TÉCNICO LISBOA

Genetic Roach Infestation Optimization

Optimization Algorithm based on Cockroaches' social life

António Gustavo Lança Brites

Thesis to obtain the Master of Science Degree in

Mechanical Engineering

Supervisor: Prof. João Miguel da Costa Sousa

Examination Committee

Chairperson: Prof. Paulo Jorge Coelho Ramalho Oliveira

Supervisor: Prof. João Miguel da Costa Sousa

Member of the Committee: Prof. Duarte Pedro Mata de Oliveira Valério

November 2017

Acknowledgements

This work is the final of an academic path. However, for me, it is even more special once it only come true due to all the support my family always gave me while I was already working. I will never forget it and I will be thankful all my entire life.

Also to my thesis advisor for all the patience and wise advises despite my lack of availability. It was really appreciated.

Finally to every single student and teacher that crossed with me and more or less contributed with a piece of who I am and where I am.

Thank you all.

Abstract

This work presents a novel adaptation of Particle Swarm Optimization (PSO) algorithm based on Roach Infestation Optimization (RIO). This algorithm is inspired on cockroaches' scientifically demonstrated behaviours. Cockroaches have been able to survive over the centuries and that is only possible due to their capacity to live in community and share information, for example the location of like dark spots to hide or food. Genetic Roach Infestation Optimization (GRIO) is more tied to roaches' life than RIO. It uses a multi-swarm and self-adaptive strategy to avoid falling into local optima and converging too soon. Using a self-adaptive inertia weight, it is possible to explore a global or a local search, according to the algorithm feedback. This algorithm allows, in the same space, the presence of two different families of cockroaches that behave independently. To keep the information for both families balanced, reproduction behaviour (based on a regular Genetic Algorithm) is carried out. In that reproduction, born child will have a crossover of parents' information. To preserve diversity some mutations are enabled. Afterwards, the proposed algorithm is tested in some benchmark functions with several different characteristics in comparison to other algorithms. Results clearly support that cockroaches' life adapted into PSO algorithm greatly improves optimization results on the most adverse functions.

Key words: Optimization, Nature, Continuous, Cockroaches, Metaheuristic, Algorithm, PSO, RIO.

Resumo

Este trabalho apresenta uma nova adaptação do Particle Swarm Optimization (PSO) baseada no Roach Infestation Optimization (RIO). Todos os comportamentos das baratas que serviram de inspiração a este algoritmo estão comprovados pela biologia. As baratas habitam neste planeta há séculos, ultrapassando diversas fases do mesmo. Isto não teria sido possível sem a capacidade delas em viver em comunidade e partilhar informação. Por exemplo, a decisão do melhor local escuro para esconder e encontrar alimento é decidida em sociedade. Este algorimo - Genetic Roach Infestation Optimization (GRIO), representa mais aspectos da vida das baratas que o RIO. Usa duas famílias distintas de baratas e um peso inercial adaptativo (que permite uma busca mais global ou local de acordo com a situação) para evitar óptimos locais. As duas famílias têm comportamentos independentes. No entanto, para haver troca de informação, entre elas, pode haver reprodução (baseada num algoritmo genético normal). Uma vez que a reprodução é entre familias diferentes, a cria terá informação de ambas as familias. Para aumentar ainda mais a diversidade, algumas crias poderão sofrer mutações no código genético. Por fim, o algoritmo proposto é testado em algumas funções referência (com características diversas) e comparado com outros algoritmos. Os resultados comprovam que os comportamentos das baratas adaptados ao PSO aumentam a possibilidade de obter os melhores resultados óptimos possíveis.

Palavras-chave: Optimização, Natureza, Contínuo, Baratas, Meta-heurísticas, Algoritmo, PSO, RIO.

Index

Acknowledgements i

Abstract ii

Resumo iii

List of Figures vi

List of Tables vii

List of Symbols viii

List of Abbreviations ix

List of Programs ix

1 Introduction 1

1.1 Objectives and Motivation 1

1.2 State Of The Art 2

1.2.1. Optimization: a General Overview 2

1.2.2. Cockroaches' social life 6

1.3 Thesis' contributions 7

1.4 Thesis' organization 7

2 Non Linear Problems: Metaheuristics 9

2.1 Particle Swarm Optimization 9

2.2 Roach Infestation Optimization 13

2.3 Other Algorithms 15

3 Genetic Roach Infestation Optimization 17

3.1 Overall Description 17

3.2 Algorithm Description 19

3.2.1. Search Behaviour 21

3.2.2. Adaptive Inertia Weight 21

3.2.3. Roaches' Reproduction 25

3.2.4. Roaches' Mutation 27

4 Results 29

4.1 Benchmarks 29
4.2 Results 31
4.2.1. Number of Function Evaluations 36
4.3 Discussion 38
4.3.1. Very Low Complexity 40
4.3.2. Low Complexity 42
4.3.3. Medium Complexity 45
4.3.4. High Complexity 48
5 Conclusions and Future Work 53
5.1 Conclusions 53
5.2 Future Work 55
6 References 57

List of Figures

Figure 1.1: Diagram for OR to solve a problem 3

Figure 2.1: Neighbourhood topologies. Global Best (a). Ring topology–Local Best (b). Wheel topology (c). 11

Figure 3.1: Parallelism between roaches' behaviours and GRIO 18

Figure 3.2: GRIO's Flowchart 20

Figure 3.3: Adaptive behaviour flowchart 22

Figure 3.4: Inertia weight oscillations. a) Oscillations for roach 1. b) Oscillations for roach 10. c) Oscillations for cockroach 20. d) Oscillations for all roaches 23

Figure 3.5: Baby roach's birth example 25

Figure 3.6: Reproduction behaviour flowchart 26

Figure 3.7: Mutations. a) Replacement mutation. b) Frameshift mutation. 27

Figure 4.1: Convergence graphs for Very Low Complexity functions. 40

Figure 4.2: Boxplots for Very Low Complexity functions. 41

Figure 4.3: Convergence graphs for Low Complexity functions. 43

Figure 4.4: Boxplots for Very Low Complexity functions. 44

Figure 4.5: Convergence graphs for Medium Complexity functions. 46

Figure 4.6: Boxplots for Medium Complexity functions. 47

Figure 4.7: Convergence graphs for High Complexity functions. 49

Figure 4.8: Boxplots for High Complexity functions. 50

Figure 5.1: Percentage of functions where GRIO scored best per group of functions 54

List of Tables

Table 1.1: Some applications of operations research [9] 3

Table 2.1: Particle Swarm Optimization 12

Table 2.2: Roach Infestation Optimization 14

Table 2.3: Probability of roaches' socialization according to number of neighbours [14] 15

Table 3.1: Genetic Roach Infestation Optimization 28

Table 4.1: Benchmark functions and their characteristics. 30

Table 4.2: Benchmark Function's specifications. 31

Table 4.3: Complete Results (bolded best results) 33

Table 4.4: Function evaluations for each tested algorithm. Bolded the ones that achieved the best result. 37

Table 4.5: Function's classification according to defined characteristics (ordered by increasing complexity). 39

List of Symbols

x – Optimization problems variables

$F(x)\ and\ Z(x)$ – Objective/fitness function

$C(x)$ – Constrains vector

D – Number of variables in a problem

X_i – Particle's position

V_i – Particle's velocity

P_{best} – Particle's best position

G_{best} – Swarm's best position

L_{best} – Neighbours' best position

w – Inertia weight

d_r – Roach's detection radius

bk – Roach's best known position

bv – Roach's best visited position

bp – Best position ever achieved

d – Distance to family's best position

ha – Hyper activity factor

$Prob\_rep$ – Probability of Reproduction

$Prob\_mut\_rep$ – Probability of Replacement Mutation

$Prob\_mut\_fram$ – Probability of Frameshift Mutation

List of Abbreviations

OR – Operation Research

RIO – Roach Infestation Optimization

PSO – Particle Swarm Optimization

GLODS – Global and local direct search

GRIO – Genetic Roach Infestation Optimization

LW-PSO – Linear Weight Particle Swarm Optimization

RW-PSO – Random Weight Particle Swarm Optimization

MCPSO – Multi-Swarm Cooperative Particle Swarm Optimization

KCPSO – Knowledge-based Cooperative Particle Swarm Optimization

List of Programs

Matlab R2012®

1 Introduction

This chapter objective is to explain this work context. It is explained why this thesis was done, motivations and goals. Also, a general overview of optimization is presented, in order to allow a reader with less optimization experience to follow this work and, maybe, become an optimization enthusiast.

The document organization and contributions are also explained so that readers can follow it in an easier and effective way.

1.1 Objectives and Motivation

This work started as a spontaneous idea. While watching a documentary (episode 7 from *Pure Science Specials* season 1) about cockroaches' behaviour it was understood that those animals have a lot of potential. For example, according to that documentary, their anatomy inspired robots that can help humanity and also some medical developments.

The documentary presented the different fields of technology that used cockroaches and also a thorough explanation of cockroaches' life. It was clear to me that these animals behave in a highly optimized way. It was impossible to think in a way to make cockroaches' life more efficient. Cockroaches, really, work as a team in order to find the best places to eat and to hide. Actually, their behaviour follows some patterns easily identified that allows them to behave in such an optimized way. In fact, after some read around those patterns, I came to the idea that, if we created those rules mathematically (the same as creating virtual cockroaches) we could find mathematical functions' best solution.

Nowadays, optimization is conquering a major role in our world [35]. Every single decision in military approaches, companies decisions and even governmental boards is taken with much more careful and advice. A decision slightly better than the opponent's one can deliver success, and a decision slightly worst can deliver failure [9].

That's why the investment in optimization resources is getting bigger and bigger [9]. Actually, a lot of companies spread around the world are creating optimization departments in order to increase profits, decrease wastes, increase productivity, reduce line setup times, increase new product development success, etc [9].

With the understanding of the increasing importance of optimization in companies and in life in general, and in a time were the future professional career must be defined, to create a master thesis in the optimization area was an obvious choice.

With the objective to create a new approach to optimization and to be creative, this thesis theme come up and work was initiated.

Summarizing: taking into consideration the importance of optimization in our world and the very efficient behaviour of cockroaches' colonies, the idea to mimic cockroaches' life into an optimization algorithm capable of solve the most difficult functions appeared.

1.2 State Of The Art

In this chapter, a brief description of optimization basics is presented. It is expected that a user not familiarized with optimization can understand the basics, and with that knowledge, follow this work. Then, cockroaches' real behaviours, in a biologic perspective, are presented.

1.2.1. Optimization: a General Overview

Humanity, even in pre-history, always sought the best option to every single decision in order to save resources, time or energy. For example a hospital service wants to reduce the time each patient waits for medical assistance, a formula one team changing tyres wants to reduce its activities time and a car manufacturer tries to reduce vehicle's energy consumption. Even in nature, every single process is carried in order to reduce energy and process time among others.

Once people started to understand optimization's potential (by saving costs in industrial processes or time in finding complex problems solution, for example), engineers and mathematicians have been directing their efforts to develop new strong, effective and efficient analytical methods that could deliver the best way to find a solution and decide an approach – activity named operations research [9, 35]. So, as we can see, operation research (OR) and optimization are intrinsically connected once researchers use a lot of optimization algorithms to underlie their decisions in order to find a best solution (there are problems where there's no unique optimal solution). Due to the evolution in operation research and the results that this concept obtains, it has been possible to companies and researchers to improve profit, process time, energy, resources among others. In table 1.1 it's possible to check some of the benefits OR brought to some organizations.

Table 1.1: Some applications of operations research [9]

Organization	Nature of Application	Annual Savings
Monsanto Corp.	Optimize production operations in chemical plants to meet production targets with minimum cost	$15 million
United Airlines	Schedule shift work at reservation offices and airports to meet customer needs with minimum cost	$6 million
IBM	Integrate a national network of spare parts inventories to improve service support	$20 million + $250 million less inventory
Delta Airlines	Maximize the profit from assigning airplane types to over 2500 domestic flights	$100 million
China	Optimally select and schedule massive projects for meeting the country's future energy needs	$425 million
Taco Bell	Optimally schedule employees to provide desired customer service at a minimum cost	$13 million
Hewlett-Packard	Redesign the sizes and locations of buffers in a printer production line to meet production goals	$280 million more revenue

OR was used by the first time on World War II when the English army created a team only to find the best way to utilize arms, vehicles and radars. Even though that team was the first official OR team, the roots of OR are decades before in the early 1800s [9, 35].

When presented with a problem any OR team is proceeding in the very same way.

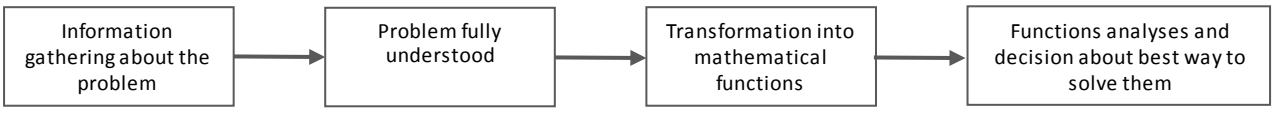

Figure 1.1: Diagram for OR to solve a problem [9]

First of all, according to figure 1.1 it's necessary to understand the problem. Several times, the OR team does not receive detailed information about the situation to be solved. That's why it is important to keep in touch with the management that presented the problem. The OR team must know very well the main objective of the problem (what do you want to find out), some constrains (what you cannot perform), time limits and so on. All this data can be obtained in a meeting with management or collected in the field (if it's a physical problem) [9].

After all the data about the problem is collected and well known, the OR team, needs to transform it in mathematical equations, as described on step 3 of figure 1.1. That way the problem can be defined in a mathematical expression, being able to be solved mathematically and simulated. So, according to Nocedal *et al* [35], we must define the following terms:

- Variable vector, also called parameters or unknowns
- Objective function that depends on $x, F(x)$.
- Vector function of x with constraints, $C(x)$.

In a more detailed way, when we have a problem and are defining it, x are our variables, where we can change values in order to get solutions. $C(x)$ are the values our variables cannot take and, finally, $F(x)$ is the function we want to maximize or minimize (it depends on the case).

As an example, that resumes the first 3 steps of figure 1.1, we can imagine that a company management asked the OR team to define what is the best hour and month to shut down the production, in order that a 3 hour intervention can be made, minimizing the impact it will produce on the company profit.

The first step the OR team executes is to understand the main goal: minimize profit reduction. Then, it is necessary to check all constraints: since company only works on a 9am to 6pm schedule the intervention cannot be made outside this hour range. Once all data about the problem is collected and well known it's time to model all this information mathematically so, in a first approach, the team identifies the variable vector x:

$$x = \begin{bmatrix} hour \\ month \end{bmatrix} \tag{1.1}$$

Once the variable vector it's identified, the team defines constraints. The only constraint from everything that it's known is the company working hours, since the company closes at 6pm and the intervention lasts 3 hours it cannot be started after 3pm. So, the following equations define all constraints:

$$hour + 3 \leq 18 \tag{1.2}$$

$$hour > 9 \tag{1.3}$$

$$month \geq 1 \tag{1.4}$$

$$month \leq 12 \tag{1.5}$$

At least but very important, it's necessary to define the objective function. That's the function that relates the variable vector to the desired objective. In this case, the OR team found out that the profit variation is established (for example) by:

$$F(x) = 5 * hour - 15 * month \tag{1.6}$$

Now, this example is fully defined mathematically and the OR team could proceed to the forth step of figure 1.1.

On the forth step, the OR team tries to develop some procedures to solve the mathematical equations. This simple case, a linear equation, could be solved by linear methods such like simplex [35].

However, in real life, problems are not as easy as the one described. Sometimes we have a lot of variables, constrains and non-linearities. In these cases, solving by linear methods would not be efficient once the time to solve it could be extremely high. So, some algorithms were created in order to solve complex problems without using that much time (metaheuristics – chapter 2).

1.2.2.Cockroaches' social life

Cockroaches are not solitaire and non-social animals [28]. That's a fact that almost nobody likes cockroaches. However, understanding cockroaches' potential, biologists have been studying these animals thorough.

Everything that we were able to find about cockroaches inspired a lot of new technologies, for example, robots with similar physiognomy that can be used to rescue survivors in disasters or scout harsh lands for military purposes and optimization [17]. In fact, this thesis mimics cockroaches' behaviours in order that mathematic functions can be optimized.

Cockroaches are very social animals that do not like to be alone and like to share information between themselves [4, 28]. Actually, if you spot one cockroach, it means one thousand are nearby hidden somewhere. In fact, such desire to live in groups, as a group (meaning, each individual acts as an agent of the group) and a genetic code very suitable to mutations allowed them to last in the planet Earth for more than 300 million years [40]. This resistance found on cockroaches also inspired human beings to create some myths about them such as: cockroaches could live without head or survive to nuclear attacks.

This thesis was inspired in cockroaches' social behaviour and cycle of life. The cockroaches' colonies have some goals to achieve. Cockroaches do not like light and they try to remain always hidden and lay eggs in dark shelters [12]. Those shelters are decided by all the cockroaches in the colony. Even if it is a big colony, cockroaches help each other to decide the best shelter and food source [12]. They do it due to their memory (because cockroaches have memory and can remember information from the past) and to their ability to socialize when meeting friends [4, 37]. So, in a resumed way, all cockroaches in the colony are trying to find a good shelter and/or food source and, when meeting friends, they are sharing the information each one got. That way, in the end, all cockroaches know what is the best spot ever found and all of them converge to the same location.

A fascinating fact about cockroaches, and proof of their very social way of living, can be observed when, for example, colony cannot fit in a shelter. In those situations, they are splitting themselves into equal groups to fit in smaller shelters. If one finds a shelter for 1000 cockroaches and there are 1001 in the colony, the group splits himself into 2 groups (500 and 501) instead of leaving a member alone [1]. Other fascinating fact proved by Lihoreau *et al* is that cockroaches get sick when they are alone. They adopt lethargic behaviours and are less likely to move or leave shelters. If the loneliness prevails, they will eventually die [28].

