Optimization Techniques for Multi Cooperative Systems MCTR 1021
Mechatronics Engineering
Faculty of Engineering and Materials Science
German University in Cairo

Multi-UAV Target Exploration

By

Team 44

Ahmed Daw
Amr Mahgoub
Nabil Hassan
Mohamed Ashraf
Mohamed Yasser
Abdulrahman Manea

Course Team

Assoc. Prof. Dr Omar Shehata

Chapter 1

Literature Review

The field of optimizing cooperative UAV area exploration has been extensively researched throughout the last decade, in terms of increasing explored area and reaching desired targets in minimal running time.

Xu et al, 2020 explored the field cooperative path planning for multi-UAV system as used in military and civil applications, such as target striking and regional surveillance, in order to increase the probability of task completion and minimizing the risk of the UAVs being captured, by proposing an optimized cooperative path planning for multi-UAV system optimizing a combination of fuel/energy consumption and risk of capture, under the constraints of exploration area and time, establishing a multi constraint objective optimization model, followed by the testing of multiple optimization algorithms, such as an improved version of the grey wolf optimization algorithm, where the algorithm is improved in three main aspects, population initialization, decoy factor updating and individual position updating, simulation results showcase that the algorithm has optimized the generated paths by lowering the path cost, while reaching faster convergence when compared to other tested techniques [1].

In their research, Qiming et al, 2021 have evaluated the use of different meta-heuristic optimization techniques in solving the UAV swarm search task. In their research, the goal was to detect targets as fast as possible, while using the least number of UAVs, in another simulation environment, the goal was to maximize the search area with the UAV swarm. The search area was divided into discrete cells of MxN size and the used UAV model was assumed to be the basic UAV model by treating each UAV as a particle in a 2-Dimensional space. Some of the constraints taken into consideration were the curve angle θ, flight speed v, flight altitude h, the table shown in Table: 1.1 provides more constraints.

Constraint name	Expression	Meaning
Curve angle constraint	$\theta_{\min} \leq \theta_i \leq \theta_{\max}$	θ_i: turning angle of the UAV at time i $\theta_{\min}$: minimum turning angle $\theta_{\max}$: maximum turning angle
Flight speed constraint	$v_{\min} \leq v_i \leq v_{\max}$	v_i: speed of the UAV at time i $v_{\min}$: minimum speed of UAV $v_{\max}$: maximum speed of UAV
Flight altitude constraint	$h_{\min} \leq h_i \leq h_{\max}$	h_i: altitude of the UAV at time i $h_{\min}$: minimum flight altitude $h_{\max}$: maximum flight altitude
Climb angle constraint	$0 \leq \alpha_i \leq \alpha_{\max}$	α_i: UAV climbing angle at time i $\alpha_{\max}$: maximum climbing angle
Subduction angle constraint	$0 \leq \beta_i \leq \beta_{\max}$	β_i: UAV subduction angle at time i $\beta_{\max}$: maximum subduction angle
Communication constraint	$d_{ab} \leq r$	d_{ab}: distance between two UAVs r: communication radius
Boundary constraint	$l_i \in D$	l_i: UAV location D: restricted area
Collaborative constraint	$d_{\min} \leq d_{ab} \leq d_{\max}$ $t_a \leq T$	d_{ab}: distance between two UAVs $d_{\min}$: shortest distance between UAVs $d_{\max}$: maximum flight range t_a: time of arrival a T: latest time of arrival a

Table 1.1: UAV Constraints Table

In their evaluation, the researchers tested the Genetic Algorithm (GA), Ant Colony (AC), Particle Swarm (PSA), with other optimizing algorithms, in GA, when minimizing the time was taken as the objective function, the total task time was reduced by 65%, however, the algorithm needed a certain number of UAVs and showed linear increase in energy consumption as the number of UAVs increases [2].

In their article, Ramasamy et al, 2022 have explored parameter tuning using genetic algorithm with a set of constraints such as fuel limitations of the UAVs and the speed limitations of the UGVs, reducing the time and/or fuel consumption were important in assessing the validity of the algorithms, taking into consideration the resource and terrain constraints. The paper compared between the Genetic Algorithm and the Bayesian Optimization, despite the fact that the GA provided relatively similar results to BO, it required 3 times more local-search optimization and required 6 times the computation time [3].

In their research, Lui et al, 2016 used quantum ant colony algorithm (QACA), to optimize the path taken in evacuation practices, while being robust and and high efficiency, comparing their findings with the traditional Ant Colony Algorithm (ACO). In their research, minimizing the time of evacuation evacuation path optimization was critical in assessing the algorithm, as less time equates to less human losses in cases of environmental crisis. The primary objective is to consume as little time as possible to evacuate all evacuees from the danger zones to safe zones, making the time the most significant factor to be considered, another objective was to minimize total density in the paths. There were three benchmark functions used to compare the results of QACA and ACO, where the tests showed that QACA was more efficient in solving the problem, as well as expand the solution as the iterations advance [4].

In their paper, Wu et al 2020 discussed the same topic of cooperative UAV-UGV exploration, but in surveillance applications. In this paper, the path planning problem was formulated to be a 0-1 optimization problem, in which the on-off states of the discrete points are to be optimized. A hybrid algorithm, combining the Estimation of Distribution Algorithm (EDA) and the Genetic Algorithm (GA) was proposed to solve the problem. The goal is to minimize the required time by the vehicles to complete a circular path, where the objective function can be expressed as shown in equation 2.1.

$$J = max(\frac{D_u}{N_u * v_u}, \frac{D_g}{N_g * v_g}) \tag{1.1}$$

Where $D_u \& D_g$ are the lengths of the circular paths, and $N_u \& N_g$ are the number of drones and UGVs and $v_u \& v_g$ are their velocities.

The decision variables are the on-off states of the active points, where each point can either be ”open” (1) or ”closed” (0), and open points must be covered during exploration.

The constraints for UAVs in this paper lied in avoiding collisions with buildings, which are represented as inaccessible grids, also, the vehicles must cover all assigned 2D grids to ensure complete coverage, where the on-off states must satisfy the coverage constraint. In their results, they validated the superiority and rationality of the proposed hybrid algorithm, where the workload can be balanced between the UAVs and UGVs, moreover, the cooperative planning yielded better results than using either systems alone, by significantly reducing the total surveillance time, the suggested algorithm combined the strengths of EDA and GA's strengths, resulting in minimizing the time of task completion, while increasing adaptability since the path is being assessed during runtime rather than before runtime, increasing flexibility in real-world applications [5].

Chapter 2

Methodology

The objective of this project is to develop an efficient algorithm for optimizing the flight paths of multiple Unmanned Aerial Vehicles (UAVs). The UAVs must navigate a defined area while reaching designated targets and avoiding restricted zones. The optimization seeks to minimize the exploration time, maximize coverage efficiency, and ensure UAVs reach their assigned targets while avoiding crossing into prohibited areas, the overall problem aims to minimize exploration time, increase coverage efficiency, the penalty for not reaching designated targets, and the penalty for entering restricted areas, with more importance on time of exploration, in simulation, it can be seen that the UAVs try to converge to the targets as quickly as possible.

2.1 Objective Function

The cost function aims to maximize the explored area while minimizing the time taken, subject to penalties for constraint violations (e.g., communication and collision avoidance).

$$ObjectiveFunction = \alpha * T_{exploration} - \gamma * C_{eff} + \delta * Penalty_{targets} \tag{2.1}$$

Where:

- α, γ and δ are weight parameters for the time of exploration, Coverage efficiency and designated targets penalty.
- $T_{exploration}$ is a measure of the total time taken by UAVs to reach their designated targets, represented as a sum of the time taken by each UAV based on their velocity and the distance covered. $T_{exploration}$ is calculated by:

$$T_{exploration} = \frac{100}{v_{UAV}} \tag{2.2}$$

where 100 is the maximum time of exploration, and v_{UAV} is the speed of each UAV, faster UAVs explore the targets in less time

- C_{eff} is a ratio between the area covered by the UAVs with the total area of the map, computing for the effective area explored. It can be calculated as follows:

$$C_{eff}(i) = \frac{C_{UAV}(i)}{Area} \tag{2.3}$$

  where $C_{eff}(i)$ is calculated as a sum of position components of each UAV, and $Area$ is predefined from the map dimensions

- $Penalty_{targets}$ is a penalty term that accounts for the distance between the UAV positions and their targets, encouraging the UAVs to stay close to the targets, where it can be calculated as follows:

$$penalty(i) = min_j ||p_{UAV}(i) - t_{target}(j)|| \tag{2.4}$$

  where $||p_{UAV}(i) - t_{target}(j)||$ is the euclidean distance between a UAV i and target j, the penalty is accounted to be the smallest distance to any of the targets, so UAVs that are far from the targets have larger penalties.

Decision Variables

The decision variables consist of the velocities of the UAVs in the x and y directions, in m/s, as well as their positions in x and y in m, a solution x_i is represented as follows:

$$x_i = [v_{xi}, v_{yi}, p_{xi}, p_{yi}] \tag{2.5}$$

where:

- v_{xi}, v_{yi} are the velocities in x and y directions (in ms^{-1})
- p_{xi}, p_{yi} are the positions in x and y (in m)

Constraints & Feasibility Checks:

The constraints are handled through a nonlinear constraint function, where 2 constraints are checked as follows:

- **Restricted Area:** For every UAV at every step, the euclidean distance between the current UAV position and the center of the restricted area is calculated, if the distance is less than the radius + some buffer distance defined in the code, a penalty is added to the solution.

