Project Report on Swarm Computing

-Rashika Moza(001352144)

Problem Statement

The collection of trash from trash cans which are present at different locations to the hub is the underlying problem which is solved by the algorithm. The company has limited number of vehicles and each vehicle has a certain limit of capacity that it can carry. The major challenge in the collection of trash is to minimize the total distance that each vehicle can travel in order to collect trash. The other challenge to minimize the distance over here is that after the vehicle's capacity is full, each vehicle has to travel back to hub and dump the garbage and again start from hub to other location.

Solution Designed

The collection of trash from different locations and optimizing the path is an instance of Vehicle Routing Problem. Particle Swarm Optimization is implemented to find optimal route and the distance travelled by the vehicles should be minimum. At the end a continues route is found from the hub to all the locations so that x number of vehicles with fixed capacities can be utilized to minimize the distance.

Swarm Computing

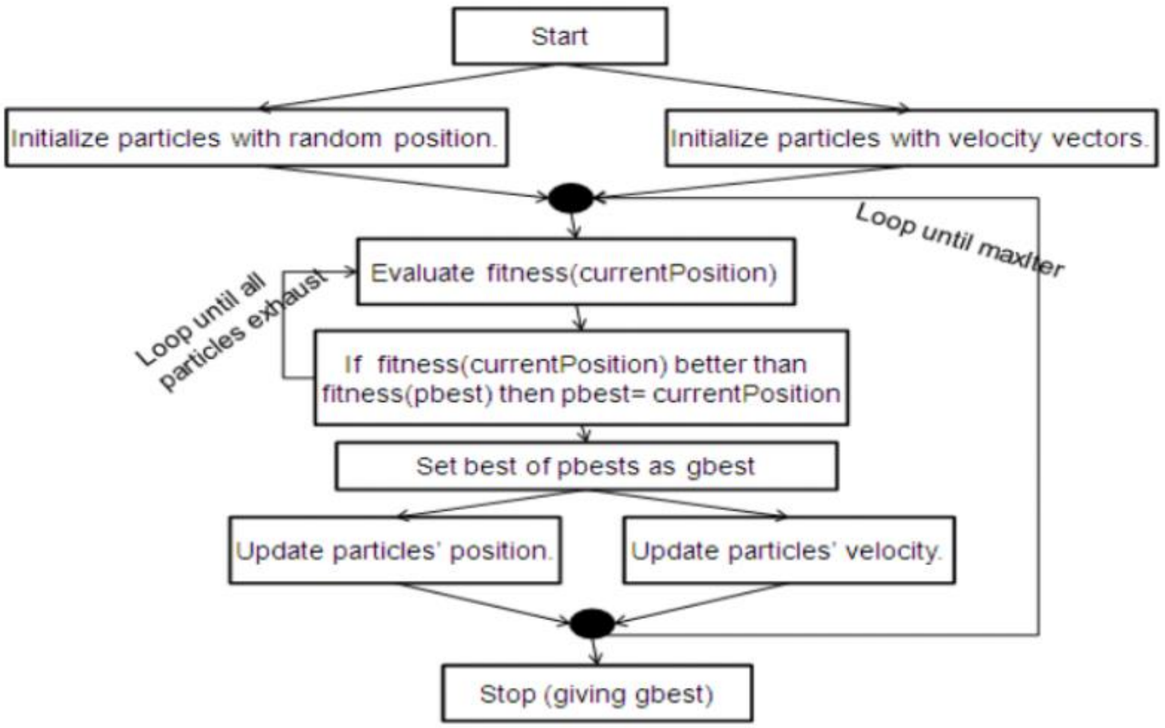
As the given problem is a NP-hard problem it is difficult to find a best solution, hence this report describes the application of particle swarm optimization algorithm that can identify an optimal solution. The aim is to find the best solution from all possible solutions turned out to be hard, near-optimal solutions.

Description about the mechanism:

It uses number of particles that constitute a group moving around in the search space looking for the best solution. It imitates the bird from a flock which is nearest to the food. Each particle is treated as a point in a N-dimensional space which adjusts its "flying" according to its own flying experience as well as the flying experience of other particles. All particles have fitness values, evaluated through the fitness function and velocities.
The two variables which are iteratively changed in PSO algorithm are the following ones:

1) Pbest (personal best)—each particle keep track of its coordinates in the solution space which are associated with the best solution that has achieves so far by that particle

2) Gbest (global best)—another best value that is tracked by the PSO is the best value obtained so far by any particle in the neighborhood of that particle.


Start
Initialize particles with random position.
Initialize particles with velocity vectors.
Loop until maxIter
Evaluate fitness(currentPosition)
Loop until all particles exhaust
If fitness(currentPosition) better than fitness(pbest) then pbest= currentPosition
Set best of pbests as gbest
Update particles' position.
Update particles' velocity.
Stop (giving gbest)

<u>Global Objective Function</u>: What should be the route of each vehicle so that all the vehicle travel in the same route which is determined by the global best from all the routes of the respective vehicles.

<u>Local Objective Function</u>: What should be the minimum distance covered by each vehicle while covering it’s route. It is the personal best value of each vehicle which gets updated in every iteration after getting compared itself to the last value.

Implementation

1) Initialize the parameters:
   trashCans: number of trashcans(demand)
   noOfTrucks: number of resources(supplier)
   noOfParticles: number of particle
   iteration: number of times in which we have to provide the solution
   maxTrashCapacity: random number(between 0-50) that will represent maximum capacity of trash

2) Design Distribution Model

   Randomly generate distance matrix which shows the distance between the trashcans

3) Build Swarm Model

   Initialize n particles with an initial solution and pbest

   Update gbest

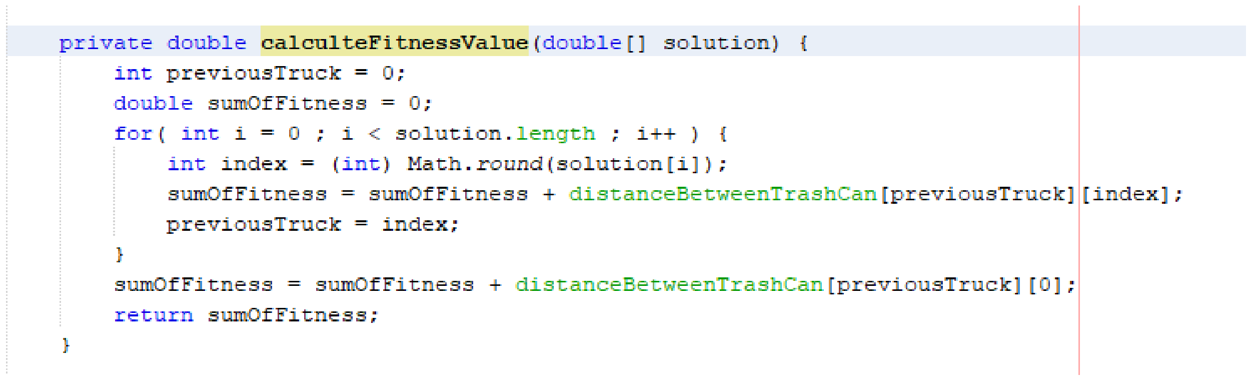
4) Iterate the no of times as specified in order to find the optimal best solution
   We will get a value of gbest with smallest fitness value

   Debug gbest to identify the route

   Identify the best route to find the total minimum distance travelled by the vehicle.

