February 5, 2024

MSc Program – Academic Year 2023-2024

(Δ3) Optimization
Project 2 Report

Nefeli Eleftheria Sextou
Student ID: 503
E-Mail: pcs00503@uoi.gr

Dept. Of Computer Science and Engineering

University Of Ioannina

Table of Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Problem Description

1.1 Introduction

The spatial arrangement of facilities (warehouses, production units, etc.) is a fundamental and critical element of the supply chain. Optimal geographical placement contributes to reducing the operational costs of companies, primarily in terms of product supply and distribution, as well as to achieving closer interaction with customers.

The present work examines a geographic location problem related to the minimization of daily operational costs of a company's warehouses. These costs stem from the daily service of a number of customers in the broader area of the city of Ioannina, Greece where products are distributed based on demand. Additional costs arise from order management and fixed operational warehouse expenses.

To solve this problem, Genetic Algorithms (GAs) and the Particle Swarm Optimization method (PSO) are employed. The algorithms' performance is statistically analyzed with the purpose of drawing conclusions about their ability to solve this specific type of problem.

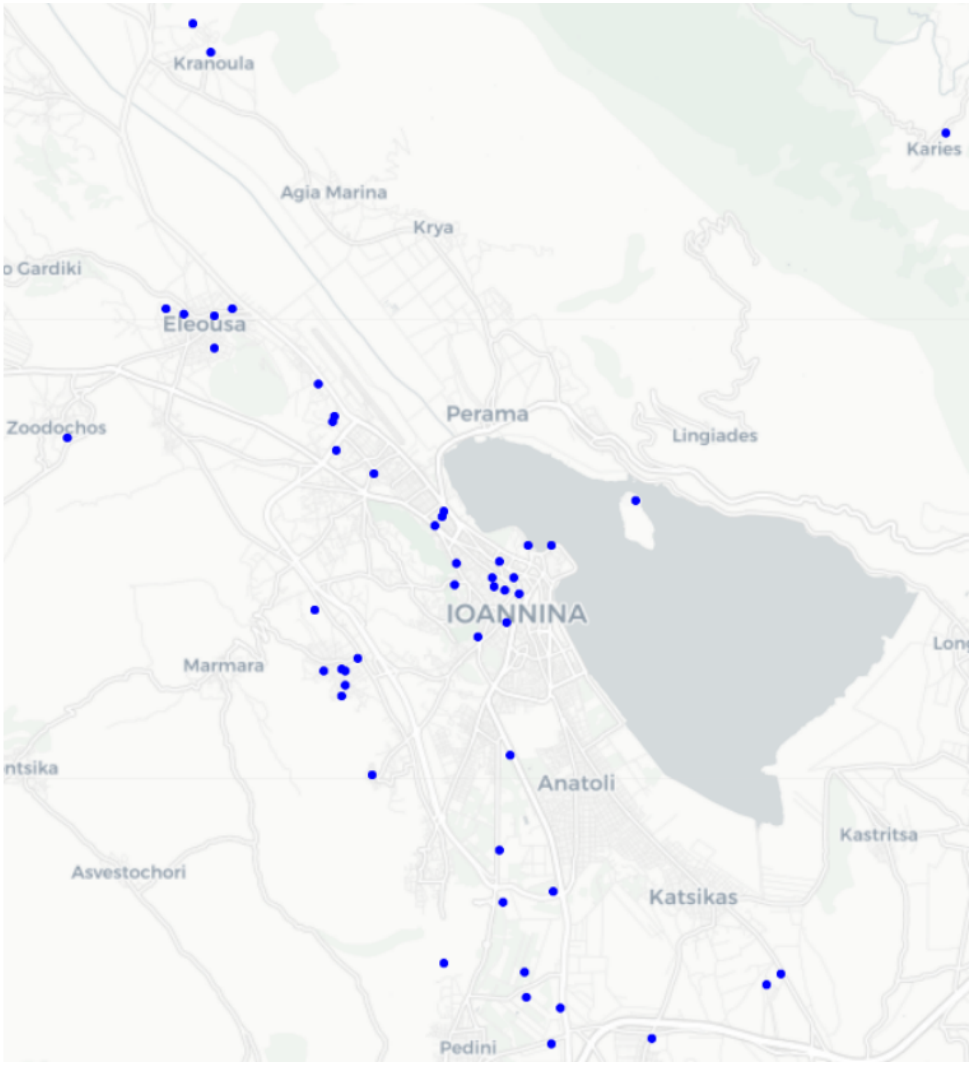


Figure 1.1: The Geographical Locations of the 50 Customers Serviced by the Company

1.2 Detailed Description

A company distributes a specific product to a total of 50 customers daily. The geographical location of each individual customer is presented in Figure 1.1. The corresponding data is provided in coordinate form (latidute and longitude) along with their mean individual demand in an xlsx file (`customer_coordinates.xlsx`).

The company intends to establish warehouses within the rectangular area defined by the maximum and minimum coordinates of the customers. The options under consideration include constructing one, two, or three warehouses in this area, with the assignment of customers to each respective warehouse. Product distribution occurs on the same day from each warehouse independently for each customer. That is, the distribution vehicle departs from the warehouse, delivers to the specific customer, and

Point	Latitude	Longtitude
North-West	39.688367	20.838671
South	39.632539	20.901448
East	39.666708	20.929109

Table 1.1: Lake Coordinates

returns to the warehouse to pick up the next order. It is assumed that the service rate for each warehouse is sufficient to cover all daily distributions assigned to it. The operational cost of each warehouse is calculated as the sum of the distances (in km) covered daily by the distribution vehicle to serve all customers assigned to that specific warehouse. This sum is then multiplied by the average operational cost per kilometer for the vehicle, set at 1.97 euros/km.

In addition to that cost, there is a daily expense for the packaging of the distributed units of the product, which is calculated (per warehouse), in a decreasing manner with regard to cost, as follows: 0.05 euros per product unit for warehouses packaging up to 500 units, 0.04 euros per product unit for warehouses packaging from 501 to 1000 units, and 0.03 euros per product unit for warehouses packaging 1001 units and above. The decreasing charge results from a discount on packaging expenses based on the quantity procured by the warehouse. Finally, there are fixed daily operational expenses for each warehouse amounting to 20.00 euros.

The total daily cost of the company is considered to be the sum of all the aforementioned costs for all the warehouses that will ultimately operate (1, 2, or 3). To simplify the problem, it is assumed that there are no other expenses for the supply of warehouses from the product manufacturing unit or constraints on the capacity of warehouses and vehicles. The overall daily cost of the company must be minimized by simultaneously selecting (a) the number of warehouses to be used, (b) their optimal placement on the map, specifically within the rectangle defined by the coordinates of the customers, and (c) the optimal assignment of customers to each warehouse.

It must be highlighted that the geographical area within the rectangle contains a lake. There can be no warehouses built within the lake or on its borders. This problem is simplified by considering the lake to be a triangle defined by three vertices, one on the North-West end, one on the South end and one on the East end as evident on Figure1.1 and as presented in coordinate form on Table 1.1

1.3 Mathematical Description

The average operational cost per kilometer for the vehicle and the fixed daily operational expenses for each warehouse are symbolized as:

$$C_{km} = 1.97 \qquad C_{op} = 20 \tag{1.1}$$

respectively.

The packaging cost per product unit for each warehouse is defined as:

$$C_{pk}(d) = \begin{cases} 0.05 & \text{if } d \leq 500 \\ 0.04 & \text{if } 500 < d \leq 1000 \\ 0.03 & \text{if } d > 1000 \end{cases} \tag{1.2}$$

where $d > 0$ represents the daily demand, in product units, that the warehouse services.

If each customer is considered to have a respective unique identifier, an integer "customer id", it is possible to describe the set of clients as follows:

$$J = \{1, 2, ..., N\} \text{ where } N = 50 \tag{1.3}$$

The corresponding set that contains the mean daily demand of each individual customer may be represented by set:

$$D = \{d_1, d_2, ..., d_N\} \text{ where } N = 50 \tag{1.4}$$

The position of each customer is described by a corresponding pair of (latitude, longitude) coordinates so each client is located at point:

$$p_j = (lat_j, long_j) \text{ where } j \in J \tag{1.5}$$

All the information described above can be found in the `customer_coordinates.xlsx` file.

