Прізвище: Жук

Ім’я: Богдан

Група: КНС-13

Дата прийняття роботи

у системі Git: xx.05.2017

Дисципліна: Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні

Перевірив: Кривий Р.З.

**Звіт до лабораторної роботи № 5**

«ГА ДЛЯ ЗАДАЧІ КОМІВОЯЖЕРА (TSP)»

1. **Мета роботи**

Ознайомитися з основними теоретичними відомостями, вивчити еволюційні оператори схрещування та мутації, що використовуються при розв’язуванні задач комбінаторної оптимізації.

**КОРОТКІ ТЕОРЕТИЧНІ ВІДОМОСТІ**

Задача комівояжераполягає 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.

Існує маса різновидів узагальненої постановки задачі, зокрема геометрична задача комівояжера (коли матриця відстаней відображає відстані між точками на площині), трикутна задача комівояжера (коли на матриці вартостей виконується нерівність трикутника), симетрична та асиметрична задачі комівояжера.   
  
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. 

1. **Індивідуальне завдання**

**Проблема полягає в наступному.** У вас є безліч міст (представлені у вигляді точок на площині з X і Y координати). 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. Мова програмування довільна.

Розробити програму для вирішення задачі комівояжера (TSP).

**Варіант № 8 (4)** Пропорційний відбір.

1. **Програмна реалізація завдання**

**Код програми на Java:**

**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;

}

// кросовер популяції

**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);

}

// мутуція нової популяції

**for** (**int** i = elitismOffset; i < newPopulation.populationSize(); i++) {

*mutate*(newPopulation.getTour(i));

}

**return** newPopulation;

}

//Застосовується кросовера до набору батьків і створення потомство

**public** **static** Tour crossover(Tour parent1, Tour parent2) {

Tour child = **new** Tour();

**int** p1 = (**int**) (Math.*random*() \* parent1.tourSize());

**int** p2 = (**int**) (Math.*random*() \* parent1.tourSize());

**for**(**int** i = 0; i < child.tourSize(); i++)

{

**if**(i>=p1 && i<=p2) child.setCity(i, **null**);

**else** child.setCity(i, parent1.getCity(i));

}

**int** n = 0;

**for** ( **int** j = 0; j < parent2.tourSize(); j++)

{

**boolean** t = **false**;

**for** ( **int** k = 0; k < child.tourSize(); k++)

{

**if**(parent2.getCity(j) == child.getCity(k)) {

t = **true**;

**break**;

}

}

**if** (t== **false**){

child.setCity(p1+n, parent2.getCity(j));

n = n+1;

}

}

**return** child;

}

//мутація

**private** **static** **void** mutate(Tour tour) {

**int** tourPos1 = (**int**) (tour.tourSize() \* Math.*random*());

**int** tourPos2 = (**int**) (tour.tourSize() \* Math.*random*());

**if** (tourPos2 < tourPos1) {

**int** q = tourPos1;

tourPos1 = tourPos2;

tourPos2 = q;

}

**for**(**int** i = 0; i <= (tourPos2 - tourPos1 +1)/2 ; i++)

{

City q = tour.getCity(tourPos1+i);

tour.setCity(tourPos1+i, tour.getCity(tourPos2-i));

tour.setCity(tourPos2-i, q);

}

}

//Вибір кандидата для кросовера

**private** **static** Tour tournamentSelection(Population pop) {

Population truncation = **new** Population(***tournamentSize***, **false**);

**for** (**int** i = 0; i < ***tournamentSize***; i++) {

**int** randomId = (**int**) (Math.*random*() \* pop.populationSize());

truncation.saveTour(i, pop.getTour(randomId));

}

Tour fittest = truncation.getFittest();

