**PARALLEL AND DISTRIBUTED COMPUTING**

**DISTRIBUTED PARAELLEL GENETIC**

**ALGORITHM ON AD HOC N/W**

Submitted to: - Prof Manoov R

Group members:-

**SIDHARTH UNNI (15BCE0075)**

**RAMEEN ABDUL AZEEZ (15BCE0257)**

***Problem statement:***

The Ad Hoc network consists of several distributed nodes. The problem statement consists of finding optimum path for travelling all the nodes in the minimum time.

The given problem is equivalent to Travelling Salesman problem. This problem is in a standard form in mathematics as finding a Hamiltonian Cycle with the minimum weight in an undirected weighted graph with n vertices in which vertices indicate cities. Edges show intercity paths and weight shows intercity distance. The number of vertices in a complete graph equals m=n\*(n-1)/2.

**PRINCIPLE**

The final method for solving the travelling salesman problem includes passing all possible paths, evaluating the relating travel distance and finding the travel with the minimum distance. The total number of passable paths for n city equals n! Consequently, for large values, finding the cost of all travels is very time consuming or sometimes impossible. The parallel process will be very useful in reducing the computing time of this problem.

**ALGORITHIM**

Steps involved in genetic algorithm are:

1. Initialization: The coordinates of nodes, distance among the nodes are noted.
2. Evaluation: Each member is then evaluated and “fitness” value is calculated.
3. Selection: The basic idea of selection is that fitter individuals are selected for second generation.
4. Crossover: In ordered crossover method we select a subset from the first parent, and then add that subset to the offspring. Any missing values are then adding to the offspring from the second parent in order that they are found.
5. Mutation: With swap mutation two location in the route are selected at random then their positions are simply swapped.
6. Repeat

**CODE**

**Node.java**

package adhoc;

public class Node {

int x;

int y;

public Node(){

this.x = (int)(Math.random()\*200);

this.y = (int)(Math.random()\*200);

}

public Node(int x, int y){

this.x = x;

this.y = y;

}

public int getX(){

7

return this.x;

}

public int getY(){

return this.y;

}

public double distanceTo(Node Node){

int xDistance = Math.abs(getX() - Node.getX()); int yDistance = Math.abs(getY() - Node.getY());

double distance = Math.sqrt( (xDistance\*xDistance) + (yDistance\*yDistance) );

return distance;

}

@Override

public String toString(){

return getX()+", "+getY();

}

}

8

**RoutesManager.java**

package adhoc;

import java.util.ArrayList;

public class Routesmanager {

private static ArrayList destinationNodes = new ArrayList<Node>();

public static void addNode(Node node) {

destinationNodes.add(node);

}

public static Node getNode(int index){

return (Node)destinationNodes.get(index);

}

public static int numberOfNodes(){

return destinationNodes.size();

}

}

9

**Routes.java**

package adhoc;

import java.util.ArrayList;

import java.util.Collections;

public class Routes{

private ArrayList routes = new ArrayList<Node>();

private double fitness = 0;

private int distance = 0;

public Routes(){

for (int i = 0; i < Routesmanager.numberOfNodes(); i++) { routes.add(null);

}

}

public Routes(ArrayList routes){

this.routes = routes;

}

10

public void generateIndividual() {

for (int nodeIndex = 0; nodeIndex < Routesmanager.numberOfNodes(); nodeIndex++) { setNode(nodeIndex, Routesmanager.getNode(nodeIndex));

}

Collections.shuffle(routes);

}

public Node getNode(int routesPosition) {

return (Node)routes.get(routesPosition);

}

public void setNode(int routesPosition, Node node) { routes.set(routesPosition, node);

fitness = 0;

distance = 0;

}

public double getFitness() {

if (fitness == 0) {

fitness = 1/(double)getDistance();

}

11

return fitness;

}

public int getDistance(){

if (distance == 0) {

int routesDistance = 0;

for (int nodeIndex=0; nodeIndex < routesSize(); nodeIndex++) {

Node fromNode = getNode(nodeIndex);

Node destinationNode;

if(nodeIndex+1 < routesSize()){

destinationNode = getNode(nodeIndex+1);

}

else{

destinationNode = getNode(0);

}

routesDistance += fromNode.distanceTo(destinationNode);

}

distance = routesDistance;

}

return distance;

}

12

public int routesSize() {

return routes.size();

}

public boolean containsNode(Node node){

return routes.contains(node);

}

@Override

public String toString() {

String geneString = "|";

for (int i = 0; i < routesSize(); i++) {

geneString += getNode(i)+"|";

}

return geneString;

}

}

13

**Network.java**

package adhoc;

public class Network {

Routes[] tours;

public Network(int networkSize, boolean initialise) { tours = new Routes[networkSize];

if (initialise) {

for (int i = 0; i < networkSize(); i++) {

Routes newRoutes = new Routes();

newRoutes.generateIndividual();

saveRoutes(i, newRoutes);

}

}

}

public void saveRoutes(int index, Routes tour) { tours[index] = tour;

}

14

public Routes getRoutes(int index) {

return tours[index];

