Cairo University

Faculty of Computers and Artificial Intelligence

[PRICING OPTIMIZATION WITH (PSO)]

Operations Research and Decision Support Department

Computational Intelligence (DS313) – 2024

NAME	ID
شادن علي حسن	20210173
نورهان عبد الله	20210437
محمد صفاء	20210537
مصطفي محمود	20200549

2024

ASSOC. PROF. AYMAN GHONEIM

Table of contents

I. Introduction
   i. Introduction to metaheuristic --3
   ii. How it works to solve optimization problems------------------------------5
   iii. Common metaheuristic algorithms--6
II. Mathematical model
   i. The problem---9
   ii. Decision variables---9
   iii. Objective function-- 9
   iv. Constraints--9
III. Code implementation
   i. Part I
      a. Encoding---11
      b. Operators--11
      c. Constraint Handling--11
   ii. Part II
      a. Logical flow of the code--12
      b. Flowchart--15
   iii. Part III
      a. Cognitive Component (c1) and Social Component (c2)---------15
      b. Radius---16
   iv. Part IV
      a. Example 1---17
      b. Example 2---18
      c. Example 3---19
IV. References--20

I. Introduction

i. Introduction to metaheuristic algorithms

What is *optimization*? In the context of mathematics, optimization, or mathematical programming, is the process of searching for the optimum solution to find an answer that minimizes or maximizes a given criterion under specified conditions.

There are several approaches to go about mathematical programming. The traditional approach to solving optimization problems that are categorized under the umbrella term of *analytical* approaches is basically a series of calculus- based optimization methods. Often these frameworks are referred to as derivate-based optimization methods, given that they rely heavily on the idea of differential algebra and gradient- oriented information to solve the problem. As such, the core idea of these methods is to *utilize the information extracted from the gradient of a differentiable function*, often from the first- or second- order derivative, as a guide to find and locate the optimal solution. The main issue here is that this could not be a practical method to approach real- world optimization problems, as these problems are often associated with high dimensionality, multimodality, non-differentiability, and discontinuous search space imposed by constraints.

The alternative approach would be to use a series of methods that are categorized under the umbrella term of *sampling*- based approaches. These, to some extent, use the simple principle of *trial- and- error* search to locate what could be the optimum solution. These methods are either based on unguided or untargeted search or the searching process that is guided or targeted by some criterion. Some of the most notable subcategories of unguided search optimization methods are sampling grid, random sampling, and enumeration-based methods. The sampling grid is the most primitive approach here, where all possible solutions would be tested and recorded to identify the best solution. In computer science, such methods are said to be based on *brute force* computation, given that to find the solution, basically, any possible solution is being tested here. As you can imagine, this could be quite computationally taxing. While this seems more manageable when the number of potential solutions is finite, in most, if not all, practical cases, this can be borderline impossible to implement such an approach to find the optimum solution. If, for instance, the search space consists of continuous variables, the only way to implement this method is to deconstruct the space into a discrete decision space. This procedure, known as *discretization*, transforms a continuous space into a discrete one by transposing an arbitrarily defined mesh grid network over the said space. Obviously, the finer this grid system, the better the chance of getting closer to the actual optimum solution.

Not only it becomes more computationally taxing to carry this task, but from a theoretical point of view, it is also considered impossible to locate the exact optimal solution for a continuous space with such an approach.