- **Safe distance:** The minimum safe distance, d_{safe} is defined in the code, if the distance between 2 UAVs is less than the defined distance, the penalty is added in a different fashion, it is added in terms of a variable called number of violations.
- **Maximum UAV velocity:** The maximum UAV velocity was pre-defined in the code, and before populating any v_x or v_y into any of the algorithms solutions, the velocity is capped to be between the inequality:

$$-v_{max} \leq v_{UAV} \leq v_{max} \tag{2.6}$$

2.2 Simulated Annealing

The Simulated annealing algorithm, introduced by Kirkpatrick, Gelett and Vecchi (1983) and Cerny (1985) takes inspiration from the process of annealing in manufacturing, where a metal slowly cools down till it eventually "freezes" at minimum energy configuration, it holds multiple advantages such as the faster rate of convergence compared to other meta-heuristic techniques, however, the algorithm may not yield the best solutions, i.e: it might yield locally optimal solutions, since the algorithm allows the acceptance of worse solutions based on randomness.

2.2.1 Code Flow

Our implementation of Simulated Annealing to solve our problem, depends on running the entire algorithm a number of times to find the best next step for each UAV, until all targets have been successfully reached, while not breaking the constraints.

After assigning the problem parameters and the SA specific parameters, the initial solution is randomly generated, where a solution [v_x,v_y,p_x,p_y] is randomly initialized, p_x & p_y are set to be the initial position of the UAVs, which is predefined in the code, and the velocities v_x & v_y are randomized to be within [-1,1].

After this initialization, the SA starts running, by generating a new solution by perturbation, which is scaled by a pre-defined step size of 1, after checking for constraints and applying the penalties if necessary, the SA then checks if this new solution will be accepted or not, if $Cost_{new} < Cost_{current}$ or by using a probability $rand < e^{\frac{(Cost_{new} - Cost_{current})}{T_{current}}}$ afterwards, if the solution is accepted, another check is performed, to check if the new accepted solution is the best overall, to be stored in an array, this is also done per SA run, so that the next step is optimized, the check is done to check if the accepted solution is the best found in the current SA run, as well as overall. Then, the position of the UAVs is updated, and the temperature is adjusted based on the cooling schedule, or reset to the initial temperature if the current SA run is over. A function which defines a unique target to every UAV has also been added, because before adding it, the algorithm failed to allow the UAVs to reach all the targets, where they would either reach 1 or 2 targets only

2.2.2 Test Cases & Evaluation

Overall, the simulated annealing succeeded in minimizing the objective function, reaching convergence where all UAVs reach a target, and we tested the effect of using different weights on the performance of the algorithm.

The initial positions of the UAVs, the targets and the restricted area locations are kept constant at every run.

In all tests, there was 1 obstacle, with 3 UAVs and 3 targets, α of the cooling is 0.8. The default case in our analysis, would have the initial temperature = 30, final temperature = 10, maximum number of iterations per run = 60, iterations per temperature change = 25, the step size = 1, $\alpha_{fitness} = 1$,$\gamma_{fitness} = 0.5$ & $\delta_{fitness} = 9$, figures 2.1 & 2.2 show the path, subplots of the temperature cooling down as the iterations goes on, fitness value changes per SA run and the overall convergence curve over the entire run of the algorithm.

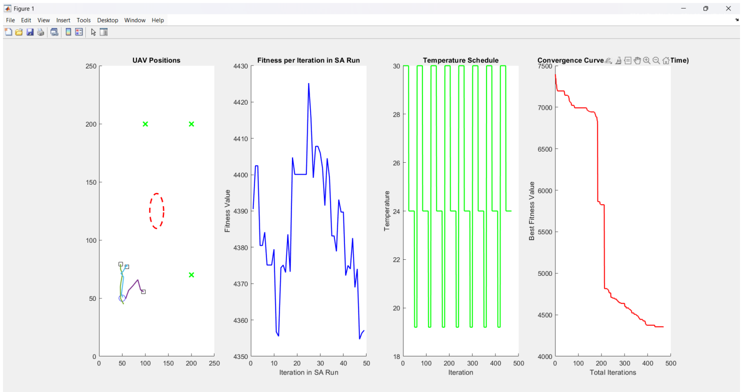

Figure 2.1: SA path during runtime, with the convergence curve, temperature variation and fitness value per run subplots

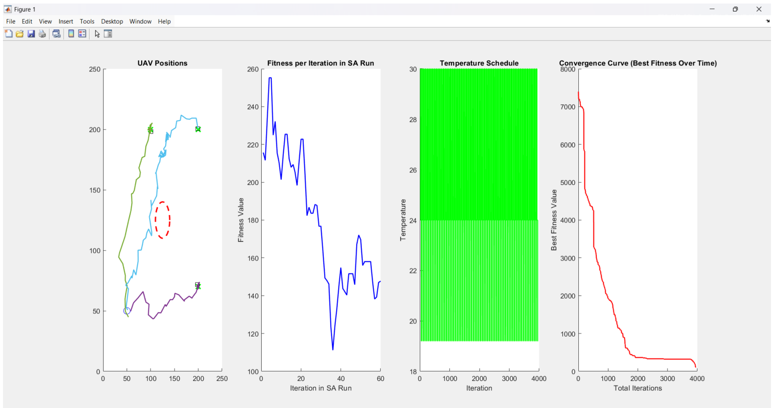

Figure 2.2: SA complete path, with the convergence curve, temperature variation and fitness value per run subplots".

Figure 2.2 shows that the SA accepts worse solutions, which justify the presence of the spikes in the 2^{nd} subplot, the algorithm reached local convergence after around 1900 iterations, at a value of approximately 350, before reaching a minimum of 111.169 after 3954 iterations, which is where all the UAVs reach their targets.

In the second test, $\gamma = 2.5, \delta = 1.5, \alpha = 3$, the objective function reached a minimum of 142.818, where it took almost 3000 iterations to reach local convergence, and 4680 iterations till the UAVs reached the targets, as shown in Figure: 2.3.

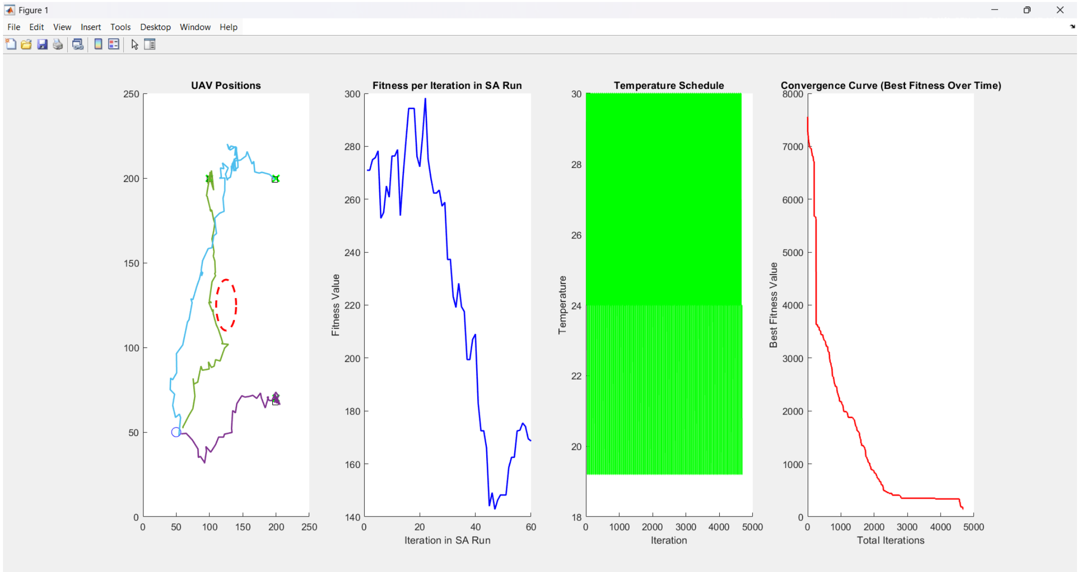

Figure 2.3: Test 2

In the third test, we changed the cooling rate to be 0.3, and the minimum temperature was adjusted to be 1, to adjust for the new cooling rate, keeping all other problem and SA parameters the same, after 2000 iterations, only 1 UAV managed to reach the target, where the other 2 UAVs are still searching for better points to get them closer to their targets, this extensive search, which caused the fitness value to get trapped in a local minima for around 10000 iterations, is due to the to aggressive decrease to the cooling rate, which causes the algorithm to explore more thoroughly, Figure: 2.4 shows the fitness plots and the generated points after 12000 iterations.

