REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
MINISTERE DE L'ENSEIGNEMENT SUPERIEURE ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE ABOU-BEKR BELKAID - TLEMCEN

LAT
Laboratoire d'Automatique

MEMOIRE

Présenté à :

FACULTE DES SCIENCES – DEPARTEMENT DE PHYSIQUE

Pour l'obtention du diplôme de :

MASTER EN PHYSIQUE

Spécialité : Physique Computationnelle

Par :

Mlle MOUALEK Amina

Sur le thème

Optimization and Simulation Using Metaheuristic Algorithms

Soutenu le 11 Octobre 2021 à Tlemcen devant le jury composé de :

M. BEKHECHI Smaine	Professeur	Université de Tlemcen	Président
M. BENDAHMANE Med Fawzi	MCB	Université de Tlemcen	Encadrant
M. MERAD Abdelkrim	Professeur	Université de Tlemcen	Examinateur
M. BRAHMI Badr Eddine Nabil	MCA	Université de Tlemcen	Examinateur

Année Universitaire : 2020 ~ 2021

Acknowledgments

First and Foremost praise is to ALLAH , the Almighty, the greatest of all, on whom ultimately we depend for sustenance and guidance. I would like to thank Almighty Allah for giving me determination and strength to go through everything. His continuous grace and mercy was with me throughout my life and ever more during the tenure of my research.

I would like to express my sincere gratitude for **Dr. BENDAHMANE Fawzi** for his continuous support and without whom i would have bee able to work on this project.

My gratitude also goes to the members of the jury **Pr. BEKHECHI Smaine**, **Pr. MERAD Abdelkrim** and **Dr. BRAHMI Nabil** for accepting to examine my work and participate with their propositions.

I would like to thank my family, my parents especially my brother **MOUALEK Djaloul Youcef** for the help and advises he provided me with from the start to the finish of this project.

Dedications

I would like to direct a huge acknowledgement to my friends and colleagues of the **Scorpio Club** (**Ayoub, Zouhir, Zaki and Nesrine**) for teaching me so much and being by my side. The group of amazing people called **Ouloulou** and my dear friends (**Mounia, Sahra, Imene, lilya, Walid, Adil**) for their presence and support.

GGEZ

Contents

List of Figures

General Introduction

One of the biggest and most common problems we encounter in applied mathematics and the computational field is the optimization of the studied function, or how to minimize and maximize a function.
One thing that is useful to know is that "Optimization" comes from the same root as "optimal", which means best. When we optimize something, we are "making it best". The objective function, $f(x)$, which is the output you're trying to maximize or minimize. and $x_1, x_2, ..., x_n$ are the variables of the said function. Mathematical Optimization is a branch of applied mathematics which is useful in many different fields : Engeneering, Mechanics, Finance and Networks to name a few.

1 Problematic

For several years, many optimization methods have been developed by the scientific community, In this work we will be looking into particular optimization methods called mataheuristics. These methods use the power of nature to solve more or less complex problems.

2 Goals

Firstly we have to discuss some important related notions (Stochastic process, general definition and classification of the algorithms) in order to understand the operating of metaheuristics methods, then we will be applying our version of the said methods on generic functions and look into the results and study their efficiency.

3 General Structure

1. **Chapter 1** : We will be looking into general notions such as stochastic processes,definition and classification of the algorithms in order to understand the operating of metaheuristics. and introduce some of the famous algorithms

2. **Chapter 2** : In this part we will make in-depth study of metaheuristics algorithms as well as some of the benchmark function that we will be applying the said algorithms into.

3. **Chapter 3** : This chapter will be dedicated to the applications of our versions of the metaheuristics into the selected functions and the study of the results,we will also be adding a comparative study with other publications.

Chapter 1

The basics of Metaheuristics

1 Introduction

The term metaheuristic comes from the greek "meta" (beyond) and heuriskein (to find). There is no clear consensus on the exact definition of heuristics and metaheuristics, but we can adapt the following :

- A heuristics is a solving specialized technique for a problem. it does not guarantee the quality of the point obtained.
- A metaheuristic is a generic heuristic that must be adapted to each problem

In this chapter we will be focusing on the definitions and basic notions needed to understand the nature and the operating of some metaheuristics.

2 Stochastic Process

The best visualisation we can have of a stochastic process is when we think of the collision of smoke particles with each other[1], these said collisions are unpredictable, random and referred to as Brownian Motion[2].

Interest rate is a variable that changes its value over time. It is not easy to predict its movements. There are two ways to classify a stochastic process[1]:

- **Discrete:** When changes in value of a variable are at fixed points in time. Only certain values can be chosen for a discrete variable.
- **Continuous:** When changes in value of a variable are continuous. Value of a continuous variable can take any value within a certain range.

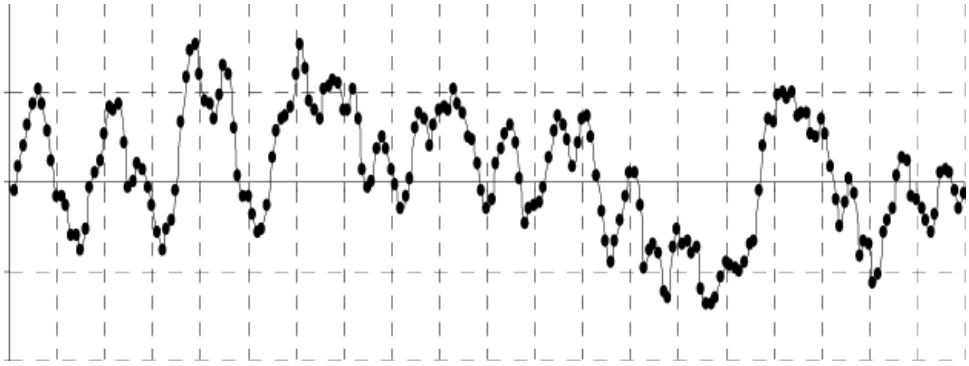


Figure 1.1: Example of a stochastic process

2.1 Markov Stochastic Process

In the study of time series data, information from the past can be used to predict future values. On the other hand, Markov processes are those where the past is irrelevant. A variable can only be predicted based on its current value. To predict future values of a variable, probability distributions must be used. As an example, the variable might follow a normal or log-normal probability distribution[1].

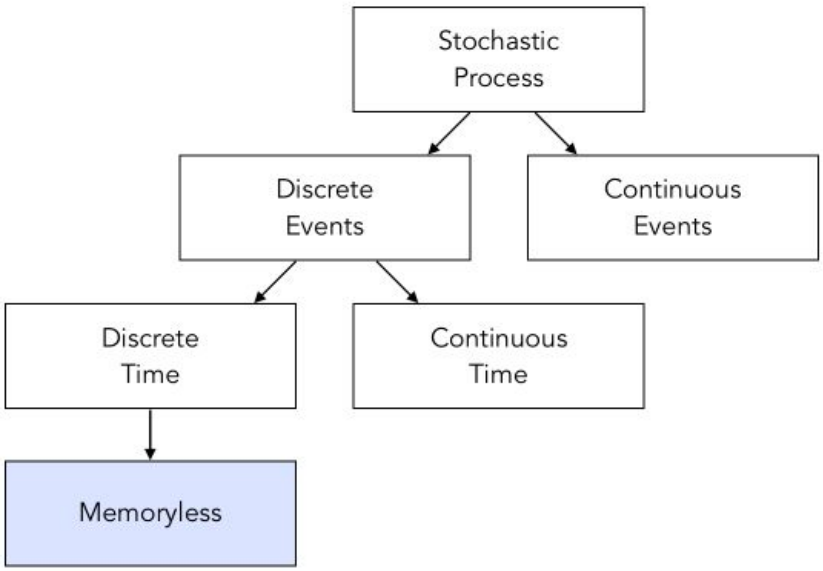


Figure 1.2: Discrete Time Markov Chain are time and event discrete stochastic process. Markov Chains rely on the Markov Property that there is a limited dependence within the process

2.2 Wiener Process

Markov stochastic process can also have a normal distribution with a mean change of 0 and variance rate of 1. This is known as Wiener process. It is a specialised form of Markov Stochastic Process.

Therefore Wiener process is where a normally distributed variable is evolved. Variable changes across two different time steps are not dependent on each other[3].

A variable follows Wiener process if following two conditions are met: Variable Change = Normal Distribution Number(mean=0, variance=1) * Sqrt(Change In Time)[1]

2.3 Gaussian Process

Gaussian process regression (GPR) models are nonparametric kernel-based probabilistic models with a finite collection of random variables with a multivariate distribution. Every linear combination is evenly distributed[4]. it is based on the notion of the Gaussian distribution to be an infinite-dimensional generalization of multivariate normal distributions. Gaussian processes are utilized in statistical modeling, regression to multiple target values, and analyzing mapping in higher dimensions[2].

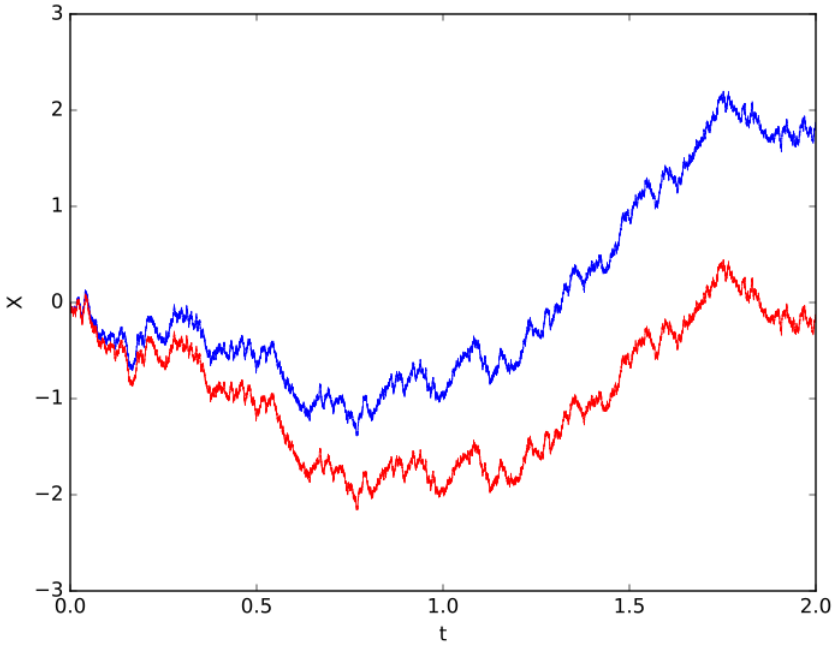


Figure 1.3: Example of Wiener process 2D

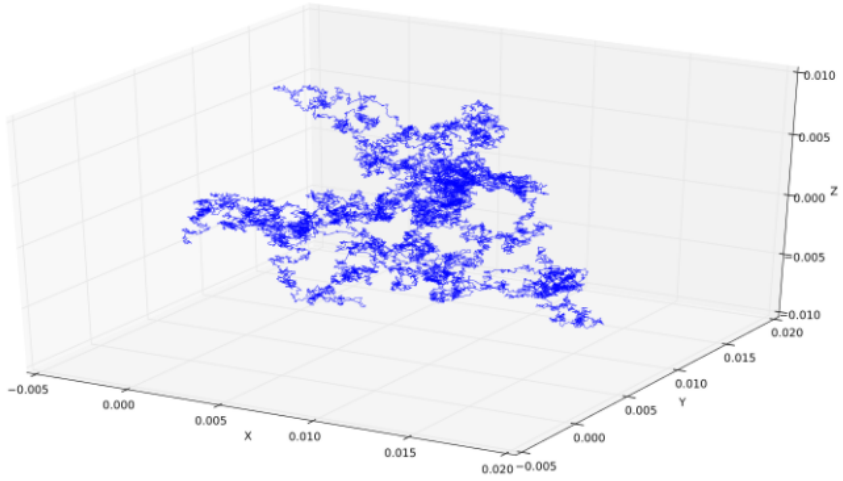


Figure 1.4: Example of Wiener process 3D

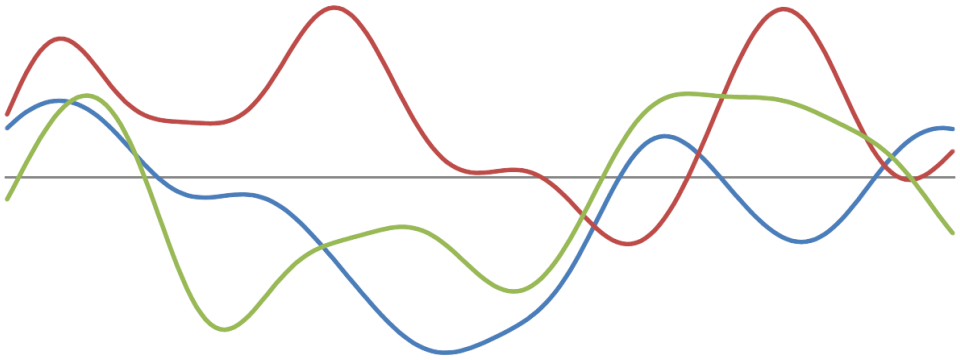


Figure 1.5: Example of Gaussian process 2D

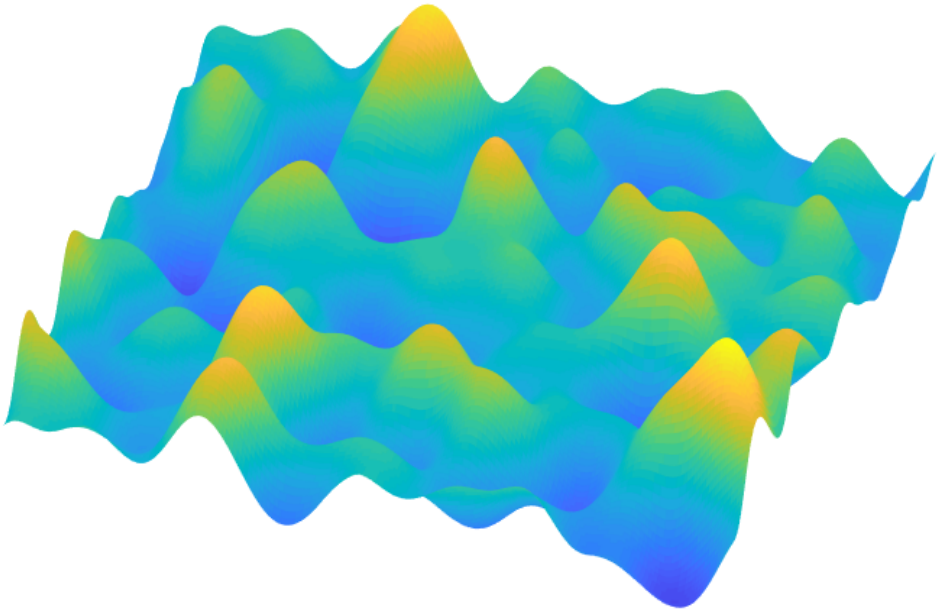


Figure 1.6: Example of Gaussian process 3D

2.4 Poisson Process

A Poisson Process is a model for a series of discrete event where the average time between events is known, but the exact timing of events is random[5]. The arrival of an event is independent of the event before (waiting time between events is memoryless).