If the company functions with M warehouses with their geographical position symbolized as w_i for $i \in I = \{1, 2, ..., M\}$. This position is also a pair of coordinates:

$$w_i = (lat_i, long_i) \text{ where } i \in I \tag{1.6}$$

If $J_i \subseteq J$ is the set of customers serviced by warehouse i, then evidently:

$$J = \cup_{i \in I} J_i \text{ and } J_k \cap J_l = \varnothing \, , \, \forall k, l \in I \, , \, k \neq l \tag{1.7}$$

The subsets containing the customers assigned to each warehouse do not overlap. The total demand serviced by warehouse i is defined as:

$$sd_i = \sum_{j \in J_i} d_j \; \forall i \in I \tag{1.8}$$

For each customer $j \in J$, we can symbolize the warehouse to which he has been assigned to with h_j. So we have:

$$h_j = i \Leftrightarrow j \in J_i \, , \; \forall i \in I \, , \, j \in J \tag{1.9}$$

The distance between a client j and warehouse i, in km, is:

$$t_{ij} = distance(w_i, p_j) \, , i \in I \, , \, j \in J \tag{1.10}$$

where the distance is the Haversine distance which is defined by the formula below:

$$t_{ij} = 2R \, arcsin(\sqrt{sin^2(0.5\delta_{lat}) + cos(lat_j \, m) \, cos(lat_i \, m) \, sin^2(0.5\delta_{long})} \,) \, , \forall i \in I \, , \, j \in J \tag{1.11}$$

where:

$$\delta_{lat} = (lat_j - lat_i) \, m \qquad\qquad \delta_{long} = (long_j - long_i) \, m \tag{1.12}$$

with $m = 0.0174532925$ the value of a degree in radians and $R = 6371$ the Earth's radius in km. This allows us to define the total approximate distance that a company vehicle will traverse in a day:

$$st_i = \sum_{j \in J_i} t_i j \; \forall i \in I \tag{1.13}$$

According to all of the above it is now possible to define the total daily cost for warehouse i:

$$C_i = C_{op} + sd_i C_{pk}(sd_i) + st_i C_{km} \tag{1.14}$$

so the optimization problem that must be solved is the minimization of the total daily cost of the company:

$$\min_{M, w_i, J_i, i \in I} \; C_{total} \triangleq \sum_{i \in I} C_i \tag{1.15}$$

1.4 Implementation Details

The vector containing the unknowns encodes the number of warehouses M as an integer, the coordinates $w_i,\ i \in I = \{1, 2, ..., M\}$ of each warehouse as real numbers, as well as the warehouse assignment h_j for each customer as integers.
The vector is of immutable size and corresponds to the maximum possible size it could have for $N = 50$ and $M = 3$, $N + 2M + 1 = 57$:

$$x = (x_1, x_2, ..., x_{57}) \tag{1.16}$$

where the components are as follows:

Index	0	1	2	3	4	5	6	7	...	56
Position	1	2	3	4	5	6	7	8	...	57
Component	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	...	x_{57}
Variable	M	lat_1	$long_1$	lat_2	$long_2$	lat_3	$long_3$	h_1	...	h_{57}

The first position encodes the variable M, positions $2 - 7$ contains the warehouse coordinates in pairs and the rest of the positions encode the warehouse assignments for the customers. If $M = 1$ the coordinates in positions $3 - 7$ are ignored and all assignments are equal to 1. If $M = 2$, the coordinates in positions $5 - 6$ are ignored and the assignments take the values 1 or 2. If $M = 3$, no coordinates are ignored and of course the assignments can take values 1, 2 or 3.

The second issue that must be tackled is the representation of real numbers as integers and coordinates because the algorithms that will be applied produce and process real valued vectors. All variables are defined within $[0.0, 1.0]$.

When it comes to expressing real numbers as integers, an integer $u \in \{1, 2, ..., U\}$ may be represented by a real value $y \in [0.0, 1.0]$ partitioning the range $[0.0, 1.0]$ in U equally sized partitions. u will be mapped to an integer value in accordance to the following formula:

$$u_y = \begin{cases} 1 & \text{if } y \in [0.0, 1/U] \\ 2 & \text{if } y \in (1/U, 2/U] \\ 3 & \text{if } y \in (2/U, 3/U] \\ ... & ... \\ U & \text{if } y \in ((U-1)/U, 1.0] \end{cases} \tag{1.17}$$

To turn real numbers that belong to $[0.0, 1.0]$ into coordinates, we can utilize the transformation below:

$$lat_i = x_{lat}\Delta_{lat} + lat_{min} \qquad lat_i = x_{long}\Delta_{long} + long_{min} \tag{1.18}$$

and

$$\Delta_{lat} = lat_{max} - lat_{min} \qquad \Delta_{long} = long_{max} - long_{min} \tag{1.19}$$

where x_{lat} and x_{long} are the real numbers that correspond to the latitude and longitude components we want to produce and $lat_{max} = 39.7506861$, $long_{max} = 20.9345622$, $lat_{min} = 39.6003201$, and $long_{min} = 20.7660648$ are the maximum and minimum latitude and longitude values of the set of customers. A special note must be made here about the case where it is necessary to do the opposite and turn coordinates into real values. That is easily done by solving Eq.1.18 considering x_{lat} and x_{long} as the unknown variables.

The final issue that must be discussed in this section is the implementation of the constraint with regard to the lake. This is achieved by transforming the coordinates on Table 1.1 into real values and then checking if the point is within or on the bounds of the triangle by leveraging the concept of barycentric coordinates.

Following [1] and [2], we consider the three vertices of the triangle in Cartesian form (real numbers) to be the North-West vertex (x_{NW}, y_{NW}), the South vertex (x_S, y_S) and the East vertex (x_E, y_E). The Cartesian coordinates of the point P being checked are (x_P, y_P). The corresponding barycentric coordinates of point P in terms of both its own Cartesian coordinates and those of the three vertices are:

$$b_1 = \frac{(y_S - y_E)(x_P - x_E) + (x_E - x_S)(y_P - y_E)}{(y_S - y_E)(x_{NW} - x_E) + (x_E - x_S)(y_{NW} - y_E)} \tag{1.20}$$

$$b_2 = \frac{(y_E - y_{NW})(x_P - x_E) + (x_{NW} - x_E)(y_P - y_E)}{(y_S - y_E)(x_{NW} - x_E) + (x_E - x_S)(y_{NW} - y_E)} \tag{1.21}$$

$$b_3 = 1 - b_1 - b_2 \tag{1.22}$$

The point is within or on the triangle if the following conditions all hold simutaneously:

$$0 \leq b_1 \leq 1 \qquad 0 \leq b_2 \leq 1 \qquad 0 \leq b_3 \leq 1 \qquad b_1 + b_2 + b_3 = 1 \tag{1.23}$$

So if the coordinates of a warehouse are within or on the triangle, a penalty equal to 1000 is added to the cost function. The more warehouses out of bounds for solutions with $M = 2$ or $M = 3$, the higher the total penalty.

Chapter 2

Optimization Algorithms

2.1 A Brief Introduction

The algorithms employed to attempt to solve the warehouse facility location-allocation problem are two state-of-the-art population-based metaheuristics.

Genetic Algorithms (GAs) were introduced by John H. Holland in the mid 1960s and are the first and most well studied type of Evolutionary Algorithms. This family of algorithms is inspired by evolutionary processes observed in nature. They rely on the idea of environmental pressure leading to competition and in turn natural selection of the fittest individuals. This concept is also commonly referred to as "survival of the fittest" [3, 4] .

Particle Swarm Optimization (PSO) on the other hand was proposed by J. Kennedy in 1995 [5] and belongs to the class of swarm intelligence algorithms. Its fundamental operational concept is the coordinated movement of a swarm of candidate solutions called "particles" towards more prospective areas of the search space by leveraging individual and collective information [6]

Algorithm 2.1 Main Scheme of the Genetic Algorithms

Require: Search Space X, popSize

1: $pop = initializePopulation(X)$
2: $gen = 1$
3: $overallBest = updateBest(pop)$
4: **while** $gen * popSize \leq 200000$ **do**
5: $\quad S = selection(pop)$
6: $\quad C = recombination(S)$
7: $\quad M = mutation(C)$
8: $\quad evaluate(M)$
9: $\quad pop = updatePopulation(M, pop)$
10: $\quad overallBest = updateBest(pop)$
11: $\quad gen++$
12: **end while**

2.2 Genetic Algorithms

The GAs applied for the purposes of the present project are two common variants. They follow the standard scheme of a GA and vary in the selection operator. One variant uses *Tournament Selection* while the other uses *Roulette-Wheel Selection*. The information is represented as real numbers (Real Valued Representation). The design of the algorithms presented in this section is based on Introduction To Evolutionary Computing by A.E Eiben and J.E Smith [3] and on graduate Optimization course notes [4].

The algorithm scheme presented in Alg.2.1 is very general but provides a clear idea regarding the series of steps that are executed. First, a set of vectors, the population is initialized within the search space. This population is evolved until the stop criterion is met through the processes of *Selection*, *Recombination* and *Mutation*. At the end of each iteration the new population is evaluated and the overall best value achieved is found and retained. The stop criterion is set to be a limit of 200000 functional operations. It must be noted that for the objective value to be obtained requires the real valued vectors to be transformed into integers or coordinates as described in Section 1.4 in Chapter1.