**return** fittest;

}

}

**public** **class** TSP {

//кількість міст

**static** **int** *nsity* = Integer.*parseInt*(CartesianFrame.*tf1*.getText());

//популяція

**static** **int** *npop*=Integer.*parseInt*(CartesianFrame.*tf2*.getText());

**static** **int** [] *x*=**new** **int**[*nsity*];

**static** **int** [] *y*=**new** **int**[*nsity*];

**static** **int** [] *xline*= **new** **int**[*nsity*];

**static** **int** [] *yline*= **new** **int**[*nsity*];

**public** **static** **void** main(String[] args) {

//створення координат міст

System.***out***.println("Кординати міст:");

**for** (**int** i = 0; i < *nsity* ; i++)

{

**int** newx = (**int**)(Math.*random*()\*200);

*x*[i]=newx;

**int** newy = (**int**)(Math.*random*()\*200);

*y*[i]=newy;

System.***out***.print("["+*x*[i]+","+*y*[i]+"],");

**if**(i%10==9 ){

System.***out***.println();

}

}

// додавання міст

**for**(**int** i = 0; i < *x*.length; i++){

TourManager.*addCity*(**new** City(*x*[i],*y*[i]));

}

Population pop = **new** Population(*npop*, **true**);

System.***out***.println("\nРозмір популяції: "+*npop*);

Date currentTimeBefore = **new** Date();

**long** timeBefore = currentTimeBefore.getTime();

pop = GA.*evolvePopulation*(pop);

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

pop = GA.*evolvePopulation*(pop);

}

Date currentTimeAfter = **new** Date();

**long** timeAfter= currentTimeAfter.getTime();;

**long** time = timeAfter-timeBefore;

//вивід результатів

System.***out***.println("Довжина шляху: " + pop.getFittest().getDistance());

System.***out***.println("Час виконання: " + time + " мс");

System.***out***.println("Знайдений шлях:");

Tour t=**new** Tour();

t=pop.getFittest();

**for**(**int** i=0;i<*nsity*;i++){

*xline*[i]= t.getCity(i).x;

*yline*[i]= t.getCity(i).y;

}

**for** (**int** i = 0; i < *nsity* ; i++)

{

System.***out***.print("["+*xline*[i]+","+*yline*[i]+"],");

**if**(i%10==9 ){

System.***out***.println();

}

}

// відображення графіка

CartesianFrame frame = **new** CartesianFrame();

//nsity = Integer.parseInt(frame.tf.getText());

frame.showUI();;