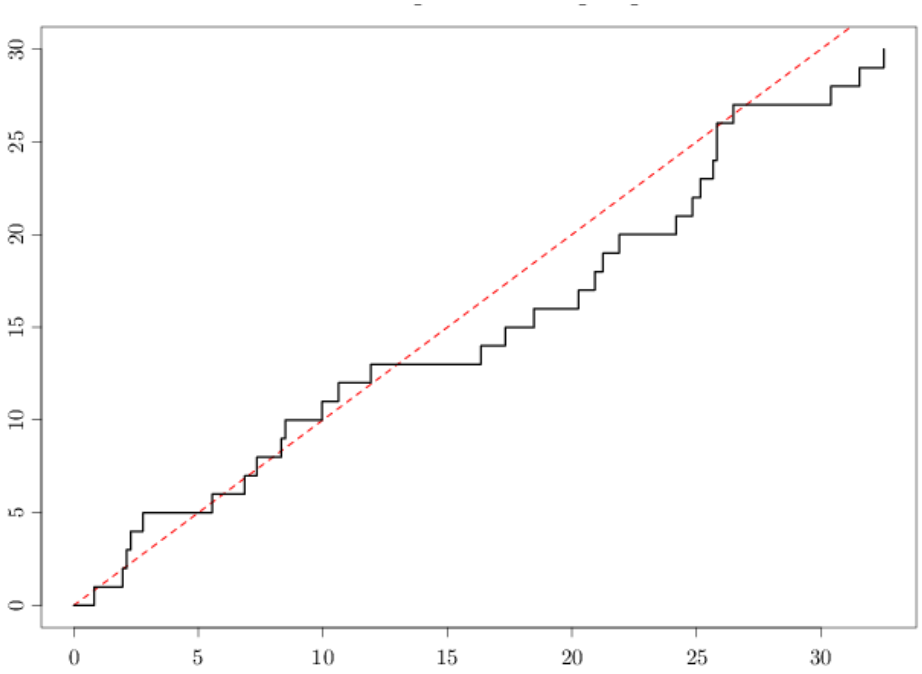


Figure 1.7: Poisson distribution sample path

3 Stochastic optimization

Stochastic optimization is a large topic, and it has been applied across a variety of fields. It includes the study of how non-deterministic or probabilistic data can be incorporated into an objective function to minimize or maximize it. In some cases, random variables representing uncertain parameters can be included in the formulation[6].

3.1 Methods for stochastic functions

Random noise can arise in several different ways. For example, when modeling physical systems, the true model may be unknown and there is uncertainty about the system parameters[7]. Or one may not know exactly what model to use because of incomplete knowledge of the problem at hand.

In addition, random errors are often present in measurements made under practical conditions. Some of those methods include :

- stochastic approximation (SA)
- stochastic gradient descent
- finite-difference SA
- simultaneous perturbation SA
- scenario optimization

3.2 Randomized search methods

While the exact data set consists of precise measurements, some methods introduce randomness into the search-process to accelerate progress. Such randomness can also make the method less sensitive to modeling errors. Further, the injected randomness may enable the method to escape a local optimum and eventually approach a global optimum. Some of these methods include :

- simulated annealing
- random search
- swarm algorithms
- evolutionary algorithms
- stochastic tunneling
- quantum annealing

These methods are the ones knows as **Metaheuristics**

4 Metaheuristics

4.1 Definition

In computer science and mathematical optimization, a metaheuristic is a high-level problem-independent algorithmic framework, most of the time inspired by natural phenomenons developed to solve complex optimization problems[8]. since a few decades now, metaheuristics are emerging as successful alternatives to more classical approaches also for solving optimization problems that include in their mathematical formulation uncertain, stochastic, and dynamic information[9].

4.2 Properties of the metaheuristics

When we compare metaheuristics to optimization algorithms and iterative methods, metaheuristics do not guarantee in the results that an exact solution (global optimum) can be found on some class of problems[8].

Although they can often find good solutions with less computational effort than optimization algorithms, iterative methods, or simple heuristics.

For that reason they are considered to be very efficient approaches for optimization problems. The implementation in a lot of metaheuristics algorithms of some form of stochastic optimization is very current,it is so that the solution found is dependent on the set of random variables generated.

Metaheuristics are especially used and shows great result when applied to combinatorial optimization in which an optimal solution is sought over a discrete search-space[9].

Most Metaheuristics are characterised as the following :

- Metaheuristics are strategies that guide the search process.
- The goal is to efficiently explore the search space in order to find near–optimal solutions.
- Techniques which constitute metaheuristic algorithms range from simple local search procedures to complex learning processes.
- Metaheuristic algorithms are approximate and usually non-deterministic.
- Metaheuristics are not problem-specific.

4.3 Classification of the metaheuristics

There are many different types of metaheuristics algorithms, and they can be classified in many ways[9].

Nature-inspired vs. non-nature inspired

One way is by the algorithm's origin, such as nature-inspired algorithms and non nature-inspired ones. Nature-inspired algorithms include genetic algorithms and ant algorithms, while Tabu Search and iterated local search are examples of non nature-inspired algorithms .

Population-based vs. single point search

Metaheuristics can be classified based on the number of solutions they work on at once: Population-based algorithms or single-solution algorithms(trajectory methods).

Dynamic vs. static objective function

Some metaheuristics can be classified based on how they use the objective function. Some algorithms keep the objective function in its original form, while others modify it during search.

One vs. various neighborhood structures :

In most metaheuristic algorithms, the fitness landscape topology does not change in the course of the algorithm. Other metaheuristics use a set of neighborhood structures to swap between different fitness landscapes.

Memory usage vs. memory-less methods

Very important to classify metaheuristics is how they use the search history. If they do not use memory, then the algorithms are considered "memoryless"or "Markovian". And if they do use memory, then they are called "memory-based".

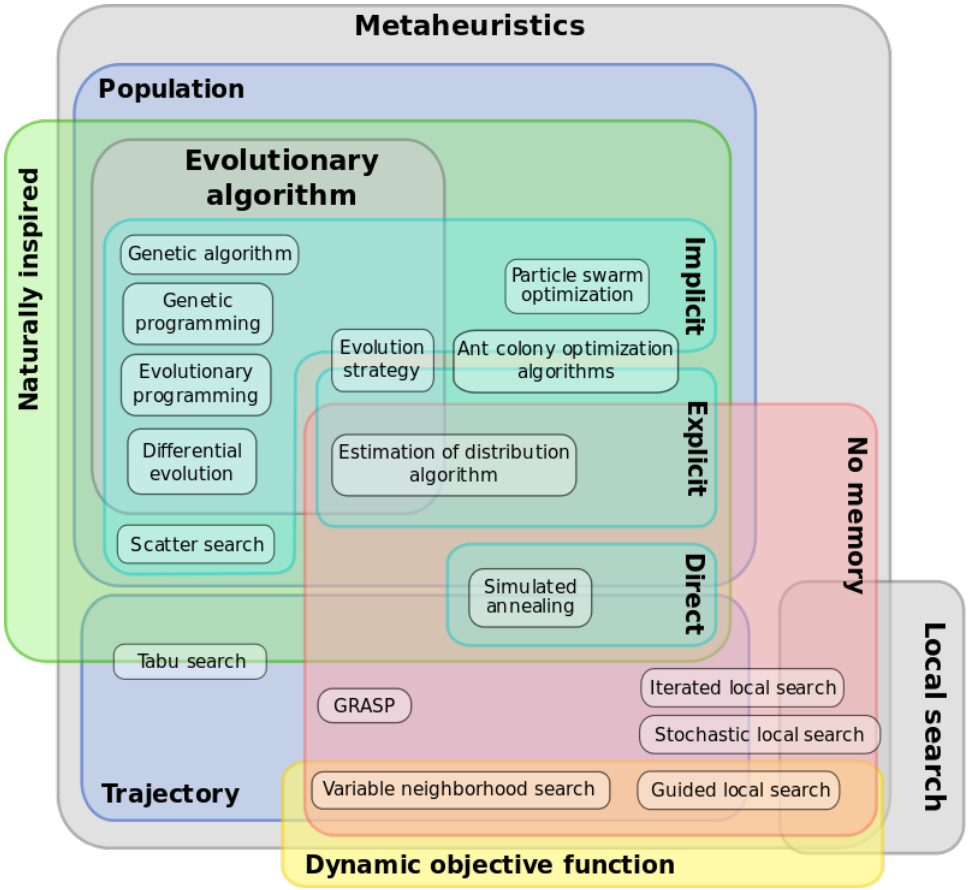


Figure 1.8: Classification of Metaheuristics

5 Nature Based metaheuristics

As we already know, the large majority of metaheuristics are Nature based. There are two main paradigms for the biological metaphores : evolutionary and swarm

- **Evolutionary Algorithms (EAs)** : simulate the biological progression of evolution at the cellular level employing selection
- **Swarm intelligence (SI)** : mimics the collective behavior of agents in a community, such as birds and insects.

5.1 Particule Swarm Algorithm

Particle swarm optimization, or PSO, was developed by Kennedy and Eberhart in 1995 and has become one of the most widely used swarm-intelligence-based algorithms due to its simplicity and flexibility[10].

Rather than use the mutation/crossover or pheromone, it uses real-number randomness and global communication among the swarm particles. Therefore, it is also easier to implement because there is no encoding or decoding of the parameters into binary strings as with those in genetic algorithms where real-number strings can also be used.

5.2 Differential Evolution

Differential Evolution (DE) is a vector-based meta-heuristic algorithm, which has some similarity to pattern search and genetic algorithms due to its use of crossover and mutation[11]. In fact, DE can be considered as a further development to genetic algorithms with explicit updating equations, which make it possible to do some theoretical analysis.

DE is a stochastic search algorithm with the self-organizing tendency and does not use the information of derivatives. Thus, it is a population-based, derivative-free method[12].

5.3 Artificial Bee Colony

An Artificial Bee Colony (ABC) is one of the most recently defined algorithms by Dervish Karaboga under the larger umbrella of swarm intelligence, motivated by the intelligent behavior of honey bees, who aim to discover food sources with progressively higher amounts of nectar[13].

5.4 Genetic Algorithm

The genetic algorithm is a specific algorithm in the family of evolutionary algorithms. Each algorithm works on the same premise of evolution but have small "tweaks" in the different parts of the lifecycle to cater for different problems[14]. Genetic algorithms are used to evaluate large search spaces for a good solution. It's important to note that a genetic algorithm isn't guaranteed to find the absolute best solution. It attempts to find the global best while avoiding local best solutions.[12]

6 Physics Based Metaheuristics

Despite the Nature dominance in the metaheuristics research There have been proposed many physics based algorithms afterwards. In fact, the number of the proposed physics based metaheuristics algorithms is not less than that of algorithms based on biology.

6.1 Gravitational Search Algorithm

GSA is inspired from Newton's laws of gravity and motion. In this algorithm, every agent is considered object. These objects' performances are measured by their masses, it is expected that at the end of the GSA run, position of the object with the heaviest mass will show the global solution[15].

6.2 Particle Collision Algorithm

PCA was inspired by nuclear collision reactions, especially scattering and absorption. The structure of PCA resembles a SA structure but it does not rely on user-defined parameters and it does not have cooling schedule[15].

6.3 Simulated Annealing

SA derived its name from the physical annealing process. Thus, like annealing initial state the algorithm is lenient and could move to a worse solution. At each iteration, the algorithm becomes more stringent for getting a better solution at each step[12].

7 Conclusion

In this chapter we have covered the basics as well as the definitions necessary for understanding metaheuristics as well as their classification and properties. we have also talked about some of the most famous algorithms that we will be looking more in the upcoming chapter.