Initialization involves producing $popSize = N$ number of random vectors with all

values being within the search space, which for the present problem is $[0.0, 1.0]$.

Selection is the stage at which individual vectors are chosen from the population for Recombination to be applied. Here, this is achieved through two different methods: Roulette-Wheel Selection and Tournament Selection.

In *Roulette-Wheel Selection*, the probability of an individual being chosen is proportional to its quality in terms of *fitness* value. The fitness value here is defined through linear ranking. This means that an individual's fitness value results from the following formula:

$$\phi_i = 2 - s + 2(s-1)\frac{\rho_i - 1}{N-1} \tag{2.1}$$

where $i = 1, 2, ...N$ is the individual's identification integer, ρ_i is the position of the individual in the ranking of the objective function values achieved by all members of the population, and $s \in [1, 2]$ is the "selective pressure" parameter. For $s = 1$ we would get individuals with the same fitness value and therefore equal probability of being chosen. For a value $s = 2$, individuals with smaller (for a minimization problem) objective function evaluations receive a higher fitness value and a higher probability of being chosen. Intermediate values lead to smaller differences amongst linear fitness values. The probability of an individual being chosen is defined as:

$$ps_i = \frac{\phi_i}{\sum_{i=1}^{N} \phi_i} \tag{2.2}$$

After the fitness values and choice probabilities are calculated, members of the population are chosen to move into the Recombination process via sampling based on the probabilities. More specifically the $[0.0, 1.0]$ interval is split into N sub-intervals, each having a size equal to the corresponding probability of each individual. Then N_s individuals are chosen on the basis of which sub-interval a random number uniformly distributed in $[0.0, 1.0]$ ends up belonging to.

Tournament Selection is based on the direct comparison of a subset of $1 \leq N_{tour} \leq N$ individuals in a "tournament". N_{tour} individuals are chosen randomly from the population and the best (min) individual is chosen based on the objective function evaluations. Selective pressure is incorporated via the choice of N_{tour}. The closer N_{tour} is to the population size, the more probable it is for the best individual to be chosen.

The *Real Valued Recombination* is the crossover operation when real valued representation is used. Say $P = \{p_1, p_2, ..., p_N\}$ is the population with individuals $p_i = (p_{i,1}, p_{i,2}, ..., p_{i,n})$ $p_{i,j} \in \mathbb{R}$, $\forall j$. Also, consider the set of chosen individuals from the

Selection process $S_c = \{p'_{i,1}, p'_{i,2}, ..., p'_{i,d}\}$ with d chosen members. The real recombination between $p'_{i,1}$ and $p'_{i,2}$ produces an offspring $O = (O_1, O_2, ..., O_n)$ where each component is chosen in accordance to the following:

$$O_j = R_j\, p'_{i,1j} + (1 - R_j)\, p'_{i,2j} \quad j = 1, 2, ..., n \tag{2.3}$$

where R_j is a uniformly distributed number in $[-\delta, 1 + \delta]$. The number $d > 0$ is used to help avoid the gradual shrinking of the area within which offspring are produced due to the concentration of random components within the search space. Individuals are chosen from S_c if a uniformly distributed random number in $[0.0, 1.0]$ is less than or equal to the recombination rate $c\_prob \in (0, 1)$.

The *Real Valued Mutation* leverages a continuous distribution. Say $p''_i = (p''_{i,1}, p''_{i,2}, ..., p''_{i,n})$ a member of the set C. This set contains only individuals from S_c for which a uniformly distributed random number in $[0.0, 1.0]$ is less than the mutation rate $m\_prob \in (0, 1)$. A component is evolved by an addition of a uniformly distributed random value as described below:

$$p''_{i,j} = p''_{i,j} + z_j \quad where \quad z_j \sim U[-\alpha, \alpha] \tag{2.4}$$

The parameter $\alpha > 0$ is chosen as $\alpha = 1 - p''_{i,j}$ in order to ensure it remains within the search space. It must be highlighted that even if this measure is used to ensure the mutated values produced remain within the search space, a safeguard has been added for each component value, regardless of if it is mutated or not, that works as follows: if a value is less than the lower bound of the search space (0.0) it is clamped to that bound. Similarly if a value is higher than the high bound of the search space (1.0) the value is clamped to the high bound.

Finally, when it comes to the population replacement step, both variants of the algorithm work with full population replacement. This means that at the end of each iteration the new population produced fully replaces the previous population. This is why N_s is chosen to be equal to N. This choice was made to favor exploration of the search space.

2.3 Particle Swarm Optimization

The employed PSO variant follows that in [6] and [7] as well as my diploma thesis [8] and course notes [4]. It is a variant of the contemporary standard PSO model, that

leverages a restarting strategy, aiming to alleviate stagnation and entrapment around local minima as well as invigorate the searcg. First, the swarm and the corresponding velocities are randomly initialized and an initial cost function evaluation is made for each particle Then, each particle is updated and limited within its bounds until the stopping criterion is met. Before obtaining the objective function value for each individual, the real valued vector's components need to be transformed into integers or coordinates as described in Section 1.4 in Chapter1. A condition for restarting is checked at the beginning of each iteration.

Let $S = \{x_1, x_2, ..., x_N\}$ be the swarm containing the set of N candidate solutions. Each candidate solution $x_i = (x_{i1}, x_{i2}, ..., x_{in})^\top$ is called a *particle*. Each particle x_i has a *velocity* $v_i = (v_{i1}, v_{i2}, ..., v_{in})^\top$, which it uses to update its position at each iteration.

Each particle also assumes a set of other particles with which, it exchanges information. This is called its *neighborhood* and has a size q. For a particle i its neighborhood is defined as :

$$NB_i = (x_{z_1}, x_{z_2}, ..., x_{z_q})^\top$$

$\{z_1, z_2, ..., z_q\} \subseteq \{1, 2, .., N\}$ represents the neighbors' indices. The *neighborhood size* is denoted as $|NB_i|$.

There are two types of neighborhood topology schemes defined for standard PSO, namely the *fully connected topology* and the *ring topology*. If a PSO implementation uses a *fully connected topology* setting it is referred to as *gbest (global best)*, while if it uses *ring topology* it is referred to as *lbest (local best)*. In *gbest*, $NB_i = S$, the neighborhood includes all other particles in the swarm. In *lbest*, $NB_i = \{x_{i-r}, x_{i-r+1}, ..., x_{i-1}, x_i, x_{i+1}, ..., x_{i+r-1}, x_{i+r}\}$, where r is the *neighborhood radius* that defines the neighborhood's size, and requires a tuning process.

The neighborhood is used to locate the neighbor with the smallest objective function evaluation. That particle is $p_{g_i} = \underset{j \text{ such that } x_j \in NB_i}{\arg\min} f(p_j)$, where g_i is its index. This particle is then used to update the velocity of x_i.

As each particle is updated across iterations, it retains the best position it has reached as $p_i = (p_{i1}, p_{i2}, ..., p_{in})^\top$. The best position for x_i is the one where the smallest objective value has been achieved, as described by the following condition:

$$p_i^{G+1} = \begin{cases} x_i & if\ f(x_i^{G+1}) < f(p_i^G) \\ p_i & otherwise \end{cases} \tag{2.5}$$

where, $i = 1, ..., N$ and G is the generation counter.

At each iteration, the velocity and position of a particle are updated according to the following equations:

$$\begin{aligned} v_{ij}^{(G+1)} &= \chi[v_{ij}^{(G)} + r_1 c_1 (p_{ij}^{(G)} - x_{ij}^{(G)}) + r_2 c_2 (p_{g_i j}^{(G)} - x_{ij}^{(G)})] \\ x_{ij}^{(G+1)} &= x_{ij}^{(G)} + v_{ij}^{(G+1)} \end{aligned} \tag{2.6}$$

where, $i = 1, 2, ..., N$, $j = 1, 2, ..., n$, G is the iteration counter and g_i is the index of the best neighboring particle. r_1, and r_2, are the random numbers uniformly distributed in [0.0,1.0], c_1 is the cognitive parameter, c_2 is the social parameter, and χ is the constriction coefficient. These parameters control the swarm's dynamic and their default values are (Clark and Kennedy 2002) $c_1 = c_2 = 2.05$ and $\chi = 0.729$.