As referred previously in this chapter, cockroaches have survived for millions of years not only due to their social behaviour but also for other reasons. The fact that cockroaches' genetic code is very mutable, allied to the fact that all males and females have the same probability to reproduce (no alphas or hierarchy) and they are not doing it with family members (no incest) [11, 27], facilitates them to adapt to different environments and makes them almost indestructible because new born roaches are always

getting different characteristics from different families. That way, the probability of a cockroach with some characteristic that allows her to survive in new environments increases.

1.3 Thesis' contributions

As it was possible to understand on previous chapter, optimization is a huge world. It is very important to define the problem in the correct way and to understand how to solve it.

Nowadays, there are a lot of options to solve continuous optimization problems (where all variables can take any real value between defined ranges) and discrete problems (where variables can only take a specific set of values) [35].

One of those options, is described in the chapter Roach Infestation Optimization (RIO), once it was the first work ever done in this field. However, this work still presents:

- A novel algorithm, more tied to cockroaches' social life than RIO to solve continuous problems based in a nature behaviour. With more complex problems appearing every single day, it is mandatory that the optimization world adapts itself to solve those problems.
- Ability to solve the most complex continuous functions with success.
- Some new combinations of swarm techniques to achieve better results

1.4 Thesis' organization

This work is divided in 5 major chapters. On the first one a brief introduction is done. That way, it is possible to understand motivations and goals. In this chapter are also briefly explained the basis of optimization and the cockroaches social life.

On chapter 2, we dive deeper in optimization algorithms. Those that are the base for Genetic Roach Infestation Algorithm, presented in this work, and some other evolutions that already were developed.

On chapter 3 are explained this new algorithm mechanisms. From a more detailed parallelism of roaches' life with the algorithm, to a detailed explanation of all the algorithm's mechanisms.

On chapter 4, benchmark functions for testing purposes are presented along with results comparing different algorithms' performance to the new one.

Finally On chapter 5 final conclusions are described and further work is suggested.

2 Non Linear Problems: Metaheuristics

Metaheuristics are methods (algorithms) to solve problems. While a heuristic is a method to solve only one specific problem (problem dependent), a metaheuristic can solve a lot of different problems (problem independent) [9]. Resuming, metaheuristics are methods to solve optimization problems that do not depend on the problem itself, that is to say, metaheuristics explore all the solution space randomly orientated, without taking advantage on any particularity of the problem itself, and based on the solutions found, it decides where to move next until it converges to one specific solution that can be the best one or not. That's why, before using a metaheuristic in a problem, we always must decide which one is the best for our problem, the number of iterations and the number of times we run the problem [9].

In this chapter are presented all the algorithms that were analysed as basis for this thesis. First it is presented Particle Swarm Optimization, the first work ever done in the flock simulation area. Then it is presented Roach Infestation Optimization, an algorithm that was the first step for this work to be possible. In the end all other related works are presented in order that we can understand the kind of strategies researchers are using to improve solutions in complex non-linear problems.

2.1 Particle Swarm Optimization

Groups of birds, insects and fishes are able to synchronize velocities, turning manoeuvres and to perform several activities (landing for example). Such behaviour has intrigued several authors that have been trying to understand the way that kind of social behaviour works. Wilson, in 1975, said that animals' social behaviour was established by some rules [41] and, since that conclusion, researchers have been trying to figure out those rules.

Craig [3] was the first author presenting some work in flocks simulation. He presented this phenomena (flocks of birds, herds and schools of fishes) as three simple rules: collision avoidance - since it is impossible that two particles (bird, fish, insect among others) exist in the same position, at the same time, every single one avoids to crash with other flock mates; velocity matching - each flock member attempts to match his own velocity with nearby mates; and flock centring - each particle tries to keep close to nearby mates.

Kennedy and Eberhart, in 1995 [22], pioneered an entire new approach to optimization problems, when inspired in real world swarm intelligence, the authors developed a new optimization algorithm based on some behaviours of birds flocking, PSO.

They defined a world whose dimensions were the variables of the chosen objective function, based on several studies mentioned above. So, if our objective function F, has D variables, there's a swarm (in a D-dimensional world) where each i^{th} particle exists on a certain position $X_i = [x_{i1} \quad x_{i2} \quad \cdots \quad x_{iD}]^T$ and

moves by a certain velocity $V_i = [v_{i1} \quad v_{i2} \quad \cdots \quad v_{iD}]^T$ (limited to V_{max}), during t iterations, trying to find the best possible position. The best location is defined proportionally to the fitness function. It means that if we want to minimize F function, the best position will be defined by the smaller value that a particle's position achieves.

In this optimization approach, particle's velocity on instant k,V_k, depends on the previous velocity V_{k-1}, on the particle's best position ever achieved, P_{best} (individual component), and on the position of the best positioned particle in all swarm G_{best} (social component). That way, it is possible to calculate the next position using the following equations:

$$v_i^{k+1} = v_i^k + C_1 r_1 (Pbest_i^k - x_i^k) + C_2 r_2 \left(Gbest_i^k - x_i^k\right) \tag{2.1}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{2.2}$$

On these equations, r_1 and r_2 are D-dimensional vectors with random values between 0 and 1. These random vectors help to keep diversity on swarm. C_1 and C_2 are cognitive and social factors, respectively. These are the parameters that define the influence of cognitive and social components on particle's velocity. Usually they are set, as suggested by Kennedy [16], in a way that C_1 plus C_2 equals 4. However, in 2001, Carlisle and Dozier [2], developed their research on cognitive and social parameters influence and found out that, for some functions, a sum lower than 4 was the best choice (like Schaffer F6). They also found out that, the values usually used on common sense of 2.05 to both C_1 and C_2 were not the best ones. They came to the conclusion that the best choice should be a ratio of 2.8 of cognitive parameter to 1.3 of social one.

Later, both Eberhart and Kennedy tried to improve PSO performance and capacity of get out from local optima by changing the global neighbourhood G_{best}, to a local neighbourhood L_{best} [6]. They established that, instead of using all swarm to define the best solution, each particle would only use its N neighbours, i.e. if we are defining two neighbours, particles' i^{th} L_{best} would be chosen between particle $i+1^{th}$ and $i-1^{th}$. According to Kennedy [21], G_{best} version provides a faster convergence but it is trapped in local minimal more easily than the neighbourhood version. Novel topologies of neighbourhood were introduced by Kennedy et al [23]. Most common approaches are the classical ones - Global best and Local best - represented on figure 2.1(a) and figure2.1 (b) and the wheel one, represented on figure 2.1 (c), where all particles share its information to a focal individual. All these topologies, and some others, were studied by Kennedy and Mendes [24]. They concluded that some topologies, in general, have better results. However it was impossible to name the ones that result in the best performance to a range of functions, once the topology depends on the problem we are solving.

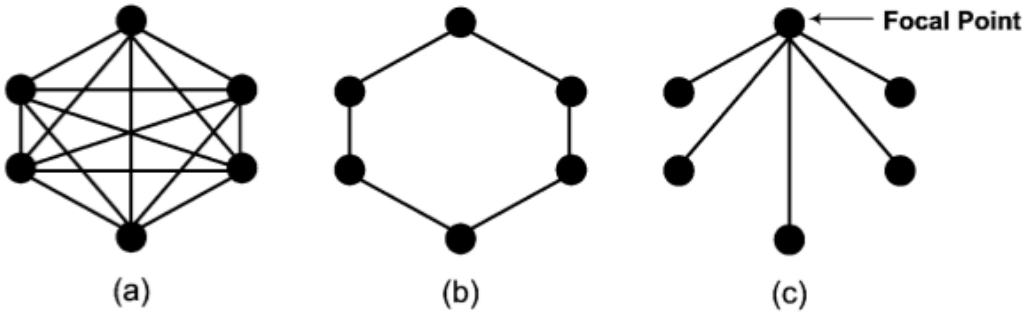

Figure 2.1: Neighbourhood topologies. Global Best (a). Ring topology–Local Best (b). Wheel topology (c).

Another advance in PSO was achieved by Eberhart and Shi, in 1998, when they presented inertia weight, w, into velocity equation [43].

$$v_i^{k+1} = wv_i^k + C_1 r_1 (Pbest_i^k - x_i^k) + C_2 r_2 \left(Gbest_i^k - x_i^k\right) \quad (2.3)$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \quad (2.4)$$

It receives a value between 0 and 1 and defines the importance of particle's previous velocity. If it is a lower value there is a preference in local search once the previous velocity it is not valorised and the final velocity will be smaller. If it is a higher value, there is a preference in global search, because the previous value is highly taken into account.

As inertial weight is a major parameter to proper convergence and it is difficult to choose a suitable fixed inertia weight, several authors have been developing new techniques to introduce a variable inertial weight that adapts while algorithm is running. So, understanding that in the beginning it is important to promote a global search in all the area and in the end a local search, Shi and Eberhart [38], proposed a linear variable inertia weight that decreases from a maximum value (they defined 1.2 but it can be adjusted) to a minimum value (defined as 0) as represented on equation 2.5. There are three types of inertia weight: constant and random; time varying (like the one represented on equation 2.5); and adaptive ones, where inertia weight is adjusted according to some feedback parameters. Nickabadi *et al* [32], concluded their research in analysing all different types of inertia weights before introducing a new adaptive one.

$$w(k) = w_{max} - \left(\frac{w_{max} - w_{min}}{k_{max}}\right) k \quad (2.5)$$

In order to get PSO fully operational, it is necessary to define all the parameters in the best possible way [2]. These parameters are maximum velocity V_{max}, cognitive parameter,C_1, social parameter, C_2, and inertia weight, w. Literature provides some clues on finding the proper values, however, that is a decision that has to be made while implementing, according to the nature of the problem.

Due to its easy implementation, few parameters to define as well as the ability to solve dynamic non-linear continuous problems PSO algorithm (represented in table 2.1) have been used in several applications such like neural networks training, structural optimization among others, for example [6, 8].

Table 2.1: Particle Swarm Optimization

Algorithm 1: Particle Swarm Optimization

Define: Fitness function F

Parameters

Initialize: Population N in random positions

Each particle's best position, P_{best}

Best Global Position, G_{best}

Compute: **for** k=1 **to** Max_iteration

for i=1 **to** N

$v_i^{k+1} = wv_i^k + C_1 r_1 (Pbest_i^k - x_i^k) + C_2 r_2 (Gbest_i^k - x_i^k)$

$x_i^{k+1} = x_i^k + v_i^{k+1}$

if $Z(x_i^{k+1}) < Z(Pbest_i^k)$

$Pbest_i^{k+1} = x_i^{k+1}$

Update $Gbest_i^k$ to $Gbest_i^k$

In order to improve its performance, a lot of PSO variations were developed and studied. The aim is always to achieve global optima in the less possible iterations.

As an example, it's possible to describe some of them: Peram et al, in 2003 [36], introduced a new PSO variant (fitness-distance-ratio PSO) where each particle is not only influenced by best one but also by it's well positioned neighbours.

Van den Bergh and Engelbrecht used multi-swarms that optimize different components of solution vector and cooperate between them [39]. In 2004, Mendes et al [31], introduced the fully informed PSO, where each particle is influenced by all neighbours and not only by the one with the best position.

Comprehensive learning PSO was presented by Liang et al [26] and suggests that better results can be obtained if particle is able to consider all friends best solution and select one of them. Krohling and Coelho [25] made a PSO version, called Evolutionary PSO using Gaussian Distribution for Solving Constrained Problems, to solve min-max problems. In this new PSO, instead of using a uniform probability distribution to generate accelerations and one swarm, it's used a Gaussian probability distribution and two swarms.

In 2007, Niu et al [34] published a new multi-swarm algorithm based on a master-slave model. In this algorithm, there are slave swarms that are looking for problem solution in an independent way and one master swarm that defines its course according to its own experience and to slave swarms experience. That way it was possible to keep solution diversification. Understanding that most of the times, the best fitness was achieved in the centre of the swarm Liu et al [29] presented a PSO where, in each iteration, a particle is transported directly to the centre.

Li and Xiao [19], in 2008, developed a multi-swarm and multi-best PSO. In this variation, particle velocity is updated using multi-personal best and multi-global best instead of one and a multi-swarm that, in the end, unites in a single one. Jie et al [18], created a knowledge billboard where information regarding search dynamics and environment is recorded and shared between all swarms. Also in 2008, Del Valle et al presented a very complete study in a large number of PSO variants and its applications [5].

At last , but still important to mention, it's Zhang and Ding's work [44], in 2011, that reports to a PSO, called Multi-Swarm Self-Adaptive and Cooperative PSO, that uses four different swarms that have a cooperative and competitive behaviour between them to avoid local optima.

2.2 Roach Infestation Optimization

Roach Infestation Optimization (RIO) [14] stands for the first algorithm based on cockroaches' social life. Havens et al, understanding that cockroaches enjoy family company and socialize between them, changed the social component in PSO's velocity update equation.

Instead of learning only from best-positioned particle, in RIO, cockroaches learn from neighbours if there's socialization. Therefore, when a cockroach can find friends in its detection radius d_r (equation 2.6), there's the probability (table 2.3) that those roaches socialize.

$$d_r = \frac{\sum_{i=1}^{N-1}\left(\sum_{j=i}^{N}\left\|x_i - x_j\right\|\right)}{\sum_{i=1}^{N-1}\left(\sum_{j=i}^{N} 1\right)} \tag{2.6}$$

That socialization is carried out by an exchange of knowledge according to equation 2.7:

$$bk_1 = bk_2 = arg\,\min\left\{F\left(bk_j\right)\right\}, j = \{1,2\} \tag{2.7}$$

Where bk is the best known position (for roach 1 and roach 2) and F is the objective function. So, every single time a roach meets another one, and they socialize, both roaches will remember the same best position ever achieved by someone.

Finally, velocity equation on RIO algorithm will be as equation 2.8 displays:

$$v_i^{k+1} = wv_i^k + C_1 r_1 (bv_i^k - x_i^k) + C_2 r_2 \left(bk_i^k - x_i^k\right) \quad (2.8)$$

Where w is inertia weight, C_1 and C_2 are cognitive and social components, respectively, r_1 and r_2 are D-dimensional vectors with random values between 0 and 1 and b_v is the best visited position (same as past best (P_{best}) position from PSO). In table 2.2, RIO is presented.

Table 2.2: Roach Infestation Optimization

Algorithm 2: Roach Infestation Optimization

Define: Fitness function Z

Parameters

A=probability to socialize (table 2.3)

Initialize: Population *N* in random positions

Each roach's best visited position, b_v

Best Known Position, b_k

Compute: **for** *k*=1 **to** Max_iteration

$d_r = \frac{\sum_{i=1}^{N-1}\left(\sum_{j=i}^{N}\|x_i - x_j\|\right)}{\sum_{i=1}^{N-1}\left(\sum_{j=i}^{N} 1\right)}$

for *i*=1 **to** *N*

for *j*=1 **to** *N*

if $j \neq i$ **&&** $rand \leq A$

$bk_i = bk_j = arg\ \min\{Z(bk_h)\}, h = \{i, j\}$

$v_i^{k+1} = wv_i^k + C_1 r_1 (bv_i^k - x_i^k) + C_2 r_2 \left(bk_i^k - x_i^k\right)$

$x_i^{k+1} = x_i^k + v_i^{k+1}$

if $Z(x_i^{k+1}) < Z(bv_i^k)$

$bv_i^{k+1} = x_i^{k+1}$

Table 2.3: Probability of roaches' socialization according to the number of neighbours [14]

Number of Neighbours	Probability of socialize
1	0.49
2	0.63
≥3	0.65

2.3 Other Algorithms

In early 1970's, John H. Holland [15], created an algorithm based on nature evolution rules, Genetic Algorithm. Basically it is an algorithm that follows Darwin's principle that says "the survival of the fittest". This algorithm creates a population of N individuals, where each individual genetic code is a possible solution. Then, during each iteration, some individuals die (according to different methods of selection) and the other ones (usually the ones with the better fitness score) reproduce between then (also selected by different methods), generating new individuals that might be more adapted then the parents [13]. This reproduction is carried on by mixing some parts of the genetic code of the father and mother. After the recombination, some mutations can be introduced in order to keep diversity [33].

Just for the record, other examples of nature based algorithms are: ant colony optimization where a group of ants decides the best path to the food using their pheromones [30], cuckoo search that mimics the cuckoo's behaviour of laying eggs in other birds nest [42] and artificial bee colony optimization where a colony of bees tries to find a group of food sources with high nectar and in the end, the one with the highest nectar [20]. In 2013, Fister Jr. *et al*, created a brief review on nature based algorithms and each one with a classification [7].

Also, to solve continuous problems, there are algorithms not connected to nature. This work is not focusing on those kind of algorithms, but GLODS (Global and Local Direct Search) was used as a reference to compare results, due to its very complete report and results [4].

3 Genetic Roach Infestation Optimization

This dissertation proposes a novel continuous algorithm: Genetic Roach Infestation Optimization algorithm. A parallelism between roaches' behaviour and techniques used in the algorithm will be presented, and all the equations will be displayed.

3.1 Overall Description

Despite the evolution from PSO to RIO, we still cannot avoid efficiently premature convergence and local optima. With RIO, we still cannot ensure that we find the global optima in an efficient way. Actually, with more complex functions, RIO can get stuck in local optima points (such like peaks and valleys). Also, RIO showed not to be so good for functions with high dimensionality (see table 4.3).

Understanding such drawbacks, in this work, a novel optimization algorithm developed to continuous problems is proposed, Genetic Roach Infestation Optimization (GRIO).

GRIO is more tied to cockroaches' behaviour than RIO and it is highly founded on some characteristics from the cockroaches' life. In figure 3.1 we can find the relation from cockroaches' behaviours to algorithm construction.