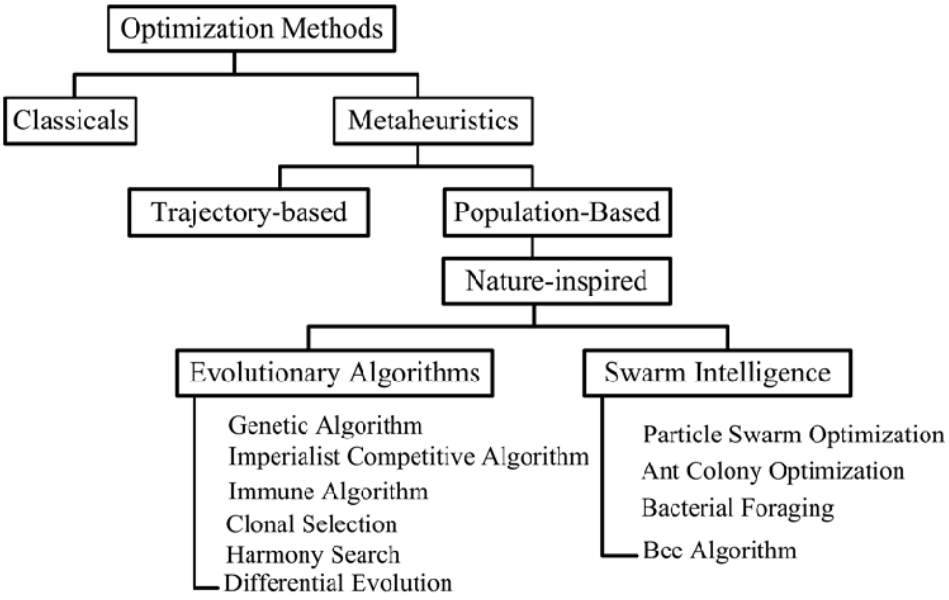


Figure 1.9: Classification of nature based metaheuristics

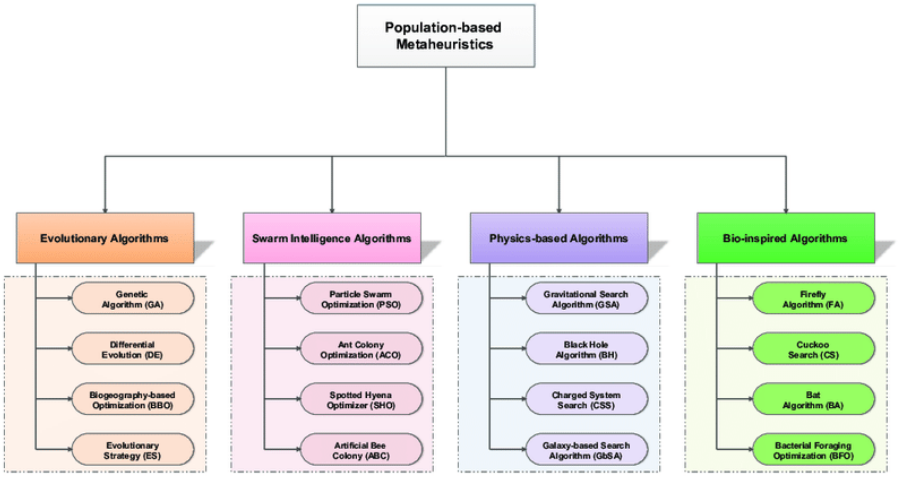


Figure 1.10: New classification of metaheuristics

Chapter 2

Algorithms and benchmark study

1 Introduction:

In this chapter we are going to look into the algorithms we previously mentioned which are the **PSO**,**DE** and **GSA**. we will also introduce the benchmark function that we will be using in our calculations later on.

2 The Algorithms:

2.1 Particule Swarm Optimization (PSO)

Optimizing parameters of multivariate systems is a general problem in computational biology. One of the many methods developed for parameter optimization is Particle Swarm Optimization (PSO), which was introduced by Kennedy and Eberhart in 1995 .

Emerging from simulations of dynamic systems such as bird flocks and fish swarms, the original algorithm is grounded on a stochastic search in multimodal search space. The idea of PSO is to have a swarm of particles "flying" through a multidimensional search space, looking for the global optimum.

By exchanging information the particles can influence each others' movements. Each particle retains an individual (or "cognitive") memory of the best position it has visited, as well as a global (or "social") memory of the best position visited by all particles in the swarm.

A particle calculates its next position based on a combination of its last movement vector, the individual and global memories, and a random component[10].

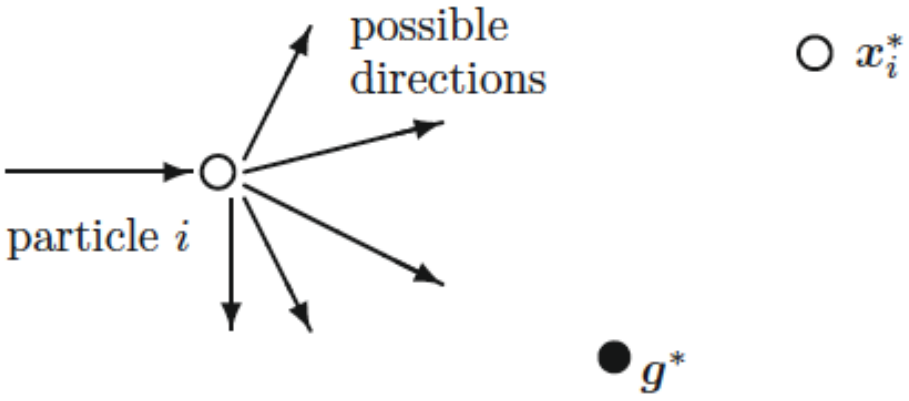


Figure 2.1: The movement of particles

The movement of particles is schematically represented in the figure above, where $x_i(t)$: the best current for particle i, and $gminf(x_i)$for $(i = 1, 2, .., n)$: is the best global current at t.

An advantage of PSO is its ability to handle optimization problems with multiple local optima reasonably well and its simplicity of implementation – especially in comparison to related strategies like genetic algorithms (GA).

In the field of cheminformatics, PSO has successfully been applied to Quantitative Structure-Activity Relationship (QSAR) modeling, including k-nearest neighbor and kernel regression, minimum spanning tree for piecewise modeling, partial least squares modeling, and neural network training [10].

Ever since its capability to solve global optimization problems was discovered, the PSO paradigm has been developed further and improved and several variations of the original algorithm have been proposed. These include the Constriction type PSO (CPSO) amongst various others .

For this method each particle is initialized at a random position in search space. The position of particle i is given by the vector $x_i = (x_{i1}, x_{i2}, ..., x_{iD})$ where D is the dimensionality of the problem. Its velocity is given by the vector $v_i = (v_{i1}, v_{i2}, ..., v_{iD})$.

Two kinds of memory were implemented that influence the movement of the particles: In the cognitive memory $p_i = (p_{i1}, p_{i2}, ..., p_{iD})$ the best previous position visited by each individual particle i is stored.
The vector $p_{best} = (p_{best1}, p_{best2}, ..., p_{bestD})$, also called "social memory", contains the position of the best point in search space visited by all swarm particles so far.
In each epoch the particle velocities are updated according to equation:

$$v_i(t+1) = wv_i(t) + n_1r_1(p_i - x_i(t)) + n_2r_2(p_best - x_i(t)) \tag{1}$$

Where w is the inertia weight, a weighting factor for the velocity, n_1 and n_2 are positive constants called "cognitive" and "social" parameter weighting the influence of the two different swarm memories, and r_1 and r_2 are random numbers between 0 and 1.

The PSO algorithms has 3 main parameters :

- Population size
- N number of iterations
- K neighbourhood

2.2 PSO Algorithm and Pseudocode

The essential steps of the PSO can be summarized as the pseudocode below:

Particle Swarm Optimization

Objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, ..., x_d)^T$
Initialize locations $\boldsymbol{x}_i$ and velocity $\boldsymbol{v}_i$ of n particles.
Find $\boldsymbol{g}^*$ from $\min\{f(\boldsymbol{x}_1), ..., f(\boldsymbol{x}_n)\}$ (at $t = 0$)
while (criterion)
  for loop over all n particles and all d dimensions
  Generate new velocity $\boldsymbol{v}_i^{t+1}$
  Calculate new locations $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1}$
  Evaluate objective functions at new locations $\boldsymbol{x}_i^{t+1}$
  Find the current best for each particle $\boldsymbol{x}_i^*$
  end for
  Find the current global best $\boldsymbol{g}^*$
  Update $t = t + 1$ (pseudo time or iteration counter)
end while
Output the final results $\boldsymbol{x}_i^*$ and $\boldsymbol{g}^*$

The PSO algorithm follows the scheme below :

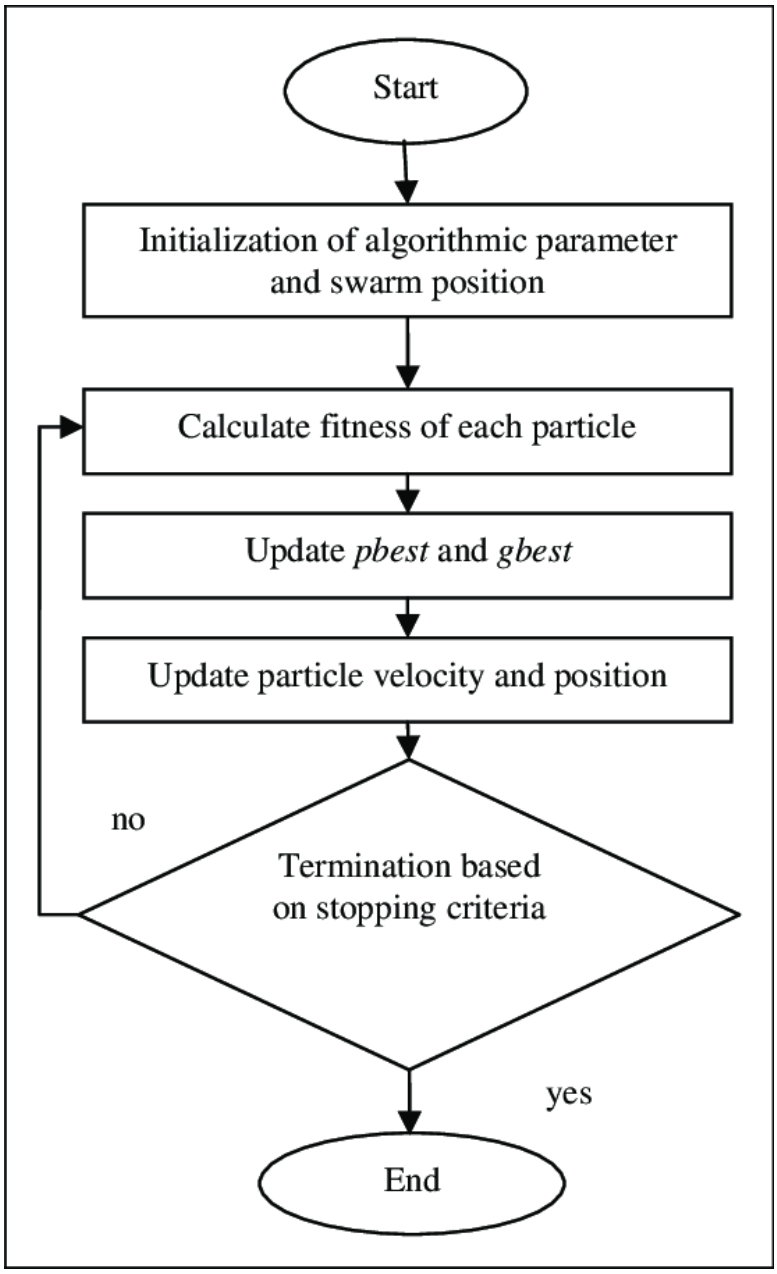


Figure 2.2: PSO algorithm

2.3 Differential Evolution (DE)

Differential Evolution is a stochastic global search optimization algorithm. It is a type of evolutionary algorithm and is related to other evolutionary algorithms such as the genetic algorithm. Unlike the genetic algorithm that represents candidate solutions using sequences of bits.

Differential Evolution is designed to work with multi-dimensional real-valued candidate solutions for continuous objective functions[16].

The algorithm does not make use of gradient information in the search, and as such, is well suited to non-differential nonlinear objective functions.

The algorithm works by maintaining a population of candidate solutions represented as real-valued vectors. New candidate solutions are created by making modified versions of existing solutions that then replace a large portion of the population each iteration of the algorithm[11].

New candidate solutions are created using a "strategy" that involves selecting a base solution to which a mutation is added, and other candidate solutions from the population from which the amount and type of mutation is calculated, called a difference vector. For example, a strategy may select a best candidate solution as the base and random solutions for the difference vector in the mutation[16].

Differential evolution consists of three main steps: mutation, crossover, and selection.

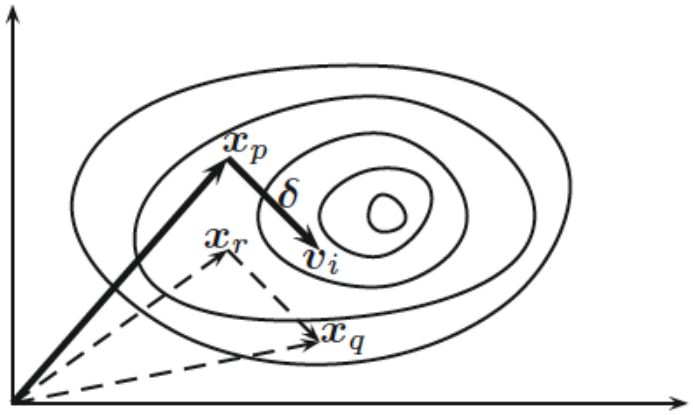


Figure 2.3: DE mutation scheme

Mutation is carried out by the mutation scheme.
For each vector (x_i): at any time or generation t, we first randomly choose three distinct vectors : (x_p, x_q, and x_r at t as seen by the figure above, and then we generate a so-called donor vector by the mutation scheme

$$\boldsymbol{v}_i^{t+1} = \boldsymbol{x}_p^t + F(\boldsymbol{x}_q^t - \boldsymbol{x}_r^t),$$

(2)

Where F in [0,2] is a parameter (differential weight). In principle, F in [0,1] is more efficient and stable.