In order to preserve the feasibility of the particles in the search space, if the update of Eq.(2.6) results in values outside of the search space, they are clamped to the relevant boundary. More specifically, if $A = [l_1, u_1] \times [l_2, u_2] \times ... \times [l_n, u_n]$ is the search space, then:

$$x_{ij}^{G+1} = \begin{cases} l_i & if \quad x_{ij}^{G+1} < l_i \\ u_i & if \quad x_{ij}^{G+1} > u_i \\ x_{ij}^{G+1} & otherwise \end{cases} \tag{2.7}$$

for all $i, j = 1, 2, ..., n$. In the implementation, it is assumed that each particle lies in $A = [0, 1]^n$, and thus:

$$x_{ij}^{G+1} = \begin{cases} 0 & if \quad x_{ij}^{G+1} < 0 \\ 1 & if \quad x_{ij}^{G+1} > 1 \\ x_{ij}^{G+1} & otherwise \end{cases} \tag{2.8}$$

for $i, j = 1, 2, ..., n$. Similarly, velocities also require clamping within specific boundaries as they control the step size of the particles. If not bounded, it is very likely that a phenomenon called *swarm explosion* will occur, i.e, velocities may uncontrollably increase in magnitude and push the particles to spread out far away from each other. In turn, this prohibits the convergence towards a good solution (swarm divergence).

In order to define a velocity boundary, a maximum velocity term must be chosen. Retaining the search space definition described in Eq.(2.7), the maximum absolute

value per velocity component is :

$$v_{max} = \frac{u_i - l_i}{k} = \alpha \cdot (u_i - l_i) \tag{2.9}$$

Algorithm 2.2 Main Scheme of Particle Swarm Optimization

Require: Search Space X, swarmSize α

1: $swarm = initializeSwarm(X)$
2: $velocities = initializeVelocities(v_{max} = \alpha(X[1] - X[0]))$
3: $evals = evaluate(swarm)$
4: $bestPositions = swarm$
5: $gen = 1$
6: $overallBest = updateBest(bestPositions)$
7: $genBest = overallBest$
8: $prevBest = genBest$
9: $noImprovement = 0 \;\; restartCounter = 0$
10: $restartIntervals = [0.05, 0.1, 0.2, 0.4, 0.8]$
11: **while** $gen * swarmSize \leq 200000$ **do**
12: $checkRestart(noImprovement, restartCounter, restartIntervals)$
13: $velocities_{new} = updateVelocities(velocities, swarm, bestPositions, \chi, c_1, c_2)$
14: $checkVelocityBounds(velocities_{new}, v_{max})$
15: $velocities = velocities_{new}$
16: $particles_{new} = updateParticles(velocities_{new}, swarm)$
17: $checkParticleBounds(particles_{new}, X)$
18: $swarm = particles_{new}$
19: $evals = evaluate(swarm)$
20: $bestPositions = updateBestPositions(swarm)$
21: $genBest = getGenBest(evals)$
22: $overallBest = updateBest(bestPositions, genBest)$
23: $checkNoImprovement(noImprovement, prevBest, genBest)$
24: $gen++$
25: **end while**

For each component, v_{max} is expressed as a percentage of the search space range. The value of α affects the convergence speed of the swarm. A common initial (uninformed) choice is $\alpha = 0.5$, but other values may be tried during a tuning phase. Eventually, the velocities are clamped as follows:

$$v_{ij}^{G+1} = \begin{cases} -v_{max} & if \quad v_{ij}^{G+1} < -v_{max} \\ v_{max} & if \quad v_{ij}^{G+1} > v_{max} \\ v_{ij}^{G+1} & otherwise \end{cases} \tag{2.10}$$

for $i, j = 1, 2, ..., n$. For the specific implementation here, this is expressed as:

$$v_{ij}^{G+1} = \begin{cases} -\alpha & if \quad v_{ij}^{G+1} < -\alpha \\ \alpha & if \quad v_{ij}^{G+1} > \alpha \\ v_{ij}^{G+1} & otherwise \end{cases} \tag{2.11}$$

The restarting strategy, is based on evaluating whether the number of non-improving iterations reaches a percentage of the overall allowed function evaluations (*budget*), which is 200000 here, or not. If it has reached one of the percentage marks, the swarm and its velocities are re-initialized and the percentage is doubled. The first restart takes place if the number of no-improvements becomes equal to 5 percent mark. Then, the no-improvements counter is set to zero so that the next restart will take place if the number of no-improvements now reaches a number equal to 10 percent mark, and so on until the final restart, which may take the 40 percent mark. The percentage cannot surpass the value, however in the coded implementation, an 80 percent mark is present since the condition is checked at the beginning of the loop and the restart counter must point somewhere in the list containing the percentage marks, without resulting in problematic outcomes.

The general scheme of the algorithm is presented in Alg.2.2.

2.4 Implementation Information

All implementations have been coded in Python (Version 3.11.5) and executed on a laptop equipped with an Intel(R) Core(TM) i7-8550U CPU (@ 1.80GHz 1.99 GHz) and 8GB of RAM on the Spyder IDE.
The corresponding .py files are:

1. Genetic Algorithms : ga.py
2. Particle Swarm Optimization : pso.py
3. Cost Function and Helper Functions: problem_functions.py

Chapter 3

Experimental Results

3.1 Initial Algorithm Parameterization

$Selective\ Pressure:$	$s=2$	$s=2$	$s=2$	$N_{tour}=25$	$N_{tour}=25$	$N_{tour}=25$
$N=N_s$	50	100	200	50	100	200

Table 3.1: GA Parameters

nb_r	0	0	0	5	5	5
N	50	100	200	50	100	200

Table 3.2: PSO Parameters

Parameter tuning is a very computationally and temporally expensive process. Due to such constraints and the extensive amount of possible parameter combinations a brief preliminary round led to parameter choices that evidently seemed to perform

better out of the mostly arbitrarily chosen values. However, while some variables were found and fixed to values, others like the population or swarm size, as well as the variant of each method, need to be evaluated through the application of statistical methods.

For the GA with the Roulette Wheel selection operator, these parameters are presented in Table 3.1 on all cases where the selective pressure is $s = 2$. The fixed parameters are : $c\_prob = 0.5$, $m\_prob = 0.2$, $N_s = N$ for full population replacement and $\delta = 0.25$. Similarly, the same table holds the same information for the Tournament Selection GA variant on the cases where the selective pressure is $N_{tour} = 25$. The rest of the parameters are the same as the fixed ones for the Roulette Wheel variant. The parameters up for futher evaluation for PSO are disclosed in Table 3.2. Values for the neighborhood radius nb_r indicate if the PSO is the *gbest* variant ($nb\_r = 0$) or the *lbest* variant ($nb\_r = 5$).

The parameters that were fixed for PSO include the *standard PSO parameters* ($\chi = 0.729$ and $c_1 = c_2 = 2.05$ and $\alpha = 0.005$. The α value was chosen to be quite small in comparison to typically chosen values like 0.5 or 0.1. This choice was tried under the assumption that the coordinate form results are quite sensitive and that very large steps within the search space $[0.0, 1.0]$ lead to much bigger steps in the geographical space post transformation into coordinate form. Indeed, larger values were observed to provided results that were larger than the results of the present choice for α by a magnitude of at least about 40.

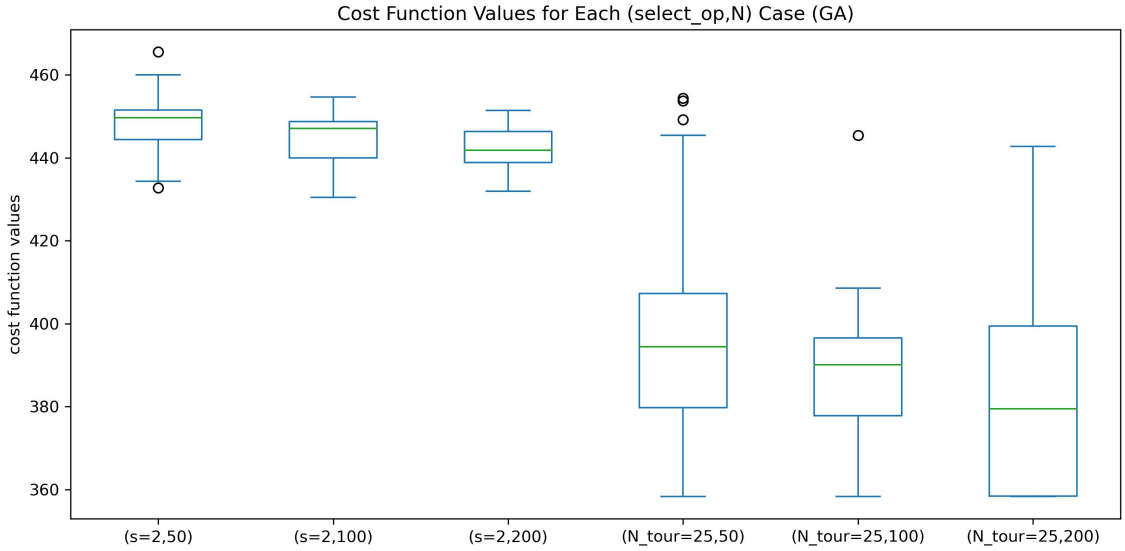
3.2 Final GA Parameterization

Selective Pressure	N	Cost Value Mean	Cost Value Median	Cost Value St.Dev	Cost Value Min	Cost Value Max
$s = 2$	50	448.420170	449.651135	7.626994	432.718523	465.527951
$s = 2$	100	445.464134	447.033719	6.052338	430.476010	454.623076
$s = 2$	200	441.724890	441.783015	4.835851	431.893735	451.400043
$N_{tour=25}$	50	398.114120	394.480284	27.445089	358.367882	454.389011
$N_{tour=25}$	100	388.492490	390.132807	18.368451	358.367887	445.416861
$N_{tour=25}$	200	383.182939	379.444482	23.673572	358.367881	442.699313