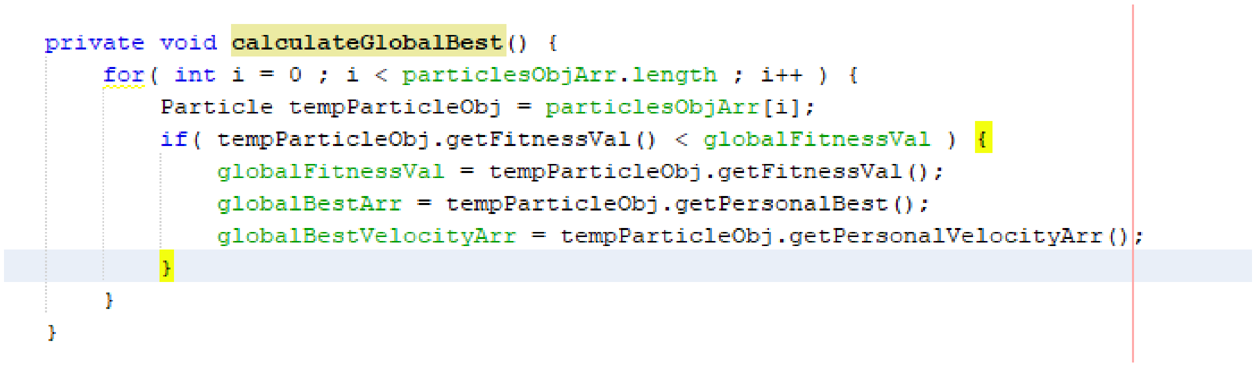
PSO Functions:

1) For each iteration, it calculates each particle’s fitness value.



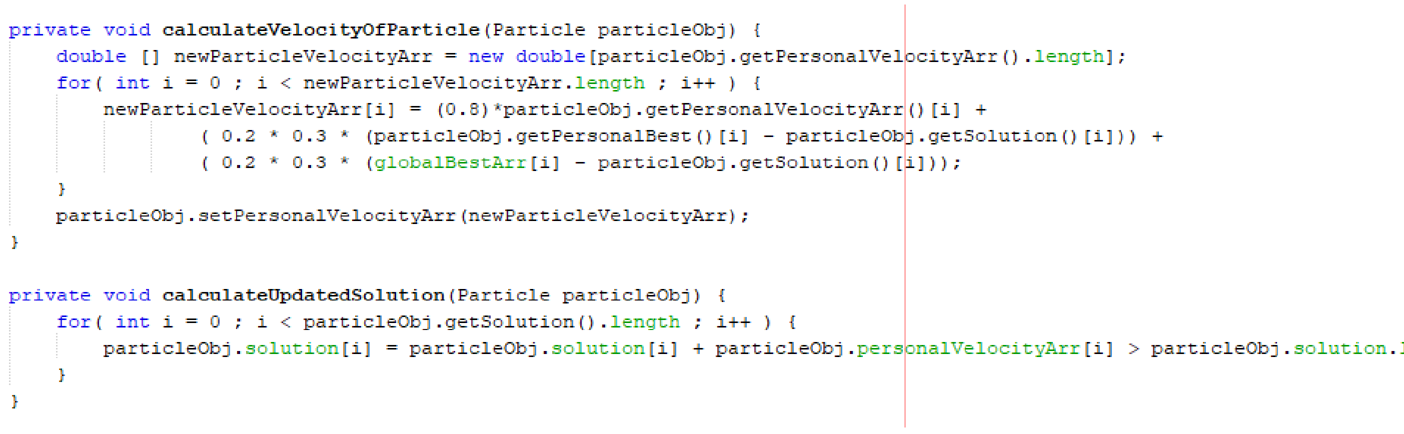
```
private double calculteFitnessValue(double[] solution) {
    int previousTruck = 0;
    double sumOfFitness = 0;
    for( int i = 0 ; i < solution.length ; i++ ) {
        int index = (int) Math.round(solution[i]);
        sumOfFitness = sumOfFitness + distanceBetweenTrashCan[previousTruck][index];
        previousTruck = index;
    }
    sumOfFitness = sumOfFitness + distanceBetweenTrashCan[previousTruck][0];
    return sumOfFitness;
}
```

2) In each iteration it computes global best value from particles’ fitness value.



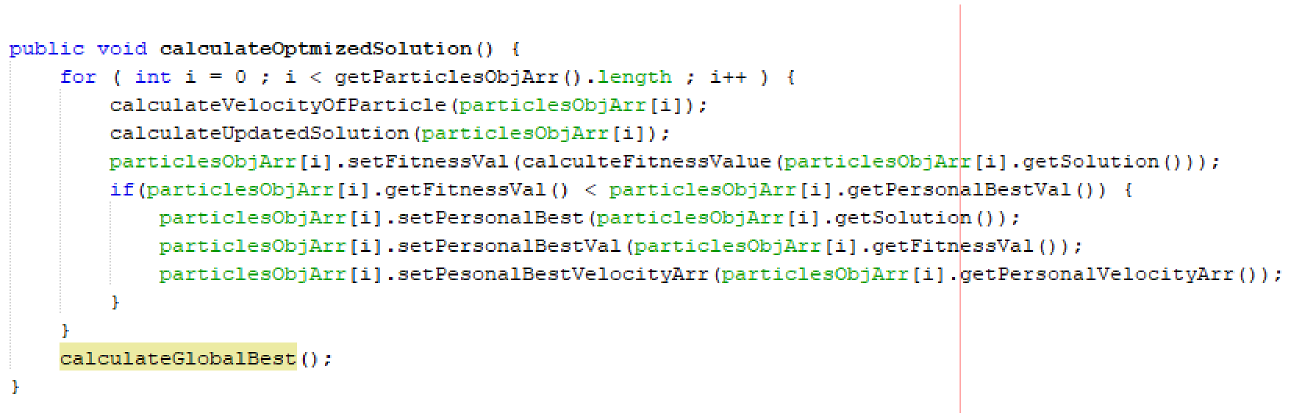
```
private void calculateGlobalBest() {
    for( int i = 0 ; i < particlesObjArr.length ; i++ ) {
        Particle tempParticleObj = particlesObjArr[i];
        if( tempParticleObj.getFitnessVal() < globalFitnessVal ) {
            globalFitnessVal = tempParticleObj.getFitnessVal();
            globalBestArr = tempParticleObj.getPersonalBest();
            globalBestVelocityArr = tempParticleObj.getPersonalVelocityArr();
        }
    }
}
```

3) It computes velocity and then updates the local solution.



