the MENDEL 2002 - 8th International Conference on Soft Computing*, pp. 80–86, 2002.

[215] M. Weber, F. Neri, and V. Tirronen, "Parallel Random Injection Differential Evolution," *Applications of Evolutionary Computation*, pp. 471–480, 2010.

[216] J. Zhang and A. Sanderson, "JADE: adaptive differential evolution with optional external archive," *Evolutionary Computation, IEEE Transactions on*, vol. 13, no. 5, pp. 945–958, 2009.

[217] V. Huang, P. Suganthan, A. Qin, and S. Baskar, "Multiobjective differential evolution with external archive and harmonic distance-based diversity measure," *School of Electrical and Electronic Engineering Nanyang, Technological University Technical Report*, 2005.

[218] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in *Proceedings of the MENDEL 2003 - 9th International Conference on Soft Computing*, pp. 41–46, 2003.

[219] M. Omran, I. Moukadem, S. al-Sharhan, and M. Kinawi, "Stochastic diffusion search for continuous global optimization," *International Conference on Swarm Intelligence (ICSI 2011), Cergy, France*, 2011.

[220] M. Clerc, "From theory to practice in particle swarm optimization," *Handbook of Swarm Intelligence*, pp. 3–36, 2010.

[221] I. C. Trelea, "The particle swarm optimization algorithm: convergence analysis and parameter selection," *Information Processing Letters*, vol. 85, no. 6, pp. 317–325, 2003.

[222] Y. L. Zheng, L. H. Ma, L. Y. Zhang, and J. X. Qian, "On the convergence analysis and parameter selection in particle swarm optimization," *International Conference on Machine Learning and Cybernetics*, vol. 3, 2003.

[223] D. R. Jones, C. D. Perttunen, and B. E. Stuckman, "Lipschitzian optimization without the lipschitz constant," *J. Optim. Theory Appl.*, vol. 79, no. 1, pp. 157–181, 1993.

[224] K. A. D. Jong, *An analysis of the behavior of a class of genetic adaptive systems.* PhD thesis, University of Michigan, Ann Arbor, MI, USA, 1975.

[225] D. Gehlhaar and D. Fogel, "Tuning evolutionary programming for conformationally flexible molecular docking," in *Evolutionary Programming V: Proc. of the Fifth Annual Conference on Evolutionary Programming*, pp. 419–429, 1996.

[226] M. M. al-Rifaie, M. Bishop, and T. Blackwell, "An investigation into the merger of stochastic diffusion search and particle swarm optimisation," in *GECCO '11: Proceedings of the 2011 GECCO conference companion on Genetic and evolutionary computation*, (Dublin, Ireland), pp. 37–44, ACM, 2011.

[227] M. M. al-Rifaie, M. Bishop, and T. Blackwell, "Resource allocation and dispensation impact of stochastic diffusion search on differential evolution algorithm; in," in *Nature Inspired Cooperative Strategies for Optimisation (NICSO 2011), Springer*, 2011.

[228] M. M. al-Rifaie, M. Bishop, and T. Blackwell, "Information sharing impact of stochastic diffusion search on differential evolution algorithm," in *Journal of Memetic Computing: Nature Inspired Cooperative Strategies for Optimization* (D. Pelta and et al, eds.), Springer Berlin Heidelberg, 2012. submitted.

[229] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y. P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," tech. rep., Nanyang Technological University, Singapore and Kanpur Genetic Algorithms Laboratory, IIT Kanpur, 2005.

Appendix A

Publications

The following publications were derived from or influenced by this work.