As referred in chapter 1.2.2, cockroaches work in team in order to find the darkest spot where all the colony can fit. In the presented approach, darkness level at some location is proportional to fitness function $Z(x)$ at that location. If we want to minimize function $Z(x)$ at location $x \in \mathbb{R}^D$, darkness increases as much as $Z(x)$ decreases. Each cockroach, in GRIO algorithm, is represented by a group of coordinates (for each dimension, which mimics his position in the space) $x \in \mathbb{R}^D$.

Because cockroaches socialize when meeting friends, when in the algorithm two cockroaches are close to each other they might stop and socialize. That socialization is imitated by sharing information between them. Then, that information is kept in their memory and they continue their path. As in the algorithm, in real life, also cockroaches have memory. Each virtual cockroach keeps in memory the best position ever achieved and the best position a friend, or the cockroach itself, already achieved. That second information is the one shared between cockroaches while socializing.

Using the fact that cockroaches split themselves in two different equal numbered groups when there are no dark spots big enough for all the group; in order to decrease the probability of being trapped in local optima, in the beginning GRIO believes that there are no dark spots big enough for all the swarm. So, a population consists in two equal sub-swarms seeking a dark spot. Those swarms are two different families and behave independently of each other. Using this independent multi-swarm approach we can increase the probability of escape from local optima and increase the solution variability.

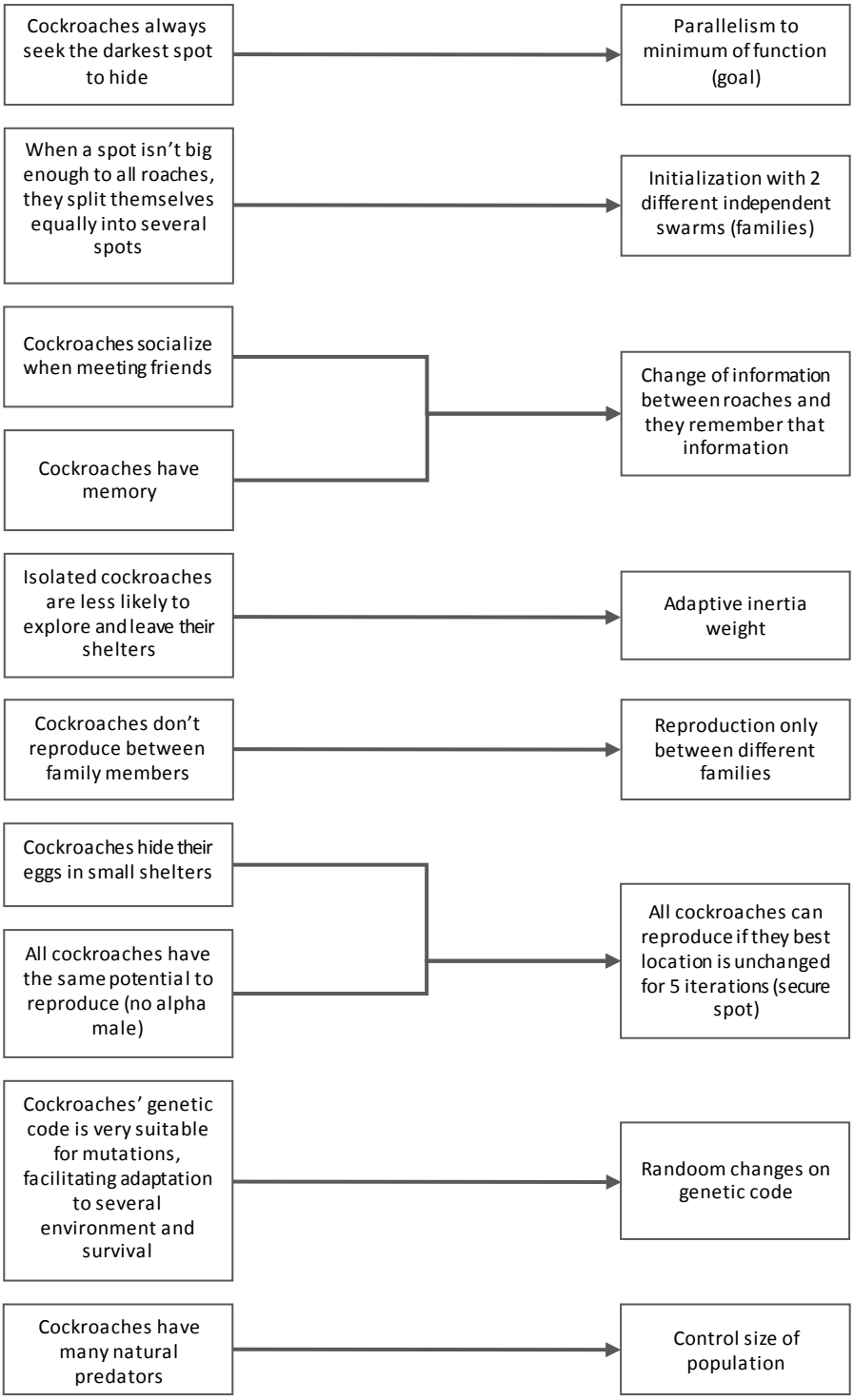

Figure 3.1: Parallelism between roaches' behaviours and GRIO

In real life, all cockroaches are able to reproduce with equal probabilities. However, they do not reproduce between family members and, that fact, is good for GRIO once it avoids individuals being generated with similar genetic codes. So, GRIO mimics these facts about cockroaches allowing cockroaches from different families to reproduce if they find a good place for that. New borns rise with

a combination of parent's knowledge. In order to promote a global search, it is possible that some random mutations occur such like in nature, where cockroaches were able to prevail due to their high susceptibility to mutations. If we allow the population to keep growing, then the algorithm execution time would be excessive so, such like in nature, there are predators that control swarms size.

A tool to control global and local search is Cockroaches' speed. The velocity is affected by the number of neighbours in the moment and the distance to the best solution.

An isolated cockroach's velocity is smaller than a surrounded one. Another velocity controller is the distance to the best location ever achieved. The closer a cockroach is to the best solution the smaller will be its velocity to promote a more local search in that area.

3.2 Algorithm Description

In this section, a detailed explanation on Genetic Roach Infestation Optimization algorithm, GRIO, is presented. As already introduced, in the previous chapter, GRIO is an algorithm with a very strong parallelism to cockroaches' life. In the algorithm, a roach is defined by her position $x \in \mathbb{R}^D$ where D is the dimension. Roaches' positions are the number of points that inserted in the function to optimize give us a solution $Z(x)$.

This algorithm, while being developed, always tries to promote diversity in all the elements of the population while promoting a global search in the beginning and a local search in the end. In fact, as we can learn from other authors in chapter 1, this is the objective for these kind of algorithms since it is the best way to find the global optima without suffering premature convergence.

The algorithm initializes with two different swarms (families of cockroaches) with the same number of elements. Those swarms behave in an independent, but similar way and only change information while reproducing (ability only possible between members of different swarms). In other words, both swarms follow the same equations and behaviours and interact only while reproducing.

Resuming, in this algorithm one roach moves, then it is checked if that roach is able to reproduce or not. If yes, it is decided if the sons are mutated or not. This is repeated for all roaches in all swarms. GRIO's flowchart can be found on figure 3.2.

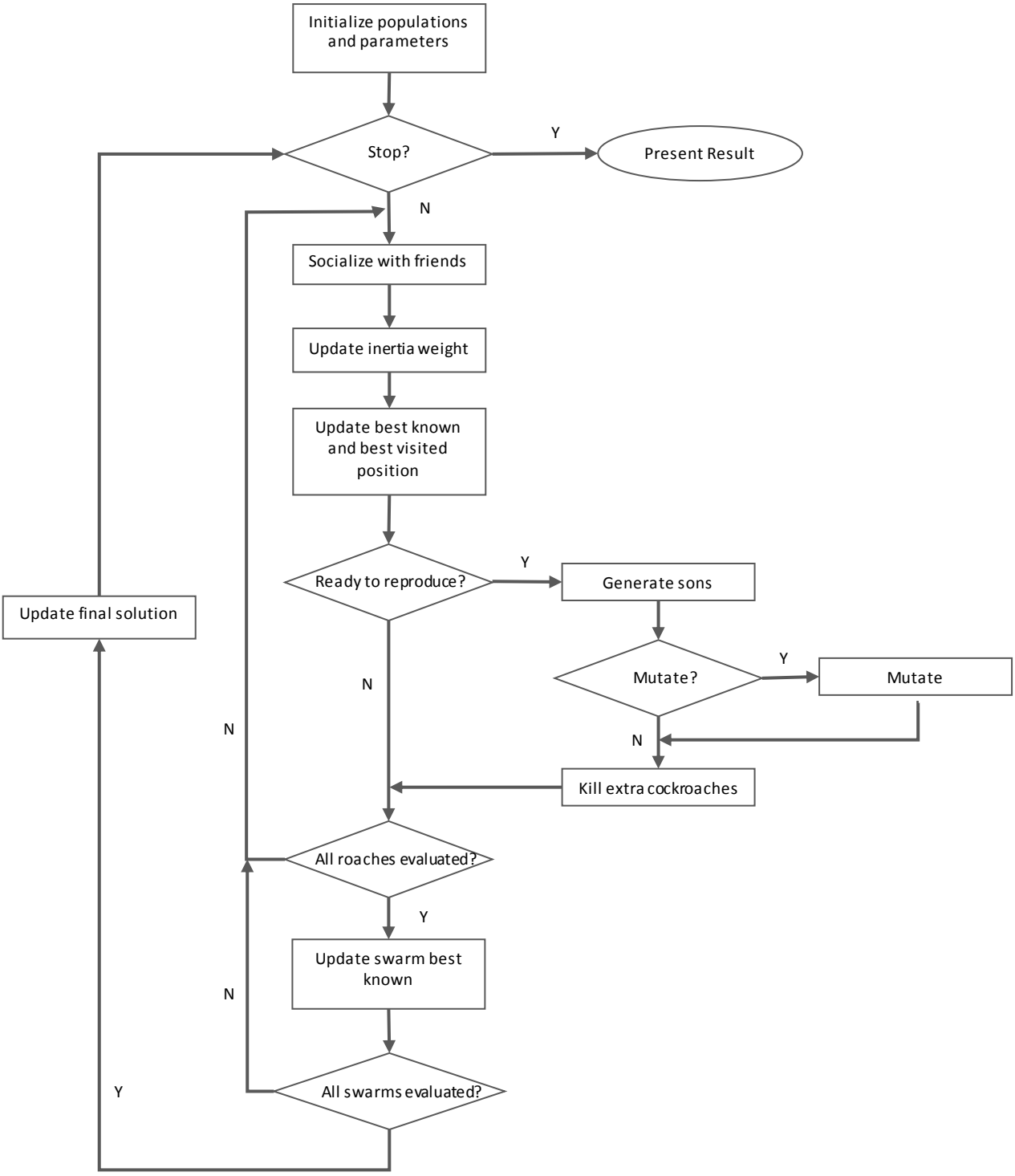

Figure 3.2: GRIO's Flowchart

3.2.1. Search Behaviour

Regarding movement, GRIO is highly based on RIO and PSO once its search behaviour is equal to the RIO search behavior. Before moving, it is necessary to verify if a cockroach has friends (in the same swarm) inside its detection radius, defined by equation 2.6 such like RIO. If there are friends in that detection radius, there is a probability (table 2.3) that these cockroaches will stop and socialize. Socialization is performed in the algorithm by comparison between roaches best-known position. So, when two roaches meet, both roaches best-known position is updated according to equation 2.7. Then, once best-known position it's updated, roach's position is calculated using equations 2.8 and 2.4, respectively. Search behaviour for GRIO can be consulted in table 2.3. To keep the information on the 2 different swarms on the same level, instead of a complete iteration (moving all roaches in the swarm) for one swarm and then for the other, it was adopted that after one roach moves in a family then, a roach from another family should move. In the end, an iteration is only completed after all roaches from both families move, in an alternated way.

3.2.2. Adaptive Inertia Weight

An important parameter in all PSO-based algorithms is inertia weight. This parameter controls velocity. Velocity, in this case, is the size of the "step" a cockroach can give in every single dimension. As one can easily see by equation 2.2, x_i^{k+1} increases as much as bigger the velocity value is. The reason why the value of velocity is so important relates to the influence of this value in a local or global search.

Inertia weight influences a lot the velocity value (as you can see on equation 2.8). In fact, it has the power to completely transform it. If it presents a lower value there is a preference in local search once the previous velocity is not valorised and the final velocity will be smaller. If it is a higher value, there is a preference in global search, because the previous value it's highly taken into account.

Because of that importance in all the algorithm, inertia weight must be chosen carefully. Algorithm convergence can be successful by applying the right value or, ruined by applying the wrong one. As inertial weight is a major parameter to proper convergence and it is difficult to choose a suitable fixed inertia weight, several authors have been developing new techniques to introduce a variable inertial weight that adapts while algorithm is running.

In this work, an adaptive linear decreasing inertia weight, adjusted to each particle/roach, is proposed. That way one can achieve better results once linear weight is adapting itself according to each situation. On figure 3.3, linear weight adaptive system is represented by a flowchart.

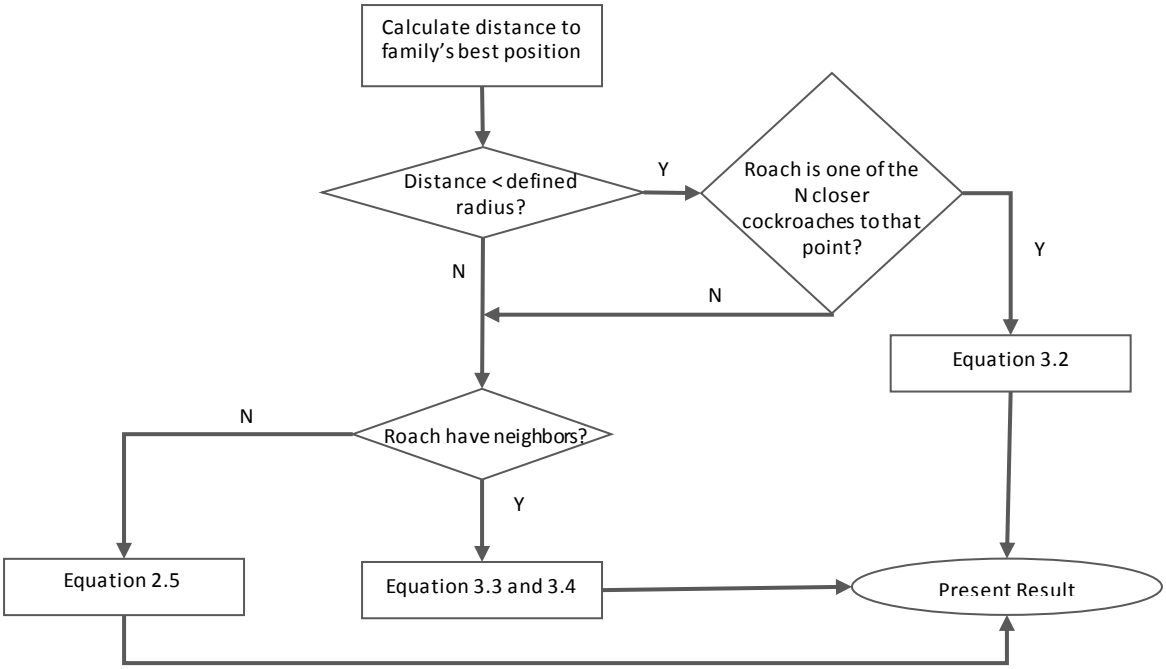

Figure 3.3: Adaptive behaviour flowchart

It is important to promote a local search in final iterations and near optimum point. So, when a cockroach is about to be moved, its distance to the family's best position ever achieved is checked (equation 3.1).

$$d = \sqrt{\sum_{j=1}^{D} (bp(j) - x(j)^2)} \tag{3.1}$$

In this equation, *D* represents problem dimension, bp is the best position ever achieved and x is the roach position.

If that distance is smaller than the maximum distance, roach is inside the $\frac{1}{4}N$ closest roaches to family's best position (with N standing for number of family members) and iterations are bigger than $\frac{3}{4}i_{max}$ (i_{max} is the highest iteration), then inertia weight is updated using equation 3.2, where N is the number of roaches in the swarm.

$$w = -\frac{0.45}{round\left(\frac{N}{4}\right)}d^2 + 0.45 \tag{3.2}$$

When roaches do not meet criteria to use equation 3.2, inertia weight follows a linear decreasing method. So, as long as iterations are increasing, inertia weight is decreasing (equation 2.5).

As accompanied cockroaches are more active than those isolated, a parameter was defined (hyper-activity, equation 3.3) that is summed to regular linear decreasing inertia weight multiplied by the number of neighbours (equation 3.4).

$$ha = \left(0.105 - 0.1\frac{i}{i_{max}}\right) Nn \tag{3.3}$$

$$w = \left(w_{max} - i\frac{w_{max} - w_{min}}{i_{max}}\right) + ha \tag{3.4}$$

As we use those equations, where Nn is the number of neighbours in a certain radius, instead of common inertia weights, we can adapt inertia weight and obtain more suitable results. In figure 3.4 are represented inertia weight's oscillations as iterations increase.