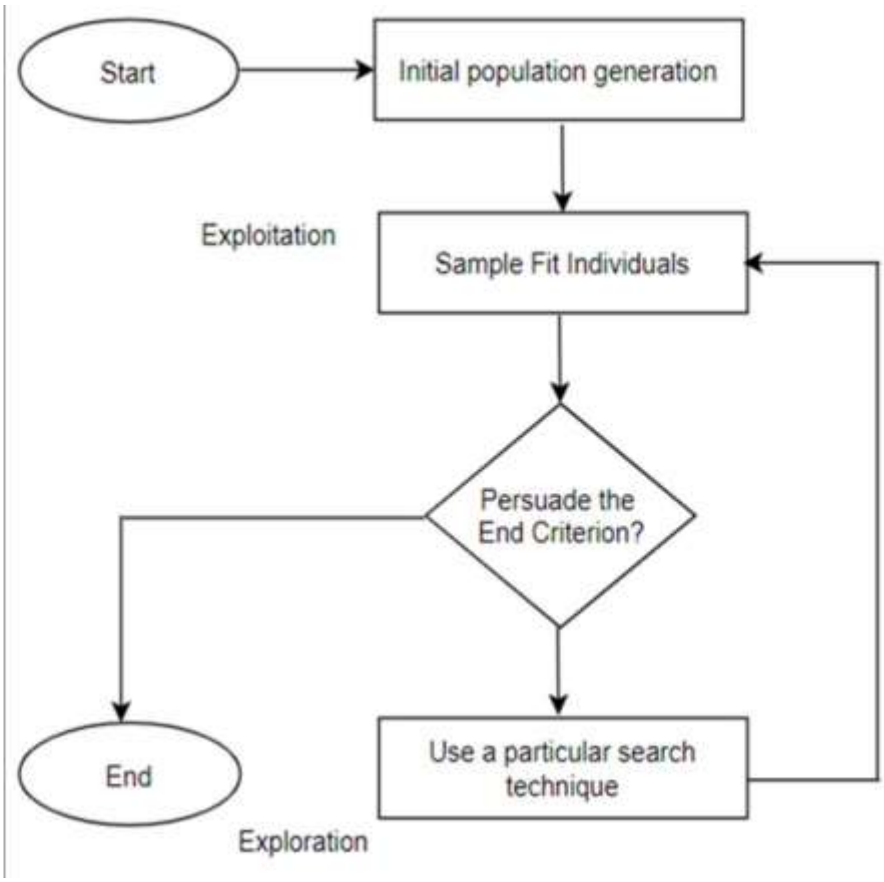
Another unguided approach is *random sampling*. The idea here is to take a series of random samples from the search space and evaluate their perform against the optimization criterion. The most suitable solution found in this process would then be returned as the optimal solution. Though this process is, for the most part, easy to execute, and the amount of computational power needed to carry this task can be managed by limiting the number of samples taken from the search space, as one can imagine, the odds of locating the actual optimum solution is exceptionally slim. This is, of course, more pronounced in complex real- world problems where there are often numerous continuous variables.

The other notable approach in the unguided search category is *enumeration-based* methods. These methods are basically a bundle of computation tasks that would be executed iteratively until a specific termination criterion is met, at which point the final results would be returned by the method as the solution to the optimization problem at hand. Its most notable drawback of all the unguided searching methods is that could not learn from their encounter with the search space to alter their searching strategies.

Alternatively, there are also *targeted* searching methods. One of the most notable features of this branch of optimization is that they can, implement what they have learned about the search space as a guiding mechanism to help navigate their searching process. As such, they attempt to draw each sample batch from what they learned in their last attempt. As a result, step by step, they are improving the possibility that the next set of samples is more likely to be better than the last until, eventually, they could gradually move toward what could be the optimum solution. It is important to note that one of the distinctive features of this approach, like any other sampling method, is that they aim to settle for a *close- enough approximation of the global optima,* better known as **near-optimal** solutions. The idea here is to possibly sacrifice the accuracy of the emerging solution to an acceptable degree to find a close- enough solution with considerably less calculation effort. One of the most well- known sub- class of the guided sampling methods is **meta-heuristic optimization algorithms**.

ii. How it works to solve optimization problems

A meta-heuristic seeks to maximize efficiency by *exploring* the search space to find near-optimal solutions. They are based on a strategy to drive the search process. The strategy can take inspiration from any natural or artificial system under observation. This can come from as diverse sources as the metallurgical process of annealing to the foraging behavior of ants. Defining a meta-heuristic around a search strategy requires us to pursue scientific and engineering goals. The scientific goal is to model the mechanism behind an inspiration like a swarm of ants. The engineering goal is to design systems that can solve practical problems. While it is impractical to define a generic framework, we can discuss some defining characteristics. Finding the ideal balance between exploration and exploitation is a crucial aspect of any meta-heuristic strategy. *Exploration* consists of exploring the entire feasible region as much as possible to evade sub-optimal solutions. *Exploitation* involves exploring the surrounding area of a promising region to find the ideal solution. This figure illustrates the exploitation and exploration flowchart.



Almost in all such meta-heuristics, we employ a *fitness function* to evaluate the candidate solutions. This is to sample the best solutions so far to focus on exploitation. Further, we use certain aspects of the search strategy to bring randomness and emphasize exploration. This is unique to every search strategy and hence quite difficult to represent using a general formulation. We can use these meta-heuristics to solve multi-dimensional real-value functions without relying on their gradient. This is a crucial point, because it implies that these algorithms can solve optimization problems that are non-continuous, noisy, and change over time as opposed to several algorithms that employ gradient descent, such as linear regression.

iii. Common metaheuristic algorithms

Genetic Algorithms (GA):

adaptive heuristic search algorithms that belong to the larger part of evolutionary algorithms. Genetic algorithms are based on the ideas of natural selection and genetics. These are intelligent exploitation of random searches provided with historical data to direct the search into the region of better performance in solution space. They are commonly used to generate high-quality solutions for optimization problems and search problems.