We can see that the perturbation

$$\boldsymbol{\delta} = F(\boldsymbol{x}_q - \boldsymbol{x}_r)$$

(3)

To the vector x_p is used to generate a donor vector v_i, and such perturbation is directed.The crossover is controlled by a crossover parameter C_r $[0, 1]$, controlling the rate or probability for crossover.
The actual crossover can be carried out in two ways: binomial and exponential.
The binomial scheme performs crossover on each of the d components or variables/parameters. By generating a uniformly distributed random number $r_i[0, 1]$, the j-th component of v_i is manipulated as

$$\boldsymbol{u}_{j,i}^{t+1} = \begin{cases} \boldsymbol{v}_{j,i} & \text{if } r_i \le C_r, \\ \boldsymbol{x}_{j,i}^t & \text{otherwise,} \end{cases} \quad j = 1, 2, \ldots, d.$$

(4)

This way, it can be decided randomly whether to exchange each component with a donor vector or not. In the exponential scheme, a segment of the donor vector is selected, and this segment starts with a random integer k with a random length L, which can include many components. Mathematically, this is to choose $k[0, d1]$ and $L[1, d]$ randomly, and we have:

$$\boldsymbol{u}_{j,i}^{t+1} = \begin{cases} \boldsymbol{v}_{j,i}^t & \text{for } j = k, \ldots, k - L + 1 \in [1, d], \\ \boldsymbol{x}_{j,i}^t & \text{otherwise.} \end{cases}$$

(5)

Selection is essentially the same as that used in genetic algorithms.
We select the fittest and, for the minimization problem, the minimum objective value. Therefore, we have:

$$x_i^{t+1} = \begin{cases} u_i^{t+1} & \text{if } f(u_i^{t+1}) \leq f(x_i^t), \\ x_i^t & \text{otherwise.} \end{cases} \tag{6}$$

The DE algorithms has 3 main parameters :

- Population size
- N number of iterations
- Crossover probability

2.4 DE Algorithm and Pseudocode

The essential steps of the DE can be summarized as the pseudocode below:

Differential Evolution

Initialize the population $\boldsymbol{x}$ with randomly generated solutions
Set the weight $F \in [0, 2]$ and crossover probability $C_r \in [0, 1]$
while (stopping criterion)
  for $i = 1$ to n,
  For each $\boldsymbol{x}_i$, randomly choose 3 distinct vectors $\boldsymbol{x}_p$, $\boldsymbol{x}_r$ and $\boldsymbol{x}_r$
  Generate a new vector $\boldsymbol{v}$
  Generate a random index $J_r \in \{1, 2, ..., d\}$ by permutation
  Generate a randomly distributed number $r_i \in [0, 1]$
  for $j = 1$ to d,
  For each parameter $\boldsymbol{v}_{j,i}$ (jth component of $\boldsymbol{v}_i$), update

$$u_{j,i}^{t+1} = \begin{cases} v_{j,i}^{t+1} & \text{if } r_i \leq C_r \text{ or } j = J_r \\ x_{j,i}^t & \text{if } r_i > C_r \text{ and } j \neq J_r, \end{cases}$$

  end
  Select and update the solution
  end
end
Post-process and output the best solution found

The DE algorithm follows the scheme below :

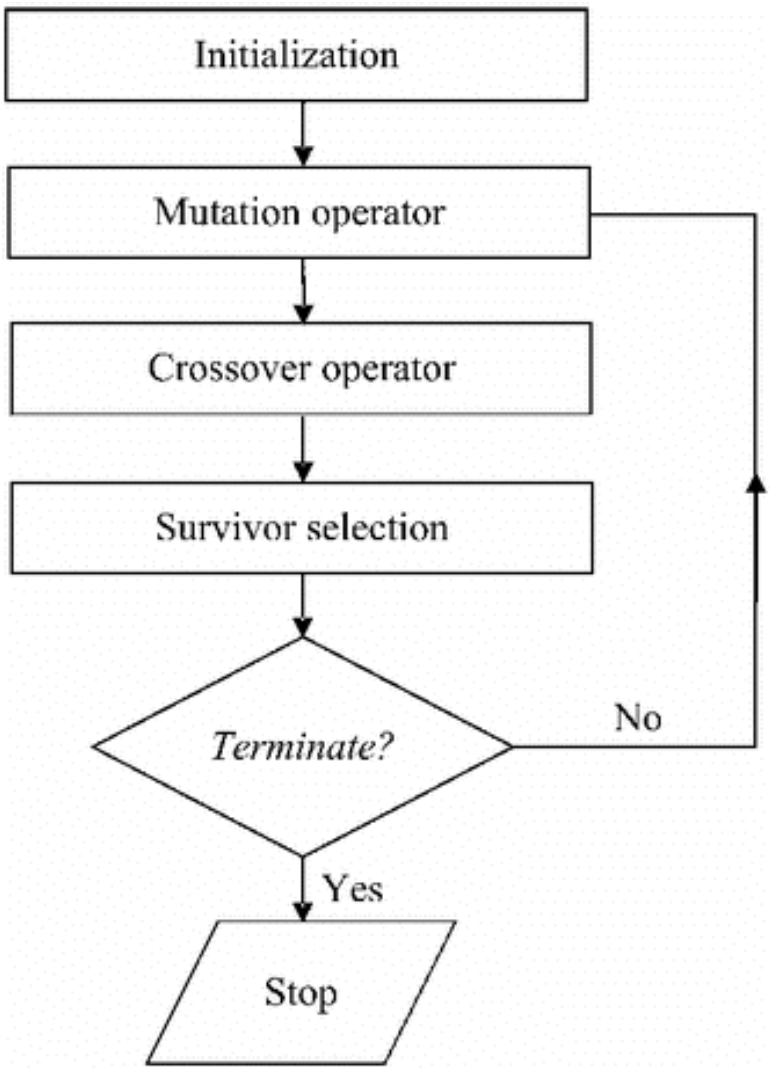


Figure 2.4: DE Algorithm

2.5 Gravitational Search Algorithm (GSA)

Gravitational search algorithm is a nature-inspired algorithm based on the mathematical modelling of the Newton's law of gravity and motion[17].
In a decade, researchers have presented many variants of gravitational search algorithm by modifying its parameters to efficiently solve complex optimization problems.

GSA is one of the famous physics inspired algorithms which has shown promising performance on numerous real-world problems.
The Gravitational Search Algorithm (GSA), proposed by Rashedi et al., is a new algorithm based on the law of gravity. The agents in GSA are considered as objects with masses. Agents attract each other by the gravity force. The greater the quality, the stronger the gravity. Therefore, the location of the agent with the largest mass is the optimal solution[18].

Suppose that there are N agents with d-dimension. The position of the$i$-th agent is

$$X_i = \left(x_i^1, x_i^2, \ldots, x_i^d\right) \quad (i = 1, 2, \ldots, N).$$

(7)

At the t-th time, the force acting on the i-th agent from the j-th agent is defined as follows:

$$F_{ij}^d = G(t) \frac{M_i(t) M_j(t)}{R_{ij}(t) + \varepsilon} \left(x_j^d(t) - x_i^d(t)\right)$$

(8)

where$Mi(t)$ and $Mj(t)$ are the masses of the i-th agent and the j-th agent, respectively, $G(t)$ is the gravitational constant at the t-th time, is a small constant, and Rij is the Euclidian distance between the i-th agent and the j-th agent.

At the t-th time, total force acting on the i-th agent is defined as follows:

$$F_i^d(t) = \sum_{j=1, j \neq i}^{N} \text{rand} \cdot F_{ij}^d(t)$$

(9)

where $rand$ is a uniform random variable in the interval [0,1].

According to the law of motion, the acceleration of the agent at the t-th time can be defined as follows:

$$a_i^d(t) = \frac{F_i^d}{M_i(t)}.$$

(10)

In each iteration process, velocity and position of the i-th agent are updated by the following two equations:

$$v_i^d(t+1) = \text{rand} \times v_i^d(t) + a_i^d(t),$$
$$x_i^d(t+1) = x_i^d(t) + v_i^d(t+1),$$

(11) and (12)

Where *rand* is a uniform random variable in the interval [0,1] and x_i and v_i are its current position and velocity, respectively.

GSA algorithms has 3 main parameters :

- Population size
- N number of iterations
- K neighbourhood

2.6 GSA Algorithm and Pseudocode

The GSA pseudocode follows the steps below :

Gravitational search algorithm (GSA)

Input: Let P search agents and a gravitational constant (G_0).
Output: An optimal solution.

Randomly initialize the swarm having P search agents;
while stopping condition does not meet **do**
Determine the fitness (f), G, and mass ($Mass$);
Determine $Kbest$ as:
$Kbest(t) = P \times \frac{final\_per + (1 - \frac{t}{maxit}) \times (100 - final\_per)}{100}$,
Determine acceleration (a) and velocity ($V1v$)
Updated the search agents as $S(t+1) = S(t) + V(t+1)$
end while

The GSA algorithm follows the scheme below :

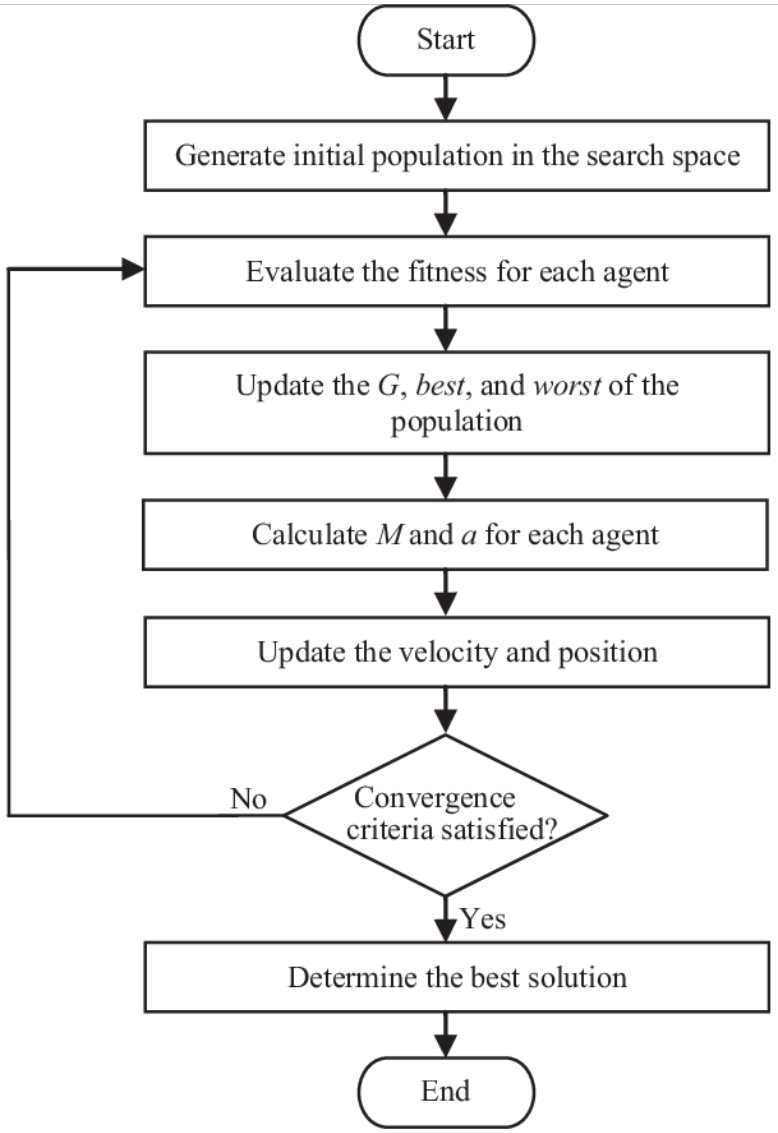


Figure 2.5: GSA Algorithm

3 The Benchmark Functions

Benchmark functions are The function, which can be used to test performance of any optimization approach and the related problem, such as constrained and unconstrained, continuous and discrete variables, and unimodal and multimodal problems[19]. In applied mathematics, these functions, known as artificial landscapes, are useful to evaluate characteristics of optimization algorithms, such as:

- Convergence rate.
- Precision
- Robustness
- General performance

In this work, we will use the Rastrigin, Ackley and Alpine functions.

3.1 Rastring function

The Rastrigin global optimization problem is a multimodal minimization problem defined as follows:

$$f_{Rastrigin}(\mathbf{x}) = 10n \sum_{i=1}^{n} \left[x_i^2 - 10\cos(2\pi x_i) \right] \tag{13}$$

Here, n represents the number of dimensions and $x_i \in$ [-5.12, 5.12] for i=1,...,n. Global optimum for this function is : $f(x_i) = 0$ for $x_i = 0$ for i=$1, ..., n$

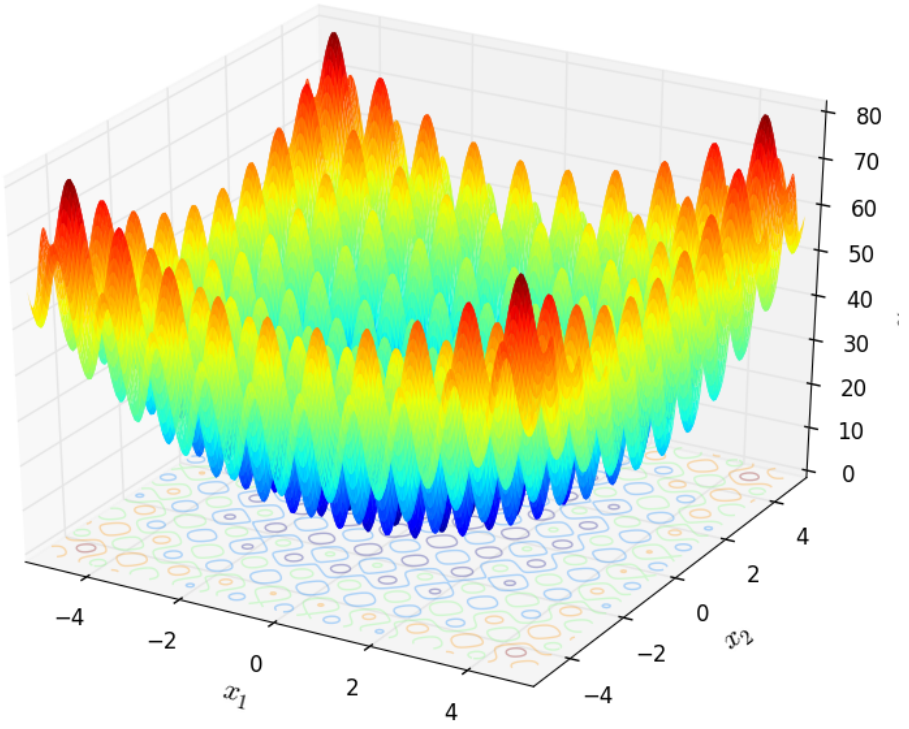


Figure 2.6: Two dimensional Rastring function

3.2 Arckley function

The Ackley global optimization problem is a multimodal minimization problem defined as follows:

$$f_{\text{Ackley}}(\mathbf{x}) = -20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e \tag{14}$$