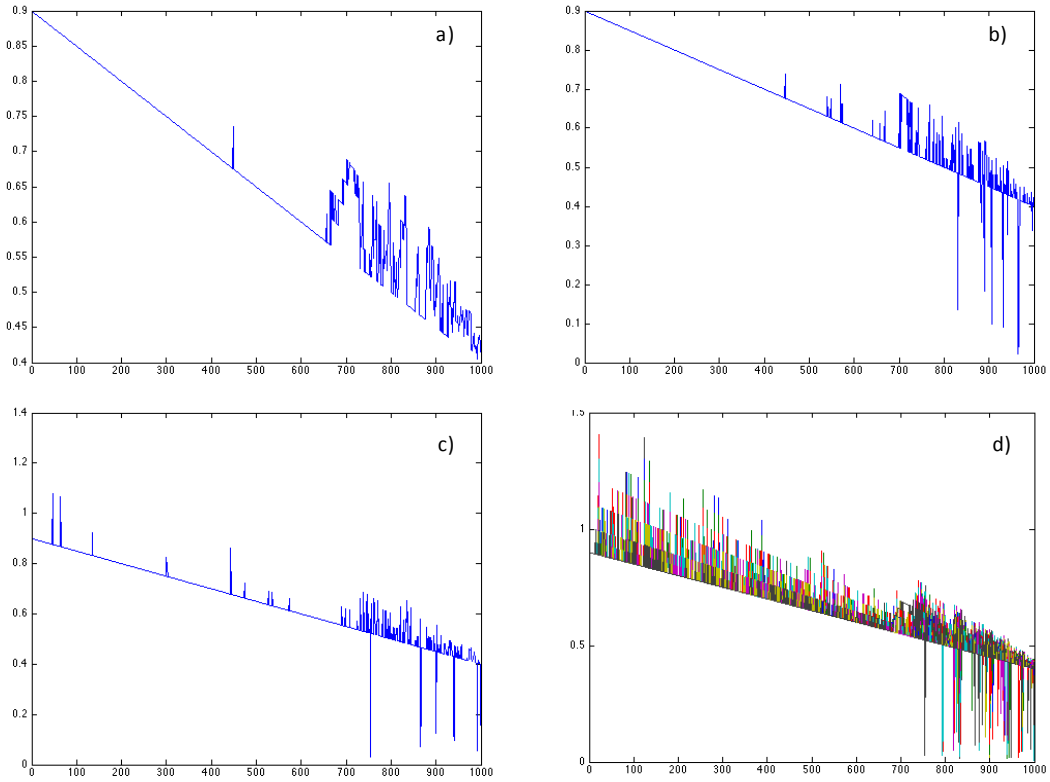

Figure 3.4: Inertia weight oscillations. a) Oscillations for roach 1. b) Oscillations for roach 10. c) Oscillations for roach 20. d) Oscillations for all roaches.

The main goal from this adaptive inertia weight is to promote the right search in the right time. It means, to have a global search in the beginning, where roaches are placed in random places and more likely to be far away from the optimal point. And a more local search in the end where it is more likely for our

roaches to be close to the optimal point. In fact, with this kind of adaptive inertia weight, we managed to decrease the inertia weight along with iterations, despite the possibility to have a local search when there is a promisor point to be optimal or near it.

Resuming, with equation 2.5, we ensure that inertia weight is decreasing linearly along with iterations, promoting a more global search in the beginning and a local search in the end. With equations 3.3 and 3.4 we can escape from local optima - when roaches are trapped in the beginning of the iterations, or in the end (if they are not one of the closest to presumed optimal point), they can have a bigger inertia weight and escape. That happens because when roaches are with neighbours (and consequently more "happy") they have more energy and inertia weight increases allowing them to move a bigger distance in one step. With hyper-activity phenomenon we can also find better solutions in the end once roaches can be putted away and come back to the best local. That way, maybe they find a better point.

As we can learn from the figure 3.4 analyses, and could expect from the way inertia weight value is established, this adaptive inertia weight has a guideline equal to linear decreasing method (coming from equation 2.5). However, sometimes it is bigger (when a cockroach has neighbours) or smaller (when a cockroach is near the best position ever achieved).

From figure 3, we can also understand that first roaches to move (roach 1 for example) are less likely to suffer variations due to hyper-activity (in first iterations) once the first particles to move are more alone and not likely to have neighbours. As opposite, for the same reason, if we study roach 20 graphic it is possible to note that that roach has more variations due to hyper activity in early iterations.

Also, we only start to see the influence of equation 3.2 in last iterations due to the condition described on equation 3.1, which only allows an inertia weight under the guideline if they are near the optimum point, in final iterations.

Using this kind of inertia weight, and analysing figure 3.4 d), we can easily see that during first iterations, roaches are wandering around all the space, increasing the probability of finding the goal point. While, in the end, despite some of them still go around (allowing an escape from a possible local optima point) roaches are promoting a local search, refining that way the already achieved best solution.

3.2.3. Roaches' Reproduction

As already referred, GRIO has a cooperative component between the two different and independent swarms. While reproducing, there is a share of knowledge from one family to another. That knowledge is shared because, if a roach is ready for reproduce, after having the sons from the chosen mate on the other swarm, those sons will be part of the family/swarm of the roach in evaluation.

Reproduction is performed like a regular genetic algorithm, chapter 2.3. First of all, the roach must be eligible to reproduce and that is possible when two conditions are satisfied:

- Roach is on fertile window (emulated with random number).
- Roach finds a secure spot to leave eggs (emulated when best known position is not changed during five iterations).

If these conditions are fulfilled at the same time, roach is able to reproduce and chooses the best-positioned roach in the opposite family. Once two reproducing roaches are selected, it's necessary to choose the "genetic code" (vector with coordinates). This vector is generated first by creating 10 times more than dimension (limited to 100 to avoid excessive running time), different random combinations of best-visited position vector (bv) and best-known position vector (bk) for each roach mating (figure 3.5). That way, one achieves a vector with information from two different memories of each roach.

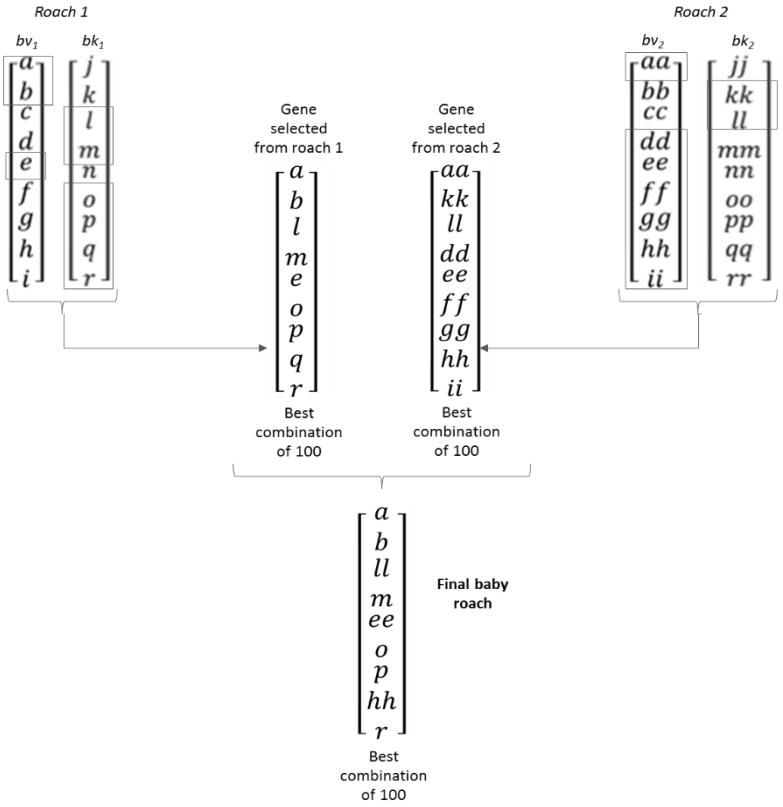

Figure 3.5: Baby roach's birth example

For each roach involved in the reproduction process, the best combination vector obtained is chosen and then a new crossover process is done and a child is generated (the child vector will represent the baby roach's actual position, best known position and best visited position). This procedure is repeated until all children (number decided by user) are born. Once it is necessary to keep swarm size to avoid extra running time, after born, N random extra roaches are killed. In figure 3.6 a flowchart of this process is represented. It was decided not to kill the N worst positioned roaches once that would affect algorithm global search and increase the risk of being trapped in local optima points.

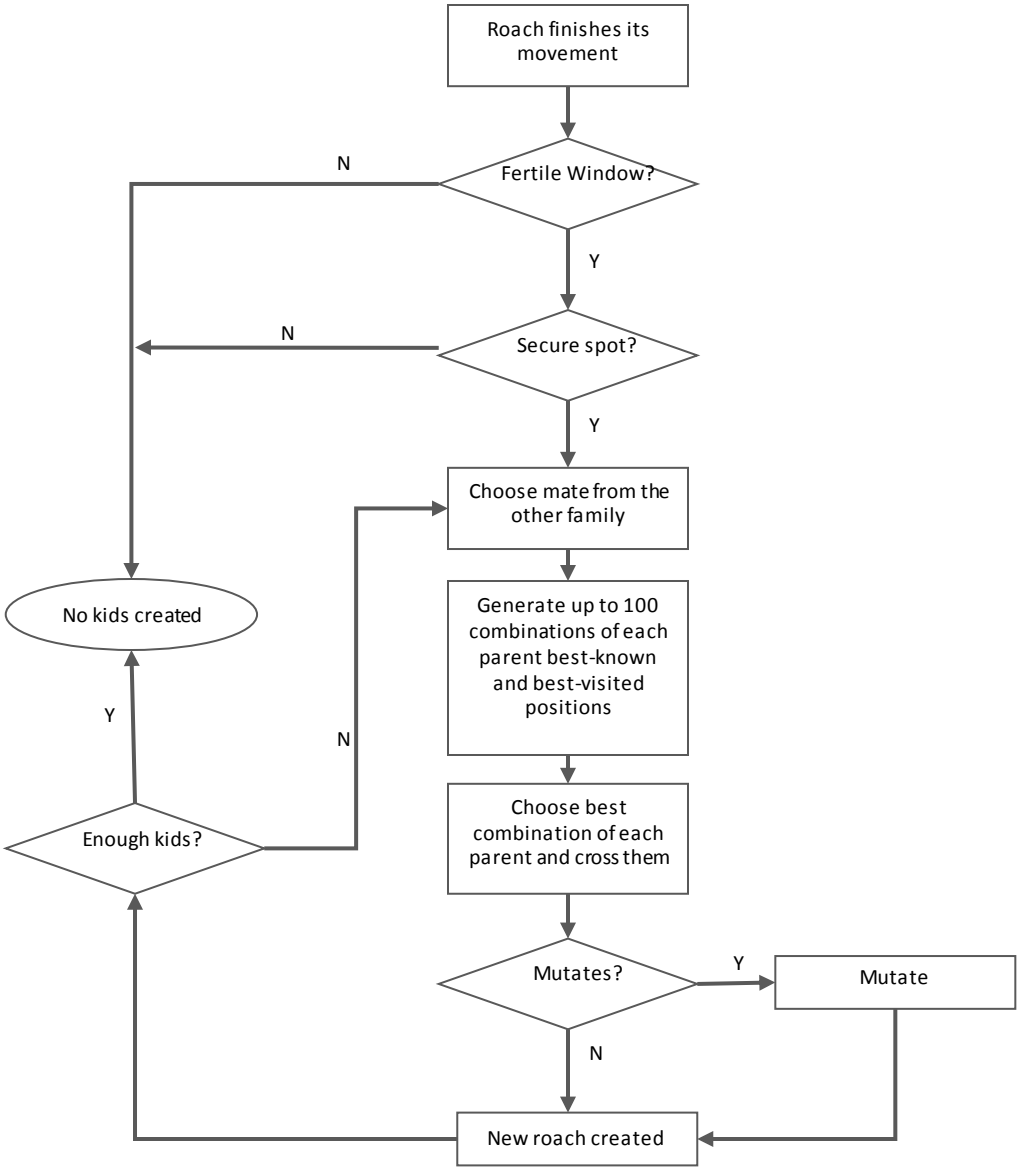

Figure 3.6: Reproduction behaviour flowchart

3.2.4. Roaches' Mutation

A major issue in PSO based algorithms is premature convergence. In order to avoid it, is necessary to keep diversity in the search space. Otherwise solutions will converge into a single spot (a good one or not) and there will be a huge unsearched and unknown space.

Understanding that, there were introduced two kinds of mutations: replacement mutation and frameshift mutation. Every time a baby roach rises, there's the opportunity of being a healthy one, or a mutant one (however only one kind of mutation will occur). That opportunity is described by a random number that must be placed between 0 and probability of replacement (Prob_mut_rep) for a replacement to happen or between probability of replacement and probability of frameshift (Prob_mut_fram) for a frameshift to happen.

- Replacement mutations: Some random values from child's vector are replaced by new values (figure 3.7, a)).
- Frameshift mutations: A random sequence of values is allocated in a new place (figure 3.7, b)).

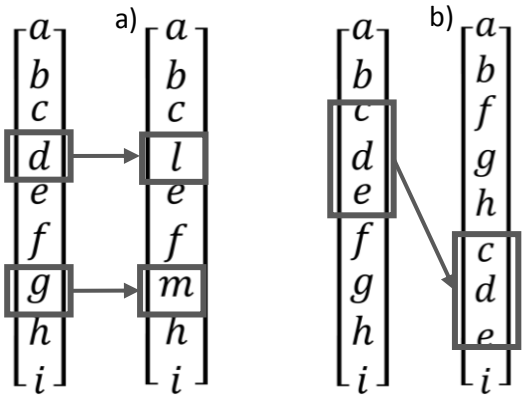

Figure 3.7: Mutations. a) Replacement mutation. b) Frameshift mutation.

All those roaches' behaviour, described before in chapter 3.2, define GRIO algorithm designed in table 3.1.

Table 3.1: Genetic Roach Infestation Optimization

Algorithm 3: Genetic Roach Infestation Optimization

Define: Fitness function Z

Parameters

A=probability to socialize (table 1)

Initialize: Population *N* in random positions

Each roach's best visited position, bv

Best Known Position, bk

Compute: **for** *k*=1 **to** Max_iteration

$$d_r = \frac{\sum_{i=1}^{N-1}\left(\sum_{j=i}^{N}\|x_i - x_j\|\right)}{\sum_{i=1}^{N-1}\left(\sum_{j=i}^{N} 1\right)}$$

for *i*=1 **to** *N*

for v=1 **to** *number_swarms*

for *j*=1 **to** *N*

if $j \neq i$ **&&** $rand \leq A$

$bk_i = bk_j = arg\min\{Z(bk_h)\}, h = \{i, j\}$

if *i* ∈ $\left(\frac{1}{4}N \text{ closest cockroaches}\right)$ **&&** $k \geq \frac{3}{4}Max\_iteration$

$w = -\frac{0.45}{round\left(\frac{N}{4}\right)}distance^2 + 0.45$

else

$w = \left(w_{max} - i\frac{w_{max}-w_{min}}{i_{max}}\right) + \left(0.105 - 0.1\frac{i}{i_{max}}\right)Nn$

$v_i^{k+1} = wv_i^k + C_1r_1(bv_i^k - x_i^k) + C_2r_2\left(bk_i^k - x_i^k\right)$

$x_i^{k+1} = x_i^k + v_i^{k+1}$

if $Z(x_i^{k+1}) < Z(bv_i^k)$

$bv_i^{k+1} = x_i^{k+1}$

if $rand \leq Prob\_rep$ **&&** bk unchanged 5 iterations

$x_A = best\ positioned\ roach\ in\ other\ family$

for $baby$=1 **to** *number_children*

Cross 10*dimen times x_A bk and bv

Cross 10*dimen times x_i bk and bv

Cross 10*dimen times previous 2 bests vectors

$x_{baby} = best\ vector\ from\ previous\ crossover$

if $rand \leq Prob\_mut\_rep$

Replace random vector's inputs by random values

elseif $rand \leq Prob\_mut\_fram$

Randomly change vector's inputs order

Eliminates randomly *Ne* = *number children* cockroaches

4 Results

In this section GRIO algorithm is tested. The goal of this section is to proof the value of this algorithm as well as the limitations. For that propose, some benchmark functions are selected and described in order that we can understand if there are different performances in different kind of problems. Also, different algorithms and one reference (from another study) were selected to compare results.

4.1 Benchmarks

Benchmark functions are very useful once they allow us to test new algorithms using functions with pre-known solutions and characteristics. So, in order to perform a valid test to the GRIO algorithm, the test functions must be carefully chosen. There are some characteristics that should be taken into consideration:

- Dimensionality: Measures the number of dimensions a function affords. Functions with more dimensions/parameters are more difficult to solve once the search space increases exponentially.
- Modality: Measures the number of peaks. As much as a function has more peaks, the probability of being trapped in local optima increases. Those peaks can be equal in depth or with a variable size. Uni-modal functions (one peak) are easier to solve.
- Separability: Measures the difficulty to optimize a function. If a dimension depends on the other the function is non-separable. If a dimension is independent from the other one the problem is separable and easier to solve than one with dimensions related to each other.
- Valleys: This phenomenon occurs when a narrow area is surrounded by steep descents. If an algorithm is trapped into such region it can be difficult to leave and slowed.

Several benchmark functions were chosen with different combinations of characteristics and possible dimensions. That way it is possible to test the proposed algorithm and understand if it has some difficulties and which characteristics raise it. In table 4.1, all chosen benchmark functions and its characteristics are outlined.

Table 4.1: Benchmark functions and their characteristics.

Function	Max Dimension	Multi-Modal	Non-Separable	Valleys
Ackley	∞	◆	◆	-
Becker lago	2	◆	-	-
Bohachevsky	2	◆	-	-
Easom	2	◆	-	-
Griewank	∞	◆	◆	-
Hartman	6	◆	◆	-
Hump	2	◆	◆	◆
Langerman	∞	◆	◆	-
LevyNo.5	2	◆	◆	-
Mccormick	2	◆	◆	-
Michalewicz	∞	◆	◆	◆
Paviani	10	◆	◆	-
Periodic	2	◆	-	-
Powell	∞	-	◆	-
Quartic	∞	◆	-	-
Rastrigin	∞	◆	◆	-
Rosenbrock	∞	-	◆	◆
Schaffer2	2	◆	◆	-
Shubert	2	◆	-	-
Sixhumpcamel	2	◆	◆	◆
Sphere	∞	◆	-	-
Threehumpcamel	2	◆	◆	◆
Wood	4	◆	◆	-

4.2 Results

In this section, using all the functions enunciated on table 4.1, results are presented. In order to test the algorithm and better understand its strengths and limitations, the results are compared with Roach Infestation Optimization (RIO), 2 variations from PSO (Linear-Weight PSO and Random-Weight PSO) and an already used state of the art for that function. Please refer to table 4.2 to check the reference algorithm for each benchmark.