They simulate the process of natural selection which means those species that can adapt to changes in their environment can survive and reproduce and go to the next generation. In simple words, they simulate "survival of the fittest" among individuals of consecutive generations to solve a problem.
Each generation consists of a population of individuals and each individual represents a point in search space and possible solution. Each individual is represented as a string of character/integer/float/bits. This string is analogous to the Chromosome.

<u>Operators of Genetic Algorithms:</u>

Once the initial generation is created, the algorithm evolves the generation using following operators:
1) Selection Operator: The idea is to give preference to the individuals with good fitness scores and allow them to pass their genes to successive generations.
2) Crossover Operator: This represents mating between individuals. Two individuals are selected using selection operator and crossover sites are chosen randomly. Then the genes at these crossover sites are exchanged thus creating a completely new individual (offspring).

3) Mutation Operator: The key idea is to insert random genes in offspring to maintain the diversity in the population to avoid premature convergence.

Simulated Annealing (SA):

It is a global search optimization algorithm that is inspired by the annealing technique in metallurgy.
'Annealing' is a technique, where a metal is heated to a high temperature and slowly cooled down to improve its physical properties. When the metal is hot, the molecules randomly re-arrange themselves at a rapid pace.
As the metal starts to cool down, the re-arranging process occurs at a much slower rate. In the end, the resultant metal will be a desired workable metal. The factors of time and metal's energy at a particular time will supervise the entire process.
Simulated annealing algorithm mimics this process and is used to find optimal (or most predictive) features in the feature selection process.

In terms of feature selection:

Set of features: represents the arrangement of molecules in the material(metal).

No. of Iterations: represents time. Therefore, as the no. of iterations decreases temperature decreases.

Change in predictive performance between the previous and the current iteration represents the change in material's energy.

Ant Colony Optimization (ACO):

inspired from the foraging behavior of ant colonies. Ants communicate with each other using sound, touch and pheromone. Pheromones are organic chemical compounds secreted by the ants that trigger a social response in members of same species. These are chemicals capable of acting like hormones outside the body of the secreting individual, to impact the behavior of the receiving individuals. Since most ants live on the ground, they use the soil surface to leave pheromone trails that may be followed (smelled) by other ants.
Ants live in community nests and the underlying principle of ACO is to observe the movement of the ants from their nests in order to search for food in the shortest possible path. Initially, ants start to move randomly in search of food around their nests. This randomized search opens up multiple routes from the nest to the food source. Now, based on the quality and quantity of the food, ants carry a portion of the food back with necessary pheromone concentration on its return path. Depending on these pheromone trials, the probability of selection of a specific path by the following ants would be a guiding factor to the food source. Evidently, this probability is based on the concentration as well as the rate of evaporation of pheromone. It can also be observed that since the

evaporation rate of pheromone is also a deciding factor, the length of each path can easily be accounted for.

Particle Swarm Optimization (PSO):

inspired by swarm behavior observed in nature such as fish and bird schooling. PSO is a Simulation of a simplified social system. The original intent of PSO algorithm was to graphically simulate the graceful but unpredictable choreography of a bird flock.

Mathematical model:

1. Each particle in particle swarm optimization has an associated position, velocity, fitness value.
2. Each particle keeps track of the particle_bestFitness_value particle_bestFitness_position.
3. A record of global_bestFitness_position and global_bestFitness_value is maintained.

II. Mathematical model

i. The problem

A manufacturer of colored TV's is planning the introduction of two new products: a 19–inch stereo color set with a manufacturer's suggested retail price of $339 per year, and a 21–inch stereo color set with a suggested retail price of $339 per year. The cost of the company is $195 per 19–inch set and $225 per 21–inch set, plus additional fixed costs of $400,000 per year. In the competitive market the number of sales will affect the sales price. It is estimated that for each type of set, the sales price drops by one cent for each additional unit sold. Furthermore, sales of the 19–set will affect sales of the 21–inch set and vice versa. It is estimated that the price for the 19–inch set will be reduced by an additional 0.3 cents for each 21–inch sold, and the price for 21–inch sets will decrease for by 0.4 cents for each 19–inch set sold. The company believes that when the number of units of each type produced is consistent with these assumptions all units will be sold. How many units of each type of set should be manufactured such the profit of the company is maximized?

ii. Decision variables

- The relevant variables of this problem are:

s1: number of units of the 19–inch set produced per year,
s2: number of units of the 21–inch set produced per year,
p1: sales price per unit of the 19–inch set ($),
p2: sales price per unit of the 21–inch set ($),
C: manufacturing costs ($ per year),
R: revenue from sales ($ per year), P: profit from sales ($ per year).