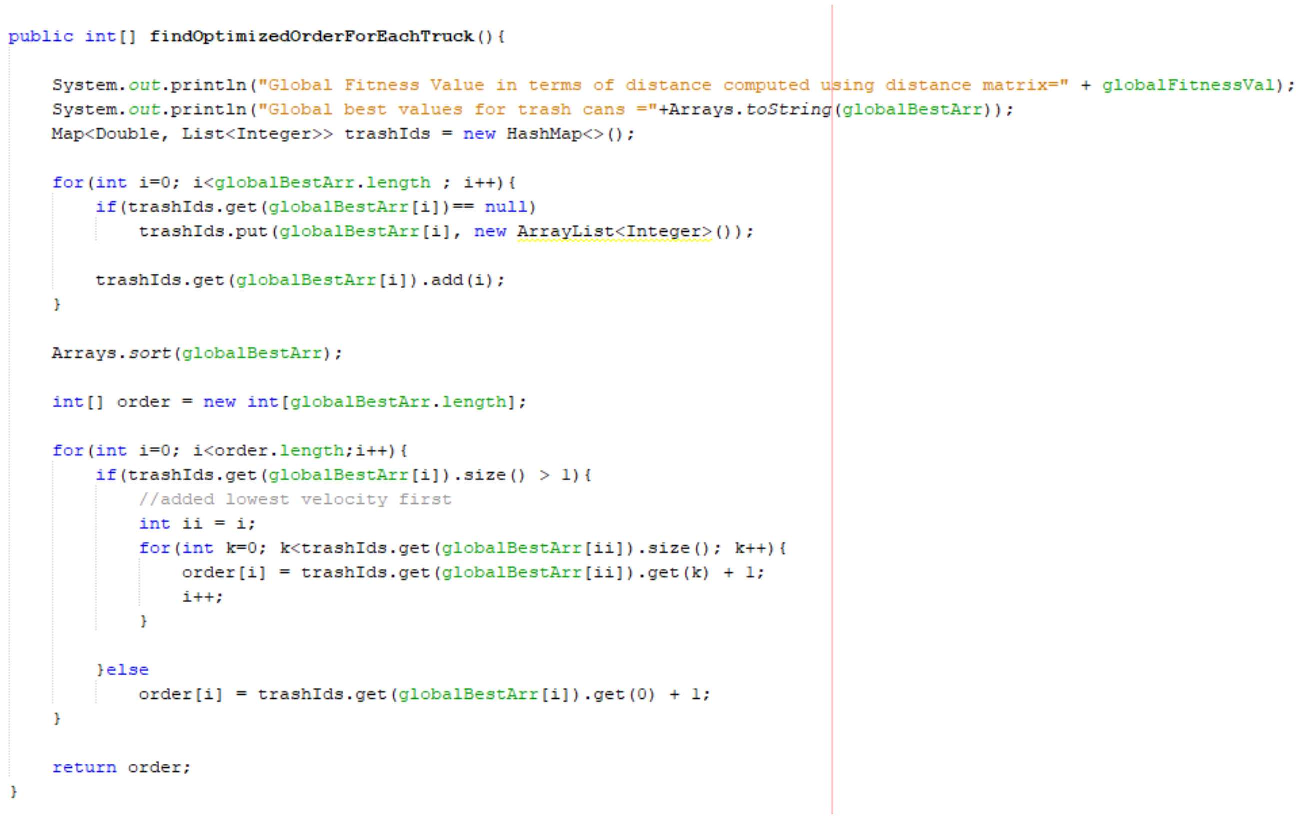
```
private void calculateVelocityOfParticle(Particle particleObj) {
    double [] newParticleVelocityArr = new double[particleObj.getPersonalVelocityArr().length];
    for( int i = 0 ; i < newParticleVelocityArr.length ; i++ ) {
        newParticleVelocityArr[i] = (0.8)*particleObj.getPersonalVelocityArr()[i] +
                ( 0.2 * 0.3 * (particleObj.getPersonalBest()[i] - particleObj.getSolution()[i])) +
                ( 0.2 * 0.3 * (globalBestArr[i] - particleObj.getSolution()[i]));
    }
    particleObj.setPersonalVelocityArr(newParticleVelocityArr);
}

private void calculateUpdatedSolution(Particle particleObj) {
    for( int i = 0 ; i < particleObj.getSolution().length ; i++ ) {
        particleObj.solution[i] = particleObj.solution[i] + particleObj.personalVelocityArr[i] > particleObj.solution.
    }
}
```

4) It computes the solution in each iteration and invoke other methods



```java
public void calculateOptmizedSolution() {
    for ( int i = 0 ; i < getParticlesObjArr().length ; i++ ) {
        calculateVelocityOfParticle(particlesObjArr[i]);
        calculateUpdatedSolution(particlesObjArr[i]);
        particlesObjArr[i].setFitnessVal(calculteFitnessValue(particlesObjArr[i].getSolution()));
        if(particlesObjArr[i].getFitnessVal() < particlesObjArr[i].getPersonalBestVal()) {
            particlesObjArr[i].setPersonalBest(particlesObjArr[i].getSolution());
            particlesObjArr[i].setPersonalBestVal(particlesObjArr[i].getFitnessVal());
            particlesObjArr[i].setPesonalBestVelocityArr(particlesObjArr[i].getPersonalVelocityArr());
        }
    }
    calculateGlobalBest();
}
```

5) This method computes the optimal order for each truck



```java
public int[] findOptimizedOrderForEachTruck(){

    System.out.println("Global Fitness Value in terms of distance computed using distance matrix=" + globalFitnessVal);
    System.out.println("Global best values for trash cans ="+Arrays.toString(globalBestArr));
    Map<Double, List<Integer>> trashIds = new HashMap<>();

    for(int i=0; i<globalBestArr.length ; i++){
        if(trashIds.get(globalBestArr[i])== null)
            trashIds.put(globalBestArr[i], new ArrayList<Integer>());

        trashIds.get(globalBestArr[i]).add(i);
    }

    Arrays.sort(globalBestArr);

    int[] order = new int[globalBestArr.length];

    for(int i=0; i<order.length;i++){
        if(trashIds.get(globalBestArr[i]).size() > 1){
            //added lowest velocity first
            int ii = i;
            for(int k=0; k<trashIds.get(globalBestArr[ii]).size(); k++){
                order[i] = trashIds.get(globalBestArr[ii]).get(k) + 1;
                i++;
            }

        }else
            order[i] = trashIds.get(globalBestArr[i]).get(0) + 1;
    }

    return order;
}
```

MultiSwarm using the concept of multithreading:

It is a transformation of Particle Swarm Optimization where multiple sub-swarms are used instead of one swarm. Each sub-swarm focuses on a specific region. This concept is implemented here by using multi-threading where each swarm has it’s own thread and each swarm consists of a group of particles which are taken as an input from user, and a total of three swarms a running simultaneously to provide their solution sets. The final solution of each swarm is then computed and we reach to a final optimal solution(which is the shortest path that is taken by each vehicle to collect trash cans from each location)