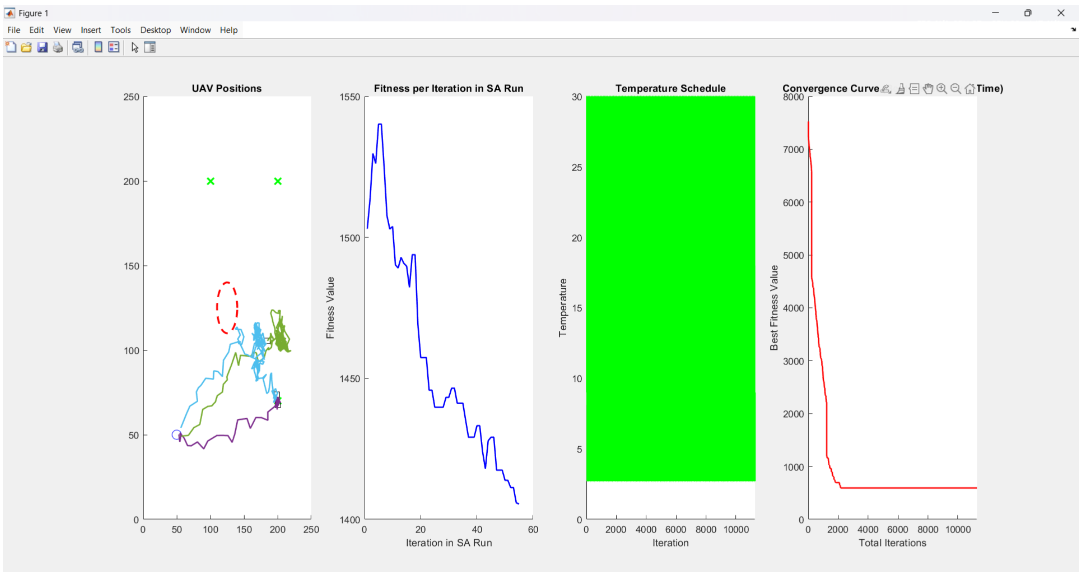

Figure 2.4: Test 3

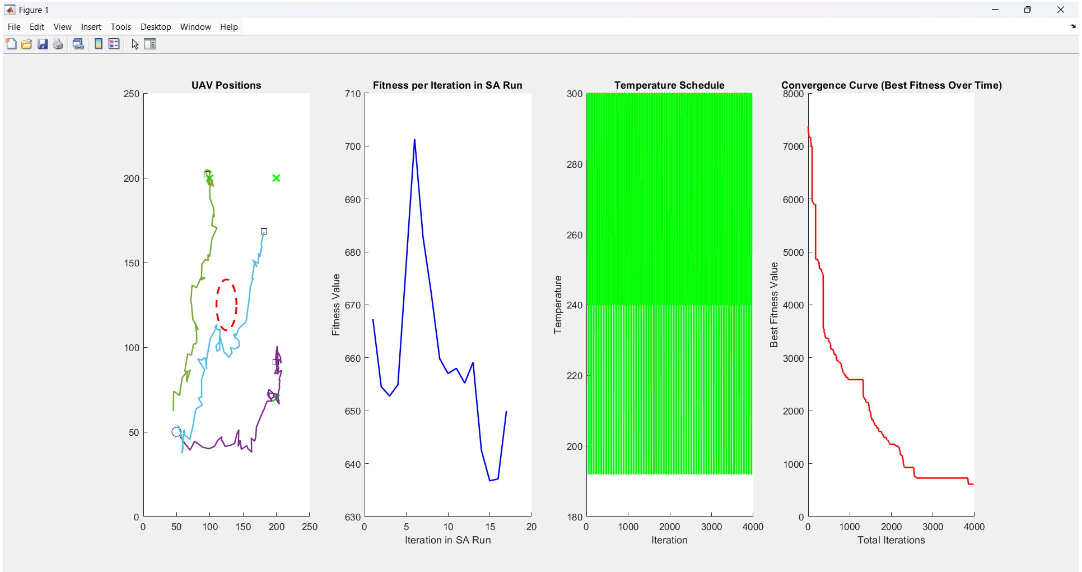

Figure 2.6: The final paths and convergence curves

In this test, we changed the initial temperature to be 300, keeping all other parameters the same, Figure 2.5, shows the path, fitness per run and convergence during runtime, it could be seen from the final path that the increased value of the temperature allows worse solutions to be accepted more often, the fitness, however did not get stuck in minima as it usually found better solutions, the UAVs reached their targets after 5160 iterations, which is slightly more than the rest of the tests, Figure: 2.6 shows the final path and the convergence curves.

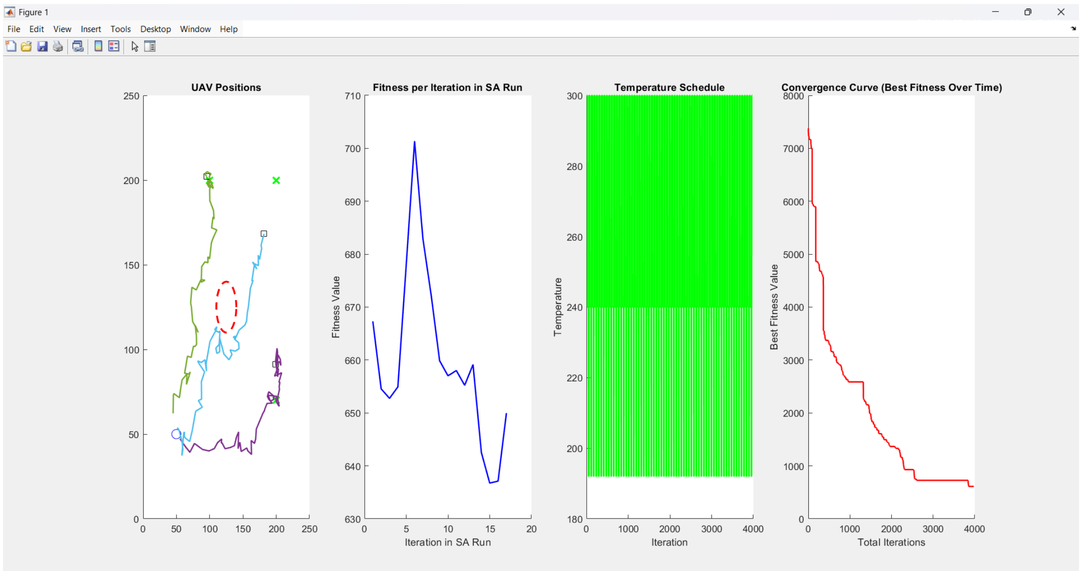

Figure 2.5: $T_i = 300$ during runtime

2.3 Genetic Algorithm

The Genetic Algorithm (GA) follows Charles Darwin's theory of evolution, through survival of the most fit individuals, allowing them to pass their traits to the next generations, in our problem, we followed a similar manner, in initializing a random population to start our GA run, and as iterations progress, the fitness value decreases showcasing the evolution.

2.3.1 Code Flow

Our problem required us to run the GA n times per point generated, where n is the number of generations, so in order to generate a single point in the path, we ran the GA algorithm n times.

Each generation consisted of m number of chromosomes, where m was a 4x3 matrix, with the columns representing all the information per UAV, so column 1 holds the information for UAV 1 and so on, and the rows are ordered as follows:[v_x,v_y,p_x,p_y]

Before the very first run of the GA, a random population is generated to fill the first generation, and their fitness values are calculated, until all generations are completed, the UAV positions and velocities are then updated with the best fit chromosomes in that run, to enter the new run and generate another point in the path, until all UAVs reach their desired targets, from each generation, the fitness values are calculated, and the population is sorted based on their fitness value, from best to worst, the chromosomes with the highest fitness, get passed to the next generation, without undergoing any change, the crossover parents are selected from the elite population, where 2 random elite chromosomes are selected to produce 2 offsprings, and the fitness value and feasibility are calculated and checked, if the offspring is feasible, it gets passed to the next generation, the number of produced offsprings is determined by a variable called GA.CrossoverRatio, which could be changed in the code

Since our problem is an arithmetic problem, the crossover is performed using a form arithmetic recombination as follows:

$$child_1 = \alpha * parent_1 + (1 - \alpha) * parent_2 \tag{2.7}$$

$$child_2 = \alpha * parent_2 + (1 - \alpha) * parent_1 \tag{2.8}$$

where α is an interpolation factor, which is a constant initialized at the start of the code.

Mutation is done in two different locations, to introduce more exploration and avoid early convergence to suboptimal solutions too quickly, firstly, the worst individuals in a generation are mutated by adding a noise to the chromosome's genes, also, mutation occurs to some of the offsprings produced from the crossover, based on the variable value GA.MutationRatio, random children are selected for mutation, the same way the worst chromosomes are mutated, the noise added can be altered using the variable GA.NoiseScale.

Constraints handling and feasibility checks were done in a similar fashion to what was previously explained in earlier sections.

2.3.2 Case Studies and Evaluation

Overall, the GA succeeded in reaching an minimum value, achieving the required functionality, but in some runs, the algorithm gets stuck in a minima, where 2 UAVs reach their designated targets, but the 3rd UAV gets stuck in the obstacle boundary, failing to reach the target, since the algorithm is unable to find a solution with a fitness value better than what was already provided, in this first run, which we will use as our benchmark for testing the effect of changing different parameters on the performance of the algorithm, the number of generations per point was 60, with each population having a population of 20 chromosomes, the elitism ratio was 0.2, the crossover ratio was 0.7 and the mutation ratio was 1-(crossover ratio + elite ratio), with $\alpha = 0.7$ and the mutation noise factor to be 0.2, the step size was 3, all UAVs started from the same position (50,50), and the targets were kept in the same position at every run (200,50), (50,200), (200,200), the obstacle was kept in the same place at every run with location (125,125) and a radius of 15.

For this first run, $\alpha_{fitness}$ = 1, γ = 0.5 and δ = 9, giving greater weight for target exploration, the graphs show that the UAVs reached their targets successfully, reaching, however, one of the UAVs got stuck in a local minima, which is shown by the flat line in the overall convergence curve in figure 2.8, the UAV got stuck trying to find a better solution to get out from the danger zone, reaching global best fitness value after generations 34860 generations, with a best cost value of 74.8016, as shown in Figure 2.7:

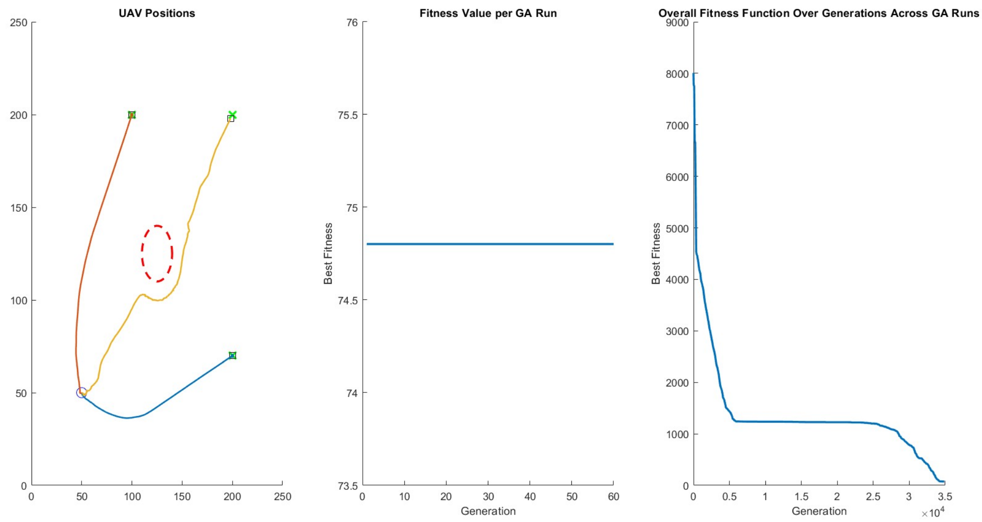

Figure 2.7: The map with UAVs in their target positions and their path plotted (Left), Fitness value per GA run (Middle), the overall convergence curve (Right).

