|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| КНм-14 | 5 | ГА ДЛЯ ЗАДАЧІ КОМІВОЯЖЕРА (TSP) |  |  |
| Євтух Олеся | |
| Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні | | Викладач: | |
| Кривий Р.З. | |

1. **Мета роботи**

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

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 № 16 (4)** Пропорційний відбір.

1. **Програмна реалізація завдання**

**Код програми на Java:**

**TSP.class:**

**package** tsp;

**import** java.awt.Graphics2D;

**import** java.util.Date;

**import** javax.swing.SwingUtilities;

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

//Кількість міст

**static** **int** *nsity*=50;

//Популяція

**static** **int** *npop*=500;

**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]));

}

//Ініціалізація популяції

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

SwingUtilities.*invokeLater*(**new** Runnable() {

@Override

**public** **void** run() {

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

frame.showUI();;

}

});

}

}

**Class CartesianFrame:**

**package** tsp;

**import** java.awt.Graphics;

**import** java.awt.Graphics2D;

**import** java.awt.RenderingHints;

**import** javax.swing.JFrame;

**import** javax.swing.JPanel;

**class** CartesianFrame **extends** JFrame {

CartesianPanel panel;

**public** CartesianFrame() {

panel = **new** CartesianPanel();

add(panel);

}

**public** **void** showUI() {

setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

setTitle("Final Results");

setSize(700, 700);

setVisible(**true**);

}

}

**class** CartesianPanel **extends** JPanel {

// x-axis координати

**public** **static** **final** **int** ***X\_AXIS\_FIRST\_X\_COORD*** = 50;

**public** **static** **final** **int** ***X\_AXIS\_SECOND\_X\_COORD*** = 600;

**public** **static** **final** **int** ***X\_AXIS\_Y\_COORD*** = 600;

// y-axis координами

**public** **static** **final** **int** ***Y\_AXIS\_FIRST\_Y\_COORD*** = 50;

**public** **static** **final** **int** ***Y\_AXIS\_SECOND\_Y\_COORD*** = 600;

**public** **static** **final** **int** ***Y\_AXIS\_X\_COORD*** = 50;

**public** **static** **final** **int** ***FIRST\_LENGHT*** = 10;

**public** **static** **final** **int** ***SECOND\_LENGHT*** = 5;

**public** **static** **final** **int** ***ORIGIN\_COORDINATE\_LENGHT*** = 6;

**public** **static** **final** **int** ***AXIS\_STRING\_DISTANCE*** = 20;

// нумерація осей

**int** xCoordNumbers = 21;

**int** yCoordNumbers = 21;

**int** xLength = (***X\_AXIS\_SECOND\_X\_COORD*** - ***X\_AXIS\_FIRST\_X\_COORD***)/ xCoordNumbers;

**int** yLength = (***Y\_AXIS\_SECOND\_Y\_COORD*** - ***Y\_AXIS\_FIRST\_Y\_COORD***)/ yCoordNumbers;

**public** **void** paintComponent(Graphics g) {

**super**.paintComponent(g);

Graphics2D g2 = (Graphics2D) g;

g2.setRenderingHint(RenderingHints.***KEY\_ANTIALIASING***,

RenderingHints.***VALUE\_ANTIALIAS\_ON***);