- The market estimates result in the following model equations,

$p1 = 339 - 0.01s1 - 0.003s2$
$p2 = 399 - 0.04s1 - 0.01s2$
$R = s1p1 + s2p2$
$C = 400{,}000 + 195s1 + 225s2$
$P = R - C.$

iii. Objective function

Maximize the profit expression:

- $P(s1,s2) = -400{,}000 + 144s1 + 174s2 - 0.01(s1)^2 - 0.007s1s2 - 0.01(s2)^2$

iv. Constraint

Subject to the constraint

- s1 + s2 <= 10,000
- 0 <= s1 <= 5000
- 0 <= s2 <= 8000

III. Code Implementation

i. Part I

a. Encoding

The encoding used for this optimization problem is real-valued, meaning the decision variables (s1 and s2) are represented as floating-point numbers. This encoding is appropriate as it allows for the representation of continuous values within the search space, reflecting the number of units of each TV set produced.

b. Operators

Initialization: The initial population of particles (representing different combinations of s1 and s2) is randomly generated within the search space.

Position Update: Particle positions (s1 and s2) are updated iteratively based on their velocities and current positions, considering the influence of personal best and local best position.

Velocity Update: Velocities of particles are adjusted based on the difference between personal best and global best positions, aiming to guide particles towards promising regions in the search space.

Personal Best Update: Each particle maintains its personal best position based on the highest fitness value it has achieved so far.

Local best update: each particle maintains its personal local best each iteration. It's determined by choose the best fitness within circle its radius = 1000.

c. Constraint handling

The constraint handling technique used in particle swarm optimization to solve pricing problem is **"Penalty schema".**

- **Penalty schema**
  The most popular idea is the penalty schema because it is straightforward in addressing the constraints issue. The penalty schema for PSO is calculating the penalty value of constraint violations and subtracting this penalty value from the objective function the NCO problems with the penalty schema can be expressed as follows:

With respect to the following q inequalities:

$$g_j(X) \leq 0, \; j = 1, \ldots, q$$

And to the following $m - q$ equalities:

$$h_j(X) = 0, \; j = q + 1, \ldots, m$$

$$V_j(X) = \begin{cases} max\left(0, g_j(X)\right) & if\ 1 \leq j \leq q \\ \left|h_j(X)\right| & if\ q + 1 \leq j \leq m \end{cases}$$

$$Maximize\ f_p(X) = f(x) - \sum_{j=1}^{m} C_j V_j^k(X)$$

Where:
fp(X): is the penalized fitness function
f (x): is the unpenalized fitness function
Cj: is a user defined constant which reflect the difficulty of satisfying the constraint j, and
k: is a user defined constant (usually 1 or 2) in order to magnify the violation.

- **Boundary schema**
  Some studies treat the infeasible regions as particles' boundaries. Thus, they will let the particles likely explore feasible regions and avoids the particles entering infeasible regions. When a particle moves to an infeasible region, drag the particle back to a closer feasible position against the infeasible region's boundary.
  In this schema, the particles are dragged back into the original feasible solution.

ii. Part II

a. Logical flow of the code

- **Parameters**

c1,c2	Congnitive and social parameters for PSO
num_particles	Number of particles in the swarm
rterations_for_solo_run	Number of iterations for PSO

Num_runs	Number of iterations for the outer loop
radius	Radius for defining the neighborhood in local best calculation
bounds	Upper and lower bounds for the search space
velocity_bounds	Upper and lower bounds for particle velocities in the initialization

- **Lists**

position	List to store the positions of particles. Each row has 2 values for s1 and s2.
velocities	List to store the velocities of particles. Each row has 2 values for s1 and s2.
xBest	List to store the personal best fitness of each particle.
optimal	List to store the best fitness value found in each iteration.
best s1, best s2	Lists to store the best positions `s1` and `s2` found in each iteration.

- **Functions**

Generate_random()	Generates a random number between 0 and 1.
Evaluate_fitness(s1,s2)	Evaluates the fitness function. It also handles the penalty for violating the linear constraint.
Calculate_velocity (x,oldVelocity,xBest,lBest)	Calculates the velocity for a particle based on its position, old velocity, personal best position, and local best position.
Calculate_distance(s1,s2)	Calculates the Euclidean distance between two particles
Calculate_lBest (particles,target,indexOFTarget)	Calculates the local best position for a particle based on its neighborhood defined by the radius.

