*дата :*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| КНC-13 | Лабораторна робота №5 | Рішення задачі комівояжера за допомогою генетичного алгоритму |  |  |
| Дробко В.М. | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р. З. | |

**Мета:** Ознайомитися з основними теоретичними відомостями про задачу комівояжера, обрати середовище розробки та мову програмування, реалізувати вирішення задачі комівояжера для [10; 50] міст за допомогою генетичного алгоритму

**Теоретичні відомості**

Задача комівояжера, Travelling Salesman Problem або TSP, полягає у знаходженні найвигіднішого маршруту, що проходить через вказані міста хоча б по одному разу. В умовах завдання вказуються критерій вигідності маршруту (найкоротший, найдешевший, сукупний критерій тощо) і відповідні матриці відстаней, вартості тощо. Зазвичай задано, що маршрут повинен проходити через кожне місто тільки один раз, в такому випадку розв'язок знаходиться серед гамільтонових циклів. Прості методи розв'язання задачі комівояжера: повний лексичний перебір, жадібні алгоритми (метод найближчого сусіда), метод включення найближчого міста, метод найдешевшого включення, метод мінімального кістяка дерева. На практиці застосовують різні модифікації ефективніших методів: метод гілок і меж і метод генетичних алгоритмів, а так само алгоритм мурашиної колонії. Всі ефективні (такі, що скорочують повний перебір) методи розв'язання задачі комівояжера — евристичні. У більшості евристичних методів знаходиться не найефективніший маршрут, а наближений розв'язок. Користуються популярністю так звані any-time алгоритми, тобто алгоритми, що поступово покращують деякий поточний наближений розв'язок.

**Індивідуальне завдання. Варіант 7(3)**

Запрограмувати генетичний алгоритм для задачі комівояжера. Мета полягає в тому, щоб знайти найкоротший маршрут, який відвідує кожне місто рівно один раз, повертаючись в кінці своєї відправної точки. Дано від 10 до 50 точок .

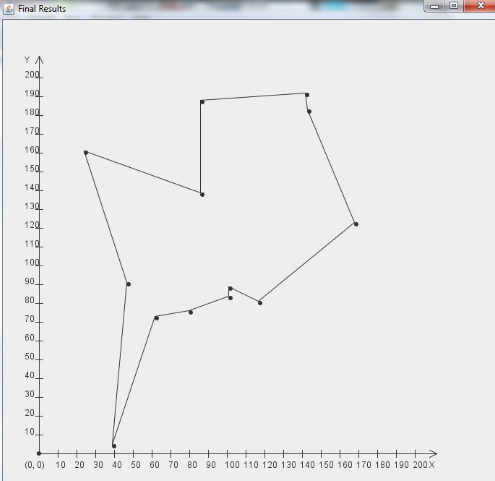
**Фрагменти коду програми**

package com.mnl.lab5;  
  
public class GA {  
  
 private static final double mutationRate = 0.015;  
 private static final int tournamentSize = 5;  
 private static final boolean elitism = true;  
  
  
 public static Population evolvePopulation(Population pop) {  
 Population newPopulation = new Population(pop.populationSize(), false);  
  
 int elitismOffset = 0;  
 if (elitism) {  
 newPopulation.saveTour(0, pop.getFittest());  
 elitismOffset = 1;  
 }  
  
 for (int i = elitismOffset; i < newPopulation.populationSize(); i++) {  
 Tour parent1 = tournamentSelection(pop);  
 Tour parent2 = tournamentSelection(pop);  
 Tour child = crossover(parent1, parent2);  
 newPopulation.saveTour(i, child);  
 }  
  
 for (int i = elitismOffset; i < newPopulation.populationSize(); i++) {  
 mutate(newPopulation.getTour(i));  
 }  
  
 return newPopulation;  
 }  
  
 public static Tour crossover(Tour parent1, Tour parent2) {  
 Tour child = new Tour();  
  
 int startPos = (int) (Math.random() \* parent1.tourSize());  
 int endPos = (int) (Math.random() \* parent1.tourSize());  
  
 for (int i = 0; i < child.tourSize(); i++) {  
  
 if (startPos < endPos && i > startPos && i < endPos) {  
 child.setCity(i, parent1.getCity(i));  
 }  
 else if (startPos > endPos) {  
 if (!(i < startPos && i > endPos)) {  
 child.setCity(i, parent1.getCity(i));  
 }  
 }  
 }  
 for (int i = 0; i < parent2.tourSize(); i++) {  
 if (!child.containsCity(parent2.getCity(i))) {  
 for (int ii = 0; ii < child.tourSize(); ii++) {  
 if (child.getCity(ii) == null) {  
 child.setCity(ii, parent2.getCity(i));  
 break;  
 }  
 }  
 }  
 }  
 return child;  
 }  
  