// x-axis

g2.drawLine(***X\_AXIS\_FIRST\_X\_COORD***, ***X\_AXIS\_Y\_COORD***,

***X\_AXIS\_SECOND\_X\_COORD***, ***X\_AXIS\_Y\_COORD***);

// y-axis

g2.drawLine(***Y\_AXIS\_X\_COORD***, ***Y\_AXIS\_FIRST\_Y\_COORD***,

***Y\_AXIS\_X\_COORD***, ***Y\_AXIS\_SECOND\_Y\_COORD***);

// x-axis стрілка

g2.drawLine(***X\_AXIS\_SECOND\_X\_COORD*** - ***FIRST\_LENGHT***,

***X\_AXIS\_Y\_COORD*** - ***SECOND\_LENGHT***,

***X\_AXIS\_SECOND\_X\_COORD***, ***X\_AXIS\_Y\_COORD***);

g2.drawLine(***X\_AXIS\_SECOND\_X\_COORD*** - ***FIRST\_LENGHT***,

***X\_AXIS\_Y\_COORD*** + ***SECOND\_LENGHT***,

***X\_AXIS\_SECOND\_X\_COORD***, ***X\_AXIS\_Y\_COORD***);

// y-axis стрілка

g2.drawLine(***Y\_AXIS\_X\_COORD*** - ***SECOND\_LENGHT***,

***Y\_AXIS\_FIRST\_Y\_COORD*** + ***FIRST\_LENGHT***,

***Y\_AXIS\_X\_COORD***, ***Y\_AXIS\_FIRST\_Y\_COORD***);

g2.drawLine(***Y\_AXIS\_X\_COORD*** + ***SECOND\_LENGHT***,

***Y\_AXIS\_FIRST\_Y\_COORD*** + ***FIRST\_LENGHT***,

***Y\_AXIS\_X\_COORD***, ***Y\_AXIS\_FIRST\_Y\_COORD***);

// малює початок координат

g2.fillOval(

***X\_AXIS\_FIRST\_X\_COORD*** - (***ORIGIN\_COORDINATE\_LENGHT*** / 2),

***Y\_AXIS\_SECOND\_Y\_COORD*** - (***ORIGIN\_COORDINATE\_LENGHT*** / 2),

***ORIGIN\_COORDINATE\_LENGHT***, ***ORIGIN\_COORDINATE\_LENGHT***);

// малює надписи "Х" "У"

g2.drawString("X", ***X\_AXIS\_SECOND\_X\_COORD*** - ***AXIS\_STRING\_DISTANCE*** / 2,

***X\_AXIS\_Y\_COORD*** + ***AXIS\_STRING\_DISTANCE***);

g2.drawString("Y", ***Y\_AXIS\_X\_COORD*** - ***AXIS\_STRING\_DISTANCE***,

***Y\_AXIS\_FIRST\_Y\_COORD*** + ***AXIS\_STRING\_DISTANCE*** / 2);

g2.drawString("(0, 0)", ***X\_AXIS\_FIRST\_X\_COORD*** - ***AXIS\_STRING\_DISTANCE***,

***Y\_AXIS\_SECOND\_Y\_COORD*** + ***AXIS\_STRING\_DISTANCE***);

// нумерація осі х

**for**(**int** i = 1; i < xCoordNumbers; i++) {

g2.drawLine(***X\_AXIS\_FIRST\_X\_COORD***+ (i \* xLength),

***X\_AXIS\_Y\_COORD*** - ***SECOND\_LENGHT***,

***X\_AXIS\_FIRST\_X\_COORD*** + (i \* xLength),

***X\_AXIS\_Y\_COORD*** + ***SECOND\_LENGHT***);

g2.drawString(Integer.*toString*(i\*10),

***X\_AXIS\_FIRST\_X\_COORD*** + (i \* xLength) - 3,

***X\_AXIS\_Y\_COORD*** + ***AXIS\_STRING\_DISTANCE***);

}

//нумерація осі у

**for**(**int** i = 1; i < yCoordNumbers; i++) {

g2.drawLine(***Y\_AXIS\_X\_COORD*** - ***SECOND\_LENGHT***,

***Y\_AXIS\_SECOND\_Y\_COORD*** - (i \* yLength),

***Y\_AXIS\_X\_COORD*** + ***SECOND\_LENGHT***,

***Y\_AXIS\_SECOND\_Y\_COORD*** - (i \* yLength));

g2.drawString(Integer.*toString*(i\*10),

***Y\_AXIS\_X\_COORD*** - ***AXIS\_STRING\_DISTANCE***,

***Y\_AXIS\_SECOND\_Y\_COORD*** - (i \* yLength));

}

//відобрадження міст на графіку

**int** x0=***X\_AXIS\_FIRST\_X\_COORD*** - (***ORIGIN\_COORDINATE\_LENGHT*** / 2);

**int** y0=***Y\_AXIS\_SECOND\_Y\_COORD*** - (***ORIGIN\_COORDINATE\_LENGHT*** / 2);

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

**int** x=x0+(TSP.*x*[i]\*xLength)/10;

**int** y=y0-(TSP.*y*[i]\*yLength)/10;

g2.fillOval(x,y,***ORIGIN\_COORDINATE\_LENGHT***, ***ORIGIN\_COORDINATE\_LENGHT***);

}

//відображення лініями знайденого марштуру

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

**int** x1,x2,y1,y2;

**if**(i+1!=TSP.*nsity*){

x1=x0+(TSP.*xline*[i]\*xLength)/10;

x2=x0+(TSP.*xline*[i+1]\*xLength)/10;

y1=y0-(TSP.*yline*[i]\*yLength)/10;

y2=y0-(TSP.*yline*[i+1]\*yLength)/10;

}

**else**{

x1=x0+(TSP.*xline*[i]\*xLength)/10;

x2=x0+(TSP.*xline*[0]\*xLength)/10;

y1=y0-(TSP.*yline*[i]\*yLength)/10;

y2=y0-(TSP.*yline*[0]\*yLength)/10;

}

g2.drawLine(x1,y1, x2, y2);