Each function was tested in its advised initialization range and modality. On table 4.2 we can check all the functions not in a descriptive way (like table 4.1 where we can access the difficulty) but its dimension, initialization range, minimum and the associated reference (work from where the function was adapted).

Table 4.2: Benchmark Function's specifications.

Function	Dimension	Initialization Range	Minimum	Reference	Algorithm
Ackley	30	$[-32,32]$	0	[18]	KCPSO
Becker lago	2	$[-10,10]$	0	[4]	GLODS
Bohachevsky	2	$[-50,50]$	0	[4]	GLODS
Easom	2	$[-100,100]$	-1	[14]	RIO
Griewank	30	$[-600,600]$	0	[34]	MCPSO
Hartman	6	$[0,1]$	-3.32236	[4]	GLODS
Hump	2	$[-50,50]$	0	[14]	RIO
Langerman	10	$[0,10]$	-1.4	[4]	GLODS
LevyNo.5	2	$[-100,100]$	-176.1375	[18]	KCPSO
Mccormick	2	$[-1.5,4;-3,3]^T$	-1.9133	[4]	GLODS
Michalewicz	10	$[-\pi,\pi]$	-9.66015	[14]	RIO
Paviani	10	$[2.001,9.999]$	-45.778	[4]	GLODS
Periodic	2	$[-10,10]$	0.9	[4]	GLODS
Powell	4	$[-10,10]$	0	[4]	GLODS

Function	Dimension	Initialization Range	Minimum	Reference	Ref. Algorithm
Quartic	30	$[0.64,1.28]$	0	[34]	MCPSO
Rastrigin	30	$[-5.12,5.12]$	0	[34]	MCPSO
Rosenbrock	30	$[-2.048,2-048]$	0	[34]	MCPSO
Schaffer2	2	$[-100,100]$	0	[4]	GLODS
Shubert	2	$[-10,10]$	-186.73	[18]	KCPSO
Sixhumpcamel	2	$[-3,3;\ -2,2\ ]^T$	-1.0316	[4]	GLODS
Sphere	30	$[-100,100]$	0	[34]	MCPSO
Threehumpcamel	2	$[-100,100]$	-1.031628	[4]	GLODS
Wood	4	$[-10,10]$	0	[4]	GLODS

Results for GRIO are presented, as already said, along with 4 other algorithms that are the basis for several PSO variations and also for this work. While the results from reference are filled directly from that work with that work's specifications (that can be checked on the table 4.2 under the column "Reference"), the other algorithms specifications were used as listed:

- LW-PSO
  - Number of evaluations: 20
  - Maximum Iteration: 1000
  - Number of Particles: 20
  - C_1 and C_2: 1.43
  - Maximum inertia weight: 0.9
  - Minimum inertia weight: 0.4

- RW-PSO
  - Number of evaluations: 20
  - Maximum Iteration: 1000
  - Number of Particles: 20
  - C_1 and C_2: 1.43
  - Maximum inertia weight: 1
  - Minimum inertia weight: 0

- RIO
    - Number of evaluations: 20
    - Maximum Iteration: 1000
    - Number of Particles: 20
    - C_1 and C_2: 1.43
    - Inertia weight: 0.7

- GRIO
    - Number of evaluations: 20
    - Maximum Iteration: 1000
    - Number of Particles in each swarm: 10
    - C_1 and C_2: 1.43
    - Maximum inertia weight (when using linear decreasing): 0.9
    - Minimum inertia weight (when using linear decreasing): 0.4
    - $Prob\_rep$: 20%
    - $Prob\_mut\_rep$: 25%
    - $Prob\_mut\_fram$: 25%

With the exception of reference, the number of evaluations, iterations and particles are exactly the same for a fair comparison. Actually, it was attempted to keep all variables on the same values to have a better comparison. Final results for all tests can be found on table 4.3.

Table 4.3: Complete Results (bolded best results)

Function		**LW-PSO**	**RW-PSO**	**RIO**	**GRIO**	**Reference**
Ackley	Best	1.244e+02	1.197e+01	4.424	**2.841e-09**	1.146e-04
	Worst	3.124e+02	1.991e+01	1.970e+01	**4.589e-07**	3.901e-04
	Mean	2.227e+02	1.709e+01	1.020e+01	**1.482e-07**	2.005e-04
	Std.	4.753e+01	2.695	4.548	**1.422e-07**	3.772e-05
Becker lago	Best	**0**	**0**	**0**	7.099e-30	9.992e-18
	Worst	**0**	**0**	**0**	6.527e-18	Na
	Mean	**0**	**0**	**0**	6.917e-19	Na
	Std.	**0**	**0**	**0**	1.733e-18	Na
Bohachevsky	Best	**0**	**0**	**0**	**0**	1.288e-14
	Worst	**0**	**0**	**0**	6.217e-14	Na
	Mean	**0**	**0**	**0**	7.383e-15	Na
	Std.	**0**	**0**	**0**	1.764e-14	Na

Function		LW-PSO	RW-PSO	RIO	GRIO	Reference
Easom	Best	**-1**	**-1**	**-1**	**-1**	Na
	Worst	**-1**	**-1**	**-1**	**-1**	Na
	Mean	**-1**	**-1**	**-1**	**-1**	**-1**
	Std.	**0**	**0**	**0**	1.209e-12	Na
Griewank	Best	9.476e-04	1.048e-01	1.351e-12	**0**	1.480e-02
	Worst	4.000e-02	1.014e+00	1.970e-02	**2.929e-13**	7.870e-02
	Mean	1.600e-02	5.668e-01	7.500e-03	**5.256e-14**	4.91e-02
	Std.	1.120e-02	2.702e-01	8.400e-03	**8.535e-14**	1.78e-02
Hartman 6	Best	-3.3223	**-3.32236**	**-3.32236**	**-3.32236**	-3.321
	Worst	-3.2032	-3.2032	-3.2032	**-3.32236**	Na
	Mean	-3.2568	-3.2866	-3.2806	**-3.32236**	Na
	Std.	5.93e-02	5.46e-02	5.690e-02	**4.301e-09**	Na
Hump	Best	**4.651e-08**	**4.651e-08**	**4.651e-08**	**4.651e-08**	Na
	Worst	**4.651e-08**	**4.651e-08**	**4.651e-08**	**4.651e-08**	Na
	Mean	**4.651e-08**	**4.651e-08**	**4.651e-08**	**4.651e-08**	4.7e00e-08
	Std.	6.661e-17	**4.839e-17**	6.661e-17	2.425e-15	Na
Langerman	Best	**-1.4**	**-1.4**	**-1.4**	**-1.4**	-2.128e-02
	Worst	-2.223e-13	-1.673e-13	-1.912e-13	**-1.4**	Na
	Mean	-1.015	-9.351e-01	-8.428e-01	**-1.4**	Na
	Std.	5.574e-01	6.503e-01	0.671	**3.482e-09**	Na
LevyNo.5**	Best	-176.1376	**-176.13758**	**-176.13758**	**-176.13758**	**-176.13758**
	Worst	-176.1376	**-176.13758**	-166.0791	**-176.13758**	-176.13757
	Mean	-176.1376	**-176.13758**	-175.2099	**-176.13758**	**-176.13758**
	Std.	8.286e-14	7.784e-10	2.3812	**1.899e-11**	1.419e-05
Mccormick	Best	**-1.9132**	**-1.9132**	**-1.9132**	**-1.9132**	**-1.9132**
	Worst	**-1.9132**	**-1.9132**	**-1.9132**	**-1.9132**	Na
	Mean	**-1.9132**	**-1.9132**	**-1.9132**	**-1.9132**	Na
	Std.	**4.441e-16**	**4.441e-16**	**4.441e-16**	**4.441e-16**	Na
Michalewicz	Best	-9.362	-2.686	-9.2799	**-9.66015**	Na
	Worst	-5.640	-1.111	-5.8654	**-9.6552**	Na
	Mean	-7.300	-1.890	-7.9299	**-9.6579**	-5.8
	Std.	0.8607	0.482	0.9422	**0.0024**	Na

Function		LW-PSO	RW-PSO	RIO	GRIO	Reference
Paviani	Best	**-45.778**	**-45.778**	**-45.778**	**-45.778**	-14.053
	Worst	**-45.778**	**-45.778**	**-45.778**	**-45.778**	Na
	Mean	**-45.778**	**-45.778**	**-45.778**	**-45.778**	Na
	Std.	**1.124e-14**	2.824e-14	1.310e-14	5.427e-10	Na
Periodic	Best	**0.9**	**0.9**	**0.9**	**0.9**	0.99
	Worst	**0.9**	**0.9**	**0.9**	**0.9**	Na
	Mean	**0.9**	**0.9**	**0.9**	**0.9**	Na
	Std.	**0**	**0**	**0**	4.299e-17	Na
Powell	Best	9.321e-10	1.129e-08	8.584e-09	**7.070e-13**	6.985e-07
	Worst	7.843e-07	2.135e-07	**7.346e-08**	2.422e-06	Na
	Mean	1.657e-07	6.412e-08	**3.190e-08**	4.267e-07	Na
	Std.	2.185e-07	5.288e-08	**1.915e-08**	6.827e-07	Na
Quartic	Best	7.390e-02	8.730e-01	1.960e-01	3.500e-03	**1.710e-04**
	Worst	11.373e-01	1.170	11.984e-01	6.800e-03	**3.800e-03**
	Mean	6.674e-01	6.841e-01	7.774e-01	5.400e-03	**1.500e-03**
	Std.	2.988e-01	3.099e-01	3.158e-01	1.100e-03	**9.385e-04**
Rastrigin	Best	9.155e+01	3.097e+02	1.244e+02	**0**	8.512e-12
	Worst	2.817e+02	2.843e+03	3.124e+02	**1.810e-10**	1.990
	Mean	1.757e+02	7.733e+02	2.227e+02	**4.296e-11**	6.985e-01
	Std.	5.051e+01	5.325e+02	4.753e+01	**4.969e-11**	7.284e-01
Rosenbrock	Best	1.227e+02	4.517e+03	4.629e+00	**2.684e-05**	6.687e-05
	Worst	5.476e+02	8.719e+05	7.724e+01	**7.904e-02**	2.218e+01
	Mean	2.710e+02	2.261e+05	2.899e+01	**1.290e-02**	2.835
	Std.	1.079e+02	2.1432+05	1.976e+01	**2.280e-02**	3.583
Schaffer2	Best	**0**	**0**	**0**	**0**	1.279e-01
	Worst	**0**	**0**	**0**	1.998e-15	Na
	Mean	**0**	**0**	**0**	1.776e-16	Na
	Std.	**0**	**0**	**0**	4.636e-16	Na
Shubert	Best	**-186.73**	**-186.73**	**-186.73**	**-186.73**	**-186.73**
	Worst	**-186.73**	**-186.73**	**-186.73**	**-186.73**	**-186.73**
	Mean	**-186.73**	**-186.73**	**-186.73**	**-186.73**	**-186.73**
	Std.	7.535e-10	5.755e-14	3.706e-14	**1.378e-14**	2.240e-05

Function		LW-PSO	RW-PSO	RIO	GRIO	Reference
Sixhumpcamel	Best	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**
	Worst	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**	Na
	Mean	**-1.0316**	**-1.0316**	**-1.0316**	**-1.0316**	Na
	Std.	2.220e-16	**2.164e-16**	2.220e-16	4.314e-15	Na
Sphere	Best	4.973e-04	1.700e-04	2.005e-11	4.732e-16	**1.429e-36**
	Worst	8.311e-00	2.128e+02	3.131e-05	1.010e-11	**5.950e-31**
	Mean	9.746e-01	2.295e+01	1.764e-06	2.510e-12	**3.8883-32**
	Std.	2.365e-00	5.048e+01	6.821e-06	3.220e-12	**1.004e-31**
Threehumpcamel	Best	2.178e-134	**1.130e-224**	3.215e-108	2.729e-30	4.649e-18
	Worst	**9.604e-129**	2.986e-01	8.255e-101	8.002e-19	Na
	Mean	**9.766e-130**	1.490e-02	8.307e-102	5.1423e-20	Na
	Std.	**2.400e-129**	6.510e-02	1.841e-101	1.738e-19	Na
Wood	Best	1.700e-03	6.698e-05	1.213e-01	**7.097e-07**	1.264e-03
	Worst	5.874e-00	7.884e-00	7.720e-00	**3.430e-01**	Na
	Mean	1.228e-00	2.631e-00	2.560e-00	**1.059e-01**	Na
	Std.	1.707e-00	2.396e-00	2.348e-00	**1.103e-01**	Na

4.2.1. Number of Function Evaluations

Other result, which can be consulted on table 4.4, is the number of function evaluations needed to achieve best result. That's to say, in order to measure the performance of an algorithm, we shouldn't only look at results but also at the number of times the algorithm needed to evaluate the objective function in order to achieve the result [4]. In order to get that analysis, on table 4.4 we have the number of objective function evaluations each algorithm needed (except the one from reference) to achieve the best value. Bolded are the algorithms that found the best result or, in case of equal values (like for example Sixhumpcamel function), the one that achieved the best result in less evaluations.

Time of processing and calculation is directly related to the number of times objective function needs to be evaluated. Ideally an algorithm must be able to find the optimal point in the less time possible. For that reason, sometimes, is not profitable to have an algorithm that solves the problem in a time that does not fit our needs [4]. For that reason, the data presented on table 4.4 is so important as the results presented on table 4.3 and, for final discussions and considerations, both tables must be crossed and considered.

It is also important to note that the number of functions evaluated presented is dependent on the way the algorithm was programed. If the algorithm does not achieve the global optima, then the number of functions evaluations is the correspondent to the 1000 iterations done (maximum). If it achieves it, then the result presented in table 4.4 is the number of evaluations that the algorithm took until it achieves the global optima.

Table 4.4: Function evaluations for each tested algorithm. Bolded the ones that achieved the best result.

Function	**LW-PSO**	**RW-PSO**	**RIO**	**GRIO**
Ackley	840e+03	840e+03	880e+03	**1840e+03**
Becker lago	381360	**102480**	239360	920e+03
Bohachevsky	241920	**56280**	113520	803160
Easom	272160	**52080**	123200	896080
Griewank	840e+03	840e+03	880e+03	**1722240**
Hartman 6	840e+03	840e+03	880e+03	**615960**
Hump	**840e+03**	**840e+03**	880e+03	920e+03
Langerman	276360	**58800**	80960	427e+03
LevyNo.5	840e+03	**840e+03**	880e+03	920+03
Mccormick	**840e+03**	**840e+03**	880e+03	920+03
Michalewicz	840e+03	840e+03	880e+03	**732908**
Paviani	213360	**66360**	73040	107800
Periodic	35280	7560	**5280**	11040
Powell	840e+03	840e+03	880e+03	**1040+03**
Quartic	840e+03	840e+03	880e+03	**1840e+03**
Rastrigin	840e+03	840e+03	880e+03	**1766400**
Rosenbrock	840e+03	840e+03	880e+03	**1840e+03**
Schaffer2	263760	**56280**	107360	823400
Shubert	197400	**23520**	46640	459080
Sixhumpcamel	27720	8400	13200	18400
Sphere	840e+03	840e+03	880e+03	**1840e+03**
Threehumpcamel	840e+03	**840e+03**	880e+03	920e+03
Wood	840e+03	840e+03	880e+03	**1040e+03**

4.3 Discussion

In this chapter results are discussed. From this section, we will be able to fully understand the advantages and drawbacks from the new proposed algorithm.

In order to better organize discussion and results analysis, all functions were grouped in four groups of complexity according to some characteristics:

- Very Low Complexity: less than 4 dimensions and one characteristic from table 4.1.
- Low Complexity: More than 3 dimensions or more than one characteristic from table 4.1.
- Medium Complexity: More than 3 dimensions and more than one characteristic from table 4.1, or more than 9 dimensions.
- High Complexity: More than 9 dimensions and more than one characteristic from table 4.1.

On table 4.5 it is possible to find all the functions summarized by classification. Bolded are the functions that GRIO scored better. If, by instance, other algorithms achieved also the better solution, the function only is bolded if GRIO got the less function's evaluations.

For each four groups of classifications, results will be discussed taking into account not only the results presented on tables 4.3 and 4.4 but also convergence and results distribution.

Convergence analysis will be supported by graphics with result at each iteration (the path that results took to achieve the best registered value). Some of those graphics, have logarithmic or exponential scale providing a better visual analysis experience.

For results distribution analysis, were considered all the 20 tests done for each function and algorithm. This way, we can analyse the kind of results an algorithm creates using boxplots. Boxplots are graphics defined by a box, a superior and inferior line, and some points outside if necessary. Bottom line of the box defines the first quartile, the second line (dividing the box) corresponds to the median and the top of the box the third quartile. The superior and inferior lines, respectively, tell us the maximum and minimum results. If some results are more distant from the box that 1.5 times of the box, then those values will be represented as points, outliers.

Using convergence graphics and boxplots, despite knowing if an algorithm achieved or not the minimum value of the function, we may analyse if the algorithm is likely to fall into local optima values, if the algorithm is not consistent on the results and, if the best result was a luck phenomenon and not due to algorithm strategies, etc.

Table 4.5: Function's classification according to defined characteristics (ordered by increasing complexity).