- M. M. al-Rifaie and M. Bishop, The mining game: a brief introduction to the stochastic diffusion search metaheuristic, *The Society for the Study of Artificial Intelligence and the Simulation of Behaviour Quarterly (AISBQ),* vol. 130, 2010.
- M. M. al-Rifaie, M. Bishop, and A. Aber, Creative or not? birds and ants draw with muscles, In *Proc of the AISB 2011: Computing and Philosophy*, University of York, York, U.K., pp. 23-30, 2011. ISBN: 978-1-908187- 03-1.
- M. M. al-Rifaie, A. Aber, and M. Bishop, Cooperation of nature and physiologically inspired mechanisms in visualisation, Chapter In *Biologically-Inspired Computing for the Arts: Scientific Data through Graphics* (A. Ursyn, ed.), IGI Global, United States, 2012. ISBN13: 9781466609426, ISBN10: 1466609427.
- M. M. al-Rifaie, M. Bishop, and T. Blackwell, An investigation into the merger of stochastic diffusion search and particle swarm optimisation, In *Proc of the GECCO '11: Proceedings of the 2011 GECCO conference companion on Genetic and evolutionary computation*, Dublin, Ireland, pp. 37-44, ACM, 2011.

- M. M. al-Rifaie, A. Aber, and R. Raisys, Swarming robots and possible medical applications, In *Proc of the International Society for the Electronic Arts (ISEA 2011)*, Istanbul, Turkey, 2011.

- M. M. al-Rifaie, M. Bishop, and T. Blackwell, Resource allocation and dispensation impact of stochastic diffusion search on differential evolution algorithm, In *Proc of the Nature Inspired Cooperative Strategies for Optimisation (NICSO 2011)*, Studies in Computational Intelligence. Springer, 2011.

- M. M. al-Rifaie, M. Bishop, and T. Blackwell, An investigation into the use of swarm intelligence for an evolutionary algorithm optimisation, In *Proc of the International Conference on Evolutionary Computation Theory and Application (ECTA 2011)*, Paris, France, 2011.

- M. M. al-Rifaie and M. Bishop, and T. Blackwell, Information Exchange In Population-Based Algorithms, *The Society for the Study of Artificial Intelligence and the Simulation of Behaviour Quarterly (AISBQ)*, vol. 134, 2011.

- A. Aber, M. M. al-Rifaie and M. Bishop, Swarms Search for Cancerous Lesions: Artificial Intelligence Use for Accurate Identification of Bone Metastasis on Bone Scans, The European Federation of National Associations of Orthopaedics and Traumatology (EFORT), 13th EFORT Congress, Berlin, Germany, 2012.

- M. M. al-Rifaie, M. Bishop, and S. Caines, Creativity and Autonomy in Swarm Intelligence Systems, In *Journal of Cognitive Computation: Computational Creativity, Intelligence and Autonomy* (M. Bishop and Y. Erden, eds.), Springer, 2012.

- M. M. al-Rifaie, M. Bishop, and T. Blackwell, Information sharing impact of stochastic diffusion search on differential evolution algorithm, In *Journal of Memetic Computing* (D. Pelta and et al, eds.), Studies in Computational Intelligence and Complexity. Springer., 2012. (submitted)

Appendix B

The Blind Men and the Elephant

It was six wise men of Indostan
To learning much inclined,
Who went to see the Elephant – (though all of them were blind),
That each by observation—might satisfy his mind.