}

}

**Результати роботи програми:**

Довжина шляху: 606

Час виконання: 196 мс

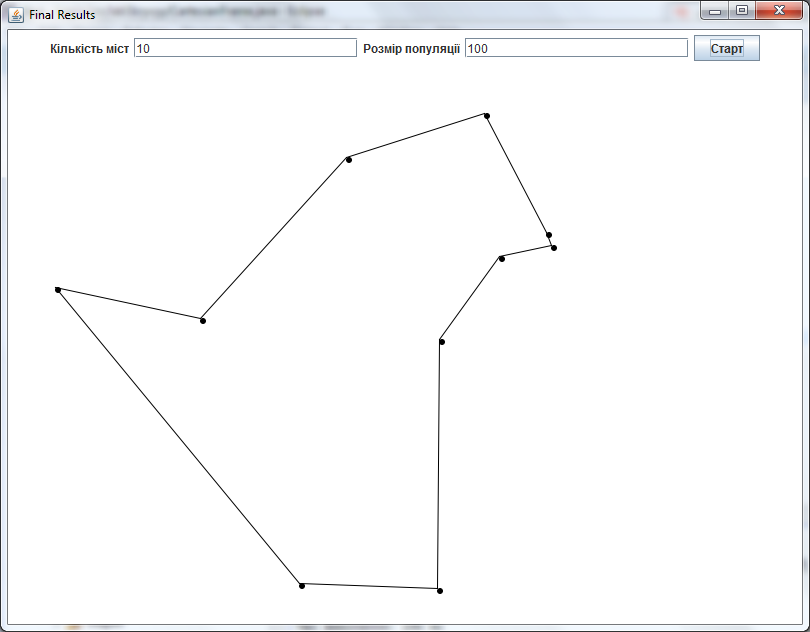


Рис. 1 Результати при запуску програми з даними для тесту 1

Довжина шляху: 683

Час виконання: 353 мс

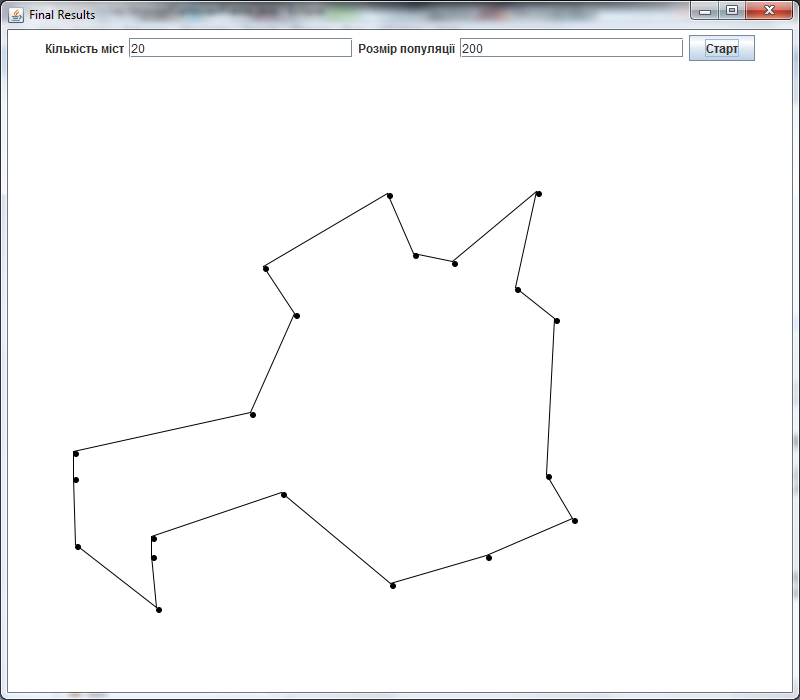


Рис. 2 Результати при запуску програми з даними для тесту 2

Довжина шляху: 1131

Час виконання: 327 мс

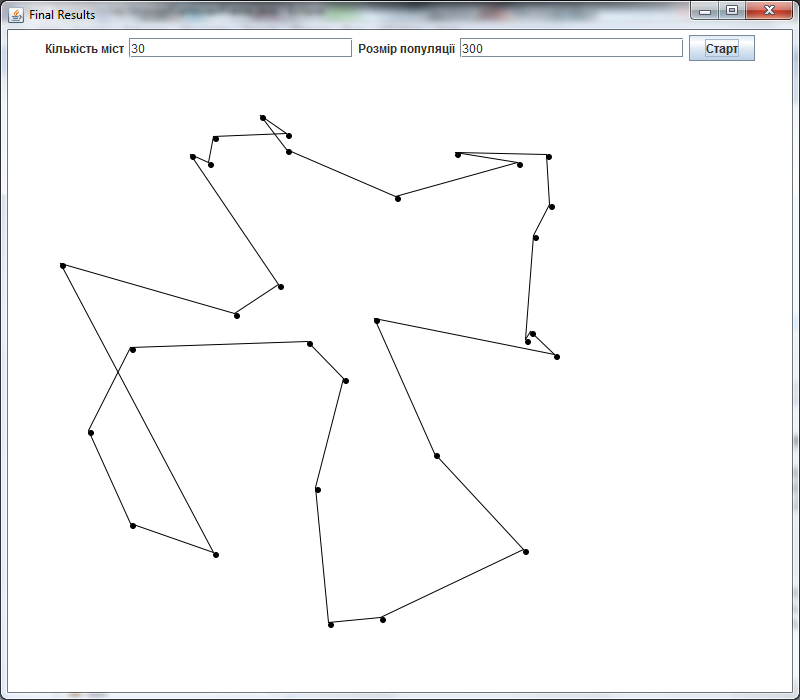


Рис. 3 Результати при запуску програми з даними для тесту 3