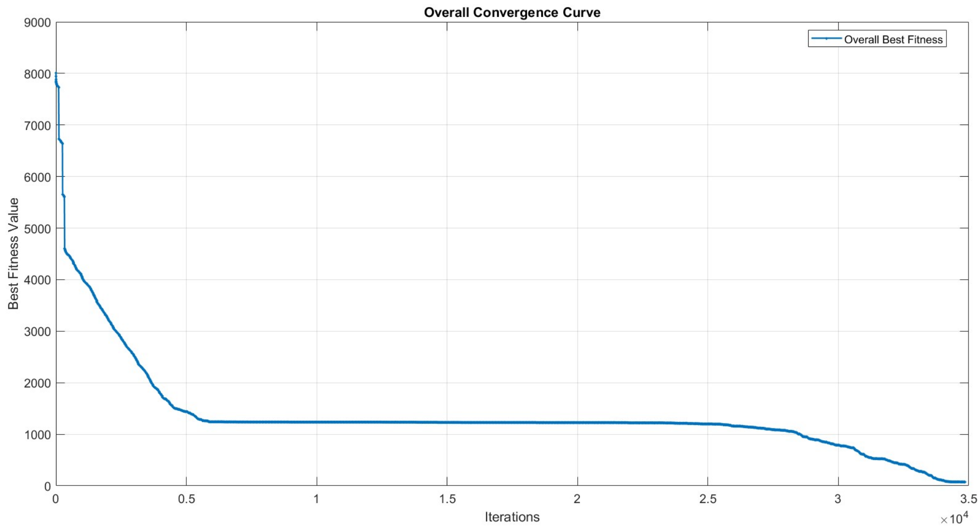

Figure 2.8: The GA overall convergence curve for the default test

In this second test, we will decrease the population size from 20 to 6, to see the effect on convergence, as shown in figures: 2.9 and 2.10

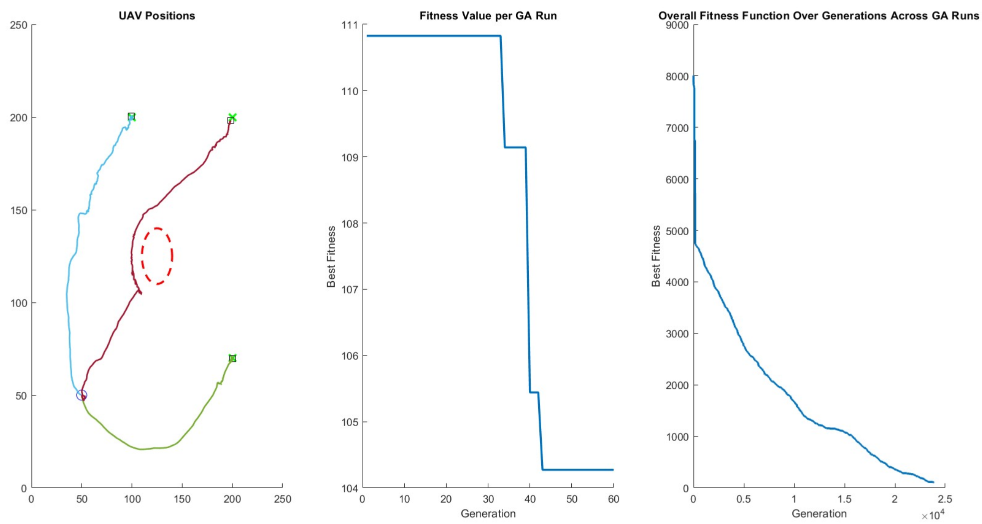

Figure 2.9: The map with UAVs in their target positions and their path plotted (Left), Fitness value per GA run (Middle), the overall convergence curve (Right).

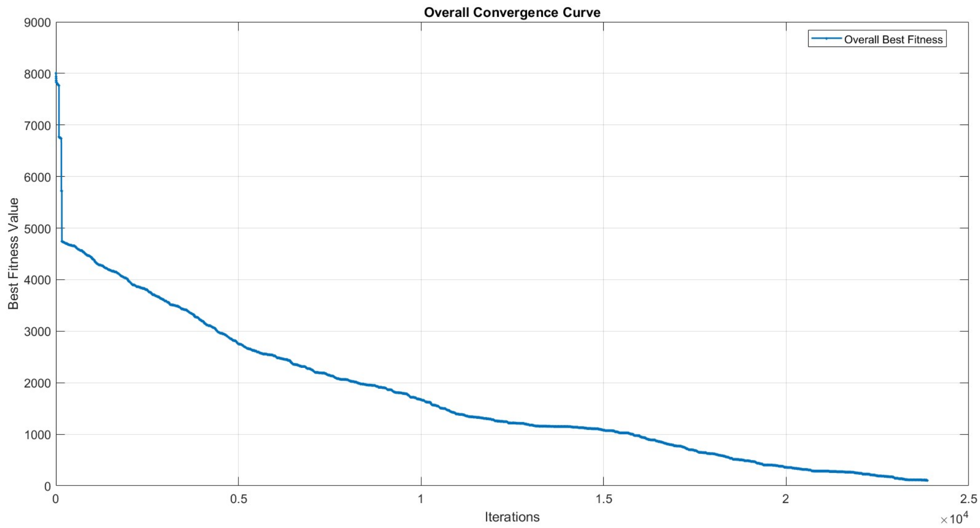

Figure 2.10: The overall convergence plot with 6 genes.

The UAVs managed to reach their targets in less iterations, however, the quality of the produced path was inferior when compared to the benchmark test path in figure 2.8, and the best overall fitness was 104.2724, which is around 40% increase in fitness.
In the 3rd test, we increased the elitism ratio to be 0.7, instead of 0.2, and the crossover ratio was subsequently changed to 0.2, the graphs can be shown in Figures 2.11 & 2.12:

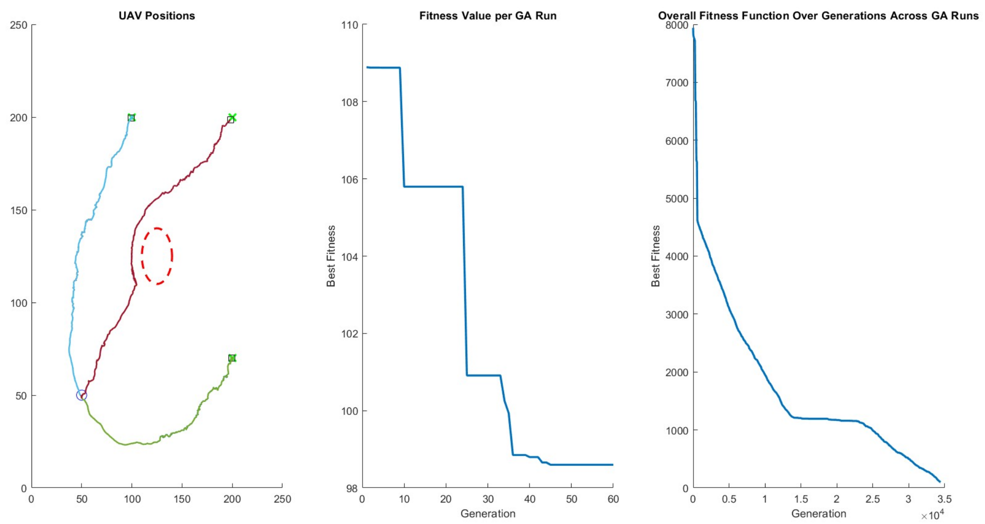

Figure 2.11: The map with UAVs in their target positions and their path plotted (Left), Fitness value per GA run (Middle), the overall convergence curve (Right).

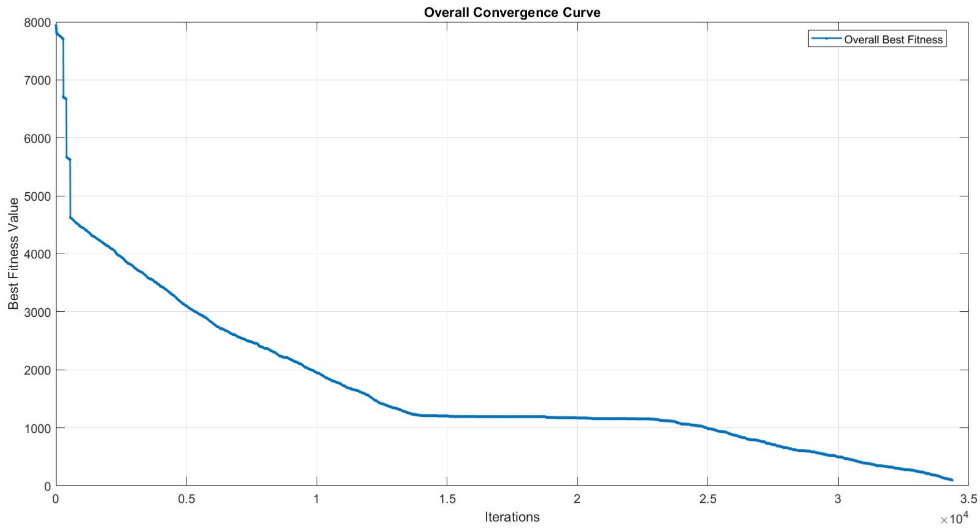

Figure 2.12: The overall convergence curve

The output path was slightly more distorted compared to the default case, and the overall best fitness reached was 98.598 in 34380 iterations, which shows around 15% increase in fitness compared to the default, this is because increasing the elitism ratio, hinders the algorithm to find better solutions, and just carries over most of the population, as they are considered elite.