Function	More or equal 4 Dimensions	More or equal 10 Dimensions	More or Equal to 2 characteristics	Group
Becker lago	-	-	-	Very Low
Bohachevsky	-	-	-	Very Low
Easom	-	-	-	Very Low
Periodic	-	-	-	Very Low
Shubert	-	-	-	Very Low
Hump	-	-	◆	Low
LevyNo.5	-	-	◆	Low
Mccormick	-	-	◆	Low
Powell	◆	-	-	Low
Schaffer2	-	-	◆	Low
Sixhumpcamel	-	-	◆	Low
Threehumpcamel	-	-	◆	Low
Hartman	◆	-	◆	Medium
Quartic	◆	◆	-	Medium
Sphere	◆	◆	-	Medium
Wood	◆	-	◆	Medium
Ackley	◆	◆	◆	High
Griewank	◆	◆	◆	High
Langerman	◆	◆	◆	High
Michalewicz	◆	◆	◆	High
Paviani	◆	◆	◆	High
Rastrigin	◆	◆	◆	High
Rosenbrock	◆	◆	◆	High

4.3.1. Very Low Complexity

In this group there are five functions (Beckerlago, Bohachevsky, Easom, Periodic and Shubert). Focusing on GRIO algorithm and its results, we can discuss that GRIO achieved always the global minimum at least once (except for Beckerlago that had a very close value). However, doing a general overview on table 4.3 we can notice that GRIO only scored better on Shubert because sometimes other algorithms failed to achieve global minimum (leading to standard deviations different from zero). Despite GRIO performs better on Shubert function, it takes much more function's evaluations to achieve that result. Taking into account function's evaluation and the fact that GRIO only can be considered with better results due to a slightly smaller standard deviation, we can conclude that GRIO algorithm is outperformed by all other algorithms used.

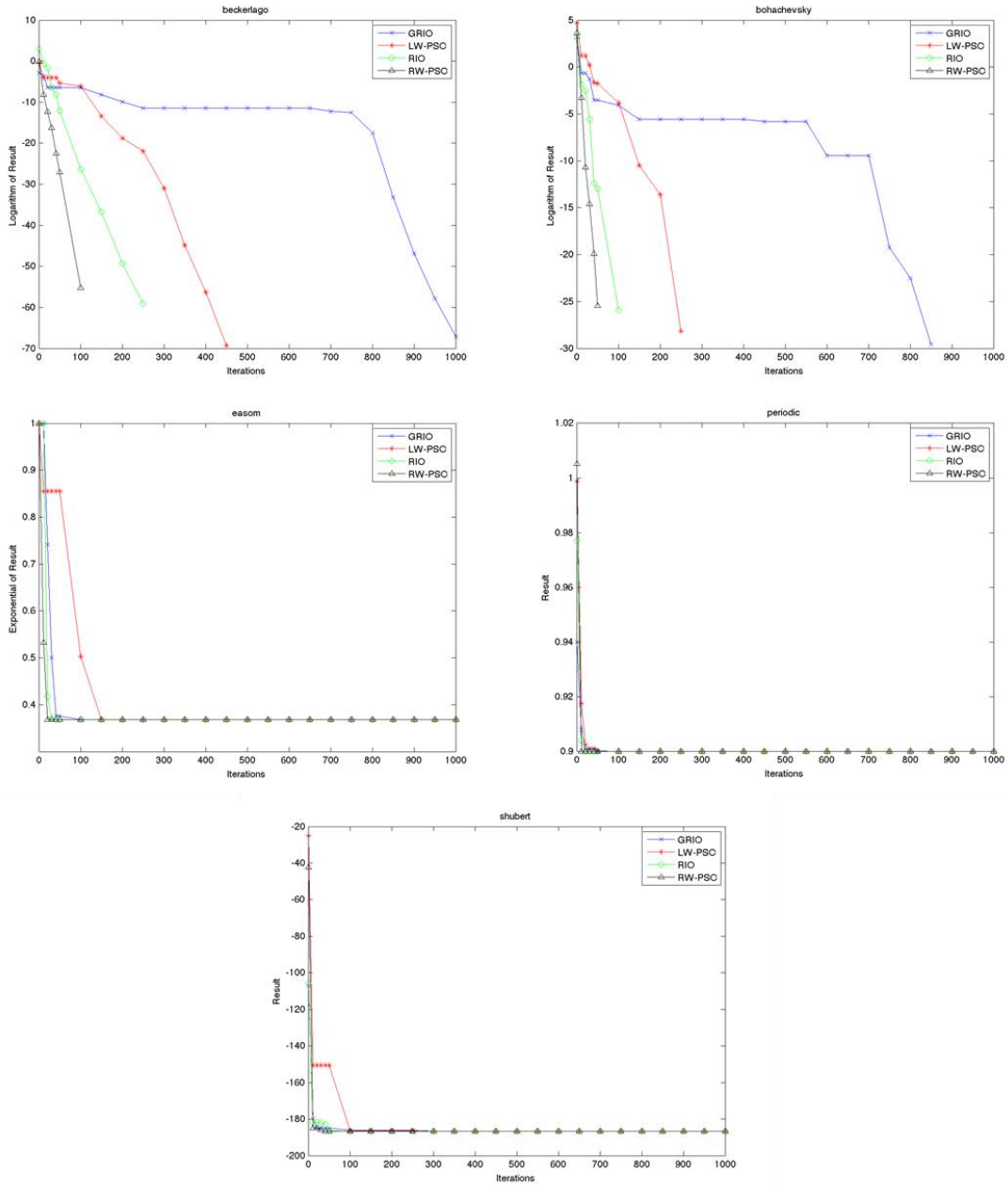

Figure 4.1: Convergence graphs for Very Low Complexity functions.

On figure 4.1 it is represented convergence for Very Low Complexity functions. When using logarithmic scale, if algorithm achieves absolute zero, data will disappear from the graph.

As we can see from figure 4.1, all algorithms achieved global minimum very quickly except for GRIO that failed to achieve it on Beckerlago function. As we can notice from the graphic of that function, GRIO only escaped from a local optima near iteration 800 and that condemned GRIO to fail.

On function Bohachevsky, despite achieving global minimum, GRIO algorithm took much more iterations than the competition and got stuck on local optima twice.

On all other functions GRIO manage to achieve global optima around the same number of iterations of other algorithms or even sooner. However if we check information on table 4.4, we can understand that per iteration, GRIO calculates more times the objective function and, despite achieving the global optima in less iterations, it required the use of more function evaluations (and that means more time for computer processing).

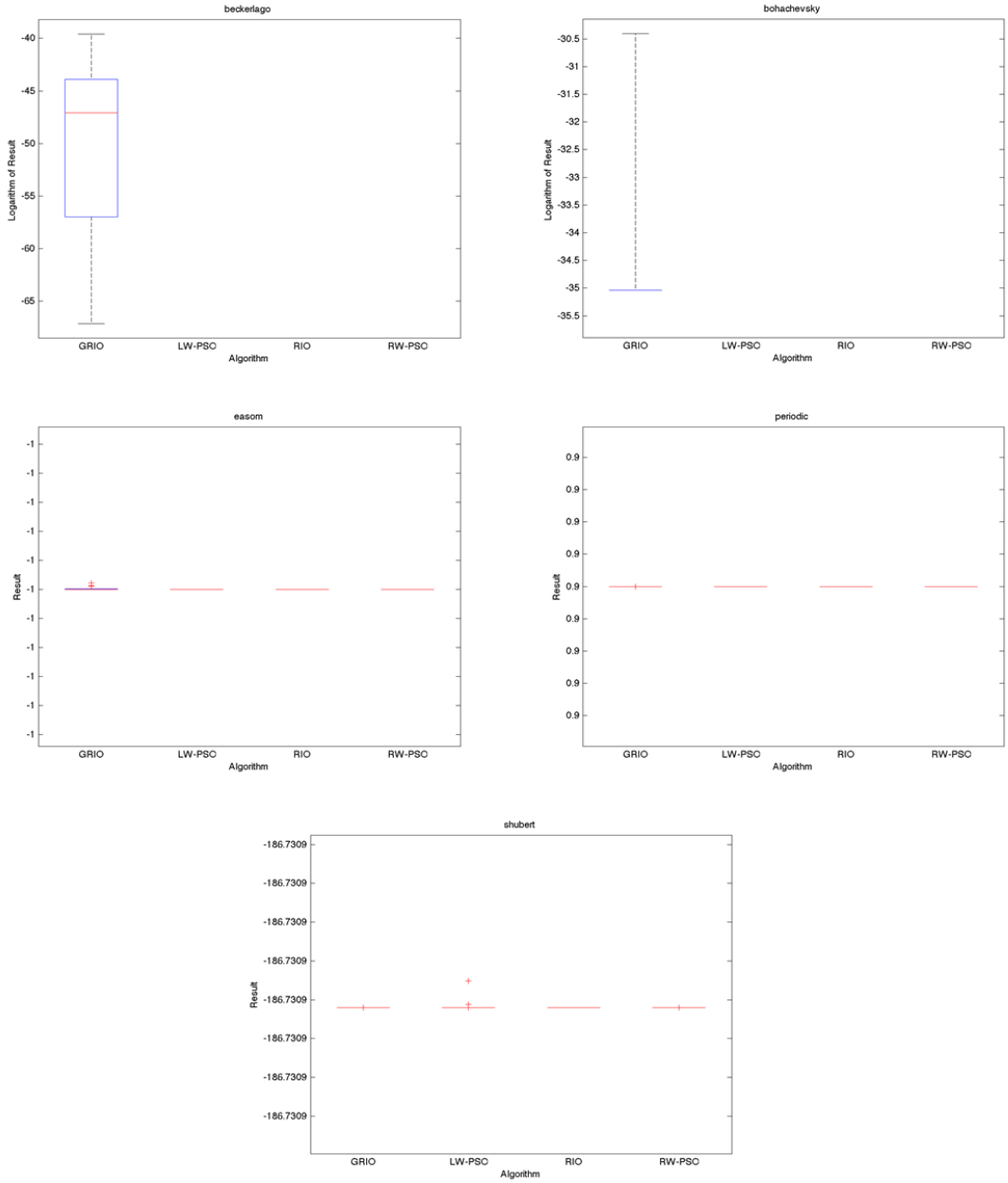

Figure 4.2: Boxplots for Very Low Complexity functions.

Regarding data dispersion on the 20 trials, we can consult figure 4.2 with all boxplots for Very Low Complexity functions. On Beckerlago and Bohachevsky functions, a logarithmic scale was used, reason why LW-PSO, RIO and RW-PSO graphs are empty because they always achieved absolute zero.

Analysing other functions graphics we can confirm, the conclusion already got by standard deviation values, that GRIO algorithm failed in achieving global optima in every single one of the twenty tests. However all other algorithms could do it except on Shubert function.

4.3.2. Low Complexity

In this section, seven functions are studied: Hump, Levy 5, Mccormick, Powell, Scaffer2, Sixhumpcamel and Threehumpcamel.

Analysing this category on table 4.3 it is possible to conclude that, once again, GRIO might not be the best algorithm.

On function Hump, GRIO always achieved the same results as other algorithms but presented a slightly bigger standard deviation. Regarding the function Levy 5, GRIO behave like in function Shubert from Very Low group: Best results always but the difference for RW-PSO is so small that the fact that GRIO uses more function evaluations makes RW-PSO a better option. On other algorithms results were not satisfactory. All algorithms achieved the exact same values in Mccormick function so the fact that GRIO needed more function evaluations makes RW and LW-PSO better choices. On Powell function GRIO managed to over perform other algorithms on best result ever achieved but was beaten in number of function evaluations, average, worst result and standard deviation. In the remaining three functions (Schaffer2, Sixhumpcamel and Threehumpcamel) it is easy to access by analysis of table 4.3 that GRIO was the worst algorithm despite it produced satisfactory scores.

Studying convergence graphs presented on figure 4.3, we can see in this category GRIO showed up with some convergence problems once in function Hump, Powell, Schaffer2 and Sixhumpcamel only in last iterations (after 800) it managed to go into the final slope. In Powell function, where it managed to achieve a better result than other algorithms, it only could do it after iteration 800 (before it was presenting the same as RW-PSO).

Still checking convergence performances, on remaining functions, all algorithms performed the same, converging almost instantly to final values.

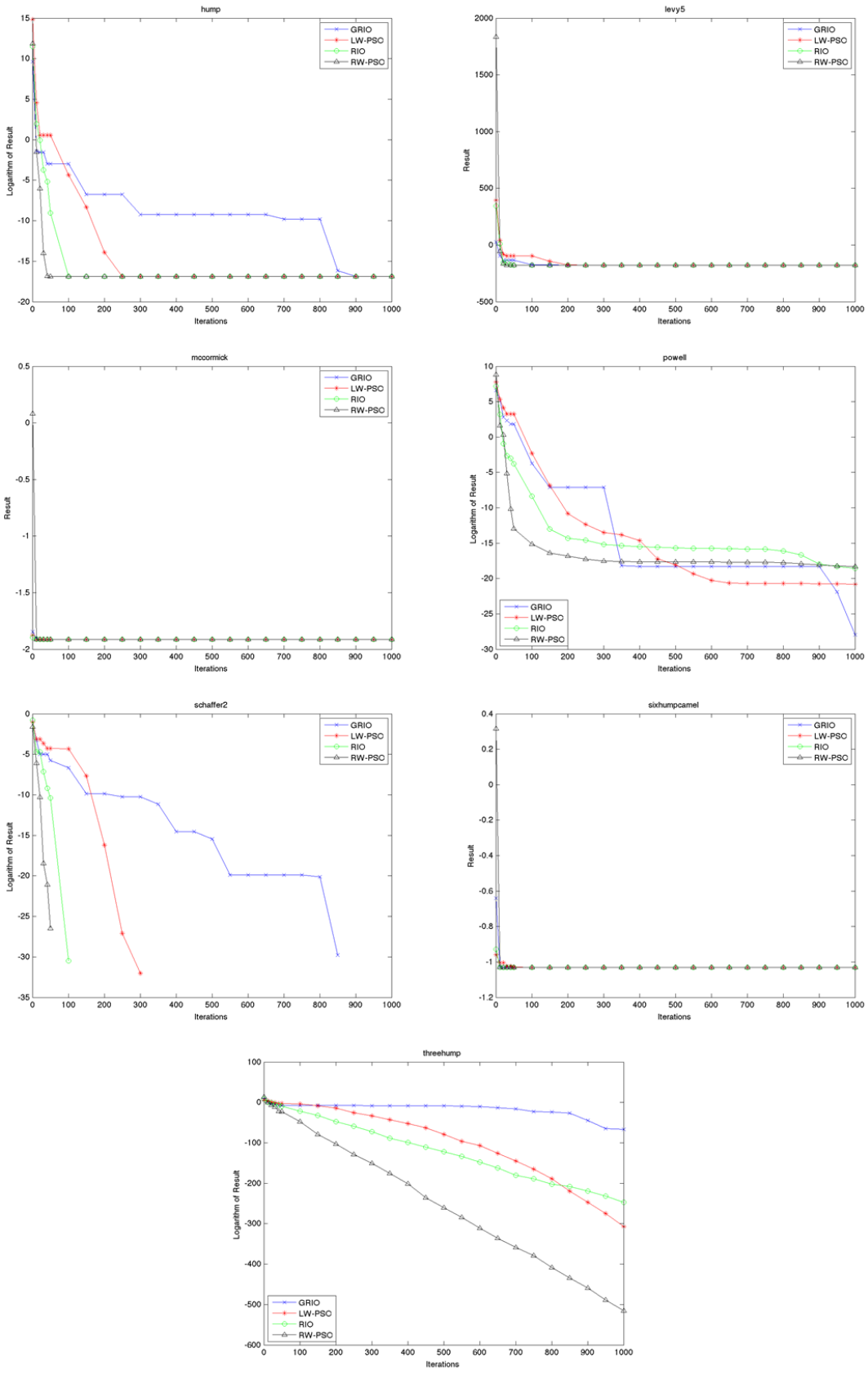

Figure 4.3: Convergence graphs for Low Complexity functions.

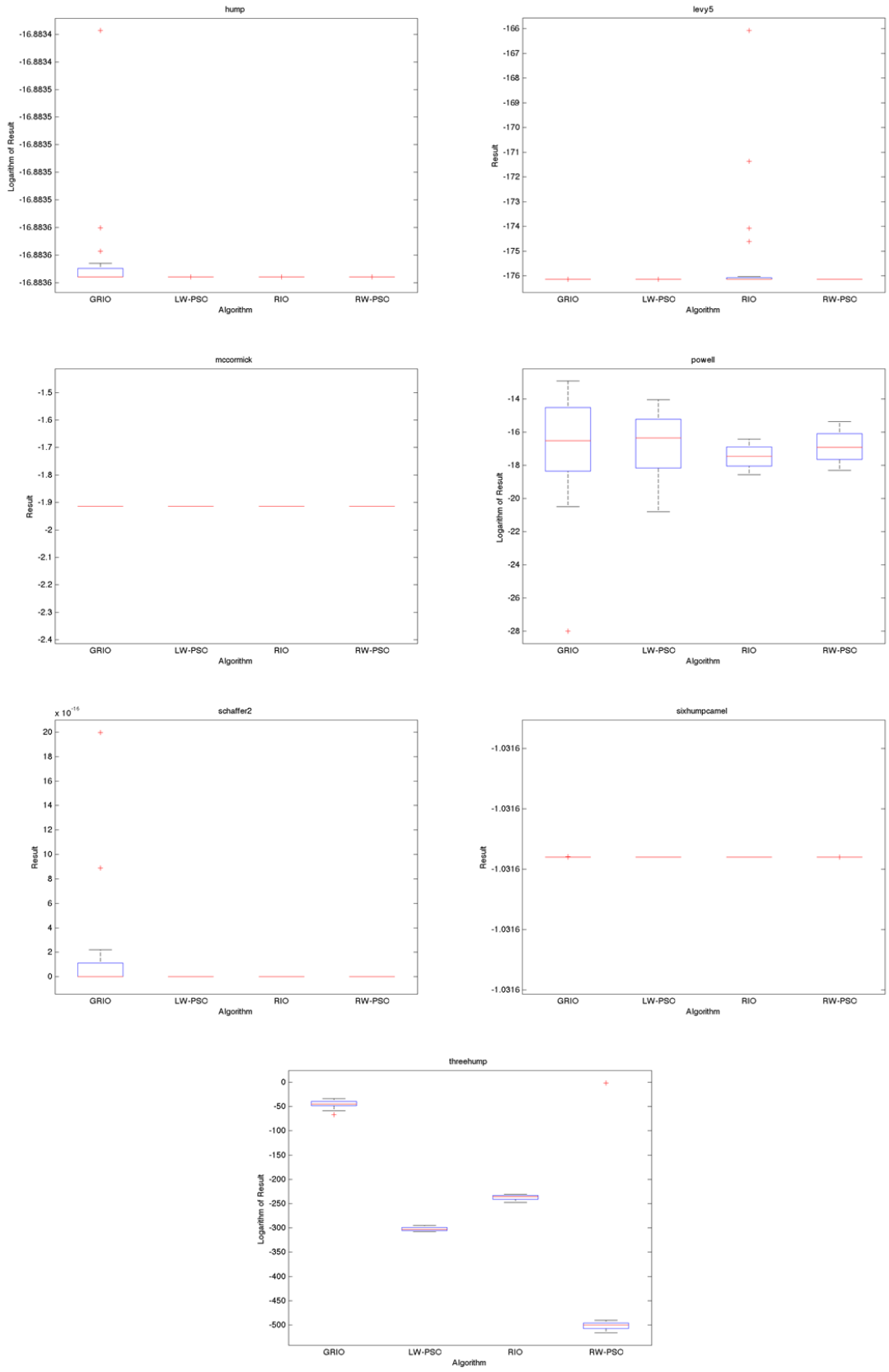

Figure 4.4: Boxplots for Low Complexity functions.