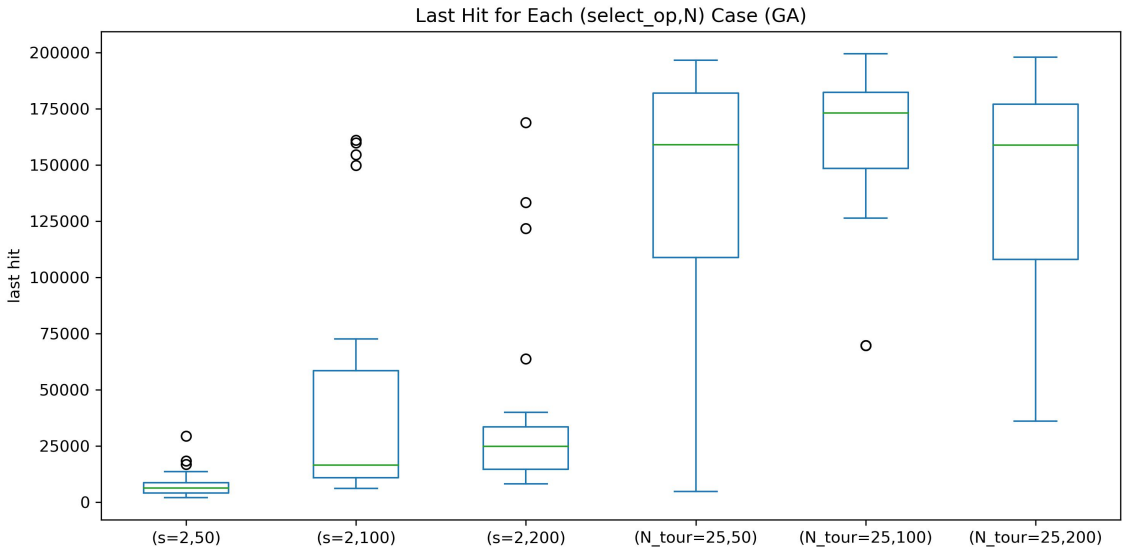
Table 3.3: Cost Value Statistical Table (GAs)

Selective Pressure	N	Last Hit Mean	Last Hit Median	Last Hit St.Dev	Last Hit Min	Last Hit Max
$s = 2$	50	7998.640000	6368	5995.452729	2099	29445
$s = 2$	100	44690.400000	16591	51971.629686	6071	161070
$s = 2$	200	37541.720000	24785	40780.208871	8106	168947
$N_{tour} = 25$	50	135860.400000	158950	57698.816773	4831	196677
$N_{tour} = 25$	100	160648.680000	173084	33711.495246	69669	199485
$N_{tour} = 25$	200	139432.400000	158801	48918.221486	36108	197988

Table 3.4: Last Hit Statistical Table (GAs)



(a) Cost Values Boxplot



(b) Last Hit Boxplot

Figure 3.1: Cost Values and Last Hit Boxplots for GAs

Variable	(2,50)	(2,100)	(2,200)	(25,50)	(25,100)	(25,200)
Cost Val	0	0	2	3	3	5
Last Hit	5	3	3	0	0	0

Table 3.5: GAs : KS Test Null Hypothesis Rejection Counts

To find the best out of the six parameterizations for the GA variants, 25 executions were run for each one. The results were collected and evaluated through the use of boxplots, basic statistical measures and statistical testing for the *Cost Function Value* and *Last Hit* variables. The statistical test applied was the Kolmogorov-Smirnov One Tailed Test (KS Test) which results in a null hypothesis rejection for the parameterizations that are both different from the others, for a significance level of 0.05, and "better" than them. This means their result distributions contain values with overall smaller magnitudes and that that difference is statistically significant.

Looking at the boxplots for *Cost Values* on Figure 3.1, it is obvious that the *Tournament Selection* GA variant results in smaller objective function values than the *Roulette-Wheel* variant. This is further corroborated by the corresponding statistical Table 3.3. The statistical table also indicates in detail that the case that performs best is the one for $N_{tour} = 25$ and $N = 200$ which achieves the smallest mean, median, min and max. The standard deviation is the second smallest amongst the three Tournament Selection variant parameterizations. It is quite large with respect to the magnitude of objective function evaluations. This could be related to the nature of the problem and the transformations used to bring coordinate values into the real search space and the other way around. Finally, the KS Test null rejection counts provided in Table 3.5 confirm that the $N_{tour} = 25$ and $N = 200$ settings are indeed the ones that lead to the best overall parameterization, given it has the highest score.

The boxplots for the *Last Hit* variable on Table ?? indicate that the smallest last hit values correspond to the *Roulette-Wheel* variant's parameterizations. It must be noted that besides a few outliers, these last hit values occur very early in comparison with the overall budget of 200000 function evaluations. Information on the *Cost Values* statistical Table 3.3, as well as the related *Last Hit* statistical Table 3.4 provide more insight with regard to what that may mean. The parameterization with smallest *Last Hit* values, the case for $s = 2$ and $N = 50$ is the one resulting in the highest *Cost Values*. The *Cost Value* results amongst *Roulette-Wheel* parameterizations are very close and with similar standard deviations. The KS Test results in Table 3.5 further confirm the difference observed between the *Roulette-Wheel* and *Tournament Selection* variants as

well as the parameterization combination for $s = 2$ and $N = 50$ resulting in the overall smallest Last Hit values. Taking all of the above into account, the Roulette wheel variants seem to achieve a value of around 445 very early in the computation and then proceeds to get stuck there, unable to take advantage of the available resources to further improve its approximation.

Given the primary goal is objective function minimization, the best parameterization is chosen on the basis of the *Cost Value* results. The obvious winner is the parameter combination for $N_{tour} = 25$ and $N = 200$. Furthermore, the *Roulette-Wheel* parameterizations may not be considered good alternatives given the difference of the chosen Tournament Selection with the best Roulette Wheel variant with regard to cost function value ($s = 2$ and $N = 200$) is at a magnitude of about 60 for the median case and about 70 for the minimum achieved case.

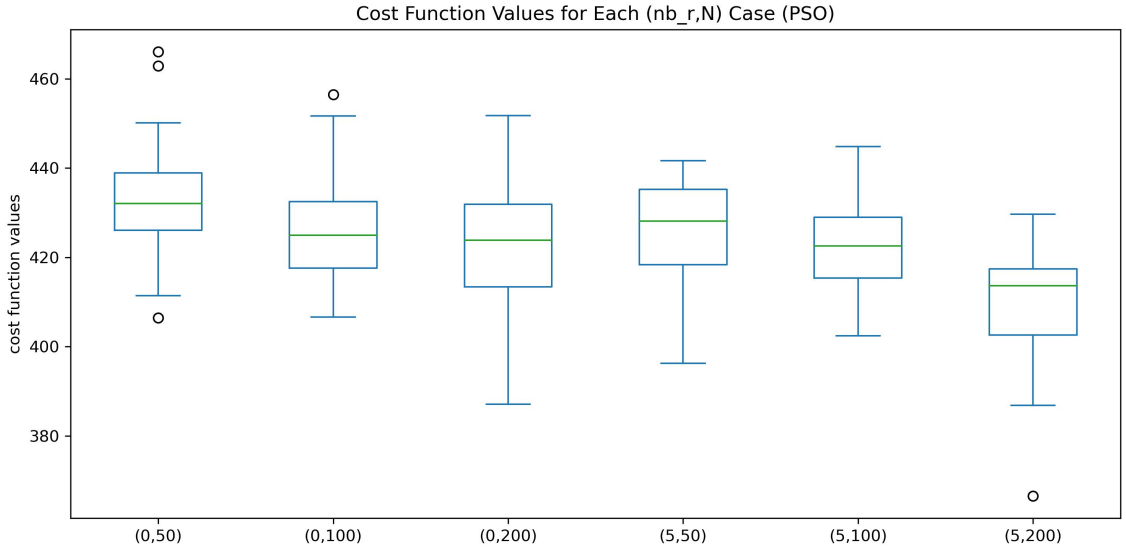
3.3 Final PSO Parameterization

nb_r	N	Cost Value Mean	Cost Value Median	Cost Value St.Dev	Cost Value Min	Cost Value Max
0	50	433.413111	432.035678	13.881304	406.458291	466.066884
0	100	425.656518	424.942921	12.289021	406.649806	456.491945
0	200	421.929134	423.887200	15.200282	387.125236	451.772748
5	50	425.763775	428.101784	12.192127	396.304405	441.646804
5	100	423.121611	422.542661	11.935331	402.408151	444.847491
5	200	410.326940	413.673383	13.828622	366.565460	429.634918