2.4 Particle Swarm

Particle Swarm Optimization takes inspiration from flocks birds in their journey in searching for food, or during immigration, consider a swarm of birds flying, the algorithm considers every bird as a particle and the entire group as a swarm, taking a set of candidate solutions as input, similar to the Genetic Algorithm, the Particle Swarm Optimization is a population based, nature inspired iterative technique, compared to the Genetic Algorithm, the PSO is known to be computationally more efficient [6].

2.4.1 Code Flow

In order to implement PSO into our problem, we followed the following approach, starting off by initializing the swarm size and the maximum iterations per PSO run, with the cognitive and social weights for velocity calculation, then initializing a random swarm to be our birds' velocities in the first run, where the positions are defined as the initial positions of the UAVs, with initialization of the global and personal bests, the neighborhood topology used was the synchronous star topology, to check all the bird's neighbors to check for global best. The inertia weight term w is set to be variable, with $w_i = 0.99$ and $w_f = 0.6$, to create a balance between exploration and exploitation for the swarm during a run, the main loop starts with calculating the fitness value for all birds in the swarm, and checking for constraints, which were explained in section 2.1, if any of the constraints were violated, a scaled penalty term would be added to the fitness value of that bird, then the personal bests of the birds are checked and updated, as well as the global bests in the swarm, to update the velocity, we first extract the position and velocity terms in our decision variables, for each UAV, the cognitive term of the velocity can be calculated as follows:

$$cognitive = c_1 * r * (personal\_best\_position - current\_position) \tag{2.9}$$

where cognitive is the term responsible for allowing the birds to move towards a previously known personal best position, c_1 is the cognitive term weight, and r is a random number from 0 to 1,

The social component of the velocity is calculated as follows:

$$social = c_2 * r_2 * (global\_best\_position - current_position) \tag{2.10}$$

where $social$ is the term responsible for pushing the swarm to move towards the neighborhood's best position, r_2 is a random number between 0 and 1 and c_2 is the social weight.

The new UAV velocities are then calculated by:

$$v_{i+1} = wv_i + social + cognitive \tag{2.11}$$

where w is the inertia weight, and v_i is the current UAV velocity. The new UAV positions can be calculated using the following equation:

$$x_{i+1} = x_i + v_{i+1} \tag{2.12}$$

The swarm is then updated with the new positions and velocities, when the number of iterations exceeds the maximum defined number, the swarm is then reinitialized, to avoid rapid convergence, since the personal and global best positions are set to be straight line very early on, so the entire swarm will exploit the same positions every time, re-initialization of the swarm after every point, introduces necessary diversity to explore more promising solutions, without the fear of rapid convergence to sub-optimal solutions.

2.4.2 Test Cases and Evaluation

The default parameters used were as follows:

- **Swarm Size** = 20 birds
- **Max iterations per run** = 60
- **Cognitive and social coefficients** = 1.5
- **Initial inertia weight** = 0.99
- **Final inertia weight** = 0.6

The default test results can be shown in figures 2.13 and 2.14, the best overall fitness reached was 88.4370 in 70740 iterations, this was because the algorithm got stuck, for most the runtime duration in a local minima, however, the PSO showed that it is capable in reaching local optimal solutions in very short number of iterations.

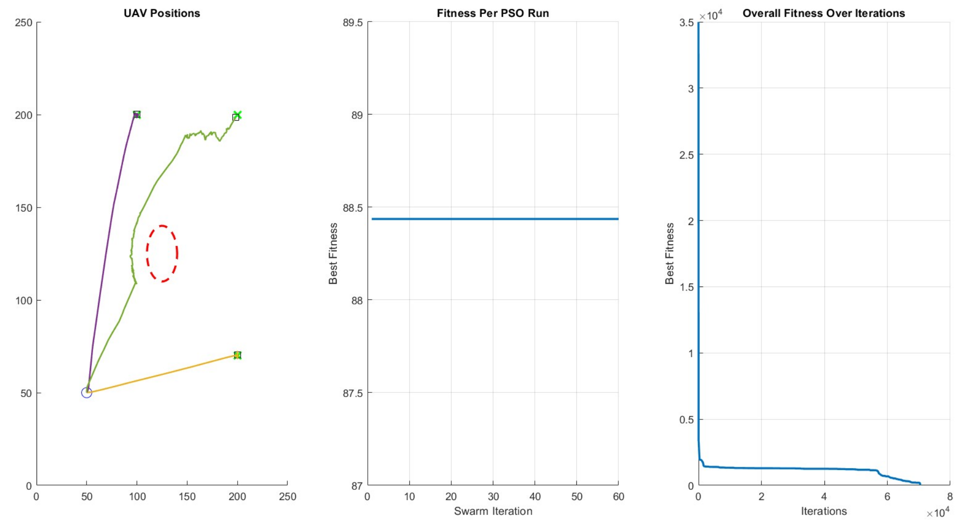

Figure 2.13: The overall output path

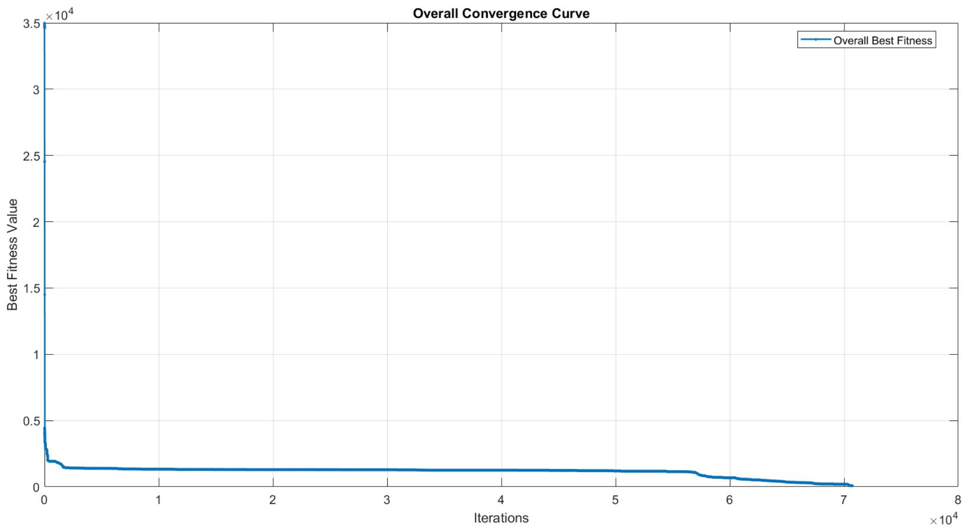

Figure 2.14: Overall convergence plot

In the following test, we tested the effect of decreasing the final inertia weight from 0.6 to 0.2, results are shown in figures 2.15. The convergence curve shows that the decreased final inertia weight value reduced exploration drastically, which can be visible as the fitness value was stuck in a local minima after 100000 iterations, and after 150000 iterations, the algorithm failed to reach global best fitness.

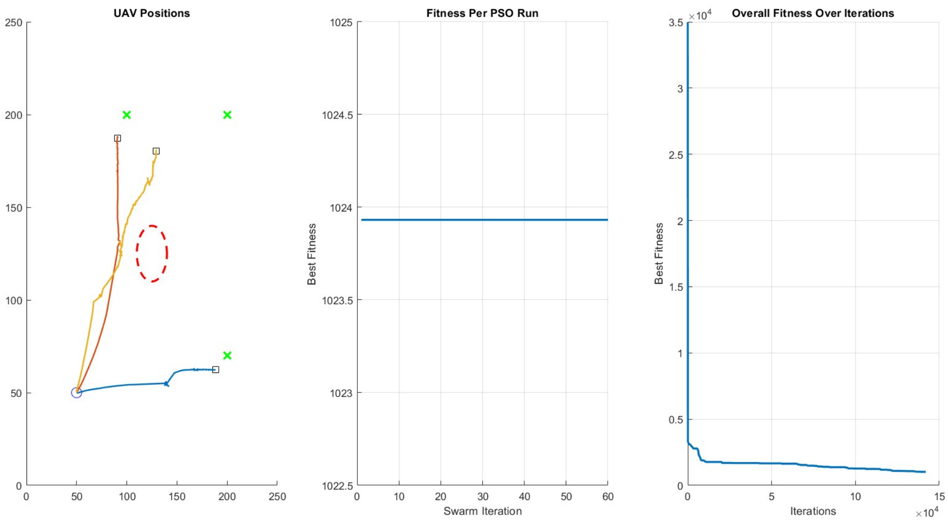

Figure 2.15: UAV positions and path, and convergence curves

The following set of tests, tested the effect of altering the cognitive and social weights, we first increased the social weight to 4, results shown in figures 2.17 and 2.18. Then following these tests, we tested the effect of the cognitive coefficient with a similar manner, by decreasing it to 0.3 as shown in figure 2.16, and increasing it to 4 as shown in figures **??** and **??**. Decreasing the social and cognitive and increasing them showed similar results, so not all cases are presented in the report.

Decreasing the cognitive coefficient to 0.3, had an effect on exploration, since this drastically decreases the effect of the cognitive section of the velocity update for the swarm, decreasing the effect of recorded personal best positions, in the fitness plot and the path, after 60,000 iterations, the algorithm was stuck in a local minima, failing to explore better solutions. The algorithm, after 95520 iterations, reached a fitness of 1040.5, while being stuck in a minima for almost the entire run, where 2 of the UAVs failed to reach their targets. A similar outcome is expected from decreasing the social weight to 0,3.