- **Steps**
    - **The outer loop:** we run code the number of times to avoid the effect of random variables
    - **The first iteration**

1. Generate position and velocity randomly for each particle between the bounds and append the values in **position** and **velocities** lists.
2. Evaluate fitness for each particle and set the value as **xBest.**
3. Get the best fitness of them all and set in the **bestFitness.**
4. Get the index of the best fitness.
5. Set **s1** and **s2** as the position for the best fitness

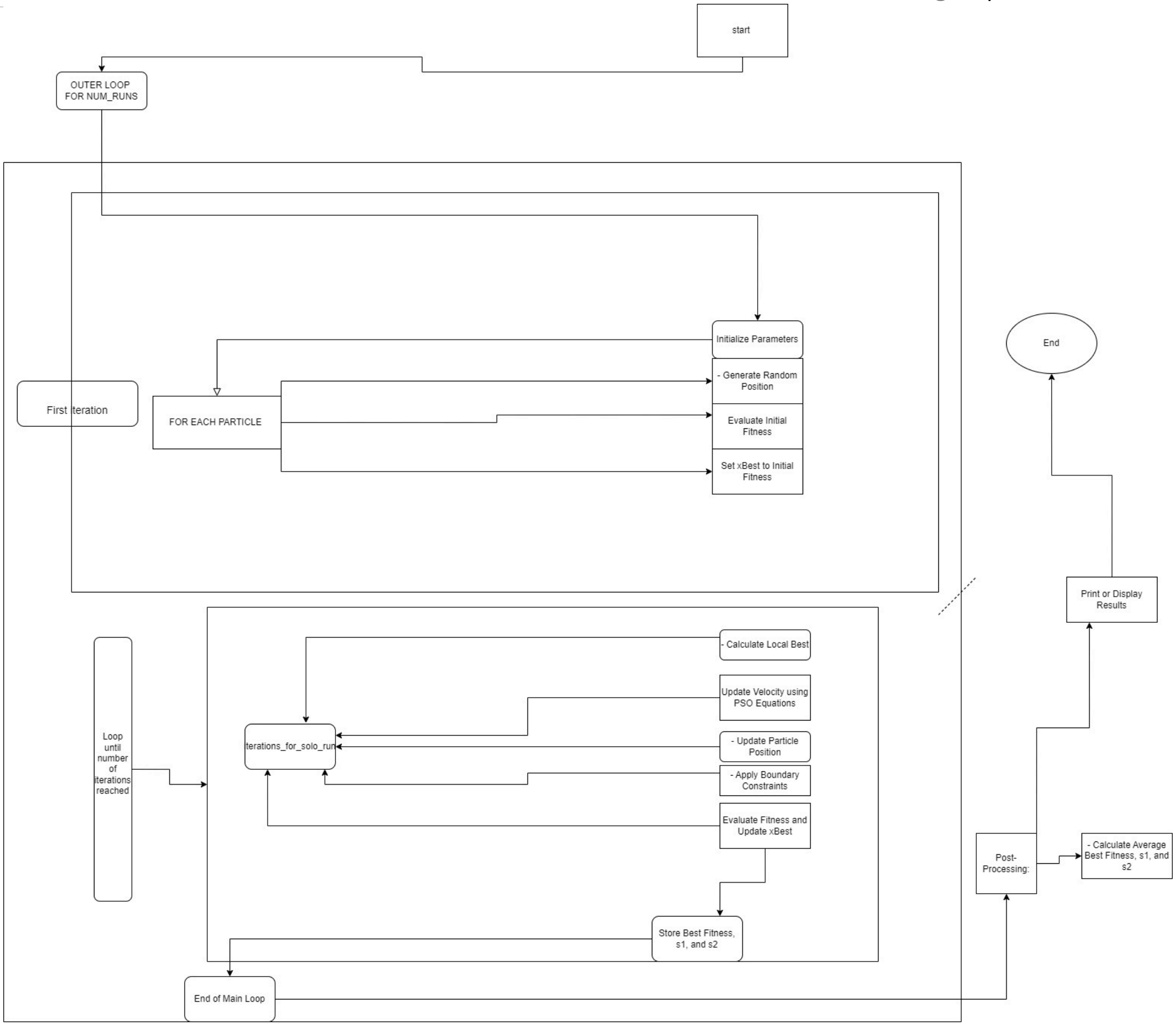
- **The next iterations**
  1. For each particle, we calculate the local best by **Calculate_lBest()** function, it's calculate by finding the best fitness in the circle of specific radius around the target particle
  2. Calculate the new velocity for s1 and s2 using **Calculate_velocity()** function using this formula:

$$\begin{aligned} v_{ij}(t+1) &= v_{ij}(t) + c_1 r_{1j}(t)[y_{ij}(t) - x_{ij}(t)] \\ &\quad c_2 r_{2j}(t)[\hat{y}_j(t) - x_{ij}(t)] \end{aligned}$$