Here, n represents the number of dimensions and $x_i \in$ [-32, 32] for i=1,...,n. Global optimum: $f(x_i) = 0$ for $x_i = 0$ for $i = 1, ..., n$

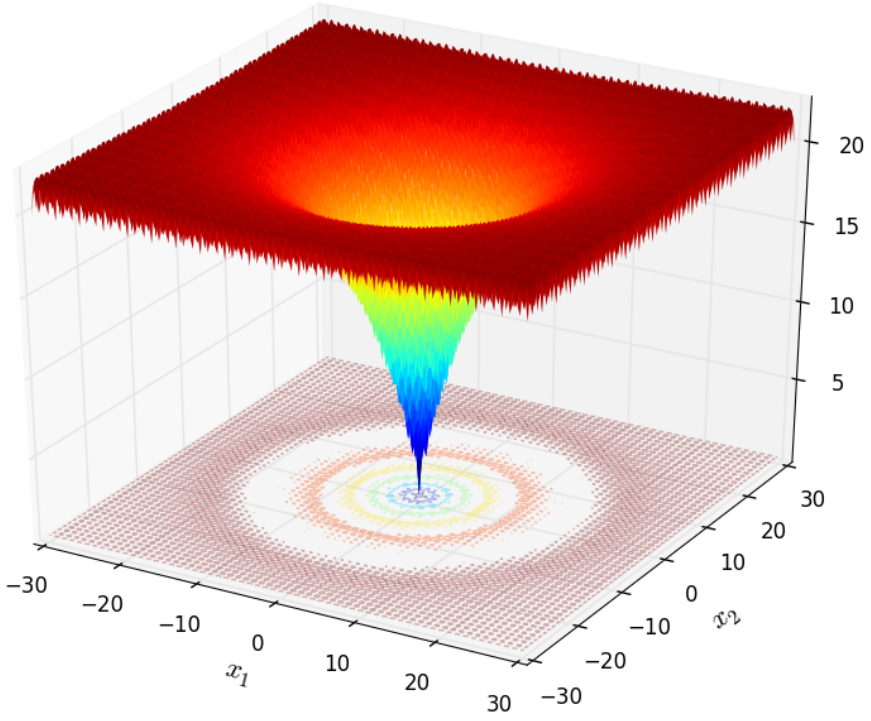


Figure 2.7: Two dimensional Arckley function

3.3 Alpine function

The Alpine global optimization problem. is a multimodal minimization problem defined as follows:

$$f_{\text{Alpine01}}(\mathbf{x}) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i| \tag{15}$$

The variable ϵ_i, $(i = 1, ..., n)$ is a random variable uniformly distributed in $[0, 1]$.

Here, n represents the number of dimensions and $x_i \in$ [-10, 10] for i=1,...,n. Global optimum: $f(x_i) = 0$ for $x_i = 0$ for $i = 1, ..., n$

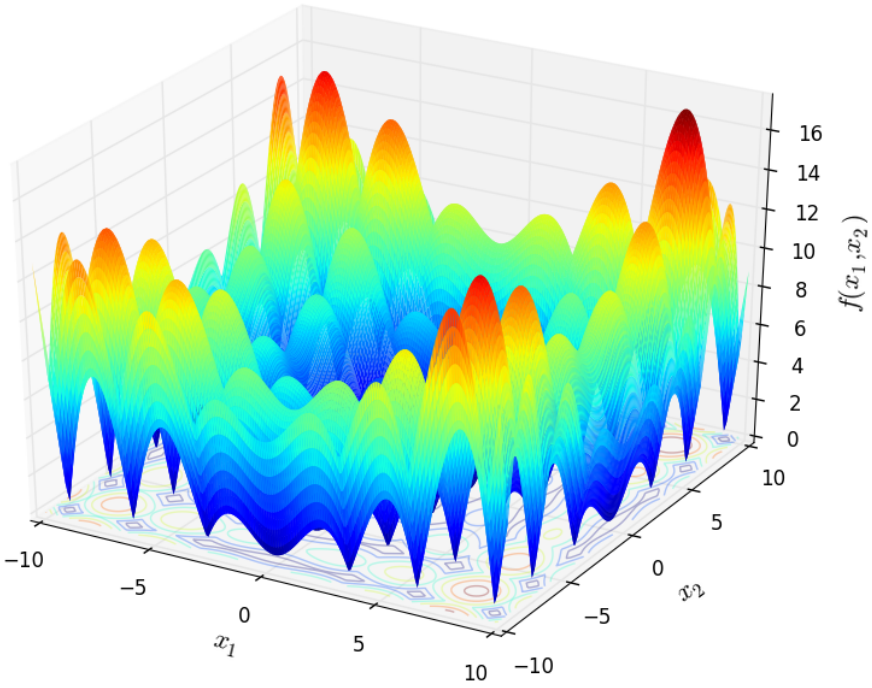


Figure 2.8: Two dimensional Alpine function

4 Constrained Benchmark functions

Most optimization problems have some sort of constraint, which limits the shape of the search space. Over the last few years, a wide variety of metaheuristics have been designed and applied to solve constrained optimization problems.
Metaheuristics are naturally unconstrained techniques for optimization. They therefore require an additional mechanism to incorporate constraints into their fitness function[20].

4.1 Definitions

Optimization problems with constraints we are using are all transformed into the following format:

$$f(\vec{x}), \vec{x} = [x_1, x_2, \ldots, x_n] \tag{16}$$

subject to the following constraints :

$$g_i(\vec{x}) \leq 0, i = 1, \ldots, q \tag{17}$$

A solution x is regarded as feasible if $g_i(\ x) \leq 0, for i = 1, ..., q$.

In our case we will be looking into two **Constrained Real-Parameter functions** which we will be naming $G1$ and $G2$ respectively.

- G_1 : is a quadratic function that goes as follow :

$$f(\vec{x}) = 5.3578547x_3^2 + 0.8356891x_1x_5 + 37.293239x_1 - 40792.141 \tag{18}$$

Subject to :

$$
\begin{aligned}
&g_1(\vec{x}) = 85.334407 + 0.0056858x_2x_5 + 0.0006262x_1x_4 - 0.0022053x_3x_5 - 92 \leq 0\\
&g_2(\vec{x}) = -85.334407 - 0.0056858x_2x_5 - 0.0006262x_1x_4 + 0.0022053x_3x_5 \leq 0\\
&g_3(\vec{x}) = 80.51249 + 0.0071317x_2x_5 + 0.0029955x_1x_2 + 0.0021813x_3^2 - 110 \leq 0\\
&g_4(\vec{x}) = -80.51249 - 0.0071317x_2x_5 - 0.0029955x_1x_2 - 0.0021813x_3^2 + 90 \leq 0\\
&g_5(\vec{x}) = 9.300961 + 0.0047026x_3x_5 + 0.0012547x_1x_3 + 0.0019085x_3x_4 - 25 \leq 0\\
&g_6(\vec{x}) = -9.300961 - 0.0047026x_3x_5 - 0.0012547x_1x_3 - 0.0019085x_3x_4 + 20 \leq 0
\end{aligned}
\tag{19}
$$

Where $78 \leq x_1 \leq 102, 33 \leq x_2 \leq 45, 27 \leq x_i \leq 45 (i = 3, 4, 5)$.
The optimum solution is x = 78, 33, 29.9952560256815985, 45, 36.7758129057882073) where $f(x^*)$ = -30665.53867178332. Two constraints are active (g_1 and g_6).

- G_2 : is a nonlinear function that goes as follow :

$$
f(\vec{x}) = -\left| \frac{\sum_{i=1}^{n} \cos^4(x_i) - 2\prod_{i=1}^{n} \cos^2(x_i)}{\sqrt{\sum_{i=1}^{n} ix_i^2}} \right|
\tag{20}
$$

Subject to :

$$
\begin{aligned}
&g_1(\vec{x}) = 0.75 - \prod_{i=1}^{n} x_i \leq 0\\
&g_2(\vec{x}) = \sum_{i=1}^{n} x_i - 7.5n \leq 0
\end{aligned}
\tag{21}
$$

Where $n = 20$ and $0 \leq x_i \leq 10$ $(i = 1, ..., n)$.
The global minimum x^* = (3.16246061572185, 3.12833142812967, 3.09479212988791, 3.06145059523469, 3.02792915885555, 2.99382606701730,2.95866871765285, 2.92184227312450, 0.49482511456933, 0.48835711005490, 0.48231642711865, 0.47664475092742, 0.47129550835493, 0.46623099264167, 0.46142004984199, 0.45683664767217, 0.45245876903267, 0.44826762241853, 0.44424700958760, 0.44038285956317),
The best we found is $f(x^*)$ = -0.80361910412559 (which, to the best of our knowledge, is better than any reported value), constraint g1 is close to being active.

As additional information for G_{01} and G_{02} respectively,

- n(1st column) is the number of decision variables,
- ρ (3rd column) is the estimated ratio between the feasible region and the search space,
- LI (4th column) is the number of linear inequality constraints,
- NI (5th column) the number of nonlinear inequality constraints,
- LE(6th column) is the number of linear equality constraints and
- NE (7th column)is the number of nonlinear equality constraints.
- a (8th column) is the number of active constraints at x.

n	Type of function	ρ	LI	NI	LE	NE	a
5	quadratic	52.1230%	0	6	0	0	2
20	nonlinear	99.9971%	0	2	0	0	1

5 Conclusion

In this chapter we saw the main algorithms based on biology, evolution and physics and explained how they worked, we also choose benchmark function with and without constraints to study that had a lot of common point with other function we encounter a lot in the field of computational physics.

Chapter 3

Applications and comparative study

1 Introduction

Now we will begin the practical part of the project, after looking into the metaheuristics algorithms in the previous chapter, we are now going to apply them onto the benchmark functions and study the results in this chapter.

2 Problematic studied

The optimization problem we can find in any mathematical problem in computational field, is finding the global optimum without having to undergo heavy calculus on multidimensional functions.

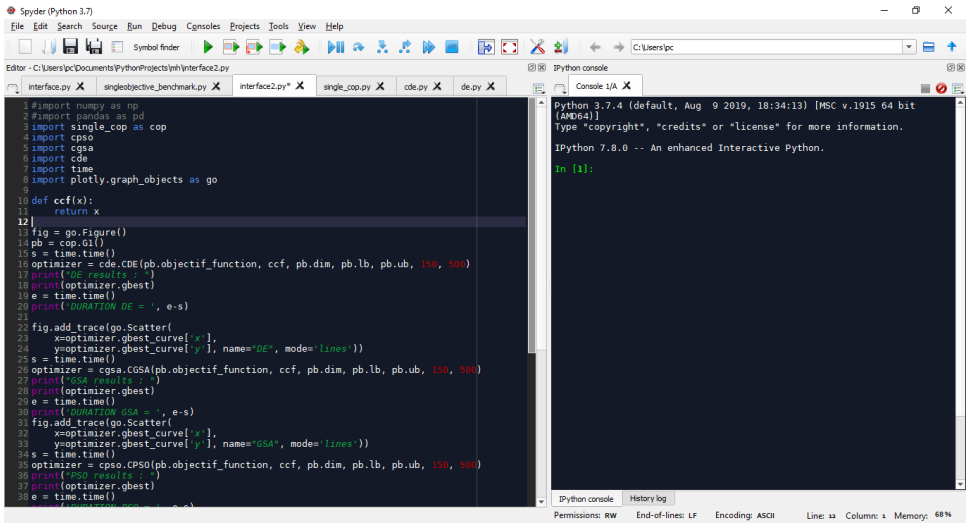
3 Process of this chapter

For this work we choose 3 metaheuristics and applied them on a 30 dimensional ($n = 30$) regular benchmark function, as well as two other Constrained Real-Parameter function with $n = 5$ and $n = 20$ respectively. After showing the results we will discuss and lay out observations for each one of them and compare our version of the algorithms to previous ones used in other publications.

