**Report on Final Project:**

**DETERMINATION OF SAFE AREAS FOR SUBMARINES USING PSO**

**1.Introduction:**

Particle swarm optimization (PSO) is a heuristic global optimization method, proposed originally by Kennedy and Eberhart in 1995.It is now one of the most commonly used optimization techniques. There is no centralized control structure dictating how individual agents should behave. The agents’ real behaviors are local, and to a certain degree random; however, interactions between such agents lead to the emergence of “intelligent” global behavior, which is unknown to the individual agents. Well known examples of SI include ant colonies, bird flocking, animal herding, bacterial growth, and fish schooling.

A naval mine is a self-contained explosive device placed in water to damage or destroy ships and submarines. Mines are deposited and left to wait until they are triggered by the approach of/ or contact with an enemy vessel. The fuses on many mines may incorporate one or more of the following sensors: magnetic, passive acoustic or water pressure displacement caused by the proximity of the vessel.

**Problem Statement:**

There are many different types of mines installed in the seas during wars. The submarines, ships, aircrafts traveling across the seas determine the existence of mines using the sonar contact system. But these sonar contact systems can go out of range or may get affected due to some distractions in the sea. Every naval craft has its own properties and tries individually to get a way away from the mine to a safer area. No individual ship in this case knows the best or optimized solution(way) to get farther from the mine. And, in this case there cannot be any centralized system controlling all the system.

**Proposed Solution:**

Therefore these individual systems(ships) have to determine their own values or properties considering the environment, behavior of other ships and also its own behavior. This helps for every ship to follow the optimized solution(way) at any time indirectly. This system is more apt to implement PSO, because PSO works with the local particles that do not have any centralized control system to calculate and present the best solution using all the local values at its place.

**PSO as the Approach:**

The aircrafts/ships/submarines are considered as the particles in the particle swarm optimization.The standard implementation of PSO , considers the position and velocity of the particle as individual and one-dimensional properties. Unlike the standard implementation of PSO, in this case, two dimensional properties are considered for the particles i.e., position is considered to have x and y co-ordinates and the velocity of the particle is derived from the sonar contact range and the depth of the particle. The choice of fitness function in the standard implementation of PSO is widely ranged among many different formulae from the real world math solution. In this case, we need to displace every ship farther from the mine. Therefore, at each iteration and for every particle’s fitness determination, we had to calculate the current distance between the mine and the particle. This calculation requires the implementation of Euclidean formula of getting the distance between the mine(Px, Py) and the target-mine particle(Tx,Ty).

**Pseudocode of the Algorithm Implemented:**

Step 1.Initialization

For each particle 𝑖 = 1, . . . , 𝑁𝑃, do

(a) Initialize the particle’s position with a uniformly distribution as 𝑃𝑖(0)∼*U*(LB, UB), where LB and UB represent the lower

and upper bounds of the search space

(b) Initialize 𝑝𝑏𝑒𝑠𝑡 to its initial position: 𝑝𝑏𝑒𝑠𝑡 = 𝑃ipos.

(c) Initialize 𝑔𝑏𝑒𝑠𝑡 to the minimal value of the swarm: 𝑔𝑏𝑒𝑠𝑡(0) = argmax𝑓[𝑃best𝑖].

(d) Initialize velocity: 𝑉𝑖 ∼ 𝑈(−|UB − LB|, |UB − LB|).

Step2. Initialize timertasks for each particle

Schedule them to a timer object using fixed rate

Step2.1

Foreach task

//updates the particle’s position and velocity

Step 2.2

//updates the gBest of the swarm

Step 3: Return to main Thread and get cancelled

**Properties:**

:Particle

* position : Position
* velocity : Velocity
* bestPosition : Position
* bestFitness : double
* fitness : double

:Swarm

- numberOfParticles : int

- particles : ArrayList<Particle>

- gbest : double

- gBestPosition : Position

:Position

- x : double

- y : double

:Velocity

- x : double

- y : double

:FitnessFunction

- mineParticle : Particle

- severityRange : double

**Equations Modified:**

newVelocityX=((w\*lastVelocityX)+(r1\*C1)\*(pBestX-lastX)+(r2\*C2)\*(gBestX-lastX));

newVelocityY=((w\*lastVelocityY)+(r1\*C1)\*(pBestY-lastY)+(r2\*C2)\*(gBestY-lastY));

particles.get(i).setVelocity(new Velocity(newVelocityX,newVelocityY));

newPosX=lastX+newVelocityX;

newPosY=lastY+newVelocityY;

particles.get(i).setPosition(new Position(newPosX,newPosY));

**Initializing Particles:**

The particles are initialized by taking random values in the range given for each attribute i.e., positionX,positionY,velocityX,velocityY.Current positions are said to be bestpositions for every particle. And also the current fitness is the best fitness for every particle.But after initializing all these particles, the gBest is calculated from the pBest values of the swarm.

**Updating the Positions and velocities:**

This function takes particle’s index in the list of particles in the swarm and updates it.The particle’s position and velocity values are calculated using the standard PSO formulae.In the real case the sonar contact reading differ according to the change in the distance between the mine and the ship. So, while calculating the newvelocity, this is also a factor in addition to the localPbest(pbest of that ship) and the globalbest(the max[ *fi =1 to n*(pbest(i))] )Note:Same as with the newVelocityY.

Time Complexity=O(1)

**Local Objective Function:**

Every Particle has one value called bestFitness, one bestPosition and the fitness value changes for each iteration.Once the particle is initialized, the particle’s current fitness is set as bestfitness and the bestposition of the particle is the initial position.But after each iteration the value of the bestfitness may change if the current fitness is greater than the prvious pBest value, it is replaced with the current fitness value.And the bestposition is set to be the currentposition.This function of updating the pbest value at one bestfitness, makes it easy to calculate the gbest of the swarm and also the next positions in succeeding iterations.

Time Complexity:O(1)

**Global Objective Function:**

During every iteration the fitnessValues array stores the pBest values of every particle following the index of the arrayList of particles in the swarm.And if the currentGBest value is lesser than any of these particles’ pBest values, the gBest value is modified and the Gbest position of this iteration is said to be the position of the particle, which has the highest pBest of all the list of particles in the swarm.

Time Complexity: O(n)

**What if we don’t use Timertask?**

If we haven’t used TimerTask for this, traditional method to adopt is the iteration through 2 for loops .

for(int i=0;i<numberOfIterations;i++)

{

for(int j=0;j<numberOfParticles;j++)

{

//updateParticle;

}

//updateSwarm;

}

Time complexity for this could have been : O(n^2).

So it is better to opt concurrent programming. The first option that comes to our mind is THREAD. Using thread.join() for concurrent computing of fitness of every particle during each iteration takes a for loop .

public void run() {

for( int i = 0; i < controller.getNumberOfIterations(); i++ ) {

//compute the pbest values of each particle

}

}

In this case there can be some Interrupted exceptions and we are not automating the process of optimization.

protected void runSwarm() {

Thread thread = new SwarmThread(controller);

try {

thread.join();

} catch(InterruptedException e) {

//catch exception

} // It's OK to interrupt this process

}

**Using Timertask:**

Using Timertask makes it even easier than using threads because these timertasks use Timer object.

**Result from Analysis:**