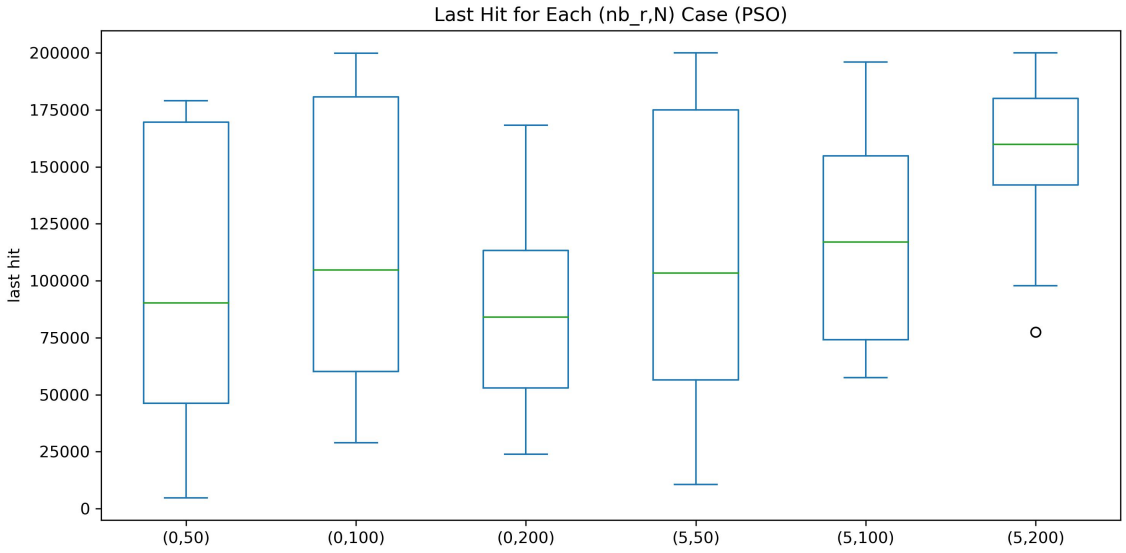
Table 3.6: Cost Value Statistical Table (PSO)

nb_r	N	Last Hit Mean	Last Hit Median	Last Hit St.Dev	Last Hit Min	Last Hit Max
0	50	106608.760000	90262	61096.385354	4747	179045
0	100	109297.400000	104769	57654.132204	28962	199905
0	200	85945.560000	84060	42531.585006	23803	168243
5	50	114112.000000	103317	54120.795883	10661	199999
5	100	117209.760000	116953	43496.262609	57515	196062
5	200	154642.120000	159847	35023.542621	77412	199996

Table 3.7: Last Hit Statistical Table (PSO)



(a) Cost Values Boxplot



(b) Last Hit Boxplot

Figure 3.2: Cost Values and Last Hit Boxplots for PSO

Variable	(0,50)	(0,100)	(0,200)	(5,50)	(5,100)	(5,200)
Cost Val	0	1	1	0	1	5
Last Hit	2	1	4	1	1	0

Table 3.8: PSO : KS Test Null Hypothesis Rejection Counts

The same experimentation and evaluation process that was applied for the GAs was also applied for the different parameterizations of PSO.

Looking at the boxplot for the *Cost Value* variable on Figure 3.2 it is evident that the *lbest* PSO with a neighborhood $nb\_r = 5$, for $N = 200$ outperforms the other

parameterizations by achieving a median and a relatively tight box around it that is lower than the rest. Generally all boxplots are neither too narrow nor too wide around their medians indicating a slight variability in the central part of the distribution but are otherwise robust in their performance. All of the above are confirmed in the corresponding statistical Table 3.6. Further evidence is provided by the KS test results on Table 3.8 where that parameterization achieves the highest score.

With regard to the *Last Hit* variable, according to the boxplots on Figure 3.2, the smallest median and overall values seem to be achieved by the gbest ($nb\_r = 0$) PSO for $N = 200$. It is noteworthy that the box is also tighter around the median indicating lower variability in the central part of the distribution than most other parameterizations. The most robust result is given by the *lbest* PSO with $nb\_r = 5$, which also happens to be the variant that has resulted in the best performance with regard to the *Cost Value* variable. The smallest value for *Last Hit* being achieved by the *gbest* ($nb\_r = 0$) PSO for $N = 200$ is also confirmed by the corresponding statistical Table 3.7 and the KS Test results on Table 3.8.

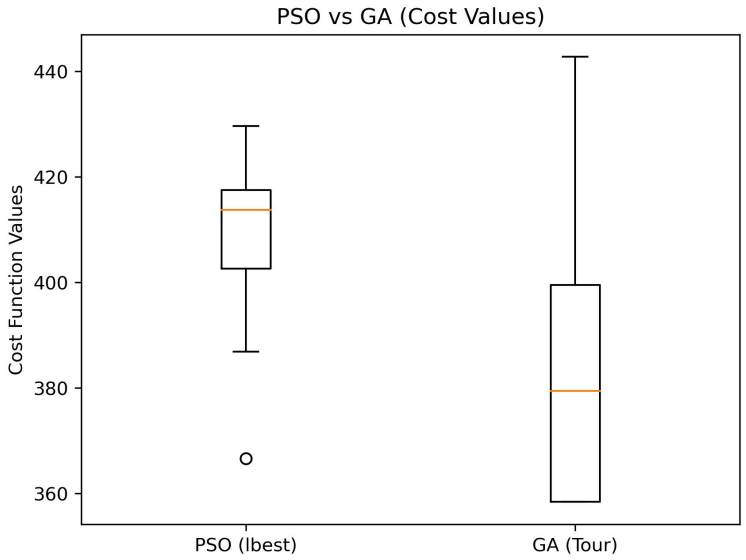
According to all of the above it is clearly deduced that the best parameterization for PSO, with regard to the primary performance variable, *Cost Value*, is the *lbest* with $nb\_r = 5$ and $N = 200$. This result is also quite robust with regard to both variables of interest.

3.4 GA vs PSO

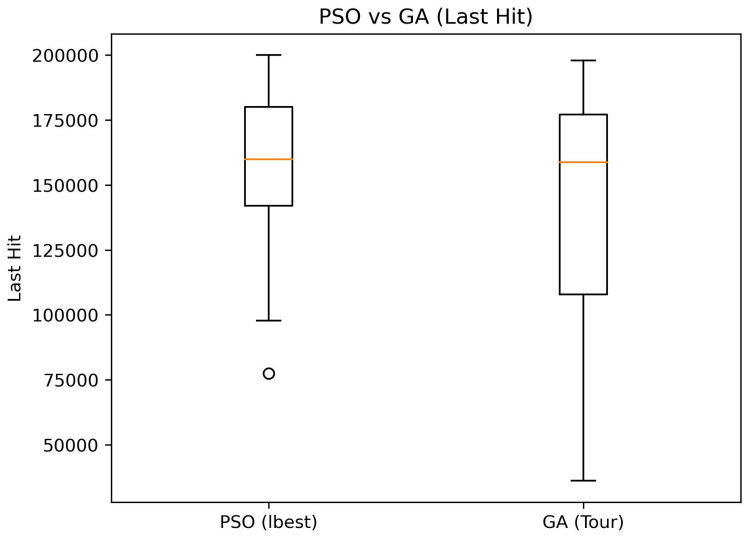
GA (Tour)	$N_{tour} = 25$	$N = 200$	$N_s = N$	$c\_prob = 0.5$	$m\_prob = 0.2$	$\delta = 0.25$
PSO (lbest)	$nb\_r = 5$	$N = 200$	$\alpha = 0.005$	$\chi = 729$	$c_1 = 2.05$	$c_2 = 2.05$

Table 3.9: Best Parameterizations for GA and PSO

The full best parameterizations for GA and PSO are provided in Table 3.9. In this section, they are compared with each other with the goal of deciding the best performing method on the facility location-allocation problem. The comparison is implemented through the same methods of comparison that were used for the GA ans PSO parameterizations separately : boxplots, statistical measures and statistical testing.