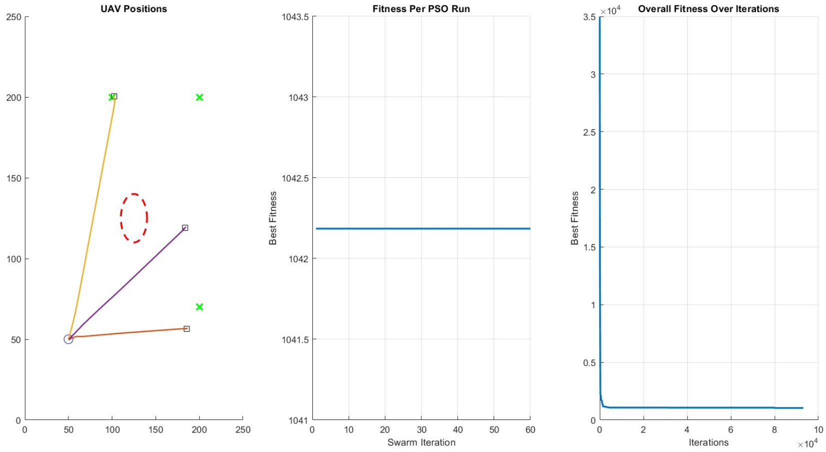

Figure 2.16: UAV positions and path, and convergence curves after 95000 iterations

Increasing the social weight to 4, showed interesting results, where the algorithm reached global convergence after around 9000 iterations, while terminating after 12480 iterations, when all the UAVs reached their targets with global best fitness of 75.0444, the output path was also more uniform compared to the default test, this is becuase of the increased weight of the social velocity component, encouraging the swarm to get heavily influenced by the most fit bird in the swarm. A similar outcome is expected from increasing the cognitive weight to 4.

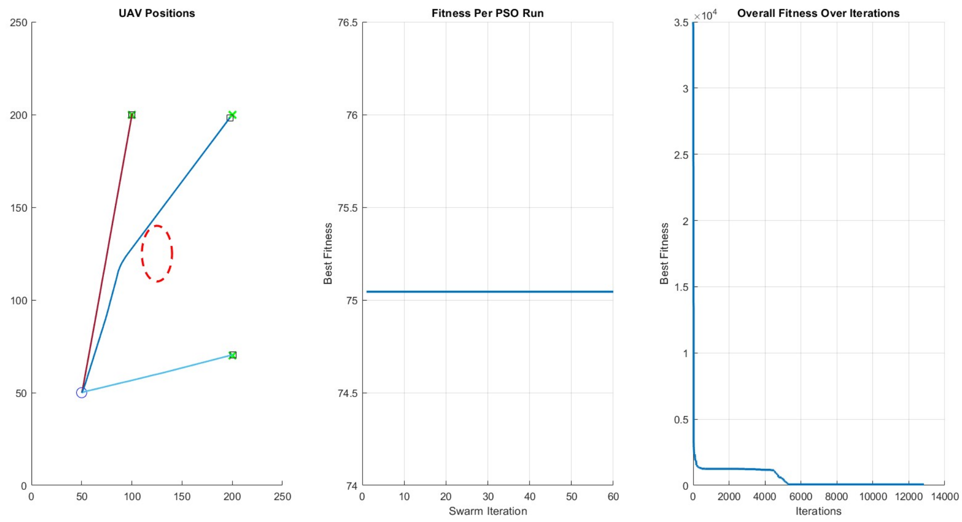

Figure 2.17: Final path and convergence plots

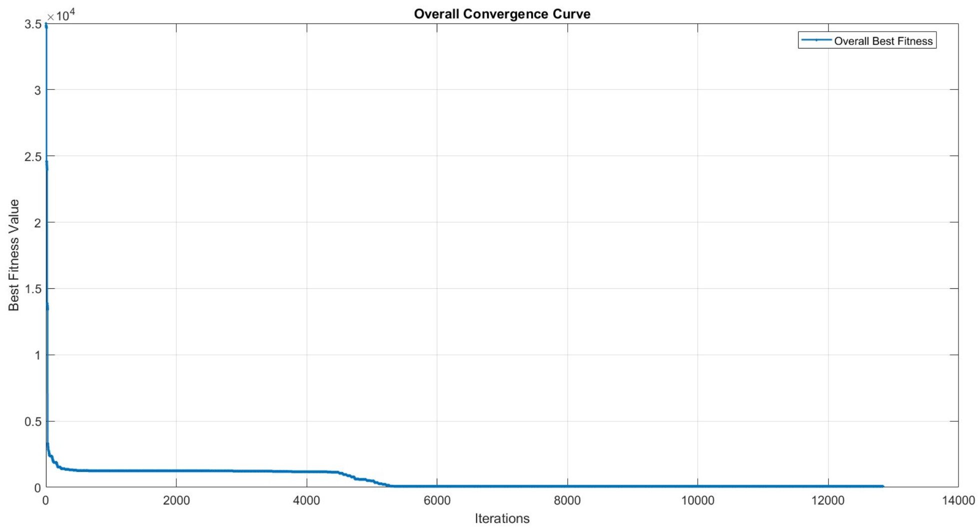

Figure 2.18: Overall fitness curve

2.5 Teaching Learning Based Optimization

2.5.1 Teaching Learning Based Optimization

Teaching Learning Based Optimization (TLBO), is a meta-heuristic population based optimization algorithm, which was introduced by Rao, et al in 2011, mimicking the behavior of students and teachers learning in a classroom, where the population is considered to be as a class, and each solution is considered to be a student in the class, the algorithm optimizes the solutions over 2 phases, the teaching phase and the learning phase, where the teaching phase relies on the most fit solution (teacher) to improve the average fitness of the classroom, where all solutions gain knowledge from their teacher, and the learning phase relies on the behavior of students learning from a partner, if the partner's information helps the student improve its fitness, it will learn from it, and if not, the student's fitness will move away from the partner [7].

2.5.2 Code Flow

Our implementation of the algorithm was mostly similar in approach to what was proposed in the literature, however, in the main while loop, which runs until all UAVs reach their targets, the fitness value gets evaluated, before the teaching phase, to select the initial teacher, after the teaching phase, to check if the teacher has changed, and to update the global best teacher, as well after the learning phase, to check if the learning phase generated a new better solution than the current best, then after completing the teaching and learning phases for the set number of iterations, the global best teacher is used to move the UAVs to the next step. Also, after each TLBO run, some of the students in the classroom get reset to share the same solution as the teacher of the previous run, and the rest get random noise from their current solutions, this is achieved by a simple check, where a random number is generated for each student, and if the random value is less than or equal to the preset number ratio, the students inherits the teacher's traits, else, it gets random noise on its current solution. Thus enhancing diversity and exploration, while keeping track of better solutions, allowing the algorithm to mix and match between exploration and exploitation.

2.5.3 Case Studies and Evaluation

In the following tests, we will test the effect of changing the algorithm's parameters on performance, in terms of the global best fitness value, how long the algorithm reached to reach that fitness value, and lastly the path shape.

The default parameters we based our evaluation were as follows:

- Classroom size: 20 students
- Maximum Iterations per run: 60 iterations

- Ratio of the old classroom to enter the new TLBO run with the previous teacher's solutions: 50%

Running the algorithm with the default parameters, provided the plot shown in Figure 2.19, and the path shown in Figure 2.20. The overall best fitness at termination was 132.9557, reached in 173 runs, and the algorithm terminated after 26880 iterations.

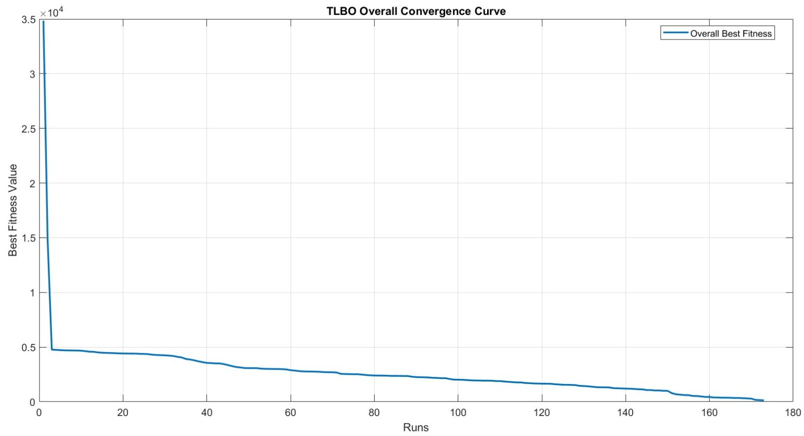

Figure 2.19: Overall best fitness convergence curve

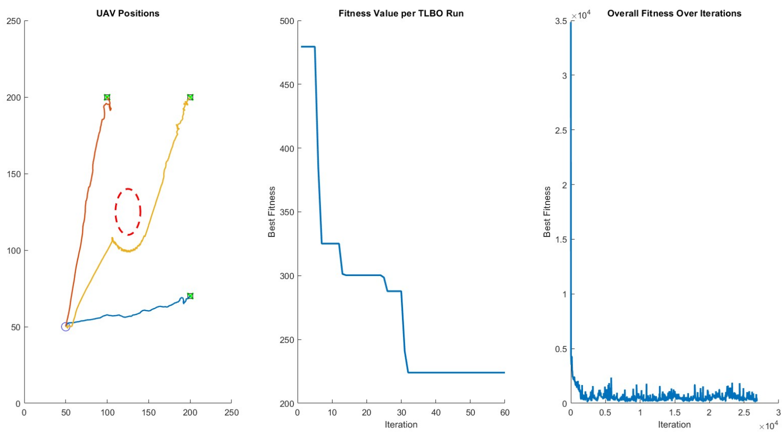

Figure 2.20: TLBO Path, with convergence curve per run and overall fitness plot.

As shown in figure 2.20, the output path looked smooth, which emphasizes that the inheritance of the teacher's solutions by most of the classroom improves the quality of solutions.