4 Presentation of tools

4.1 The software : Spyder

Spyder is an open source cross-platform IDE and was built specifically for data science.
It integrates the essentials libraries for data science, such as NumPy, SciPy, Matplotlib and Pandas.
Features : Spyder contains features like a text editor with syntax highlighting, code completion and variable exploring, which you can edit its values using a Graphical User Interface (GUI). The image below shows the interface of **spyder** :



4.2 The Hardware

The experiments were all carried out on a computer which offers acceptable performance, the characteristics of which are as follows :

CPU : Intel(R) Core(TM) i5-6300U (2.50 GHz)
RAM : 8.00 GB
System type : 64-bit operating system, x64-based processor

5 Applications

5.1 Rastrigin test

We applied the PSO, GSA and DE algorithms all together on the 30 dimensional Rastrigin function (seen in the second chapter of this work) to find the global optimum, first with 500 then 10000 iterations, and got the following results :

500 iterations :

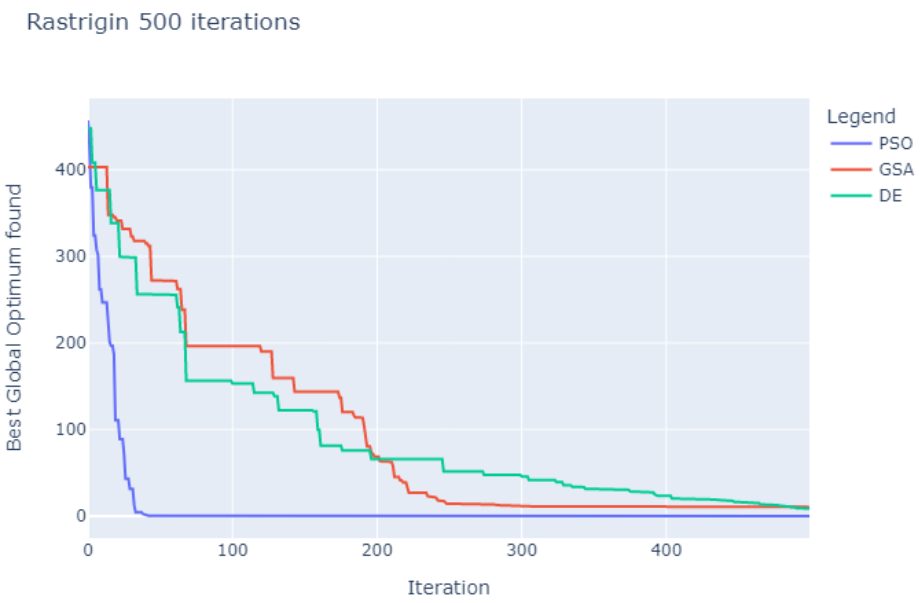


Figure 3.1: The evolution curve of the optimal solution compared to the number of iterations for the Rastrigin function at 500 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(0.), 'position': array([-1.87954296e-10, -7.56119388e-09, -5.89812919e-10, -1.85269997e-09,
       -1.18669897e-09,  2.02442124e-10,  1.45514346e-10,  2.90853938e-10,
       -4.01509324e-09,  2.31752814e-11,  1.86330945e-09, -6.10075900e-10,
       -1.31142277e-09, -2.47721289e-10, -1.25182203e-09, -3.00830055e-10,
        3.65579872e-10,  1.90349961e-09, -3.94372117e-09,  6.31887395e-10,
        1.88781813e-10,  1.21066338e-10,  2.95918973e-09,  2.31832809e-09,
        2.97891678e-11, -4.95815281e-09, -3.27129400e-09, -3.12990556e-09,
       -4.37320693e-09,  5.01629082e-09]), 'index': array(34, dtype=int64)}
DURATION PSO =  0.47028684616088867
GSA results :
{'fitness': array(10.44570384), 'position': array([ 1.45242737e-02, -8.53615955e-04, -4.32057687e-03, -9.81039892e-03,
       -9.98570015e-01, -9.88602002e-01, -5.77803195e-04,  1.01194512e+00,
        1.18450860e-02, -8.16636060e-04,  3.58967312e-04, -1.11564025e-02,
       -3.22708865e-03, -2.44113414e-03,  9.92912002e-01, -4.16641973e-03,
       -2.79312214e-03,  9.90748424e-01, -9.99793880e-01, -2.77792546e-03,
        8.20206792e-03,  9.86304390e-01,  9.99292648e-01,  7.83095478e-03,
       -9.75080342e-01, -2.21962296e-02,  8.59440316e-04, -4.18374837e-03,
        9.89353670e-01, -1.77440036e-02]), 'index': array(0, dtype=int64)}
DURATION GSA =  0.3204624652862549
DE results :
{'fitness': array(3.50905102), 'position': array([ 1.35660656e-02,  1.05192313e-02,  1.43885345e-02, -1.80423731e-02,
        2.50576448e-02,  1.02946368e-02, -3.41897568e-03,  1.81655906e-02,
       -1.86871004e-02, -9.94622867e-01,  1.57349446e-02,  2.64468270e-02,
        3.96799402e-03,  1.36613187e-02,  7.17499515e-03, -1.06490832e-02,
       -2.54406310e-02, -3.82203752e-02, -1.10542364e-02,  1.33870976e-02,
        2.76898865e-03,  9.78885546e-01, -5.47575413e-03,  1.02256400e-02,
        1.73425734e-02, -7.49118330e-03, -5.19403411e-03, -1.91927403e-05,
        2.65683407e-02, -1.43881304e-02]), 'index': array(16, dtype=int64)}
DURATION DE =  1.4007771015167236
```

Observations

For a number of 500 iterations and $n = 30$, The PSO shows great results by being the only algorithm to find the optimal $f(x) = 0$ with values of $x_1, x_2...x_n$ close to the optimum values 0.
The optimum result for GSA was $f(x) = 10$ although with the best time performance, and DE results $f(x) = 3.5$with the longest time performance.

10000 iterations :

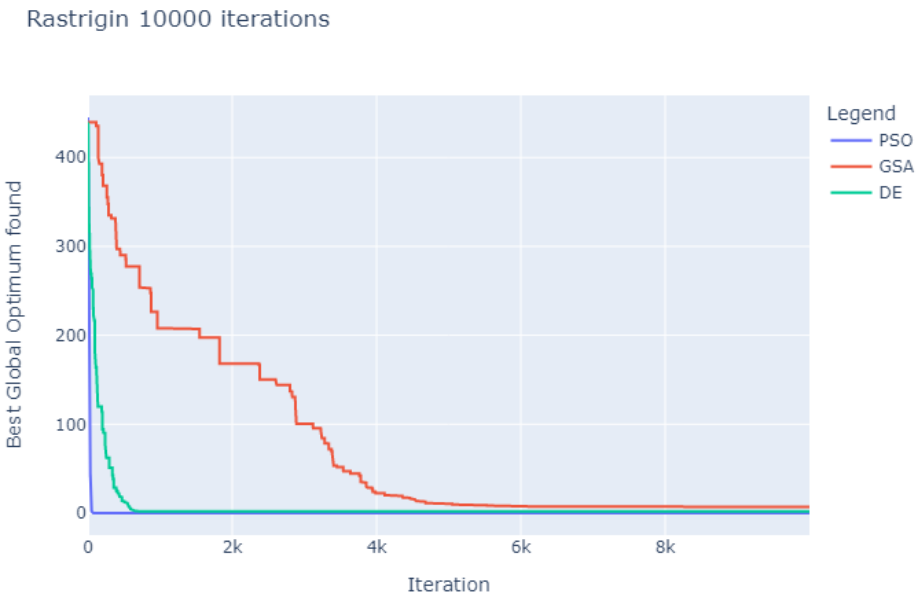


Figure 3.2: The evolution curve of the optimal solution compared to the number of iterations for the Rastrigin function at 10k iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(0.), 'position': array([-1.87390234e-10,  4.57771146e-10, -5.90485676e-10, -1.00244949e-10,
        2.56695461e-09,  5.15790158e-10,  2.87811416e-10,  1.04935182e-09,
       -5.02529778e-10, -4.09611907e-09,  1.31075889e-09,  3.76760776e-09,
       -1.67248306e-10, -2.17827012e-11,  9.01389329e-10, -4.66702626e-10,
       -6.24305411e-09, -4.30611124e-09,  2.22219161e-10, -2.21898429e-10,
        6.18884810e-11, -9.27317408e-10, -9.82529450e-10,  9.58474705e-10,
        6.71399373e-11,  5.45368161e-09,  6.69041265e-09, -4.34708484e-10,
        2.30862242e-10,  9.74359469e-10]), 'index': array(25, dtype=int64)}
DURATION PSO =  8.886280536651611
GSA results :
{'fitness': array(7.04460359), 'position': array([-4.34283361e-03, -1.33525745e-04,  9.97826210e-01, -1.73536396e-03,
       -7.80329049e-03, -1.30142678e-03,  5.53598710e-03, -9.95948360e-01,
       -3.26991573e-05, -3.70097221e-03, -1.22787544e-03, -4.07473197e-03,
       -2.08340203e-03, -1.43359001e-03,  8.13031242e-03,  1.28510563e-03,
       -9.98909875e-01,  6.65840158e-03, -9.90991786e-01,  2.49869799e-03,
        9.90865529e-01,  9.99340905e-01, -4.26414069e-03,  2.68539062e-03,
        9.96114242e-01, -8.29106324e-04,  3.65459229e-03,  1.21221679e-03,
        2.80802883e-03,  2.11547828e-03]), 'index': array(29, dtype=int64)}
DURATION GSA =  6.258137226104736
DE results :
{'fitness': array(1.98991811), 'position': array([ 3.52589298e-09,  1.42355602e-09, -6.53243280e-10,  9.94958643e-01,
        9.19882880e-10, -1.11463422e-09, -4.18737609e-09, -3.84043851e-10,
       -1.43559251e-09, -1.54244056e-09, -2.51341660e-09, -6.55463501e-09,
       -1.18619617e-09,  2.74260363e-09, -1.73444026e-09, -5.24812777e-09,
        1.29270299e-09,  9.94958633e-01, -2.99733254e-09,  3.72666337e-09,
        2.26928191e-09,  3.90136069e-09, -1.51807485e-09, -4.87252974e-09,
       -3.06984998e-09,  7.61417348e-09,  8.33956798e-09,  6.26600495e-09,
       -1.69282308e-09, -3.59944955e-09]), 'index': array(17, dtype=int64)}
DURATION DE =  26.318841457366943
```

Observations

For a more significant number of iterations which is 10k and $n = 30$, The PSO shows again the best results by being the only algorithm to find the optimal $f(x) = 0$ with values of $x_1, x_2...x_n$ close to the optimum values 0.
The other two algorithms do not reach the value, in regards of time the GSA still shows the best performance.

5.2 Ackley test

We applied PSO, GSA and DE algorithms all together on the 30 dimensional Ackley function (seen in the second chapter of this work) to find the global optimum, first with 500 then 5000 iterations, and we got the following results :

500 iterations :