  3. Calculate the new position using this formula:

$$x_i(t+1) = x_i(t) + v_i(t+1)$$

  4. Check boundaries: if it's out of boundaries, set it as boundaries.
  5. Calculate the new fitness using **Evaluate_fitness()** function.
  6. Reset new **xBest** by set the max between the current fitness and the current **xBest.**
  7. Reset **bestFitness** by set the max between the current **bestFitness** and the current fitness and reset the corresponding **s1** and **s2**.
- **At the end of each run**
  1. Append the **bestFitness** into the **optimal** list
  2. Append the corresponding **s1** and **s2** to **bestS1** and **bestS2** lists.
  3. Clear all lists.



iii. Part III

Cognitive Component (c1) and Social Component (c2):

There are many parameters in particle swarm optimization algorithm which considered an important thing in this algorithm, parameters are: o Cognitive Component (c1) and Social Component (c2): These parameters determine the influence of the personal best and global best weights(positions), respectively, on the expected risk (velocity) update equation. The cognitive component (c1) controls the particle's tendency to move towards its personal best position, while the social component (c2) determines its tendency to move towards the local best position. The optimal values of c1 and c2 depend on the problem, sum of c1 and c2 should be 4.

In our project we consider that c1 and c2 both equal 2, because we tried many values, but the final solution is better in the first case.

Example 1: when we set c1=3 and c2=1

```
==================================================================================
Average of average best fitness =  447086.74787414615
Average of average S1 =  4500.973558382024
Average of average S2 =  3010.769119582777
==================================================================================
The best solution =  515075.6027354374
s1 for best solution =  4785.250163768691
s2 for best solution =  2493.6367288010288

Process finished with exit code 0
```

Example 2: when we set c1=1 and c2=3

```
==================================================================================
Average of average best fitness =  454639.629073107
Average of average S1 =  4536.439812604333
Average of average S2 =  3745.58241430419
==================================================================================
The best solution =  539440.4211694765
s1 for best solution =  4990.454827805103
s2 for best solution =  2453.689072138106

Process finished with exit code 0
```

Example 3: when we set c1=2 and c2=2

```
==================================================================================
Average of average best fitness =  460033.5313588091
Average of average S1 =  4415.525617235995
Average of average S2 =  2920.810147423684
==================================================================================
The best solution =  546898.2546230537
s1 for best solution =  4987.142004006133
s2 for best solution =  4285.81573012146

Process finished with exit code 0
```

Radius:

- 0 <= s1 <= 5000
- 0 <= s2 <= 8000

  So, our search space area = 40,000

  So, we tried to choose a suitable radius to trade off between exploration and exploitation

Example 1: radius = 2000

```
=====================================================================================
Average of average best fitness =  418704.05277724995
Average of average S1 =  4366.448741988203
Average of average S2 =  3234.2464327530286
=====================================================================================
The best solution =  513789.9008512695
s1 for best solution =  4623.365382816702
s2 for best solution =  3171.064816064637

Process finished with exit code 0
```

Example 2: radius = 1000

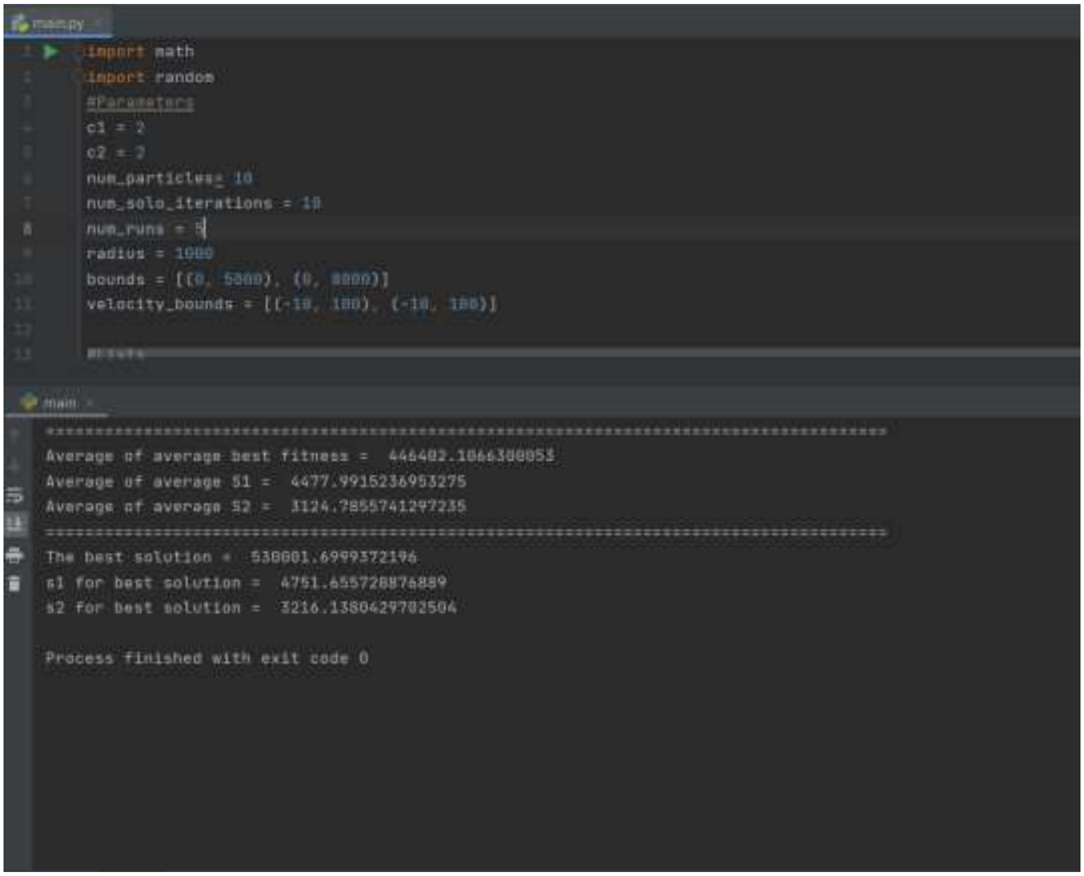
```
=====================================================================================
Average of average best fitness =  418704.05277724995
Average of average S1 =  4366.448741988203
Average of average S2 =  3234.2464327530286
=====================================================================================
The best solution =  513789.9008512695
s1 for best solution =  4623.365382816702
s2 for best solution =  3171.064816064637

Process finished with exit code 0
```