In the first test, shown in figures 2.21 and 2.22, we decided to change the ratio of the classroom that inherit the previous teacher's traits, from 0.5 to 0.1, so that more students inherit the previous teacher's position and velocity, this effect was visible on the performance, where the UAVs reached their targets in 92 runs, or 31000 iterations, reaching a global minimum fitness of 125.0940,

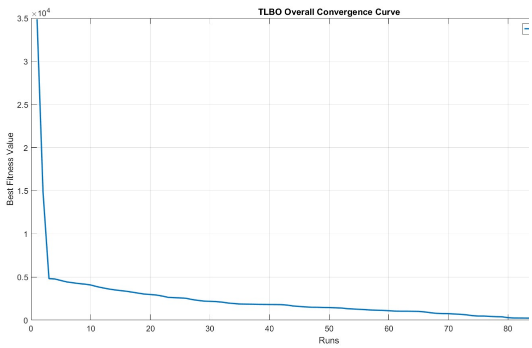

Figure 2.21: Overall fitness convergence plot per number of runs

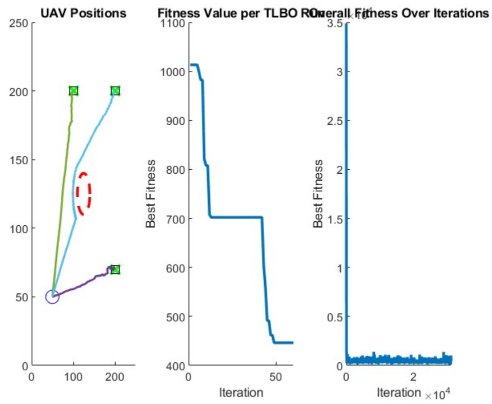

Figure 2.22: TLBO Path, with convergence curve per run and overall fitness plot.

In the second test, we explored the effect of drastically increasing and decreasing the classroom size, on the quality of solutions and fitness achieved, shown in figures 2.23, 2.25, and 2.24.

We first decreased the number of students in the classroom to 5 students, as shown in figures 2.23. The UAVs failed to reach their targets after more than 100,000 iterations, reaching a global best fitness of 333.0532, showcasing the importance of having an adequate classroom size.

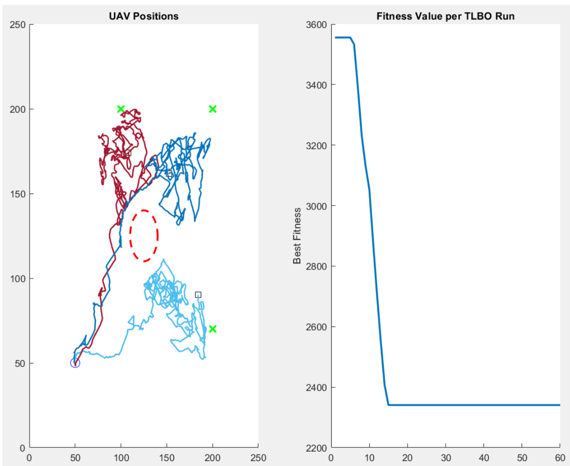

Figure 2.23: Path taken by UAVs after almost 80,000 iterations

We then increased the classroom size to 100 students, as shown in figures 2.25 and 2.24. The algorithm took longer to terminate, due to the increased computation required to iterate over an increased number of students for every iteration, however, the quality of the path has been significantly increased, as well as the overall best fitness. The algorithm found its best fitness after 102 runs, reaching a global minima of 99.3593 in around 35000 iterations.

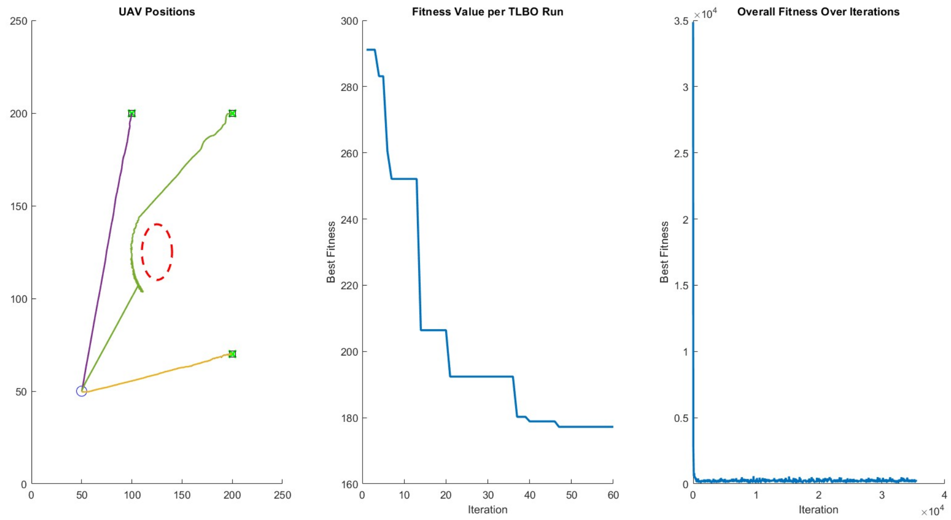

Figure 2.24: UAVs path with 100 students, convergence plot per 1 TLBO run at termination and fitness curve over the entire run

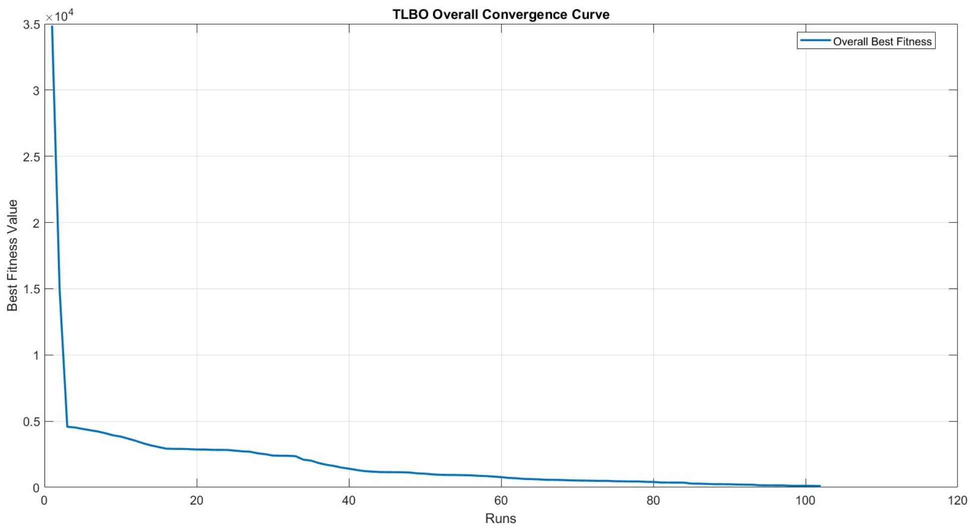

Figure 2.25: The overall best fitness convergence curve

2.6 Performance Indices

Standard Deviation, Mean Fitness & Best Overall Fitness:

In order to compare the performance of all the algorithms, we ran the algorithms using the same parameters 15 times, to calculate the standard deviation, mean fitness and the best fitness during those 15 runs, the table shows the results:

	Standard Deviation	Mean Fitness	Best Fitness
SA	26.9377	111.5836	85.4256
GA	6.7540	76.62864	72.5574
PSO	4.38347	89.10830	81.8662
TLBO	12.26142	137.69886	122.1275

Based on the table above, the PSO showed the best standard deviation compared to the GA and the SA, due to the reduced effect of randomizing values, where the SA heavily relies on random numbers to accept solutions, and random solutions are constantly generated, on the contrary to population based techniques, where the "elite" or "global best" solutions, are carried through the iterations.The TLBO, however, showed a standard deviation being a middle ground between the performance of the PSO, GA and the SA, in our implementation, the TLBO relied on randomness in generation of new solutions after every run, which might have caused this increase in standard deviation, and the higher values of mean and best overall fitness recorded.

The following figures show the difference in the overall convergence curve at each of the algorithms' overall fitness per iteration in a run.