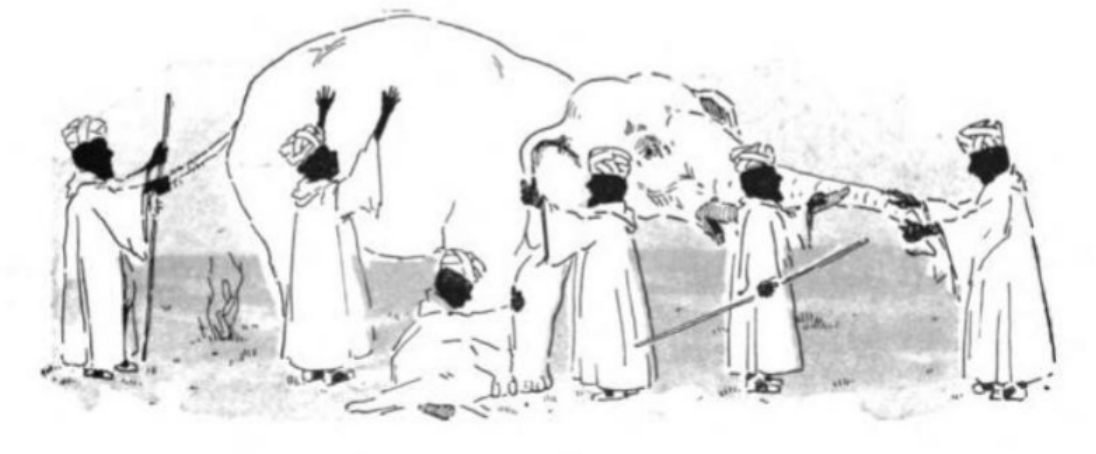
The First approached the Elephant,
And happening to fall
Against his broad and sturdy side—At once began to bawl:
"God bless me! But the Elephant—Is very like a wall!

The Second, feeling of the tusk,
Cried, "Ho! What have we here?
So very round and smooth and sharp – To me 'tis mighty clear
This wonder of an Elephant – Is very like a spear.

The Third approached the animal,
And happening to take
The squirming trunk within his hands, – Thus boldly up and spake:
"I see," quote he, "the Elephant—Is very like a snake!"

The fourth reached out his eager hand,

Figure B.1: The blind men and the elephant



Source: From Charles Maurice Stebbins & Mary H. Coolidge, Golden Treasury Readers: Primer, American Book Co. (New York), p. 89

And felt about the knee:
"What most this wondrous beast is like, is mighty plain," quoth he;
"Tis clear enough the elephant is very like a tree."

The Fifth who chanced to touch the ear,
Said: "Even the blindest man
Can tell what this resembles most; – Deny the fact who can,
This marvel of an Elephant – Is very like a fan!"

The Sixth no sooner had begun
About the beast to grope,
Than seizing on the swinging tail –That fell within his scope,
"I see", said he, "the Elephant – Is very like a rope!

And so these men of Indostan Disputed loud and long, Each in his own opinion –Exceeding stiff and strong Though each was partly in the right –And all were in the wrong!

John Godfrey Saxe

Appendix C

The Restaurant Game

"A group of delegates attends a long conference in an unfamiliar town. Each night they have to find somewhere to dine. There is a large choice of restaurants, each of which offers a large variety of meals. The problem the group faces is to find the best restaurant, that is the restaurant where the maximum number of delegates would enjoy dining. Even a parallel exhaustive search through the restaurant and meal combinations would take too long to accomplish. To solve the problem delegates decide to employ a Stochastic Diffusion Search.

Each delegate acts as an agent maintaining a hypothesis identifying the best restaurant in town. Each night each delegate tests his hypothesis by dining there and randomly selecting one of the meals on offer. The next morning at breakfast every delegate who did not enjoy his meal the previous night, asks one randomly selected colleague to share his dinner impressions. If the experience was good, he also adopts this restaurant as his choice. Otherwise he simply selects another restaurant at random from those listed in 'Yellow Pages'.

Using this strategy it is found that very rapidly significant number of delegates congregate around the best restaurant in town." [51]

There is however a pitfall in this metaphor which is illustrated in the following scenario:

In a pathological case, consider two diners (D1 and D2) with two restaurants (R1 and R2) in the town, each serving just one meal. Also it is known that:

- diner D1 likes both of the meals at restaurant R1 and R2,
- but diner D2 only enjoys the meal at restaurant R2.

In this case, if

- diner D1 initially chooses restaurant R1 for his meal
- and diner D2 chooses R2,

neither will ever leave the restaurants of their choice.

Therefore, since diner D1 never leaves R1, the diners will never converge on restaurant R2, where most of the diners enjoy having their meals in.