And radius = 1000 gives the best solution.

iv. Part IV

Example 1:

num_particles= 10
num_solo_iterations = 10
num_runs = 5

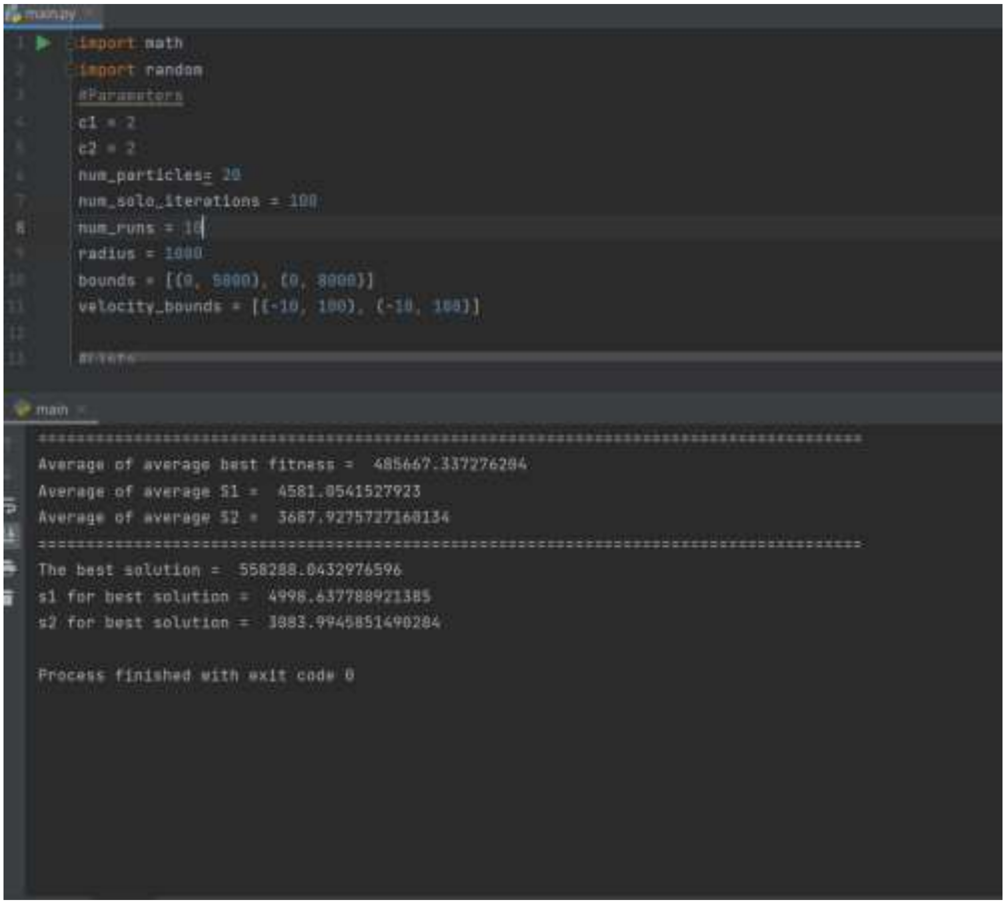


Example 2:

num_particles= 20

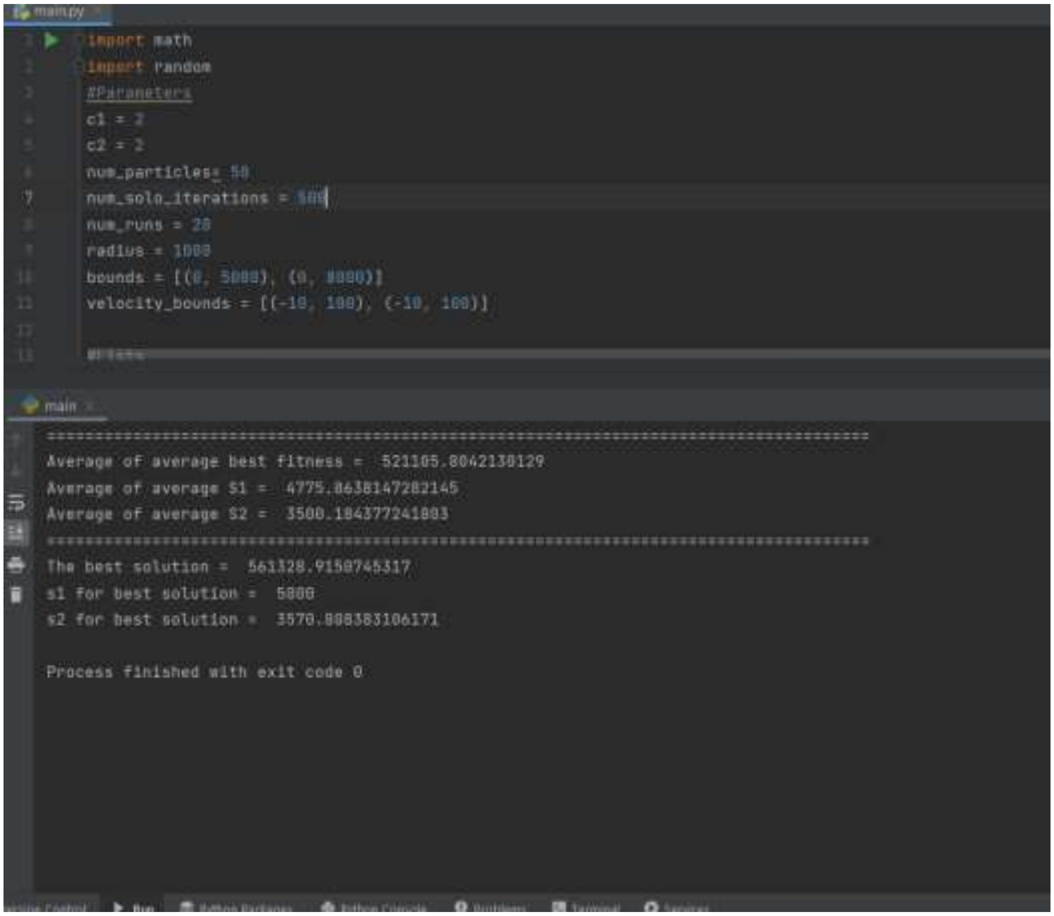
num_solo_iterations = 100

num_runs = 10



Example 3:

num_particles= 50
num_solo_iterations = 500
num_runs = 20


main.py
import math
import random
#Parameters
c1 = 2
c2 = 2
num_particles= 50
num_solo_iterations = 500
num_runs = 20
radius = 1000
bounds = [(0, 5000), (0, 8000)]
velocity_bounds = [(-10, 100), (-10, 100)]
main
Average of average best fitness = 521105.8042130129
Average of average S1 = 4775.8638147282145
Average of average S2 = 3500.184377241803
The best solution = 561328.9150745317
s1 for best solution = 5000
s2 for best solution = 3570.808383106171
Process finished with exit code 0

IV. References

1. Computational Intelligence based Optimization Algorithms (from Theory to Practice) - Babak Zolghadr- Asli
2. Bozorg- Haddad, O., Solgi, M., & Loáiciga, H.A. (2017). Meta- heuristic and evolutionary algorithms for engineering optimization. John Wiley & Sons. ISBN: 9781119386995
3. Bozorg- Haddad, O., Zolghadr- Asli, B., & Loaiciga, H.A. (2021). A handbook on multi- attribute decision- making methods. John Wiley & Sons. ISBN: 9781119563495 Metaheuristic Algorithms for Optimization: A Brief Review by Vinita Tomar, Mamta Bansal and Pooja Singh
4. geeksforgeeks.org
5. https://www.machinelearningplus.com/machine-learning