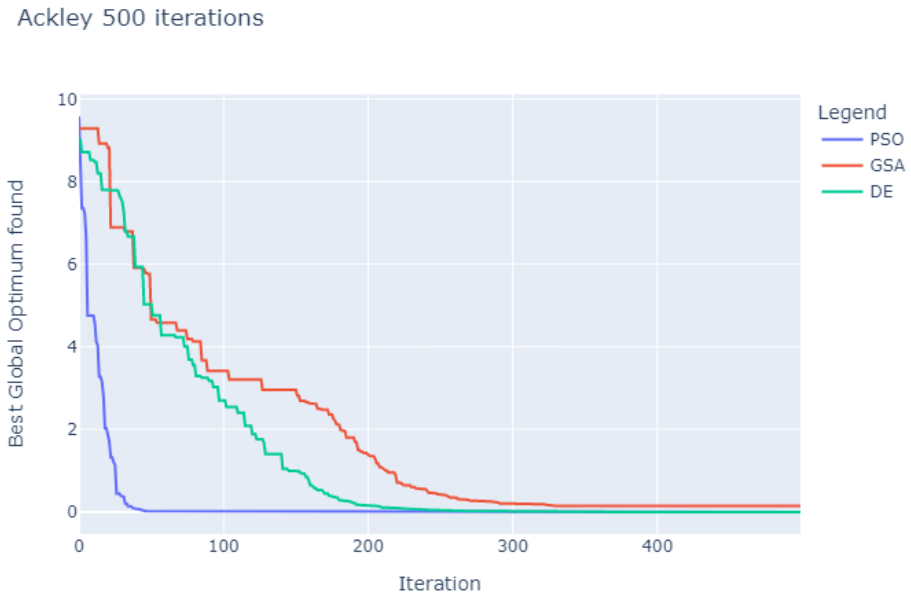


Figure 3.3: The evolution curve of the optimal solution compared to the number of iterations for the Ackley function at 500 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(4.4408921e-16), 'position': array([ 8.05802354e-17, -9.32921945e-17,  7.71415293e-17,  6.57037034e-16,
       -6.17563649e-16, -4.08876336e-16,  5.57705695e-17, -1.12360659e-16,
       -1.28722036e-16, -2.12805008e-17,  1.92475237e-16, -8.51897536e-18,
       -3.74777589e-16,  4.88266135e-17,  3.90722172e-17,  4.65695748e-16,
        7.28107262e-18, -8.50005814e-18,  5.05978952e-16,  1.71954365e-16,
        2.38188424e-17,  3.39806277e-17, -1.29229069e-16,  4.79832068e-16,
       -1.30109088e-16, -1.55823104e-16, -1.25768492e-16, -4.36950080e-17,
        2.06230865e-16,  3.74664811e-17]), 'index': array(13, dtype=int64)}
DURATION PSO =  0.374053955078125
GSA results :
{'fitness': array(0.17837514), 'position': array([ 0.01826428,  0.02673514, -0.00563696,  0.00438673,  0.00175171,
        0.01664826, -0.03915624,  0.02186012, -0.0017168 ,  0.01708907,
       -0.01102709, -0.04116752,  0.01807483,  0.00903832,  0.03388132,
        0.03457044,  0.02128524, -0.01360521, -0.01457405, -0.01808317,
       -0.00648008,  0.0404482 , -0.00203608,  0.02361136, -0.00059799,
        0.01705132,  0.0505825 ,  0.03799862, -0.11888953,  0.00256262]), 'index': array(39, dtype=int64)}
DURATION GSA =  0.3274857997894287
DE results :
{'fitness': array(0.00039393), 'position': array([ 4.39583745e-05, -1.52398929e-04, -1.32393201e-05, -1.54049445e-04,
       -1.24022367e-05, -4.20992186e-05, -2.58783060e-05,  4.72817298e-05,
       -2.89841335e-05,  1.56980331e-05, -2.36276608e-05,  1.12157300e-04,
       -2.68787793e-04, -9.36488918e-05,  2.10054935e-05,  4.82585348e-05,
       -1.42860787e-04, -2.77742083e-06, -1.95112235e-05,  1.81259365e-05,
       -8.07575409e-05, -1.37079888e-04, -1.26916258e-04,  1.18067672e-04,
        4.10591609e-05,  1.05268811e-04,  4.43842627e-05, -1.93158252e-04,
        6.63519531e-05,  7.32472279e-05]), 'index': array(31, dtype=int64)}
DURATION DE =  1.5996408462524414
```

Observations

For the Ackley test with a number of 500 iterations and $n = 30$, none of the three algorithm finds the exact value of $f(x) = 0$. Althought the algorithms all get to a close result from the best with PSO being the closest.

5000 iterations :

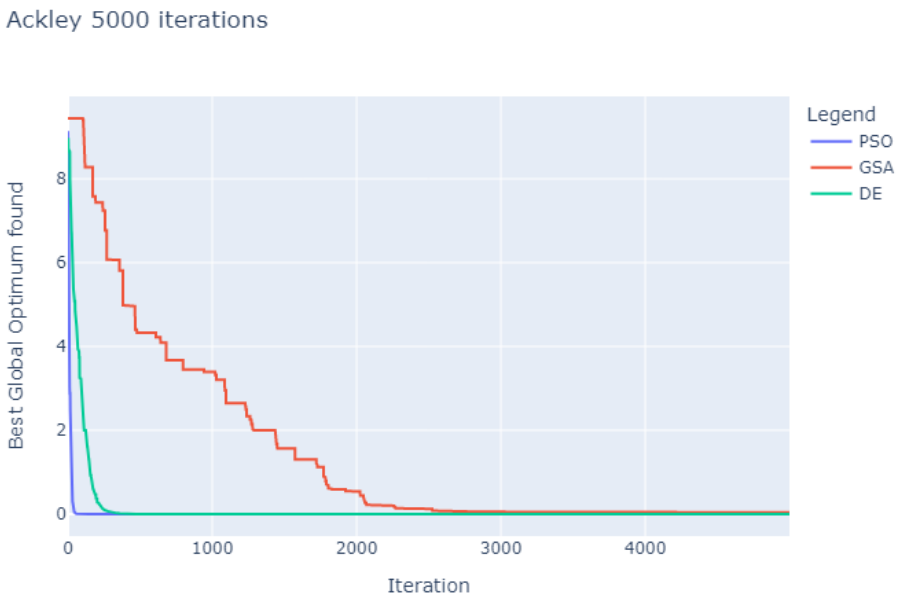


Figure 3.4: The evolution curve of the optimal solution compared to the number of iterations for the Ackley function at 5000 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(4.4408921e-16), 'position': array([ 5.67813423e-17,  6.00255078e-18, -5.85919197e-20,  9.86431759e-17,
       -3.09530792e-17, -3.19810699e-17, -4.34907015e-18,  1.04558466e-15,
       -3.06839212e-17,  3.67015798e-18,  1.63285354e-17,  5.32980079e-17,
        6.11428631e-18, -4.23071938e-17,  4.49839196e-16, -2.11530703e-16,
       -3.32853931e-16, -1.15031111e-16,  6.72761530e-17,  6.80859322e-18,
        3.36652737e-17,  5.88161829e-16, -1.12092829e-16,  2.55095135e-16,
        1.61591768e-17, -4.66706010e-18,  1.92552984e-16,  7.06354387e-17,
        1.52640699e-18, -1.16533398e-16]), 'index': array(14, dtype=int64)}
DURATION PSO =  4.08333945274353
GSA results :
{'fitness': array(0.03399829), 'position': array([ 1.23725491e-02,  1.39849534e-02,  3.75154551e-04,  1.26331082e-04,
       -4.37324197e-03, -3.15286964e-03,  6.25113752e-03, -1.35736005e-02,
        1.74623484e-02,  3.59522919e-03,  4.46626393e-03, -1.74454990e-02,
       -2.16803986e-03, -5.99161188e-03, -3.20186692e-03,  8.70982522e-03,
        9.78177905e-04,  6.73420914e-03,  1.24597031e-03,  3.03140704e-03,
       -1.78235835e-03, -4.92877034e-03, -6.24780856e-03,  9.59315292e-03,
       -8.33027943e-03, -3.97190772e-04, -4.03358743e-06,  4.75667413e-03,
       -1.21802481e-03,  1.01810106e-02]), 'index': array(21, dtype=int64)}
DURATION GSA =  3.0712969303131104
DE results :
{'fitness': array(7.54951657e-15), 'position': array([-1.07199894e-15,  2.07253362e-15,  3.59092984e-16,  2.79996777e-17,
       -2.58900087e-15,  2.82845074e-15, -4.30906933e-15, -4.55795067e-15,
        1.57020653e-15,  4.19009813e-16, -2.29587853e-15, -7.71880032e-16,
        3.41703867e-15, -4.50362901e-16,  2.04451697e-15,  4.87197777e-15,
        1.08634946e-15,  1.39973911e-16, -1.22086858e-15, -1.71605845e-15,
        3.42564521e-15,  4.05577128e-15, -5.88639293e-17, -9.80198112e-17,
        1.06480936e-15,  5.07140561e-15,  3.54955704e-16, -5.16608322e-16,
       -7.95221166e-16,  1.03711473e-15]), 'index': array(22, dtype=int64)}
DURATION DE =  13.471330881118774
```

Observations

Even with a greater number of iterations which was 5k and $n = 30$, once again none of the three algorithm finds the exact value of $f(x) = 0$. Here again PSO being the closest to the global optimum value and GSA taking the least time to find the result.

5.3 Alpine test

We applied PSO, GSA and DE algorithms all together on the 30 dimensional Alpine function (seen in the second chapter of this work) to find the global optimum, first with 100 then 1000 iterations, and got the following results :

500 iterations :

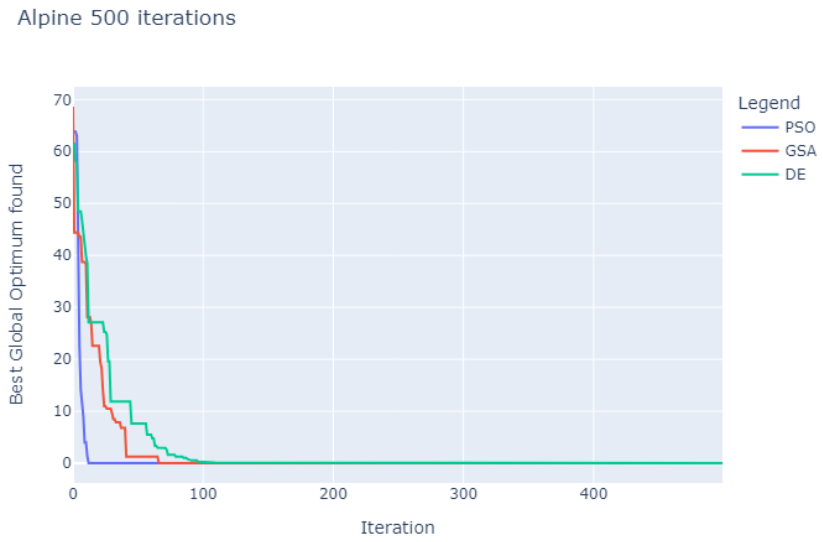


Figure 3.5: The evolution curve of the optimal solution compared to the number of iterations for the Alpine function at 500 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(0.), 'position': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'index': array(35, dtype=int64)}
DURATION PSO =  0.3321232795715332
GSA results :
{'fitness': array(1.99356003), 'position': array([0.31861537, 0.12108874, 3.24405379, 0.11793136, 0.15914629,
       0.16385657, 0.46671151, 0.24136672, 0.21170284, 0.20965238,
       0.32696311, 0.13976859, 0.11143612, 0.09376815, 0.3461296 ,
       0.22618751, 0.22580522, 3.23860152, 0.23643508, 0.27872583,
       3.24490833, 3.24517461, 0.19868578, 3.24256126, 0.2091104 ,
       0.16316762, 3.24164547, 0.33916433, 0.28719241, 0.2663723 ]), 'index': array(21, dtype=int64)}
DURATION GSA =  0.27880358695983887
DE results :
{'fitness': array(0.), 'position': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'index': array(33, dtype=int64)}
DURATION DE =  1.7824914455413818
```

Observations

For the Alpine function using 500 iterations with $n = 30$. The PSO and DE both get the best results by find the optimal $f(x) = 0$ with values of $x_1, x_2...x_n = 0$, the GSA is the only one that doesn't find that result.

2000 iterations :

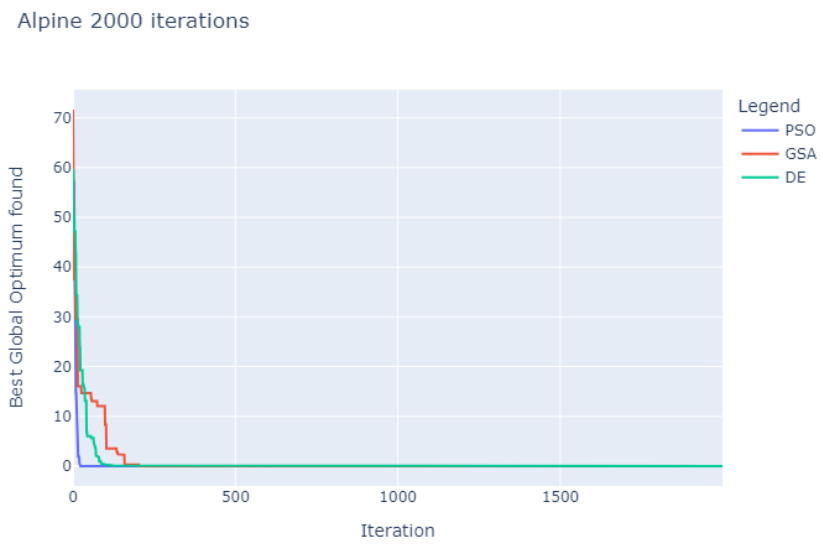


Figure 3.6: The evolution curve of the optimal solution compared to the number of iterations for the Alpine function at 2000 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(0.), 'position': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'index': array(26, dtype=int64)}
DURATION PSO =  1.465442180633545
GSA results :
{'fitness': array(0.), 'position': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'index': array(0, dtype=int64)}
DURATION GSA =  1.106048822402954
DE results :
{'fitness': array(0.), 'position': array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.]), 'index': array(8, dtype=int64)}
DURATION DE =  5.0091376304626465
```

Observations

This time for 2K iterations and $n = 30$, all three algorithms get to the global optimum $f(x) = 0$ with values of $x_1, x_2 ... x_n = 0$. With GSA showing the best performance in time.

5.4 Constrained G_1 test

We applied PSO, GSA and DE algorithms all together on the 5 dimensional Constrained G_1 function (seen in the second chapter of this work) to find the global optimum, first with 500 then 3000 iterations, and got the following results :

500 iterations :

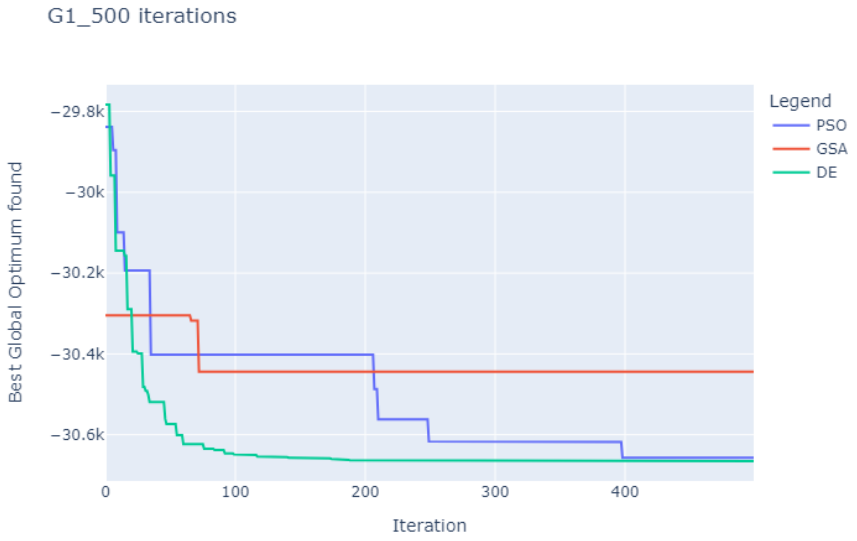


Figure 3.7: The evolution curve of the optimal solution compared to the number of iterations for the constrained G_1 function at 500 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(-30657.09741797), 'position': array([78.        , 33.        , 30.04785178, 45.        , 36.64573574]),
'index': array(9, dtype=int64)}
DURATION PSO =  1.9640560150146484
GSA results :
{'fitness': array(-30443.94005365), 'position': array([78.        , 33.        , 30.94379588, 45.        , 35.42421689]),
'index': array(118, dtype=int64)}
DURATION GSA =  2.831359624862671
DE results :
{'fitness': array(-30665.30586507), 'position': array([78.        , 33.00020889, 29.99671494, 45.        , 36.77219036]),
'index': array(134, dtype=int64)}
DURATION DE =  6.005842685699463
```

Observation

We notice in this case that the DE function is the metaheuristics that approaches the most the given optimal solution thought being at the bottom in time performance.

3000 iterations :

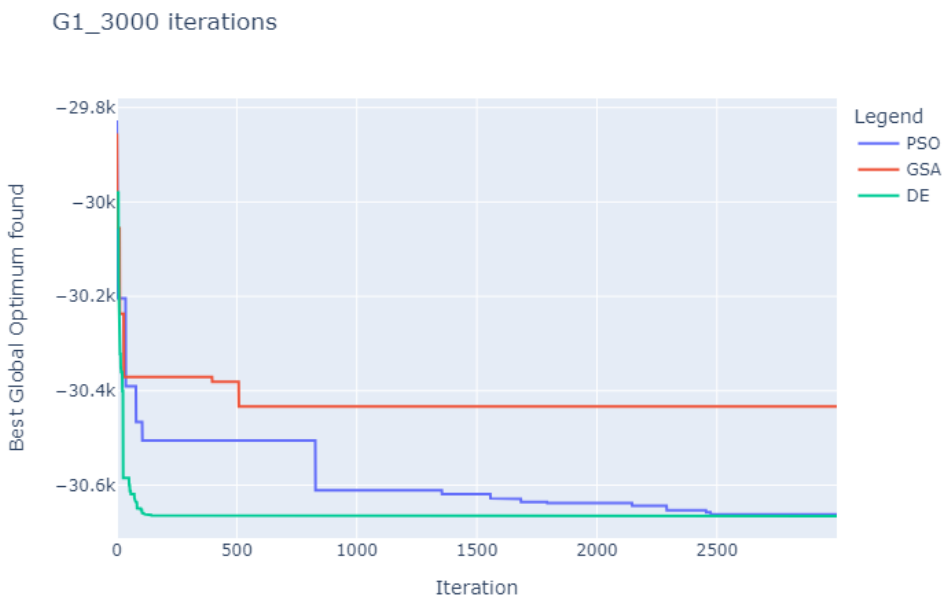


Figure 3.8: The evolution curve of the optimal solution compared to the number of iterations for the constrained G_1 function at 3000 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(-30661.61248741), 'position': array([78.        , 33.        , 30.00973027, 45.
, 36.76465588]), 'index': array(52, dtype=int64)}
DURATION PSO =  11.463290691375732
GSA results :
{'fitness': array(-30433.01844776), 'position': array([78.        , 33.        , 31.07179387, 45.
, 34.93930536]), 'index': array(18, dtype=int64)}
DURATION GSA =  12.2842435836792
DE results :
{'fitness': array(-30665.53867178), 'position': array([78.        , 33.        , 29.99525603, 45.
, 36.77581291]), 'index': array(145, dtype=int64)}
DURATION DE =  34.43457889556885
```

Observations :

with 3k iteration the DE function is the only metaheuristic to gets the optimal results to a precision of $10e-12$, while the two others are close to that number.