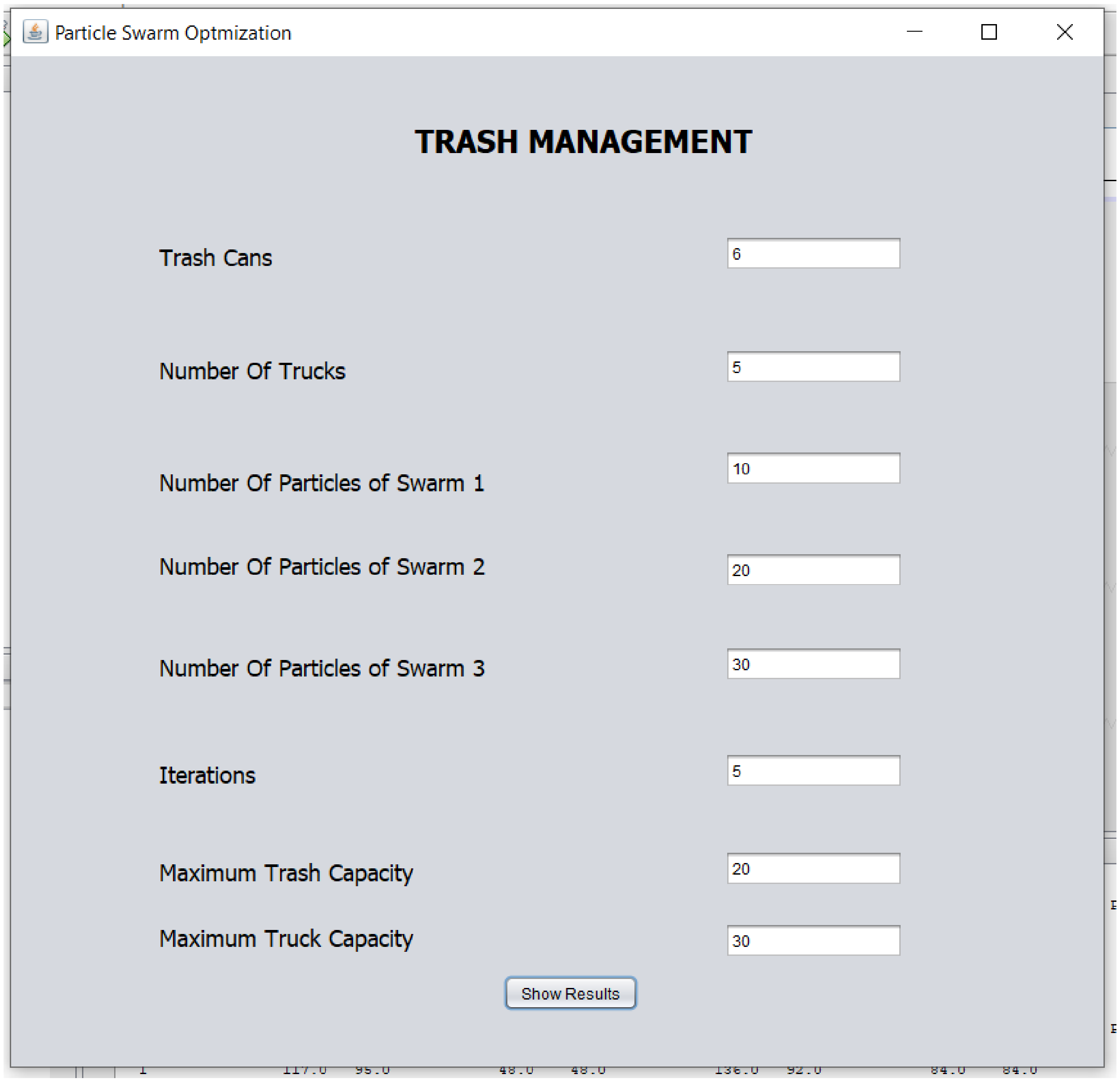
Concept of TimerTask:

It is used to implement the concept of multi-swarm where tasks are scheduled in background thread

(The ParticleModel class extends TimerTask class inorder to make the execution concurrent)

Console Output/Animation/UI

The following screenshot shows the input taken from the user from which the output is determined.


Particle Swarm Optmization

TRASH MANAGEMENT

Trash Cans	6
Number Of Trucks	5
Number Of Particles of Swarm 1	10
Number Of Particles of Swarm 2	20
Number Of Particles of Swarm 3	30
Iterations	5
Maximum Trash Capacity	20
Maximum Truck Capacity	30

The following console output shows the distance matrix which is generated using random numbers. Then Trash Can Capacities are printed(5 trash cans with random capacities upto the limit entered by the user).

Here Swarm1 contains 3 particles and it’s global best is 51.

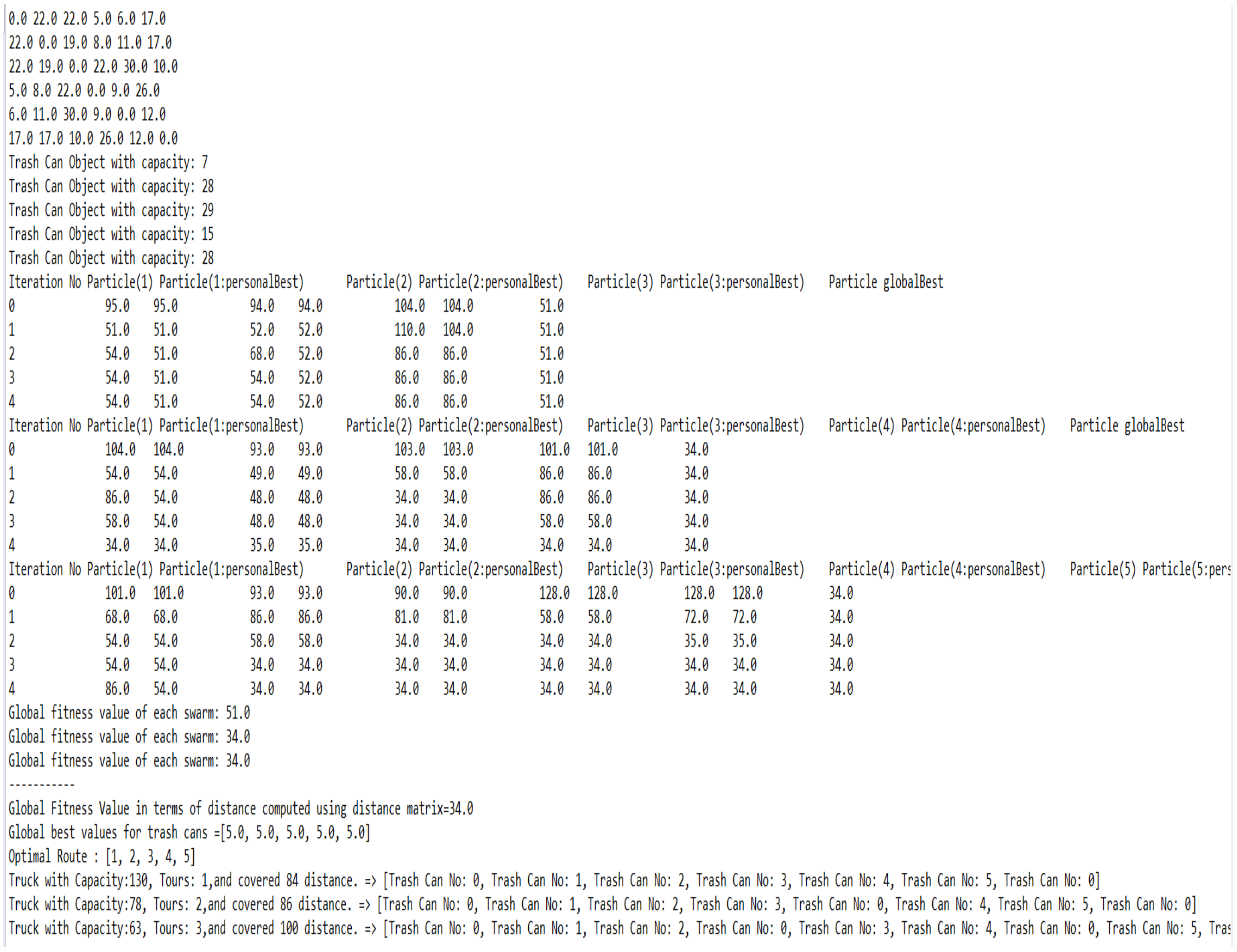
Swarm2 contains 4 particles and it’s global best is 34.

Swarm3 contains 5 particles and it’s global best is 34.

After computing the global best of these 3 swarms, 34 is taken as the global fitness value in terms of distance computed using distance matrix.