Results dispersion can be consulted on figure 4.4. On Hump function GRIO's box we can observe 3 superior outlier values and a much bigger box than other algorithms. Meaning that GRIO is not being as consistent as other algorithms on final solutions provided. Schaffer 2 boxplots follow the same examination as Hump boxplots. On Levy 5 function we only can access that RIO algorithm produced results much more deviated than the others. On Powell function GRIO scored the best value as it was already noticed from previous material analysis. However, that result can be considered an outlier once it is very different from other values between first and third quartiles. According to figure 4.4, on Powell function, RIO is the most reliable algorithm. In Mccormink and Sixhumpcammel functions, boxplots are not a good material to bring conclusions, so understandings from other graphs and tables must be taken into account. Threehumpcamel boxplot clarify that GRIO is not for sure the best way to solve this function, even if it presents consistent results, once other algorithms (led by RW-PSO) are consistent as well but around a lower point.

4.3.3. Medium Complexity

Hartman, Quartic, Sphere and Wood were the functions classified as Medium Complexity. Those are functions with 10 or more dimensions or between 4 and 9 dimensions but with 2 characteristics at least from table 4.1.

On Hartman function, GRIO clearly outperformed all other algorithms achieving always global minima value. Quartic and Shpere functions were the only two cases where reference algorithm achieved better results that all the tested ones (despite GRIO algorithm achieved the best results among the tested one and always very close to reference one, that had different conditions). Checking results for Wood function, we can see that GRIO algorithm got a very good best result comparing to other algorithms. Overall GRIO scored better than all other tested algorithms (in the same conditions).

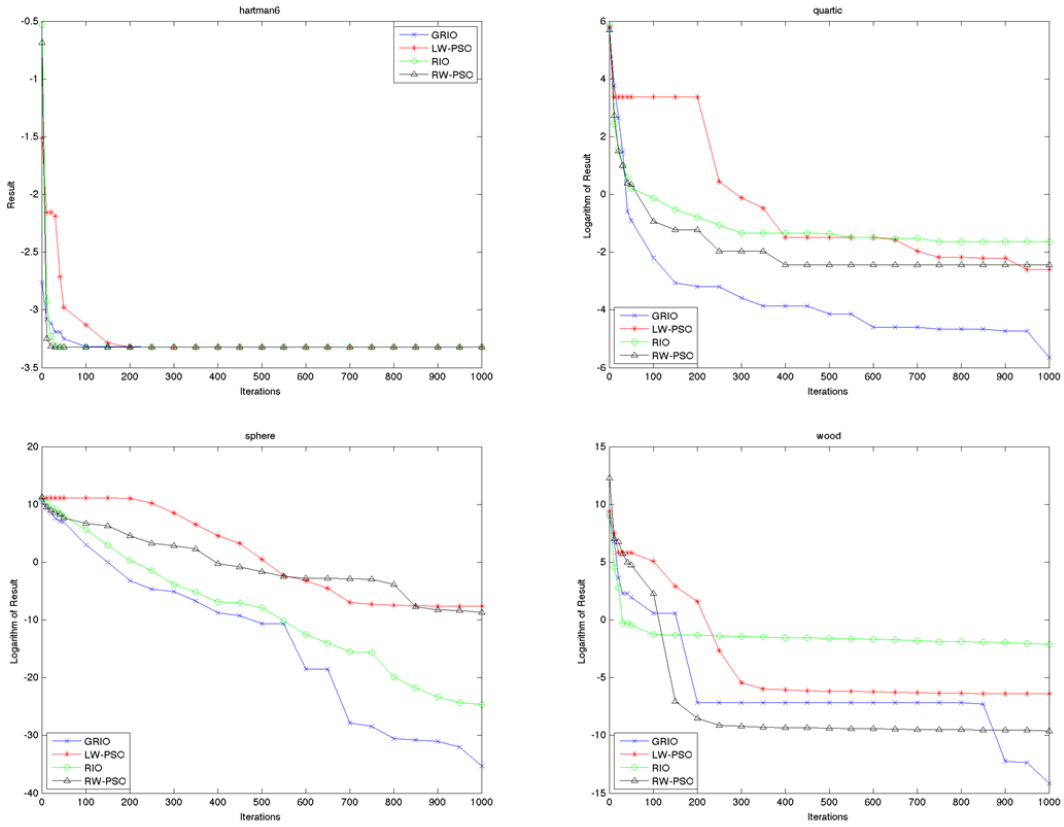

Figure 4.5: Convergence graphs for Medium Complexity functions.

Studying convergence from figure 4.5 we can see that in every single function, GRIO presented a faster convergence and more effective. In Hartman's case, it is difficult to access GRIO better results once the scale is not the best one. However, as already referred, it is easy to notice GRIO's more accurate results from table 4.3. In all other cases we can see that all other algorithms were trapped into local optima points and, for them, in contrast to GRIO, it was difficult to escape to a better solution. RIO, managed to not being trapped into local optima values on Sphere function, however RIO's convergence was too slow when compared to GRIO's one.

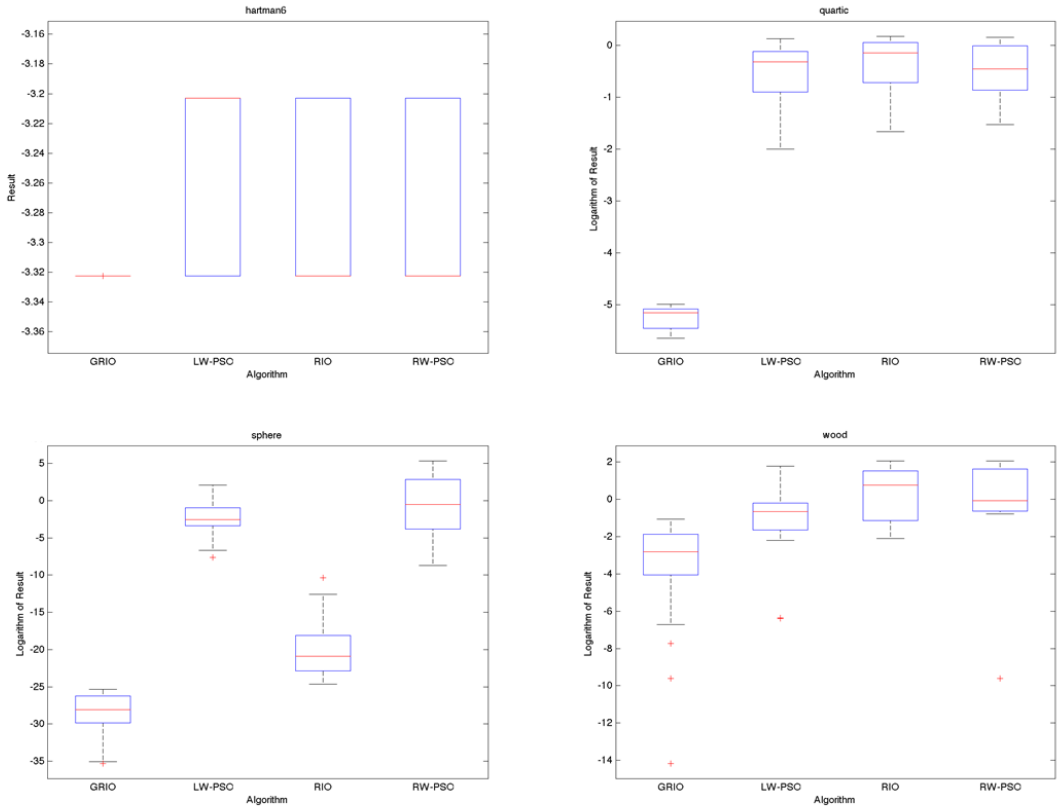

Figure 4.6: Boxplots for Medium Complexity functions.

Reading figure's 4.6 boxplots, some interesting results might be noticed. For Hartman function, it is possible to confirm table 4.3 results. Actually, other algorithms' were able to achieve optimum value such like GRIO. However, in all other tests the range of results were much bigger than GRIO's. In Quartic and Sphere GRIO clearly has a less range of values, meaning that results are consistent among a better result than all other algorithms. At least, Wood function looks to be a function susceptible to outliers' values once 3 out of 4 algorithms got inferior outliers. GRIO best result is an outlier result but, if we exclude outliers, GRIO would still be the best algorithm to choose.

4.3.4. High Complexity

Finally High Complexity functions are studied. Those functions are the ones where all other algorithms fail to get consistent and satisfactory results: Ackley, Griewank, Langerman, Michalewickz, Paviani, Rastrigin and Rosenbrock.

Looking at table 4.3, we can observe that GRIO scored the better results in all of them. In Ackley function, GRIO failed to achieve the absolute 0, however managed to get an impressive 2.841e-09 with a standard deviation of 1.422e-07 while reference one only got as best result 1.146e-04. On Griewank function GRIO scored the absolute 0 as best result and only a deviation of 8.535e-14 (smaller than all best results from other studied algorithms). On Langerman function, all other studied algorithms (except reference one) were able to achieve global minima but, only GRIO, could do it in a consistent way. GRIO followed the same on Michalewickz function as on Griewank function. Achieved the global optimum and scored an average very close to the minimum. On Paviani function, all algorithms tested achieved the same (global minimum) in every single test and low deviations. On Rastrigin function, once again, GRIO managed to achieve global optima and an average of 4.296e-11 that considering that the best result on tested algorithms never went to decimal numbers (except on reference that achieved 8.512e-12) is very good. Finally, on Rosenbrock function, GRIO achieved better values in all fields again. Tested algorithms were all very far away from global optima so the comparison in this function will be done with reference. Despite GRIO and reference algorithm scored almost the same on best result: on worst score, average score and standard deviation the difference is huge so, again, in this function, GRIO is the best option available.

Regarding convergence, interpretation from figure 4.7 is quite easy. If we check functions where GRIO was not the only one to achieve the optimum point, we can see that all algorithms got a fast convergence to that point (even before the iteration 100). However, in functions where other algorithms failed, due to lack of capacity to avoid local minima points, GRIO could overcome those points and keep improving its score. If we look to GRIO's line on Ackley and Rosenbrock functions (the only ones that GRIO could not find the minimum in 1000 iterations) it is easy to spot that with more iterations the line should keep decreasing, once that the algorithm was not stuck in local points like other tested algorithms, and global minimum should, eventually, be found.

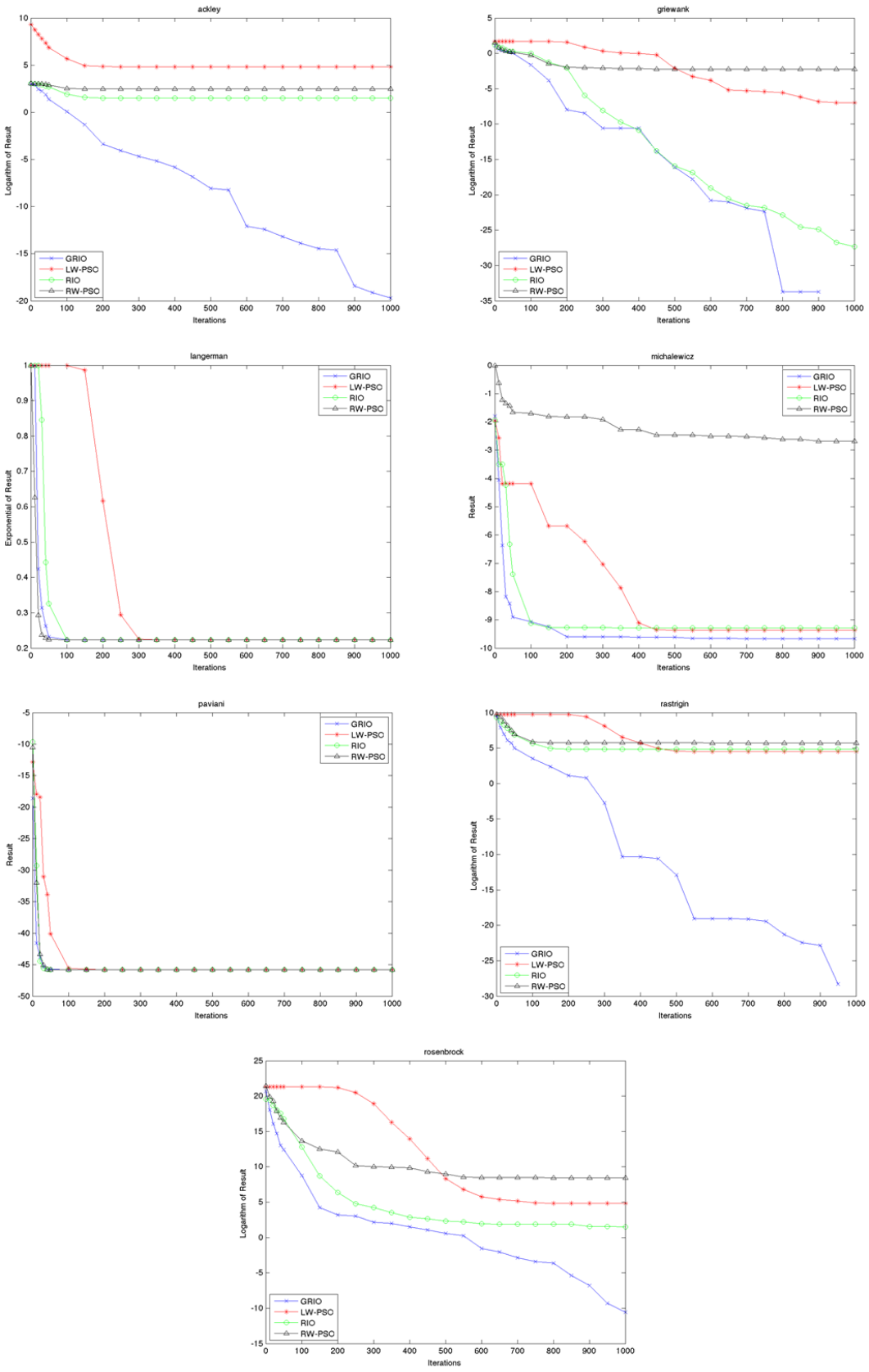

Figure 4.7: Convergence graphs for High Complexity functions.