5.5 Constrained G_2 test

We applied PSO, GSA and DE algorithms all together on the 20 dimensional Constrained G_2 function (seen in the second chapter of this work) to find the global optimum, first with 500 then 3000 iterations, and got the following results :

500 iterations :

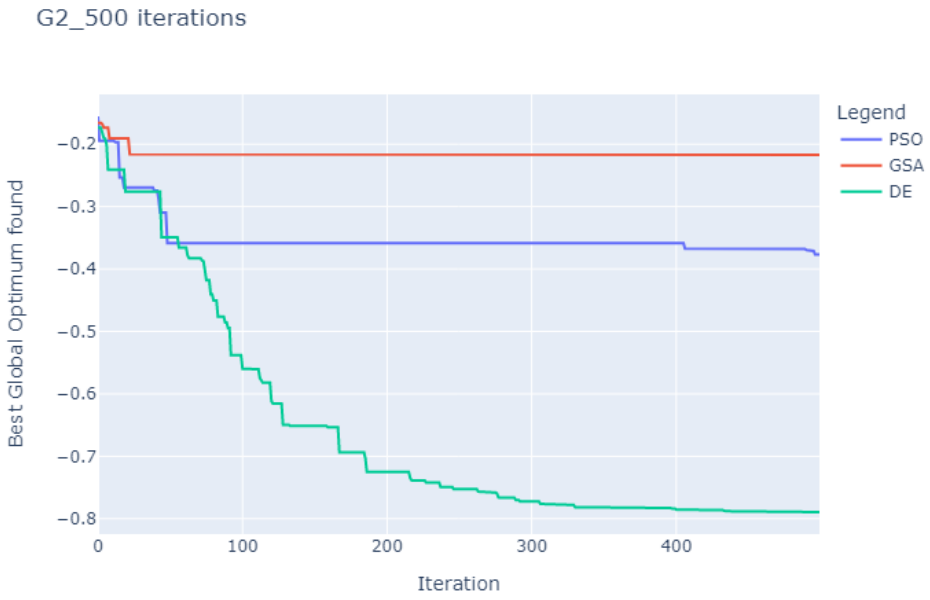


Figure 3.9: The evolution curve of the optimal solution compared to the number of iterations for the constrained G_2 function at 500 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(-0.37658639), 'position': array([3.21390094, 5.92451397, 3.17302767, 3.09521068, 4.66505054,
       2.94675182, 3.45394983, 3.36124916, 3.12596548, 3.23765918,
       1.29646675, 3.43859738, 3.17918246, 3.28084955, 3.30543524,
       0.60191441, 2.8306176 , 3.25829324, 3.16245587, 0.46865051]), 'index': array(79, dtype=int64)}
DURATION PSO =  1.075178861618042
GSA results :
{'fitness': array(-0.21694125), 'position': array([8.59538934, 9.74066583, 4.23409378, 2.70555984, 3.03854035,
       5.06637116, 8.87636694, 3.51793402, 0.56396599, 2.22107919,
       0.02829224, 3.42506731, 2.97538903, 1.58726608, 3.62690286,
       6.39747168, 3.14899205, 0.75779818, 3.32749595, 0.81808383]), 'index': array(112, dtype=int64)}
DURATION GSA =  4.113409757614136
DE results :
{'fitness': array(-0.78941802), 'position': array([3.09692233, 3.12416164, 3.0614493 , 3.03257059, 2.95921852,
       2.97981003, 2.87322104, 2.93806063, 2.9410492 , 0.43735197,
       0.43136081, 0.36920316, 0.39095181, 0.41208913, 0.41768657,
       0.32234147, 0.39522315, 0.38554228, 0.36789423, 0.45462062]), 'index': array(33, dtype=int64)}
DURATION DE =  5.020646810531616
```

Observation

We notice in this case again that the DE function is the metaheuristics that approaches the most the given optimal solution thought being at the bottom in time

performance. while the other two show quite a poor precision performance.

3000 iterations :

Figure 3.10: The evolution curve of the optimal solution compared to the number of iterations for the constrained G_2 function at 3000 iterations

The results and duration for each experimentation are as follows :

```
PSO results :
{'fitness': array(-0.41071969), 'position': array([2.71023255, 6.00937712, 2.54637536, 3.27819872, 3.28067412,
       3.47507522, 3.05262628, 3.18779521, 2.91820463, 3.04621506,
       0.31974952, 2.99260604, 2.88131826, 3.13813285, 2.88659995,
       0.13218732, 1.87666807, 2.77501503, 3.28144727, 2.79087695]), 'index': array(68, dtype=int64)}
DURATION PSO =  6.682046890258789
GSA results :
{'fitness': array(-0.2582543), 'position': array([9.09486876, 0.86435538, 0.72601855, 3.70395667, 6.02249079,
       3.68739239, 8.55607496, 3.11243398, 2.51636189, 3.67993994,
       0.91551073, 0.05222921, 0.41651099, 3.35126972, 2.44324482,
       0.41091015, 1.29076476, 2.63712026, 0.73463838, 2.82497998]), 'index': array(58, dtype=int64)}
DURATION GSA =  24.866761207580566
DE results :
{'fitness': array(-0.80357364), 'position': array([3.15752863, 3.13491296, 3.08891259, 3.06456394, 3.02819017,
       2.99461465, 2.96228962, 2.92298569, 0.48505044, 0.49920369,
       0.48626028, 0.47873082, 0.4722126 , 0.46724015, 0.45225875,
       0.45665105, 0.45263184, 0.45096041, 0.44544668, 0.43650251]), 'index': array(143, dtype=int64)}
DURATION DE =  29.628002405166626
```

Observations

with 3k iterations the DE function is the only metaheuristic to gets the optimal results to a precision of $10E$-4, while the two others barely approach that result.

6 General discussion of the results

The algorithms we used on test function show very good efficiency in both result accuracy and time performance. For the regular benchmark function the PSO has proven itself the most efficient by finding the global optimum expected nearly everytime.
In the case of Constrained function the DE algorithms is the only metaheuristic to get to the global optimum despite showing the poorest time performances.
The GSA algorithms does the expected work of a metaheuristic job by "approaching" the global optimum by only with a significant among of iterations.

7 Comparaison with other publications

In their gray wolf optimizer article published in 2014, Seyedali Mirjalili, Seyed Mohammad Mirjalili and Andrew Lewis, propose a comparative study between multiple metaheuristics which are the Gray Wolf Optimizer(GWO), PSO, GSA and DE[21].
These metaheuristics were applied to multiple benchmark functions, two of these function were the Rastrigin and the Ackley functions.
The table below showcases the result of the article version of the PSO, GSA and DE applied on the said benchmark functions for a standard number of iterations along side with results of our version :

	Comparaison with GWO article					
	PSO		GSA		DE	
	GWO Article Results	Our Results	GWO Article Results	Our Results	GWO Article Results	Our Results
Rastrigin	46.70423	0	25.96841	9.079263	69.2	1.989918
Ackley	0.276015	4.4408E-16	0.062087	0.037802	9.7E-8	1.4654E-14
		Value of the Global Optimum	0			

From the table above we can clearly notice the performance of our version being superior as they are by far closest to the optimum result which is 0.

There is another comparative study, this time only between the PSO and GSA in another article published in 2009 by Esmat Rashedi, Hossein Nezamabadi-pour and Saeid Saryazdi, GSA: A Gravitational Search Algorithm[18].
Both Algorithms were applied on the Rastrigin and the Ackley functions with 1000 iterations.

The table below showcases the results of the article results along side with our versions of the algorithms results applied on the same functions with the same number of iterations :

	Comparaison with GSA article			
	PSO		GSA	
	GSA Article Results	Our Results	GSA Article Results	Our Results
Rastrigin	55.1	0	15.32	13.26822
Ackley	9,00E-09	4.440E-16	6.9E-6	0.05287
		Value of the Global Optimum	0	

The table above shows how much our version of the PSO is delivering better performances.
As for the GSA, both results are close to one another.

By looking at both of the studies shown above, we can clearly see how metaheuristics have improved in delivering precise results.
The algorithm showcased in this work are more efficient and robust than their predecessors as the study and results shown prove.

8 Conclusion

In this chapter we have completed the practical study of our algorithms applied to the selected function and discussed all the results found, as well as a comparative study with previous publications.

Conclusion and Perspective

In recent years, various Heuristic optimization methods have been developed. Some of these algorithms are inspired by swarm behaviors in nature others but Evolution and others are Physics based.

In the first chapter of this work we covered the basic notions and definition that helped us understand the operating of metaheuristics

In the second chapter we studied the methods that we were going to base our applications on as well as benchmark functions we used to test the performances of the said methods.

And finally in the last chapter we put three metaheuristics to test by applying them to both constrained and unconstrained benchmark functions we showcased and discussed the results and compared them with previous studies.

The three metaheuristics are :

1. The particle swarm algorithm inspired by swarm behavious.
2. The differential evolution inspired by evolutionary process.
3. The gravitational search algorithm inspired by Newtonian physics.

The three algorithms showcased in this report have all proven themselves efficient methods to turn to when it comes to not only on unconstrained problems but also on constrained problems.
For these reasons it would be very interesting to apply these methods on real existing physics problems.

Bibliography

[1] Overview Of Stochastic Process, Farhad Malik, 2018 — Medium bibitemtexbook Gaussian Process Regression From First Principles, Ryan Sanders — Towards Data-Science

[2] An Elementary Introduction to the Wiener Process and Stochastic Integrals Tam´as Szabados Technical University of Budapest, 1994

[3] Gaussian processes - From scratch, Peter Roelants, 2018 — GitHub

[4] The Poisson Distribution and Poisson Process Explained, Will Koehrsen, 2019 — Towards Data-Science

[5] Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control, James C. Spall, 2003

[6] Optimization for Simulation: Theory vs. Practice, Fu. M. C., 2002 — INFORMS Journal on Computing

[7] A survey on metaheuristics for stochastic combinatorial optimization; Leonora Bianchi; Marco Dorigo; Luca Maria Gambardella; Walter J. Gutjahr, 2008

[8] Metaheuristics in Combinatorial Optimization : Overview and Conceptual Comparison, Blum, C.; Roli, A. ,2003

[9] Optimized Particle Swarm Optimization (OPSO) and its application to artificial neural network training, Michael Meissner; Michael Schmuker; Gisbert Schneider, 2006 —BMC

[10] Biologically-Inspired AI : Differential Evolution, Particle Swarm Optimization, and Firefly Algorithms, James Le, 2019 — Data Notes

[11] Metaheuristic Algorithms : A Comprehensive Review, MohamedAbdel-Basset; LailaAbdel-Fatah; Arun KumarSangaiah, 2018 — Intelligent Data-Centric Systems

[12] Understanding Artificial Bee Colony (ABC)Algorithm Using Numeral Example, Ankit Singh, 2019 — Medium

[13] Evolutionary Algorithms: genetic algorithms, Ishal Hurbans — Grokking Artificial Intelligence Algorithms

[14] Physics based Metaheuristic Optimization Algorithms for Global Optimization, Bilal Alatas; Umit Can, 2015 — American Journal of Information Science and Computer Engineering

[15] Differential Evolution Global Optimization With Python, Jason Brownlee

[16] Gravitational search algorithm: a comprehensive analysis of recent variants, Himanshu Mittal, Ashish Tripathi, Avinash Chandra Pandey, Raju Pal, 2020 — Multimed Tools Appl 80

[17] Two Kinds of Classifications Based on Improved Gravitational Search Algorithm and Particle Swarm Optimization Algorithm", Advances in Mathematical Physics, Hongping Hu, Xiaxia Cui, Yanping Bai, 2017 — Advances in Mathematical Physics

[18] GSA: A Gravitational Search Algorithm, Esmat Rashedi, Hossein Nezamabadipour and Saeid Saryazdi, 2009 — Information Sciences

[19] A survey on applications and variants of the cuckoo search algorithm, Mohammad Shehaba Ahamad, Tajudin Khadera Mohammed, AzmiAl-Betarb, 2017 — Applied Soft Computing

[20] Problem Definitions and Evaluation Criteria for the CEC 2006 Special Session on Constrained Real-Parameter Optimization,J. J. Liang, Thomas Philip Runarsson, Efr´en Mezura-Montes, Maurice Clerc, P. N. Suganthan, Carlos A. Coello Coello, K. Deb

[21] Gray Wolf Optimizer, Seyedali Mirjalili, Seyed Mohammad Mirjalili and Andrew Lewis, 2014 —Advances in Engineering Software