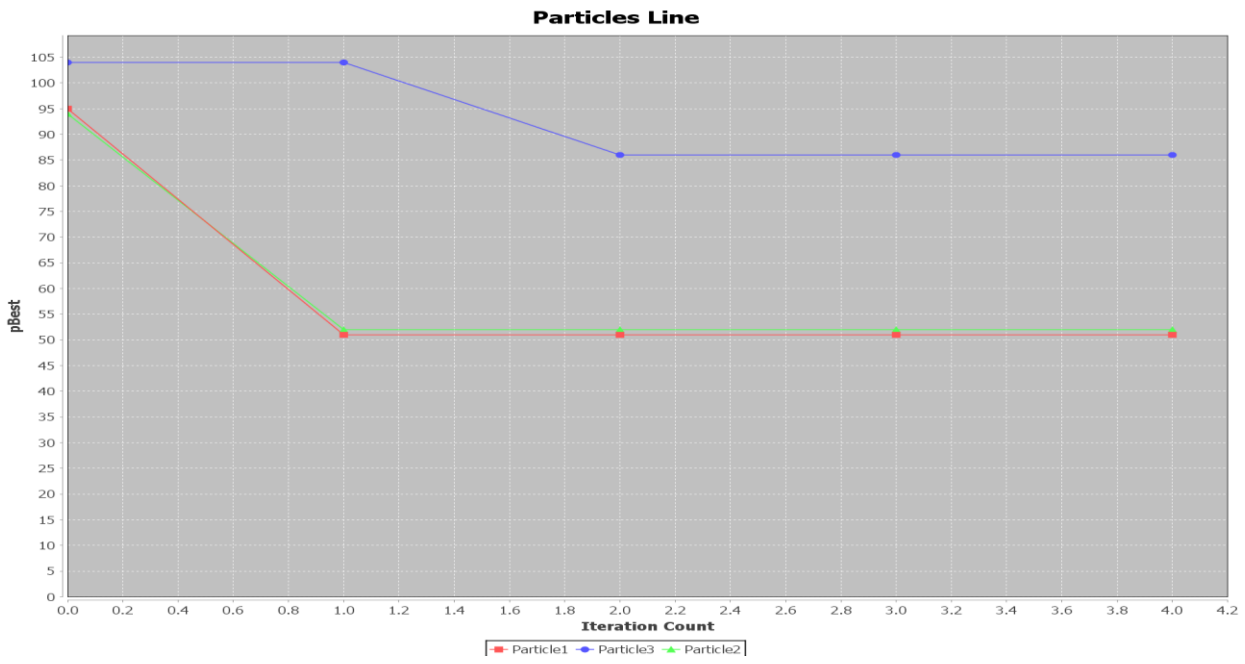
The optimal route is then calculated which in this case is 1,2,3,4,5

Finally each truck has to route through this route and as the truck capacity is different of each truck, they carry certain amount of trash, dump it at hub and hence the distance is different for all the trucks.

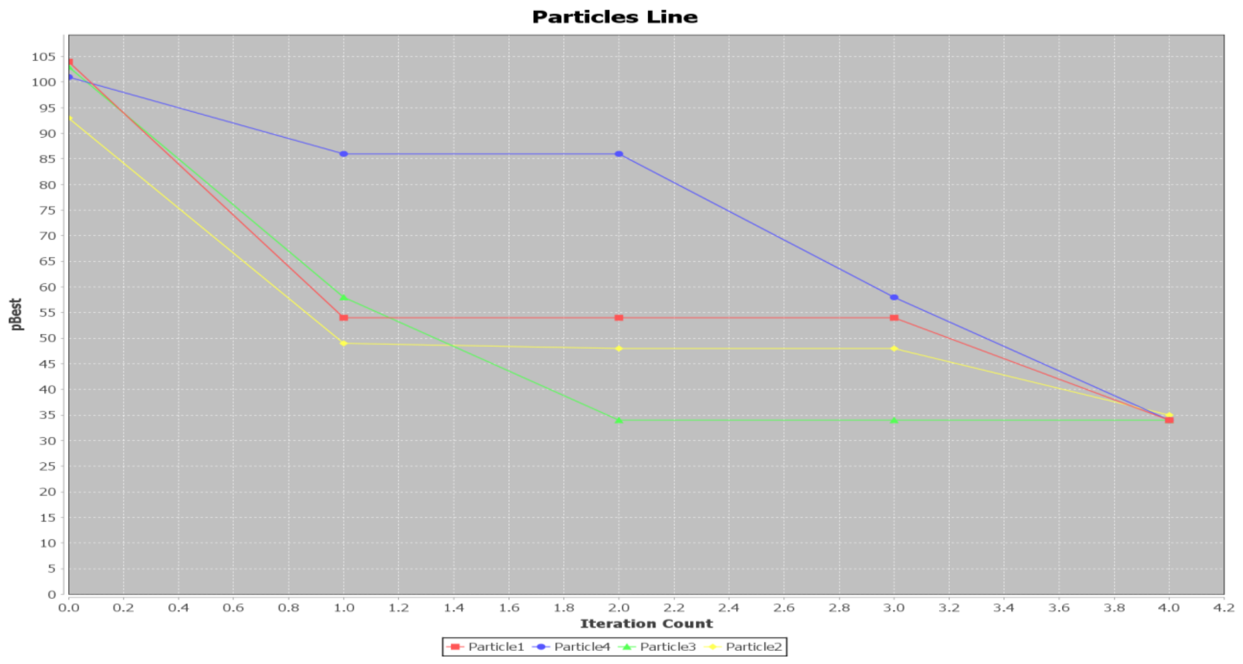


```
0.0 22.0 22.0 5.0 6.0 17.0
22.0 0.0 19.0 8.0 11.0 17.0
22.0 19.0 0.0 22.0 30.0 10.0
5.0 8.0 22.0 0.0 9.0 26.0
6.0 11.0 30.0 9.0 0.0 12.0
17.0 17.0 10.0 26.0 12.0 0.0
Trash Can Object with capacity: 7
Trash Can Object with capacity: 28
Trash Can Object with capacity: 29
Trash Can Object with capacity: 15
Trash Can Object with capacity: 28
Iteration No Particle(1) Particle(1:personalBest)       Particle(2) Particle(2:personalBest)    Particle(3) Particle(3:personalBest)    Particle globalBest
0               95.0    95.0            94.0    94.0            104.0   104.0           51.0
1               51.0    51.0            52.0    52.0            110.0   104.0           51.0
2               54.0    51.0            68.0    52.0            86.0    86.0            51.0
3               54.0    51.0            54.0    52.0            86.0    86.0            51.0
4               54.0    51.0            54.0    52.0            86.0    86.0            51.0
Iteration No Particle(1) Particle(1:personalBest)       Particle(2) Particle(2:personalBest)    Particle(3) Particle(3:personalBest)    Particle(4) Particle(4:personalBest)    Particle globalBest
0               104.0   104.0           93.0    93.0            103.0   103.0           101.0   101.0           34.0
1               54.0    54.0            49.0    49.0            58.0    58.0            86.0    86.0            34.0
2               86.0    54.0            48.0    48.0            34.0    34.0            86.0    86.0            34.0
3               58.0    54.0            48.0    48.0            34.0    34.0            58.0    58.0            34.0
4               34.0    34.0            35.0    35.0            34.0    34.0            34.0    34.0            34.0
Iteration No Particle(1) Particle(1:personalBest)       Particle(2) Particle(2:personalBest)    Particle(3) Particle(3:personalBest)    Particle(4) Particle(4:personalBest)    Particle(5) Particle(5:pers
0               101.0   101.0           93.0    93.0            90.0    90.0            128.0   128.0           128.0   128.0           34.0
1               68.0    68.0            86.0    86.0            81.0    81.0            58.0    58.0            72.0    72.0            34.0
2               54.0    54.0            58.0    58.0            34.0    34.0            34.0    34.0            35.0    35.0            34.0
3               54.0    54.0            34.0    34.0            34.0    34.0            34.0    34.0            34.0    34.0            34.0
4               86.0    54.0            34.0    34.0            34.0    34.0            34.0    34.0            34.0    34.0            34.0
Global fitness value of each swarm: 51.0
Global fitness value of each swarm: 34.0
Global fitness value of each swarm: 34.0
-----------
Global Fitness Value in terms of distance computed using distance matrix=34.0
Global best values for trash cans =[5.0, 5.0, 5.0, 5.0, 5.0]
Optimal Route : [1, 2, 3, 4, 5]
Truck with Capacity:130, Tours: 1,and covered 84 distance. => [Trash Can No: 0, Trash Can No: 1, Trash Can No: 2, Trash Can No: 3, Trash Can No: 4, Trash Can No: 5, Trash Can No: 0]
Truck with Capacity:78, Tours: 2,and covered 86 distance. => [Trash Can No: 0, Trash Can No: 1, Trash Can No: 2, Trash Can No: 3, Trash Can No: 0, Trash Can No: 4, Trash Can No: 5, Trash Can No: 0]
Truck with Capacity:63, Tours: 3,and covered 100 distance. => [Trash Can No: 0, Trash Can No: 1, Trash Can No: 2, Trash Can No: 0, Trash Can No: 3, Trash Can No: 4, Trash Can No: 0, Trash Can No: 5, Tras
```