(a) Cost Values Boxplot



(b) Last Hit Boxplot

Figure 3.3: Cost Values and Last Hit Boxplots (GA vs PSO)

Variable	**PSO (lbest)**	**GA (Tour)**
Cost Val	0	1
Last Hit	0	0

Table 3.10: GA vs PSO : KS Test Null Hypothesis Rejection Counts

Starting with the *Cost Value* variable, from the boxplots in Figure 3.3 it is clear that the Tournament Selection GA results in much smaller values than the *lbest* PSO.

Experiment ID	Cost Value	Feasibility	Last Hit
1	358.3678826657905	1	147743
2	379.4450857875246	1	174142
3	389.9391792543482	1	162507
4	367.4845285121478	1	177093
5	428.3277654767023	1	138383
6	395.3850291698011	1	168505
7	358.4372597421212	1	63902
8	363.4108396992541	1	152356
9	379.4444818973665	1	49944
10	383.4084095464337	1	180188
11	368.2948441181675	1	43899
12	358.3678814253733	1	107937
13	379.18545669796055	1	183729
14	379.4484416203725	1	179752
15	358.43729343191546	1	109732
16	408.00355445115326	1	98378
17	399.4640752851046	1	168524
18	401.5066343771231	1	189392
19	358.4372643738172	1	86946
20	379.36123772211363	1	170387
21	416.50262045627886	1	197988
22	442.6993133928429	1	158801
23	409.3398690866934	1	186574
24	358.43726267956913	1	36108
25	358.43727572893096	1	152900

Table 3.11: Full Results for the Best GA Tournament Selection

However, PSO provides more robust results while the GA has a wide box. Despite that, the the interquartile range for the GA is toward the lower end containing better values than that of PSO. The statistical Tables 3.3 and 3.6 provide more details that help put numerical values to the observed results. The deduced difference is proven to be statistically significant by the one tailed KS test, with the GA being the best performing algorithm, as shown on Figure 3.10.

Looking at the boxplots for the *Last Hit* variable on Figure 3.3, there is significant overlap between the two boxplots with PSO being more robust. Both algorithms have *Last Hit* values above the mark of half of the budget. PSO's robustness is most likely attributed to the incorporated restart strategy, since if it gets stuck on a Cost Value 40% through the budget, a restart will occur, rejuvenating the search. While, the GA has a wider box, the median is almost identical to that of PSO's and the distribution

is skewed toward the budget limit. The statistical Tables 3.3 and 3.6 do not give clear enough information that enables us to discern if any observed differences are important. The one tailed KS Test is not helpful either, given it tests for difference and smaller values simultaneously. This is why the Wilcoxon Rank Sum test is applied to further examine this. The resulting *p-value* is equal to 0.3567187999388043, which is much greater than the chosen significance level of 0.05. It can be concluded that there is no statistically important difference with respect to the *Last Hit* variable. Overall, they tend to use up the majority of the available budget, indicating convergence to a good local minimum approximation or possible stagnation.

The best performing algorithm is found to be the *Tournament Selection* GA with the parameter settings provided in Table 3.9. The full results of the experiments for this case are provided on Table 3.11.

3.5 Implementation Information

The data used and provided for this section were produced via Jupyter Notebook scripts using popular Python libraries like Pandas, Numpy and Scipy. The folders corresponding to each section are listed below:

1. Final GA Parameterization 3.2 : ga_results, ga_stat_scripts
2. Final PSO Parameterization 3.3 : pso_results, pso_stat_scripts
3. GA vs PSO 3.4: ga_vs_pso

Chapter 4

Final Results

Experiment ID	Latitude w_1	Longitude w_1	Latitude w_2	Longitude w_2
1	39.610267118058495	20.85782107850307	39.67409739099832	20.835734692849694
7	39.61184360293569	20.85735184965597	39.67494175745181	20.835333273492626
12	39.61026844738384	20.857811552150633	39.67410208849791	20.835728555968924
15	39.61186748288785	20.857359783415287	39.67494164374507	20.83533779099476
19	39.611845417877525	20.857354860792018	39.67494826307262	20.83534133703354
24	39.61184490100549	20.857354698241238	39.67494742127474	20.835341048692772
25	39.61185137524392	20.857350417043396	39.67495179353588	20.835335978278792

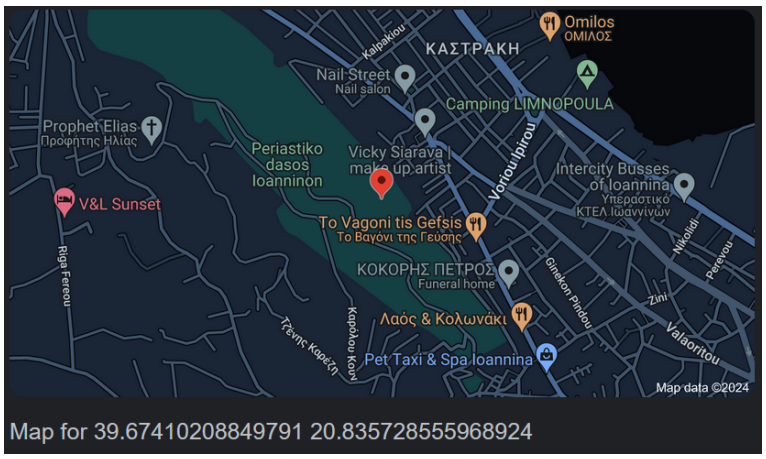
Table 4.1: Coordinates for the Warehouses for Cost Values around 358

Experiment ID	Latitude w_1	Longitude w_1	Latitude w_2	Longitude w_2
2	39.70768505940675	20.794203126935138	39.661978087694486	20.844004001392417
9	39.707689491516454	20.79420525889027	39.661983611855526	20.844004366549722
13	39.60992898029574	20.855883482830095	39.67234743858961	20.836911585107654
14	39.707682877563755	20.794170990308753	39.66198678423629	20.844004707015294
20	39.608631970413995	20.860394159434385	39.67234086529716	20.83672955749186

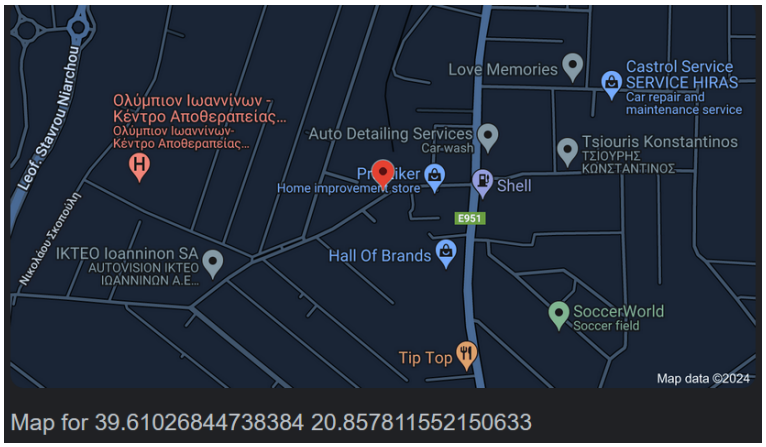
Table 4.2: Coordinates for the Warehouses for Cost Values around 379

Examining Table 3.11 in Chapter 3 for the *Tournament Selection* GA that was found to be the best performing method, it can be observed that if we look only at the whole number part, some values are repeated. Specifically the minimum integer value is 358 and is repeated 7 times (Experiments 1, 7, 12, 15, 19, 24, 25). This is a strong indication that a local minimum's approximations have been found. The next most

repeated value, looking only at the whole number part is 379 (Experiments 2, 9, 13, 14, 20). The corresponding coordinates for the groups of values are provided in Tables 4.1 and 4.2 repspectively. It must be noted that all resulting solutions are feasible solutions with two warehouses ($M = 2$). Feasibility is set to 1 when there are no awarded penalties for the resulting solution. This means that both warehouses' coordinates do not violate the lake constraint.