 private static void mutate(Tour tour) {  
 for(int tourPos1=0; tourPos1 < tour.tourSize(); tourPos1++){  
 if(Math.random() < mutationRate){  
 int tourPos2 = (int) (tour.tourSize() \* Math.random());  
 City city1 = tour.getCity(tourPos1);  
 City city2 = tour.getCity(tourPos2);  
  
 tour.setCity(tourPos2, city1);  
 tour.setCity(tourPos1, city2);  
 }  
 }  
 }  
 private static Tour tournamentSelection(Population pop) {  
 Population tournament = new Population(tournamentSize, false);  
 for (int i = 0; i < tournamentSize; i++) {  
 int randomId = (int) (Math.random() \* pop.populationSize());  
 tournament.saveTour(i, pop.getTour(randomId));  
 }  
  
 Tour fittest = tournament.getFittest();  
 return fittest;  
 }  
}

**Результат виконання**

**№1.** Кількість міст = 13. Розмір популяції = 101. Загальна довжина шляху = 602.

Шлях: [144,182],[143,191],[87,187],[87,138],[25,160],[48,90],[40,4],[63,72],[81,75],[102,83],   
[102,88],[118,80],[169,122]



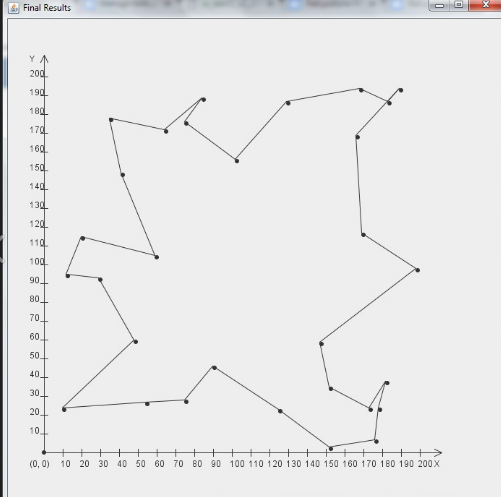
**№2.** Кількість міст = 30. Розмір популяції = 241. Загальна довжина шляху = 915.

Шлях:

[60,104],[42,148],[36,177],[65,171],[85,188],[76,175],[103,155],[130,186],[169,193],[184,186],

[190,193],[167,168],[170,116],[199,97],[148,58],[153,34],[174,23],[183,37],[179,23],[177,6],

[153,2],[126,22],[91,45],[76,27],[55,26],[11,23],[49,59],[30,92],[13,94],[21,114]



**№3.** Кількість міст = 49. Розмір популяції = 549. Загальна довжина шляху = 1712.

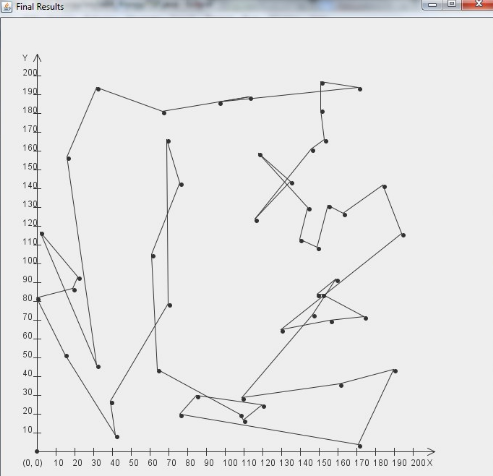
Шлях: [147,160],[154,165],[152,181],[152,196],[172,193],[98,185],[114,188],[68,180],[33,193],[17,156],

[33,45],[3,116],[23,92],[20,86],[1,81],[16,51],[43,8],[40,26],[71,78],[70,165],

[77,142],[62,104],[65,43],[109,19],[111,16],[121,24],[86,29],[77,19],[172,3],[191,43],

[162,35],[110,28],[148,72],[160,91],[150,83],[153,83],[175,71],[157,69],[131,64],[195,115],

[185,141],[164,126],[156,130],[150,108],[141,112],[145,129],[119,158],[136,143],[117,123]



**Висновок**

Під час виконання лабораторної роботи я ознайомився із поняттям задачі комівояжера, та реалізував програму для розв’язку задачі комівояжера за допомогою генетичного алгоритму.