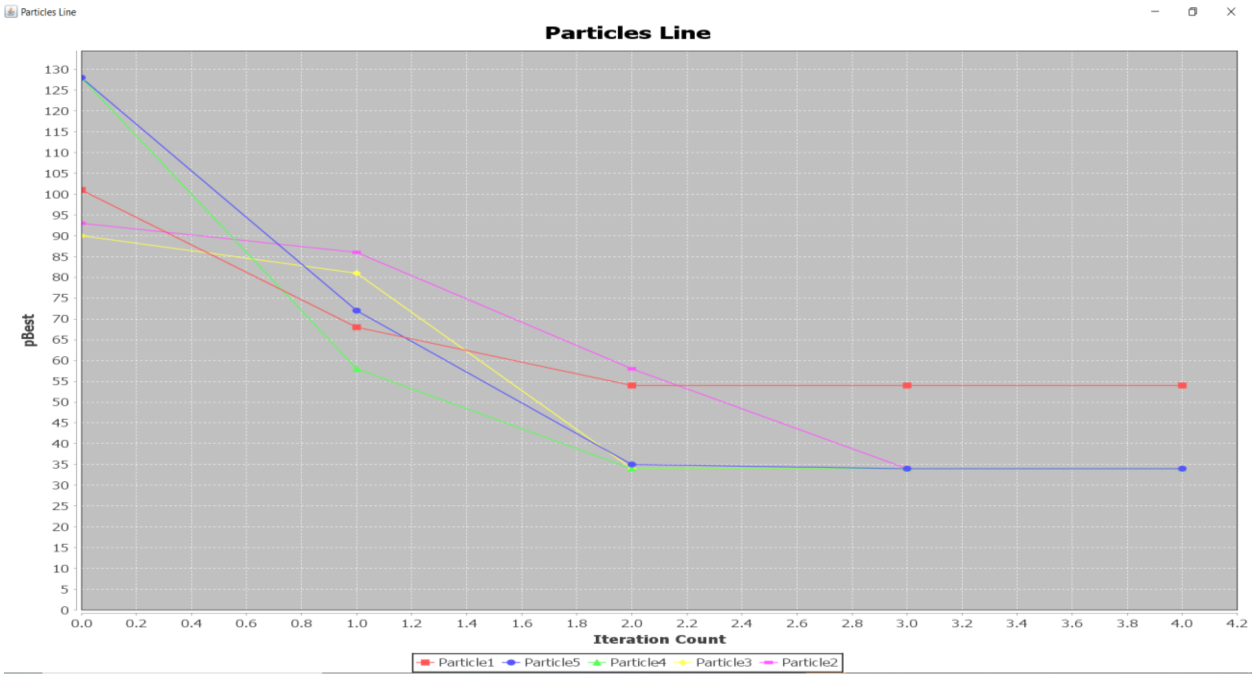
Line Graph of Swarm 1



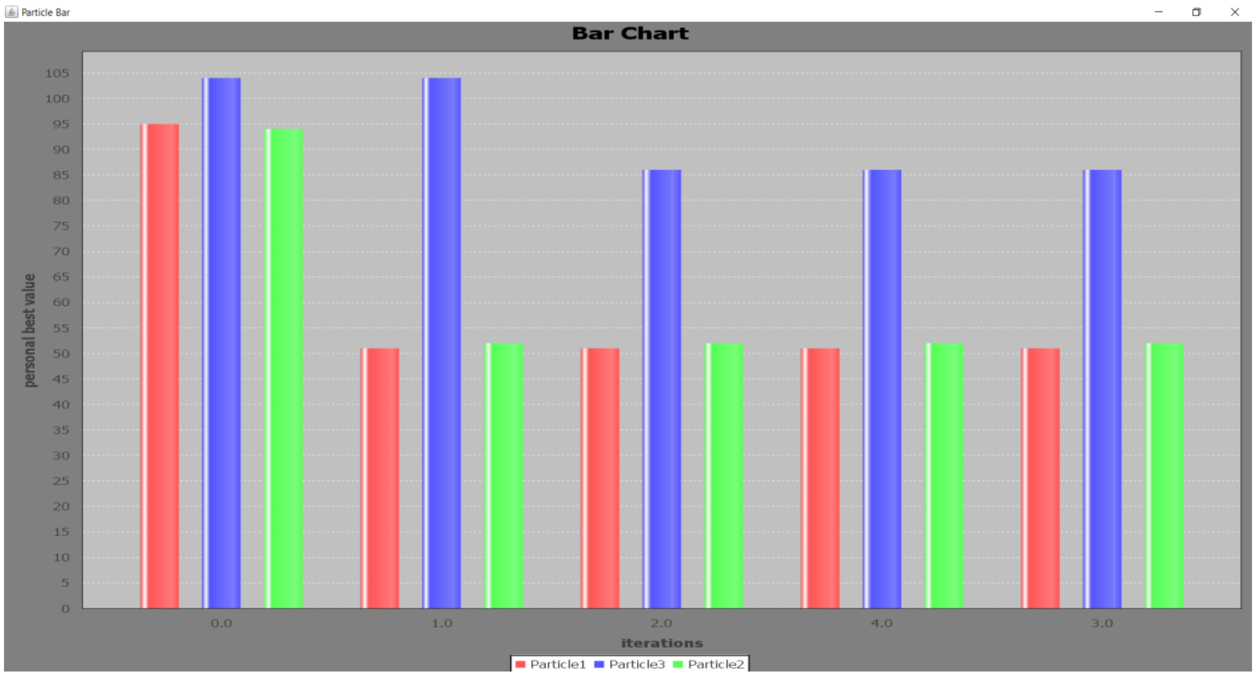
Line Graph of Swarm2



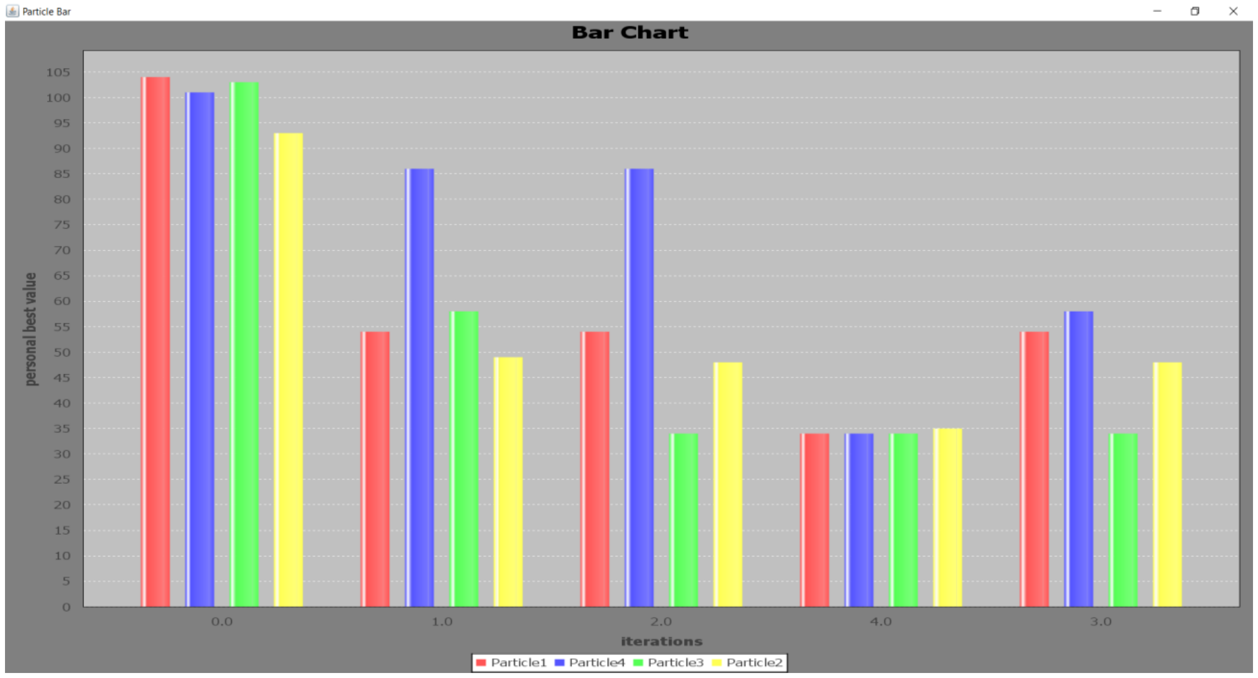
Line Graph Of Swarm 3



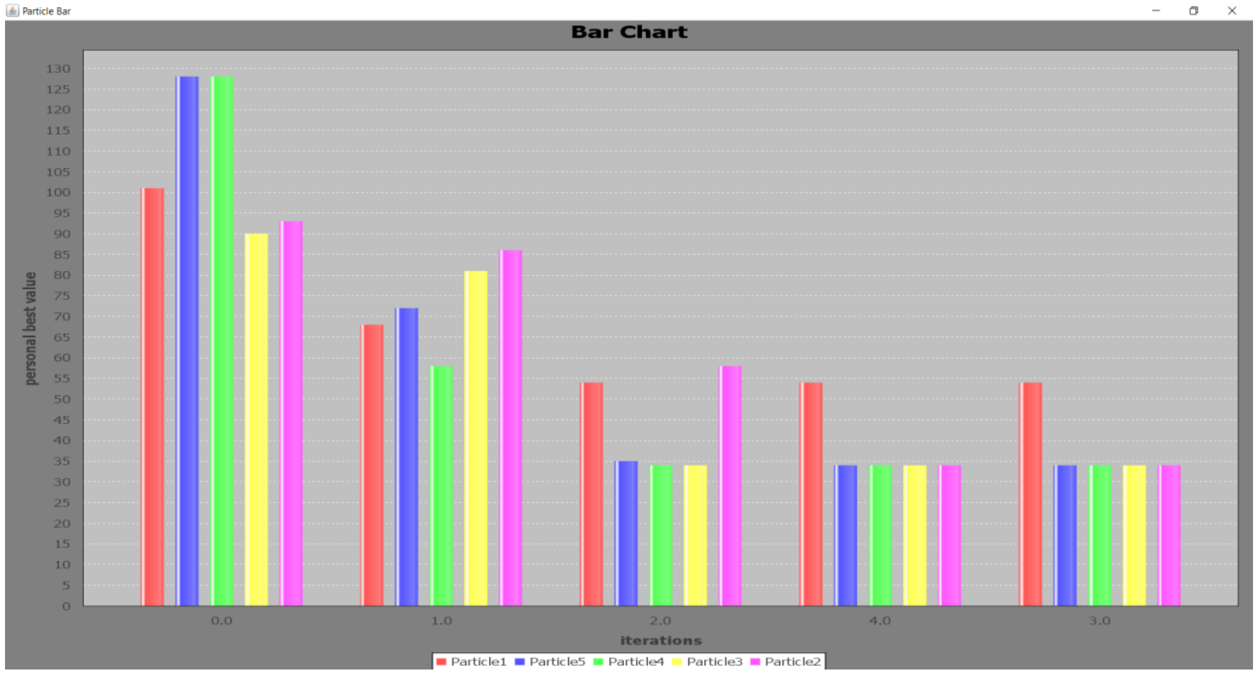
Bar Graph Of Swarm 1



Bar Graph Of Swarm 2

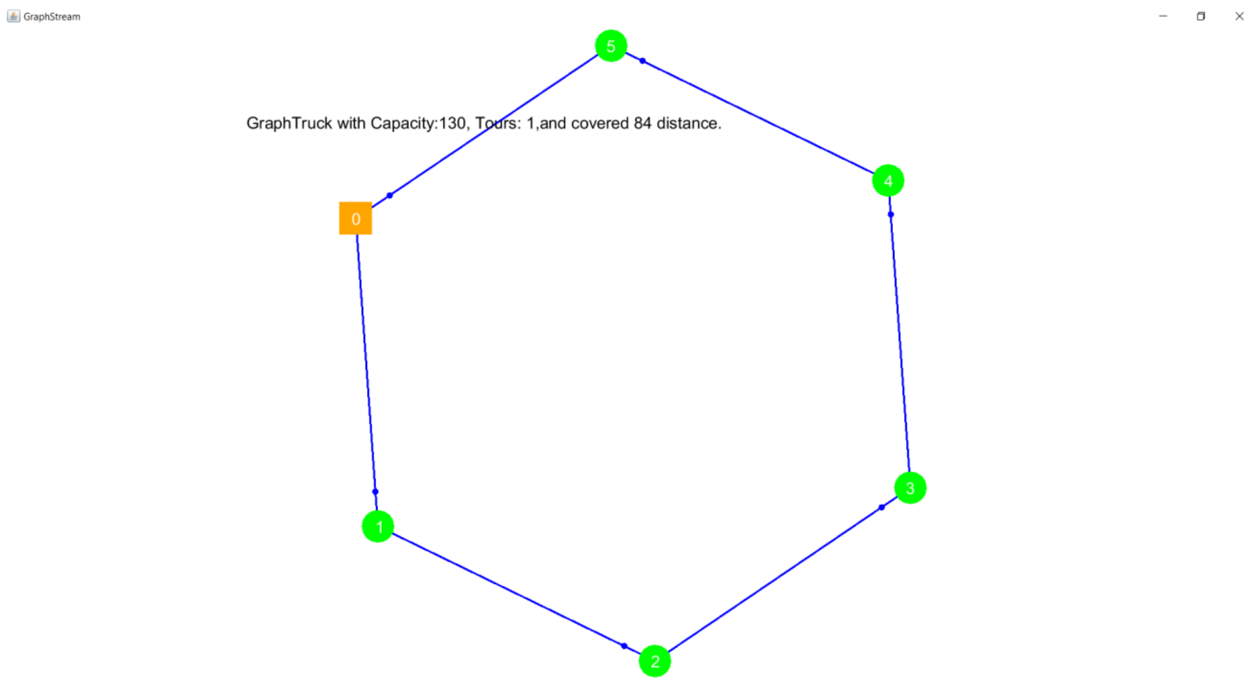


Bar Graph Of Swarm 3



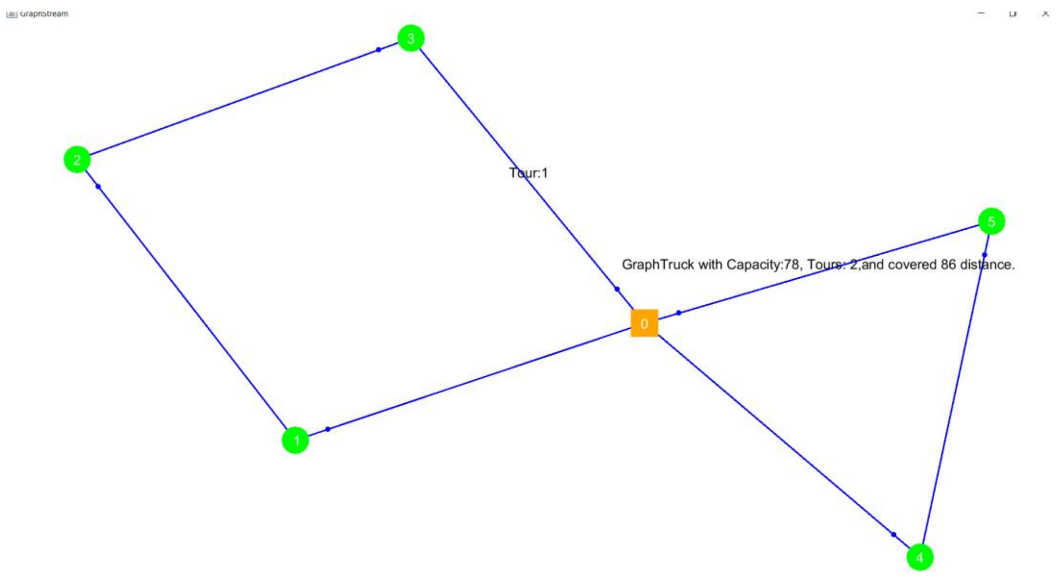
Animation Depicting the distance covered by the trucks in the optimal route

1) Truck Capacity=130 , Distance Covered=84



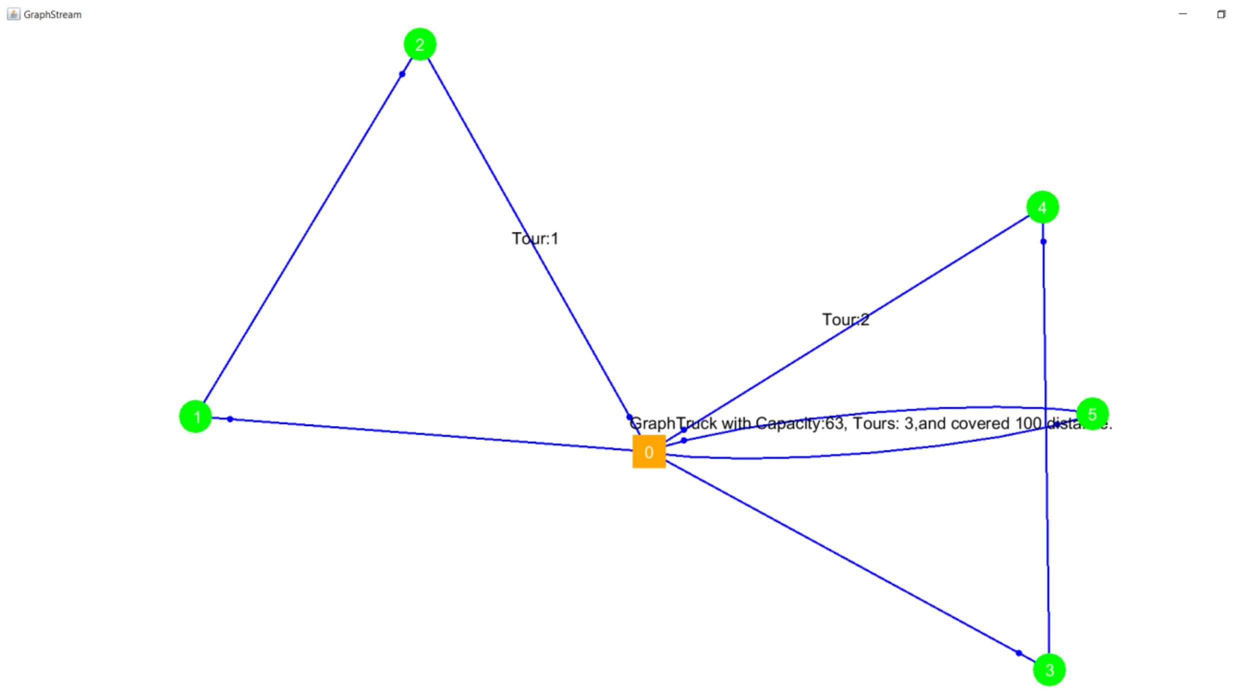
GraphTruck with Capacity:130, Tours: 1,and covered 84 distance.

2) Truck Capacity=78, Distance Covered=86



GraphTruck with Capacity:78, Tours: 2,and covered 86 distance.

3) Truck Capacity=63, Distance Covered=100



GraphTruck with Capacity:63, Tours: 3,and covered 100 distance.

Conclusion

As the given problem set is a NP-hard problem, it is very difficult to decide what could be the best solution. Hence, we have limited the number of iterations in which we can decide what could be the best solution from those iterations. PSO is implemented in order to solve this problem and it works for medium size inputs (neither too small nor too large).

References:

1) https://www.researchgate.net/publication/50996638_Application_of_Particle_Swarm_Optimization_to_Formative_E-Assessment_in_Project_Management
2) https://www.youtube.com/watch?v=DiAdlCOtHVQ
3) https://www.youtube.com/watch?v=Rn1kcmG9AUU
4) https://www.journaldev.com/1050/java-timer-timertask-example
5) https://www.sciencedirect.com/science/article/pii/S1568494618301017
6) https://www.geeksforgeeks.org/introduction-to-particle-swarm-optimizationpso/