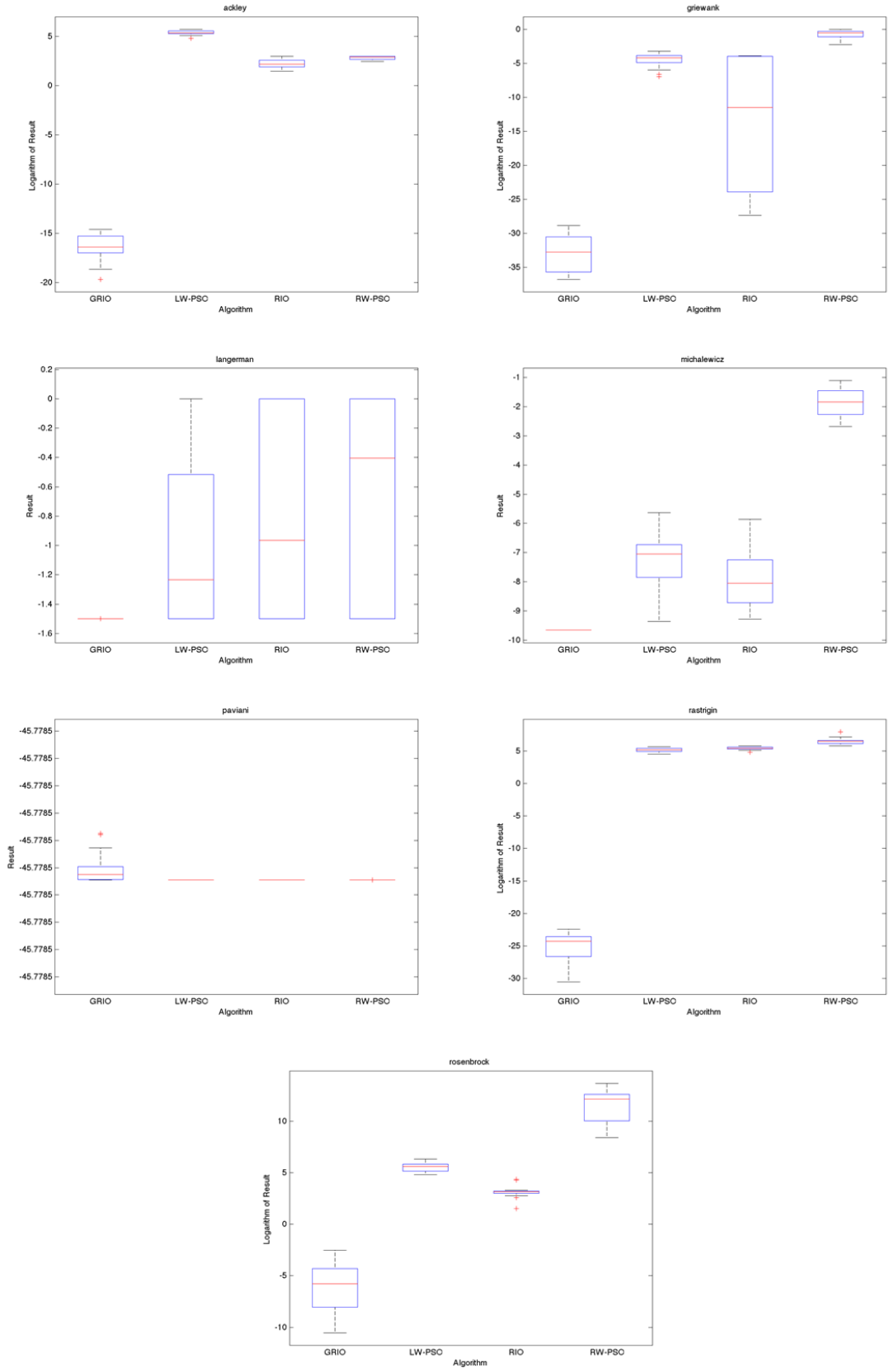

Figure 4.8: Boxplots for High Complexity functions.

Checking figure 4.8 for distribution analysis, some results can be easily validated. On Ackley's graph, we can see that GRIO's best result was an outlier one. Nevertheless, as it is easily observed, the worst result is much better than all algorithms best result and the median line is more tied to the inferior values than superior ones. Griewank function, where GRIO was able to achieve global optima, shows as well a good consistency in all tests done. Actually, RIO, the second algorithm with best score, $1.351e-12$, achieved that score with a very big range of results among all tests. On Langerman function, figure 4.8 reveals, in an easy way to read, that GRIO algorithm was the most consistent among all the others despite the best result is the same (equal to -1.4, global minima). Paviani was the only one where GRIO could not perform the best. Despite having achieved global optima, results among the 20 tests done to RIO were slightly inconsistent that other 3 tested algorithms. Finally, on Michalewickz, Rastrigin and Rosenbrock functions, GRIO clearly outperformed other algorithms achieving more consistency in the results produced and more accuracy. From these three functions, GRIO managed to find out global minima in all of then except Rosenbrock. However, in all of them the worst result obtained by GRIO is better than the best obtained by other algorithms.

5 Conclusions and Future Work

In this final chapter, all learnings from previous chapters are analysed aiming to a final conclusion. It is supposed to analyse all previous work and explain all advantages and drawbacks from this algorithm, learned before.

Questions like applicability, future work and potential are answered as well.

5.1 Conclusions

This work, as explained on chapter 1, was inspired by a documentary. Meaning it is a piece of creativity that gave a lot of fun to build. It was really rich to start something from the beginning until the end. To leave some ideas that looked promisor in order to move on with other ones. To look for what is done in the optimization world and to try to adapt and improve this work. To succeed with results and produce an algorithm able to solve the most complex continuous problems.

After the final algorithm development, 3 other algorithms were selected and programmed, to act as a fair comparison (with the same number of maximum iterations, the same values in constants, and so on). Also results were compared with a reference (using the values provided in their papers). That way we could compare with algorithms in the same conditions and algorithms more complex with ideal conditions for that algorithm. GRIO and 3 other algorithms in the same conditions were tested each one 20 times for each function once those algorithms are based in probabilities and produce different results each time they are run. By doing this, it was possible to study the consistency of the algorithm once it is supposed to run a problem the smaller number of times possible and, still, produce a result that might be acceptable for the problem. All the results are described in chapter 4.2 and interpretation is conducted on chapter 4.3.

Looking at chapter 4.2 we are able to see a table with all the results resumed and the number of function evaluations. It is easy to note that GRIO demands more function evaluations than basic algorithms used on the comparison. Actually, this can be a drawback at first sight but, if more function evaluations makes GRIO achieve the result in early iterations it can be an advantage. Also it is easy to see that GRIO is not the top performer in a lot of functions. This first observations that can be made, by doing an overview on chapter 4.2 show us that, for sure, GRIO would not be the best option for some kind of functions and, if it is, it might take more function evaluations that other simpler algorithms.

On chapter 4.3, results discussion, doubts raised in previous paragraphs are answered. With all benchmark functions split in groups according to their complexity and graphics to study algorithms behaviours, it was easy to spot that first thoughts were right: GRIO is better only to a certain group of functions.

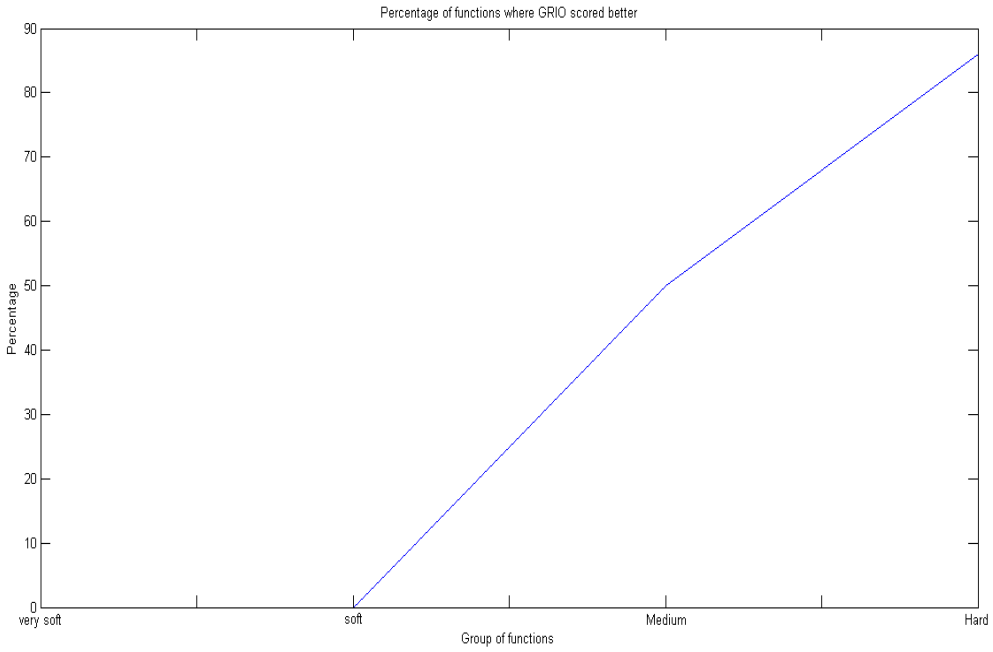

Figure 5.1: Percentage of functions where GRIO scored best per group of functions

As it is seen on figure 5.1, on very Low and Low Complexity functions, all other algorithms were better than GRIO. That happens because GRIO achieves global minima but uses more function evaluations or fails more times in achieving global minima. This last option is the one that occurs more often and is a pattern observed in almost all functions with few dimensions. Most of anti-premature convergence strategies from GRIO are based in changing information between dimensions, meaning that if there are few dimensions, few information is changed as well. Then with adaptive inertia weight techniques, that information can be worked and polished. So, GRIO does not outperform other algorithms in functions with few dimensions.

In fact, it is proved that GRIO's performance improves as complexity increases. On High Complexity functions, due to its escape from local optima techniques, convergence velocity regulation among other details, GRIO was able to achieve results not seen with any other algorithm.

Concluding, we can say that GRIO allows optimization of High Complexity, high-dimensional and continuous functions. In fact, looking at table 4.3, and focusing only on the most difficult benchmark functions, we can spot that GRIO outperformed with a big distance all other algorithms. Even the reference ones! On Ackley function GRIO achieved an impressive best result of $2.841e-09$ and average of $1.482e-07$ while reference one achieved a mean of $2.005e-04$. On Griewank function reference could obtain a best result and means on the order of $e-2$ while GRIO achieved the absolute zero and an average of $5.256e-14$. Michalewickz is another example of success once that GRIO was able to achieve global minima and an average of -9.6579 and reference algorithm only achieved -5.4.

Paviani was the only example where GRIO was outperformed by tested algorithms (only the standard deviation was a little worst) but still was able to beat reference by a long distance. Another very difficult function is Rastrigin, where GRIO achieved global minima and an amazing average of $4.296e-11$ even though reference only got a best result of $8.512e-12$ and a mean result of $6.985e-01$. Finally we have Rosenbrock, as well in the group of High Complexity functions and one of the most difficult in all this group. GRIO was not able to achieve global minima scoring only a best result of $2.684e-05$ with reference achieving $6.687e-05$. However, even thou the best result was only slightly better than reference one, analysing GRIO's mean value of $1.290e-02$ and reference mean value of 2.835 we can conclude that GRIO always achieved consistent results not seen before in these function and that is a big advance for this specific function.

As a final point, analysing all the work done and previous conclusions, we can conclude that this work was concluded with success once that the main objective was achieved: to produce an algorithm capable of solving the most difficult continuous functions. Taking into account professional life and continuous increase of optimization in industries, this work was also rich in knowledge earned in optimization, its challenges and how to overcome them. Also, it was an amazing experience to learn how to create something new, test it, compare it and analyse the results. Taking into account all challenges that professional life brings to a young engineer, all these competences are a must to have and I am glad this work helped me to master them.

5.2 Future Work

Now that we know about this algorithm potential, some steps should be taken in order to transport these algorithm to real applications.

A discrete version of the algorithm must be made once, all real-life problems are modelled as discrete. One of those problems could be related to logistics' industries in order to optimize delivery routes taking into account the kind of vehicle, transportation, deadlines, fuel's consumption, etc.

Another potential work, in the robotic area, for an algorithm like this can be the design of robotic cockroaches. Each one of these robots would be an individual inside a swarm that could behave like real roaches. Those robotic roaches could work together in order to find something that can be measured by sensors.

6 References

[1] Amé, J.-M., et al., Collegial decision making based on social amplification leads to optimal group formation. Proceedings of the National Academy of Sciences, 2006. 103(15): p. 5835-5840.

[2] Carlisle, A. and G. Dozier, An Off-The-Shelf PSO. 2001.

[3] Craig, W.R., Flocks, herds and schools: A distributed behavioral model. SIGGRAPH Comput. Graph. %@ 0097-8930, 1987. 21(4): p. 25-34.

[4] Custódio, A.L. and J.F.A. Madeira, GLODS: Global and Local Optimization using Direct Search. Journal of Global Optimization, 2015. 62(1): p. 1-28.

[5] del Valle, Y., et al., Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems. Evolutionary Computation, IEEE Transactions on, 2008. 12(2): p. 171-195.

[6] Eberhart, R. and J. Kennedy. A new optimizer using particle swarm theory. in Micro Machine and Human Science, 1995. MHS '95., Proceedings of the Sixth International Symposium on. 1995.

[7] Fister Jr, I., et al., A brief review of nature-inspired algorithms for optimization. arXiv preprint arXiv:1307.4186, 2013.

[8] Fourie, P.C. and A.A. Groenwold, The particle swarm optimization algorithm in size and shape optimization. Structural and Multidisciplinary Optimization, 2002. 23(4): p. 259-267.

[9] Frederick, S.H. and J.L. Gerald, Introduction to operations research, 4th ed. 1986: Holden-Day, Inc. 888.

[10] Gautier, J.Y., P. Deleporte, and C. Rivault, Relationships between ecology and social behavior in cockroaches. The ecology of social behavior, 1988: p. 335-351.

[11] Gillott, C., Entomology. 2005: Springer Netherlands.

[12] Halloy, J., et al., Social Integration of Robots into Groups of Cockroaches to Control Self-Organized Choices. Science, 2007. 318(5853): p. 1155-1158.

[13] Hancock, P.J.B., An empirical comparison of selection methods in evolutionary algorithms, in Evolutionary Computing: AISB Workshop Leeds, U.K., April 11–13, 1994 Selected Papers, T.C. Fogarty, Editor. 1994, Springer Berlin Heidelberg: Berlin, Heidelberg. p. 80-94.

[14] Havens, T.C., et al. Roach Infestation Optimization. in Swarm Intelligence Symposium, 2008. SIS 2008. IEEE. 2008.

[15] Holland, J.H., Adaptation in Natural and Artificial Systems. 1975: MIT Press.

[16] James, K., The Behavior of Particles, in Proceedings of the 7th International Conference on Evolutionary Programming VII %@ 3-540-64891-7. 1998, Springer-Verlag. p. 581-589.

[17] Jayaram, K. and R.J. Full, Cockroaches traverse crevices, crawl rapidly in confined spaces, and inspire a soft, legged robot. Proceedings of the National Academy of Sciences, 2016. 113(8): p. E950-E957.

[18] Jie, J., et al., Knowledge-based cooperative particle swarm optimization. Applied Mathematics and Computation, 2008. 205(2): p. 861-873.

[19] Junliang, L. and X. Xinping. Multi- Swarm and Multi- Best particle swarm optimization algorithm. in Intelligent Control and Automation, 2008. WCICA 2008. 7th World Congress on. 2008.

[20] Karaboga, D., An idea based on honey bee swarm for numerical optimization. 2005, Technical report-tr06, Erciyes university, engineering faculty, computer engineering department.

[21] Kennedy, J. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. in Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. 1999.

[22] Kennedy, J. and R. Eberhart. Particle swarm optimization. in Neural Networks, 1995. Proceedings., IEEE International Conference on. 1995.

[23] Kennedy, J., R.C. Eberhart, and Y. Shi, Swarm intelligence. 2001. Kaufmann, San Francisco, 2001. 1: p. 700-720.

[24] Kennedy, J. and R. Mendes. Population structure and particle swarm performance. in Evolutionary Computation, 2002. CEC '02. Proceedings of the 2002 Congress on. 2002.

[25] Krohling, R.A. and L. dos Santos Coelho, Coevolutionary particle swarm optimization using Gaussian distribution for solving constrained optimization problems. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on, 2006. 36(6): p. 1407-1416 %@ 1083-4419.

[26] Liang, J.J., et al., Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. Evolutionary Computation, IEEE Transactions on, 2006. 10(3): p. 281-295.

[27] Lihoreau, M., C. Zimmer, and C. Rivault, Kin recognition and incest avoidance in a group-living insect. Behavioral Ecology, 2007. 18(5): p. 880-887.

[28] Lihoreau, M., L. Brepson, and C. Rivault, The weight of the clan: even in insects, social isolation can induce a behavioural syndrome. Behavioural Processes, 2009. 82(1): p. 81-84.

[29] Liu, Y., et al., Center particle swarm optimization. Neurocomputing, 2007. 70(4–6): p. 672-679.

[30] Marco, D., S. Thomas, and tzle, Ant Colony Optimization. 2004: Bradford Company.

[31] Mendes, R., J. Kennedy, and J. Neves, The fully informed particle swarm: simpler, maybe better. Evolutionary Computation, IEEE Transactions on, 2004. 8(3): p. 204-210.

[32] Nickabadi, A., M.M. Ebadzadeh, and R. Safabakhsh, A novel particle swarm optimization algorithm with adaptive inertia weight. Applied Soft Computing, 2011. 11(4): p. 3658-3670.

[33] Nitasha, d.t.k., Study of Various Mutation Operators in Genetic Algorithms. Ijcsit, 2014. 5(3): p. 4519-4521.

[34] Niu, B., et al., MCPSO: A multi-swarm cooperative particle swarm optimizer. Applied Mathematics and Computation, 2007. 185(2): p. 1050-1062.

[35] Nocedal, J. and S.J. Wright, Numerical Optimization, Second Edition. Numerical optimization, 2006: p. 497-528.

[36] Peram, T., 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. 2003.

[37] Sakura, M. and M. Mizunami, Olfactory Learning and Memory in the Cockroach Periplaneta americana. Zoological Science, 2001. 18(1): p. 21-28.

[38] Shi, Y. and R. Eberhart, Parameter selection in particle swarm optimization, in Evolutionary Programming VII, V.W. Porto, et al., Editors. 1998, Springer Berlin Heidelberg. p. 591-600.

[39] Van den Bergh, F. and A.P. Engelbrecht, A Cooperative approach to particle swarm optimization. Evolutionary Computation, IEEE Transactions on, 2004. 8(3): p. 225-239.

[40] Wada-Katsumata, A., J. Silverman, and C. Schal, Changes in Taste Neurons Support the Emergence of an Adaptive Behavior in Cockroaches. Science, 2013. 340(6135): p. 972-975.

[41] Wilson, E.O.U.h.b.g.p.b.i.P.F.-j., Sociobiology: The New Synthesis. 1975: Belknap Press of Harvard University Press.

[42] Yang, X.-S. and S. Deb, Engineering optimisation by cuckoo search. International Journal of Mathematical Modelling and Numerical Optimisation, 2010. 1(4): p. 330-343.

[43] Yuhui, S. and R. Eberhart. A modified particle swarm optimizer. in Evolutionary Computation Proceedings, 1998. IEEE World Congress on Computational Intelligence., The 1998 IEEE International Conference on. 1998.

[44] Zhang, J. and X. Ding, A Multi-Swarm Self-Adaptive and Cooperative Particle Swarm Optimization. Engineering Applications of Artificial Intelligence, 2011. 24(6): p. 958-967.