Довжина шляху: 1432

Час виконання: 644 мс

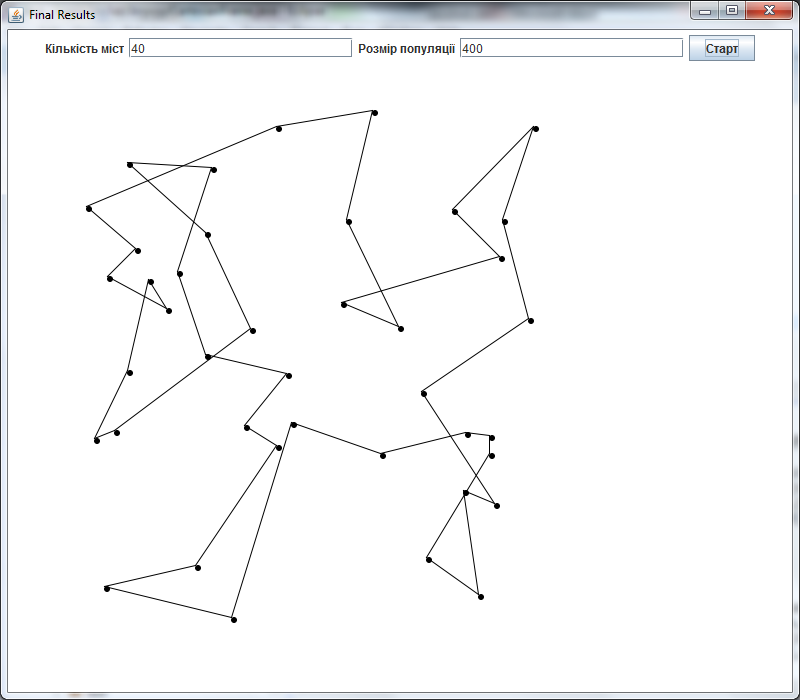


Рис. 4 Результати при запуску програми з даними для тесту 4

Довжина шляху: 1888

Час виконання: 1337 мс

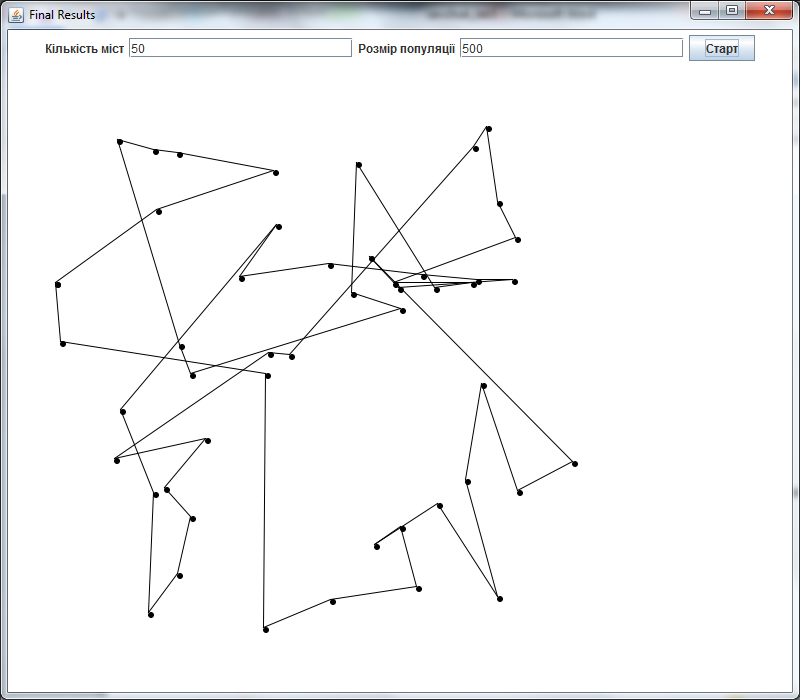


Рис. 5 Результати при запуску програми з даними для тесту 5

1. **Висновки**

У результаті виконання лабораторної роботи я розробила програму, яка реалізовує генетичний алгоритм для розв’язання задачі комівояжера.