}

public Routes getFittest() {

Routes fittest = tours[0];

for (int i = 1; i < networkSize(); i++) {

if (fittest.getFitness() <= getRoutes(i).getFitness()) { fittest = getRoutes(i);

}

}

return fittest;

}

public int networkSize() {

return tours.length;

}

}

15

**GA.java**

package adhoc;

public class GA {

private static final double mutationRate = 0.015; private static final int routesnamentSize = 5; private static final boolean elitism = true;

public static Network evolveNetwork(Network net) {

Network newNetwork = new Network(net.networkSize(), false);

int elitismOffset = 0;

if (elitism) {

newNetwork.saveRoutes(0, net.getFittest());

elitismOffset = 1;

}

for (int i = elitismOffset; i < newNetwork.networkSize(); i++) {

Routes parent1 = routesnamentSelection(net);

Routes parent2 = routesnamentSelection(net);

16

Routes child = crossover(parent1, parent2);

newNetwork.saveRoutes(i, child);

}

for (int i = elitismOffset; i < newNetwork.networkSize(); i++) { mutate(newNetwork.getRoutes(i));

}

return newNetwork;

}

public static Routes crossover(Routes parent1, Routes parent2) {

Routes child = new Routes();

int startPos = (int) (Math.random() \* parent1.routesSize()); int endPos = (int) (Math.random() \* parent1.routesSize());

for (int i = 0; i < child.routesSize(); i++) {

if (startPos < endPos && i > startPos && i < endPos) {

17

child.setNode(i, parent1.getNode(i));

}

else if (startPos > endPos) {

if (!(i < startPos && i > endPos)) {

child.setNode(i, parent1.getNode(i));

}

}

}

for (int i = 0; i < parent2.routesSize(); i++) {

if (!child.containsNode(parent2.getNode(i))) {

for (int ii = 0; ii < child.routesSize(); ii++) {

if (child.getNode(ii) == null) {

child.setNode(ii, parent2.getNode(i));

break;

}

}

}

}

return child;

}

18

private static void mutate(Routes routes) {

for(int routesPos1=0; routesPos1 < routes.routesSize(); routesPos1++){

if(Math.random() < mutationRate){

int routesPos2 = (int) (routes.routesSize() \* Math.random());

Node city1 = routes.getNode(routesPos1);

Node city2 = routes.getNode(routesPos2);

routes.setNode(routesPos2, city1);

routes.setNode(routesPos1, city2);

}

}

}

private static Routes routesnamentSelection(Network net) {

Network routesnament = new Network(routesnamentSize, false);

for (int i = 0; i < routesnamentSize; i++) {

int randomId = (int) (Math.random() \* net.networkSize()); routesnament.saveRoutes(i, net.getRoutes(randomId));

19

}

Routes fittest = routesnament.getFittest();

return fittest;

}

}

**adhoc\_GA.java**

package adhoc;

public class adhoc\_GA {

public static void main(String[] args) {

Node node = new Node(60, 200);

Routesmanager.addNode(node);

Node node2 = new Node(180, 200);

Routesmanager.addNode(node2);

Node node3 = new Node(80, 180);

Routesmanager.addNode(node3);

Node node4 = new Node(140, 180);

Routesmanager.addNode(node4);

Node node5 = new Node(20, 160);

Routesmanager.addNode(node5);

Node node6 = new Node(100, 160);

Routesmanager.addNode(node6);

20

Node node7 = new Node(200, 160);

Routesmanager.addNode(node7);

Node node8 = new Node(140, 140);

Routesmanager.addNode(node8);

Node node9 = new Node(40, 120);

Routesmanager.addNode(node9);

Node node10 = new Node(100, 120);

Routesmanager.addNode(node10);

Node node11 = new Node(180, 100);

Routesmanager.addNode(node11);

Node node12 = new Node(60, 80);

Routesmanager.addNode(node12);

Node node13 = new Node(120, 80);

Routesmanager.addNode(node13);

Node node14 = new Node(180, 60);

Routesmanager.addNode(node14);

Node node15 = new Node(20, 40);

Routesmanager.addNode(node15);

Node node16 = new Node(100, 40);

Routesmanager.addNode(node16);

Node node17 = new Node(200, 40);

Routesmanager.addNode(node17);

Node node18 = new Node(20, 20);

Routesmanager.addNode(node18);

Node node19 = new Node(60, 20);

Routesmanager.addNode(node19);

Node node20 = new Node(160, 20);

21

Routesmanager.addNode(node20);

Network net = new Network(50, true);

System.out.println("Initial distance: " + net.getFittest().getDistance());

net = GA.evolveNetwork(net);

for (int i = 0; i < 100; i++) {

net = GA.evolveNetwork(net);

}

System.out.println("Finished");

System.out.println("Final distance: " + net.getFittest().getDistance());

System.out.println("Solution:");

System.out.println(net.getFittest());

}

}

**OUTPUT**

Initial distance: 1879

Finished

Final distance: 928

Solution:

|100, 120|140, 140|140, 180|180, 200|200, 160|180, 100|180, 60|200, 40|160, 20|120, 80|100, 40|60, 20|20, 20|20, 40|60, 80|100, 160|80, 180|60, 200|20, 160|40, 120|