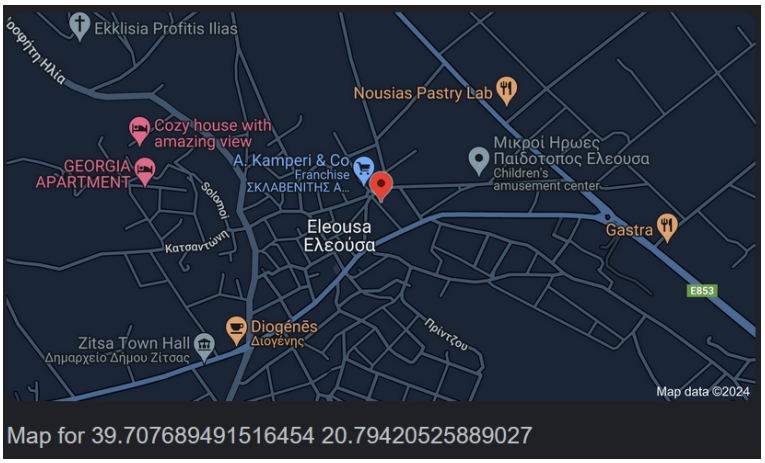
(a) Warehouse 1



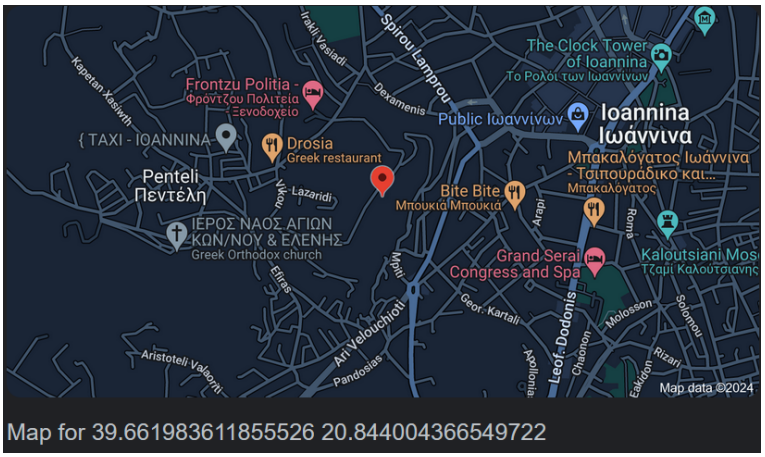
(b) Warehouse 2

Figure 4.1: Warehouse Locations for Cost Value = 358.3678814253733 (zoom)

For the 358 case, it is observable that the coordinate pairs, at first glance, are very close to each other. It is confirmed through individual lookup of each coordinate pair on Google Maps that they all point to almost exactly the same point. The smallest *Cost Value* is 358.3678814253733 (Experiment 12) and the corresponding coordinate's result is depicted in Figure 4.1



(a) Warehouse 1



(b) Warehouse 2

Figure 4.2: Warehouse Locations for Cost Value = 379.4444818973665 (zoom)

For the 379 case, it is observable that while the coordinates are quite close, they do present some variability. Through individual lookup of each coordinate on Google Maps the coordinates of experiments 13 and 20 point to the same locations found for the 358 case, while the rest point to different locations. The smallest Cost Values are 379.18545669796055 and 379.36123772211363 which unsurprisingly correspond to experiments 13 and 20 respectively. From the experiment results that remain, the minimum cost value for this case is 379.4444818973665 and the corresponding coordinates' results are depicted in Figure 4.2.

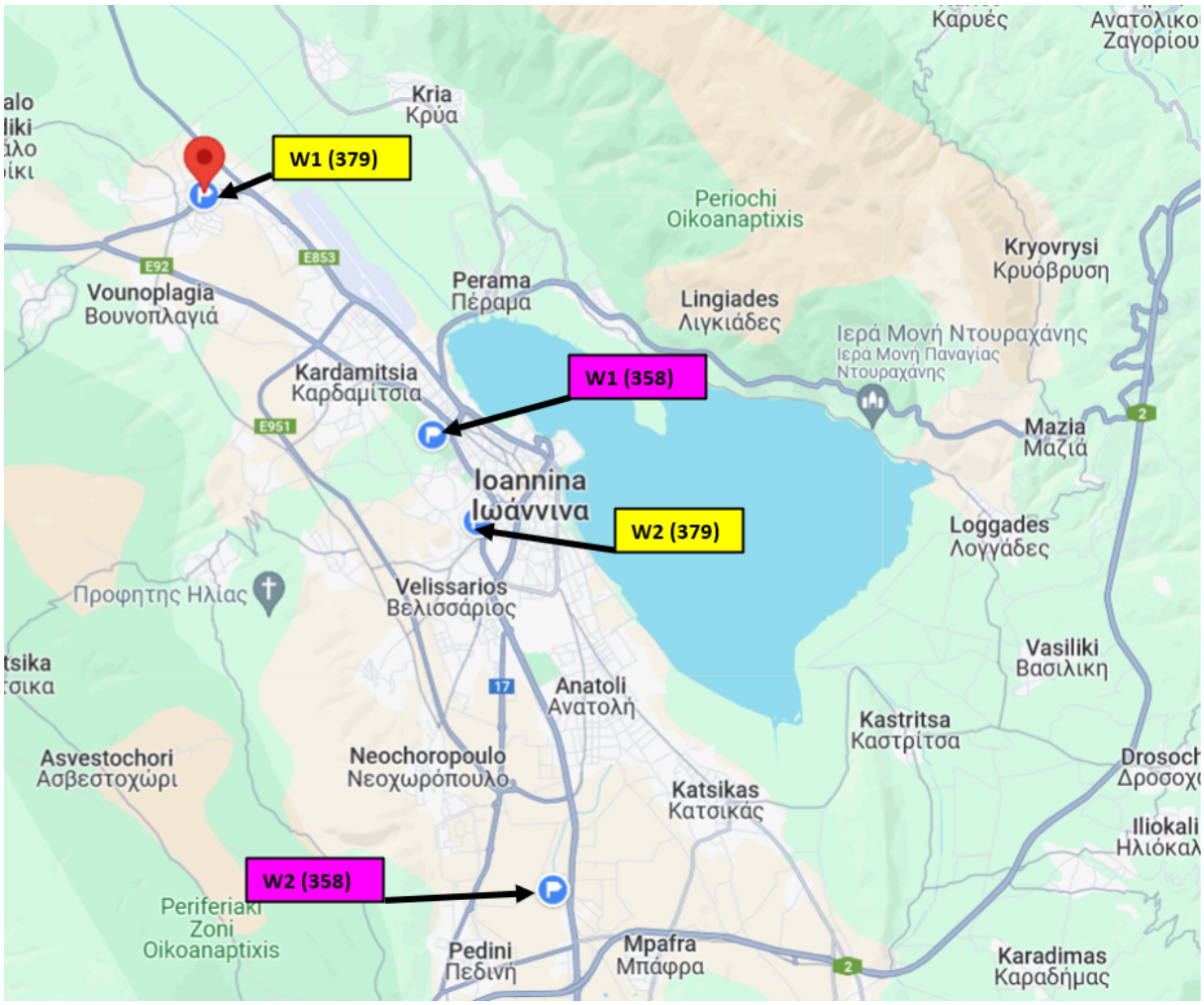


Figure 4.3: Warehouse Locations On The Map

The two solutions seem to be vastly different. So, to help attain a better understanding of where they fall on the map, the warehouses for case 358 are noted with magenta labels and the warehouses for case 379 are noted with yellow labels on the map depicted on Figure 4.3. Cross referencing with the geogoraphical distribution of customers depicted in Chapter 1 Figure 1.1, and taking into consideration the lack of many realstic constraints like a limit on available vehicle, the solutions found seem intuitively rational.

Ultimately, the final solution is the one that corresponds to the smallest cost value found, meaning the warehouses are at the locations found for the case of the cost value being equal to 358.3678814253733. The second solution could be utilized as an alternative if any insurmountable problem is encountered with the locations found to correspond to the primary solution.

Bibliography

[1] Webpage, https://en.wikipedia.org/wiki/Barycentric_coordinate_system.

[2] Webpage, https://www.scratchapixel.com/lessons/3d-basic-rendering/ray-tracing-rendering-a-triangle/barycentric-coordinates.html.

[3] A. Eiben and J. Smith, *Introduction to Evolutionary Computing*, ser. Natural Computing Series. Springer Berlin Heidelberg, 2015. [Online]. Available: https://books.google.gr/books?id=bLIYCgAAQBAJ

[4] K. Parsopoulos, "Optimization, graduate course notes 2023-2024."

[5] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, vol. 4, 1995, pp. 1942–1948 vol.4.

[6] D. Souravlias, K. Parsopoulos, I. Kotsireas, and P. Pardalos, *Algorithm Portfolios: Advances, Applications, and Challenges*, 12 2020.

[7] K. Parsopoulos and M. Vrahatis, *Particle Swarm Optimization and Intelligence: Advances and Applications*, 01 2010.

[8] N. E. Sextou, "Optimized multi-criteria decision analysis through median ranking and analytical hierarchy process," undergraduate Diploma Thesis 2023.