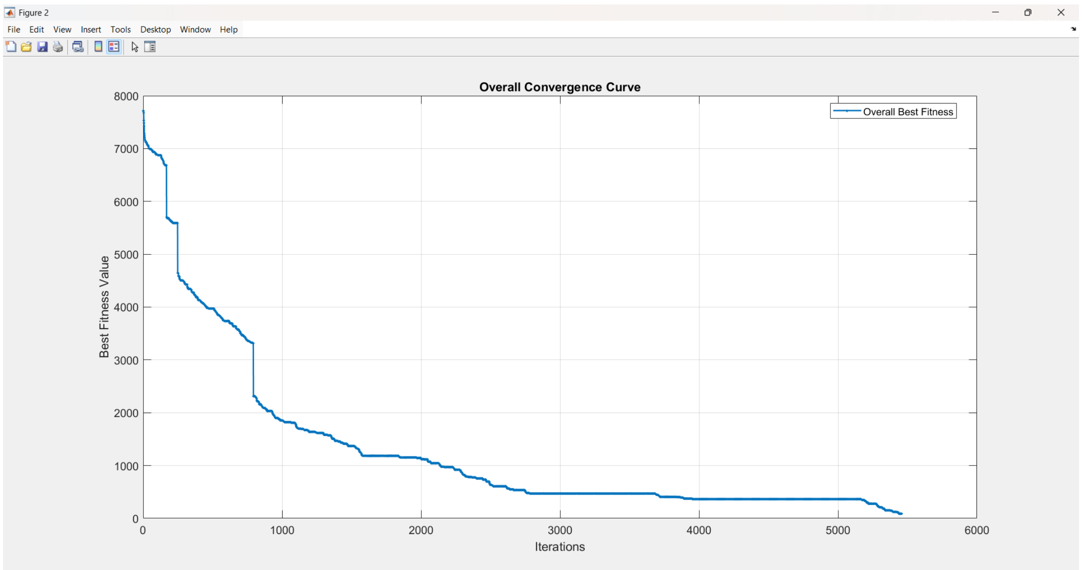

Figure 2.26: SA Overall Fitness Convergence Curve

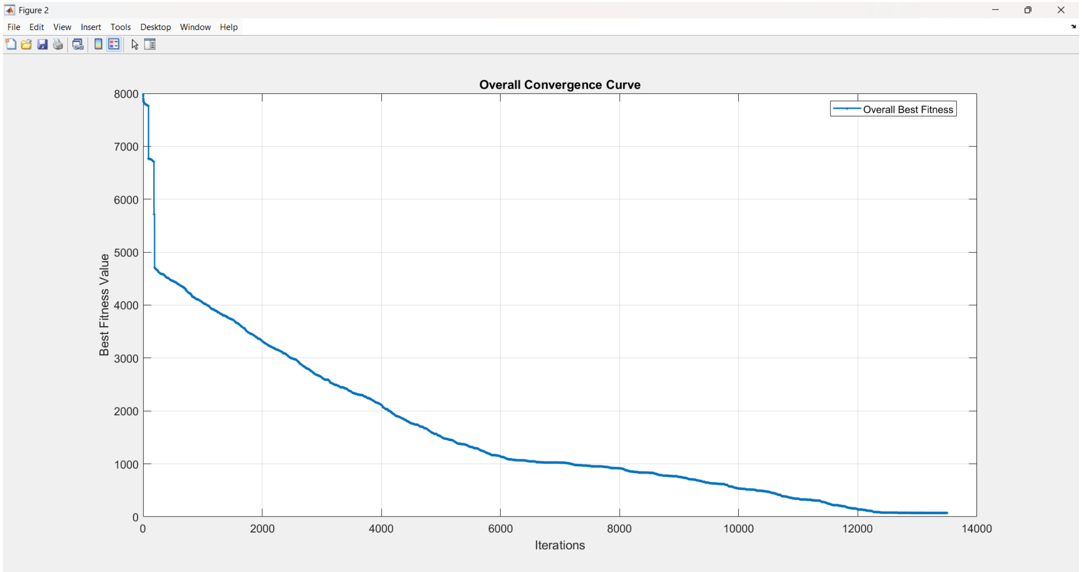

Figure 2.27: GA Overall Fitness Convergence Curve

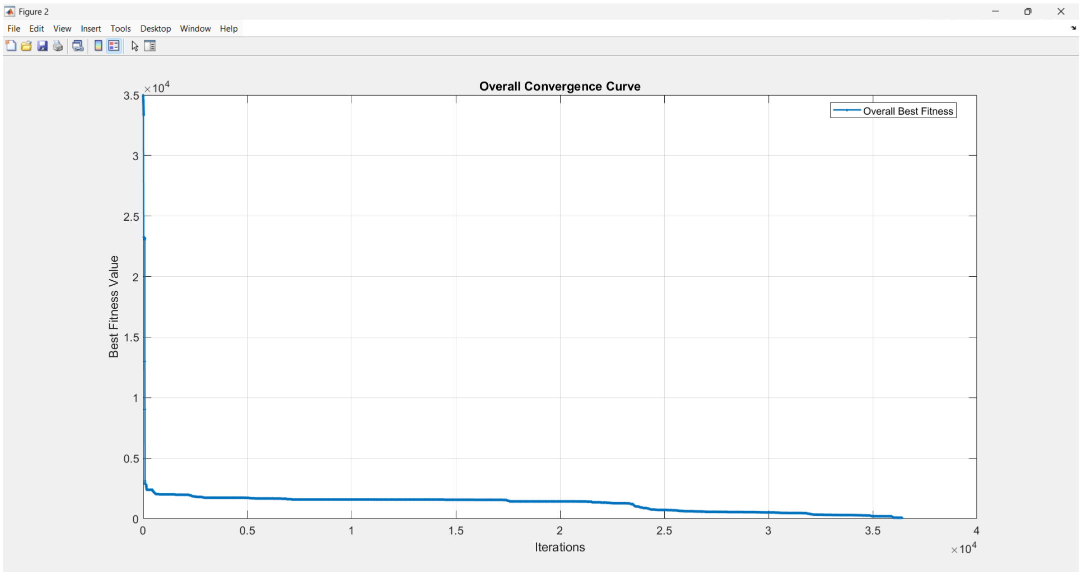

Figure 2.28: PSO Overall Convergence Curve

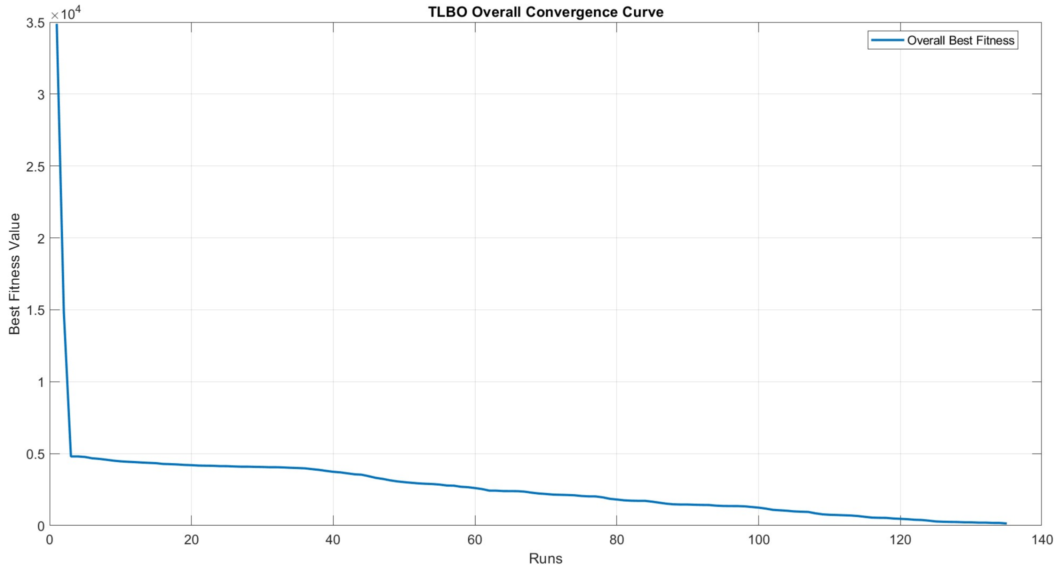

Figure 2.29: Overall TLBO convergence plot

Figures 2.26, 2.27, 2.28, and 2.29 can give a clear insight on the algorithms' performance, where the SA took the least number of iterations to reach global best fitness value, on the contrary to the population based techniques, the PSO and the GA, where both took considerably more iterations to reach global best fitness, where the GA took around 13000 iterations, and the PSO reached global best in almost 28000 iterations, however, the PSO was much faster than the other 2 algorithms in reaching a fitness value close to the best fitness, where it took almost 500 iterations to reach a locally best fitness, getting stuck in a local minima for most of the run, until the swarm finds a better solution to reach global best fitness, where all the UAVs reach their targets, the local minima appears when one of the UAVs get stuck trying to escape the restricted region area, iterating to find a solution with a better fitness to help the UAV escape its current region. The TLBO, shown in figure 2.29 reached its global minimum in around 33000 iterations, which is similar to the PSO's performance in this run, however the TLBO reached a worse fitness value when compared to all the other algorithms, due to the increased randomization of new population, and inaccurate position updates where the implementation does not create a balance between exploration and exploitation.

Chapter 3

Conclusion

Overall, the algorithms showed varying success rates in minimizing the fitness function based on our problem formulation, with the PSO being the most consistent, in terms of performance indices, while the GA was the best performing in terms of quality of produced path. Population based techniques proved that their outputs are much more reliable, compared to other meat-heuristic techniques, such as the SA. Fine tuning the algorithm specific parameters is crucial to an algorithms success, as shown in the test cases in the previous chapter, also, a solid problem formulation can make or break an algorithm, since the problem formulation is the base of an optimization problem.

For future work, We would like to explore the effect of adding multiple static obstacles, as well the possibility of dynamic obstacles, alongside with tuning the problem formulation to reflect more realistic optimization scenarios, furthermore, testing more algorithms and using reinforcement learning algorithms to solve our optimization problem.

Bibliography

[1] Cheng Xu, Ming Xu, and Chanjuan Yin. Optimized multi-uav cooperative path planning under the complex confrontation environment. *Computer Communications*, 162:196–203, 2020.

[2] Zhu Qiming, Wu Husheng, and Fu Zhaowang. A review of intelligent optimization algorithm applied to unmanned aerial vehicle swarm search task. In *2021 11th International Conference on Information Science and Technology (ICIST)*, pages 383–393. IEEE, 2021.

[3] Subramanian Ramasamy, Md Safwan Mondal, Jean-Paul Reddinger, James Dotterweich, James Humann, Marshal Childers, and Pranav Bhounsule. Heterogenous vehicle routing: comparing parameter tuning using genetic algorithm and bayesian optimization. pages 104–113, 06 2022.

[4] Min Liu, Feng Zhang, Yunlong Ma, Hemanshu Roy Pota, and Weiming Shen. Evacuation path optimization based on quantum ant colony algorithm. *Advanced Engineering Informatics*, 30(3):259–267, 2016.

[5] Yu Wu, Shaobo Wu, and Xinting Hu. Cooperative path planning of uavs & ugvs for a persistent surveillance task in urban environments. *IEEE Internet of Things Journal*, 8(6):4906–4919, 2020.

[6] Hiren Kumar Thakkar, Hrushikesh Shukla, and Prasan Kumar Sahoo. Chapter 2 - metaheuristics in classification, clustering, and frequent pattern mining. In Sushruta Mishra, Hrudaya Kumar Tripathy, Pradeep Kumar Mallick, Arun Kumar Sangaiah, and Gyoo-Soo Chae, editors, *Cognitive Big Data Intelligence with a Metaheuristic Approach*, Cognitive Data Science in Sustainable Computing, pages 21–70. Academic Press, 2022.

[7] R Venkata Rao, Vimal J Savsani, and Dipakkumar P Vakharia. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. *Computer-aided design*, 43(3):303–315, 2011.