}

}

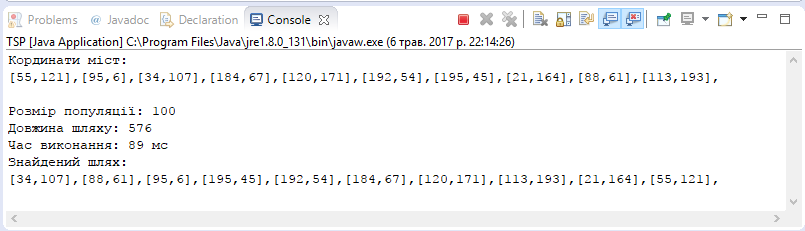
}

**Результати роботи програми:**

*Таблиця 1*

*Результатами роботи програми з різними параметрами*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Тест №** | 1 | 2 | 3 | 4 | 5 |
| **Кількість міст** | 10 | 20 | 30 | 40 | 50 |
| **Популяція** | 100 | 200 | 150 | 300 | 500 |
| **Час виконання, мc** | 89 | 135 | 162 | 278 | 396 |
| **Довжина шляху** | 576 | 754 | 1031 | 1413 | 1779 |



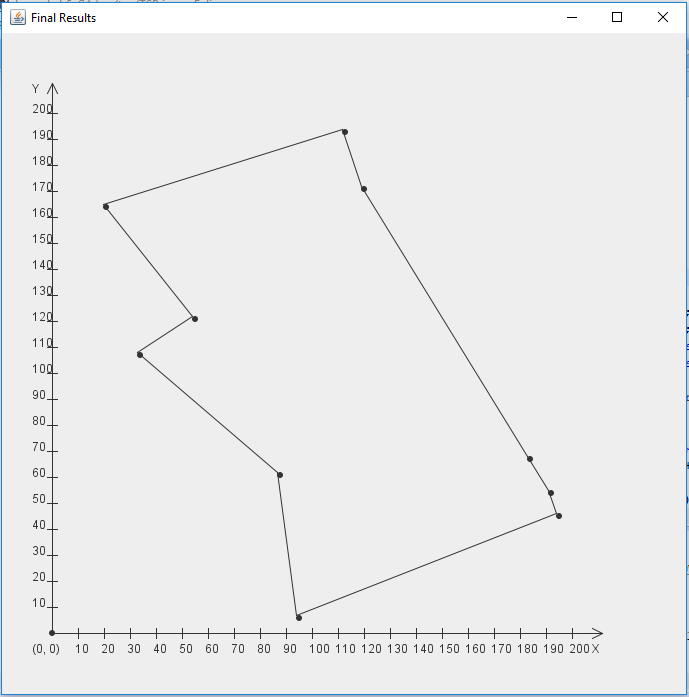
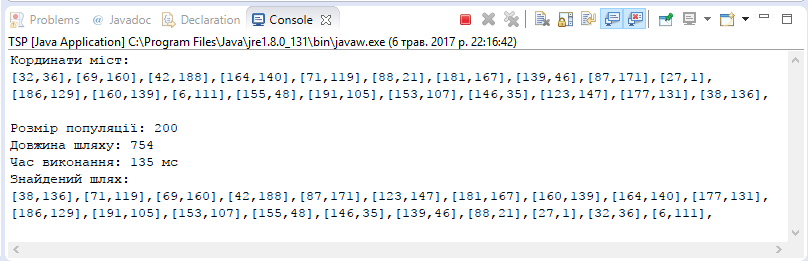


Рис. 1 Результати при запуску програми з даними для Тесту №1



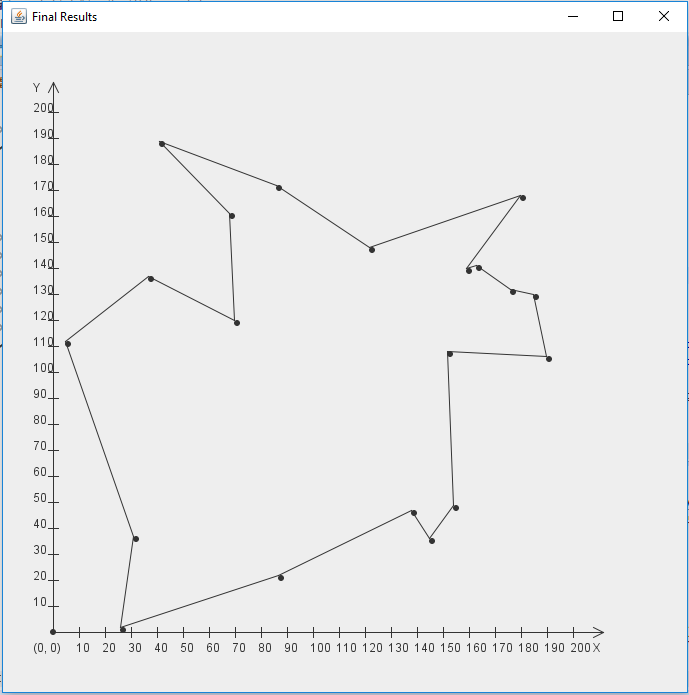


Рис. 2 Результати при запуску програми з даними для Тесту №2

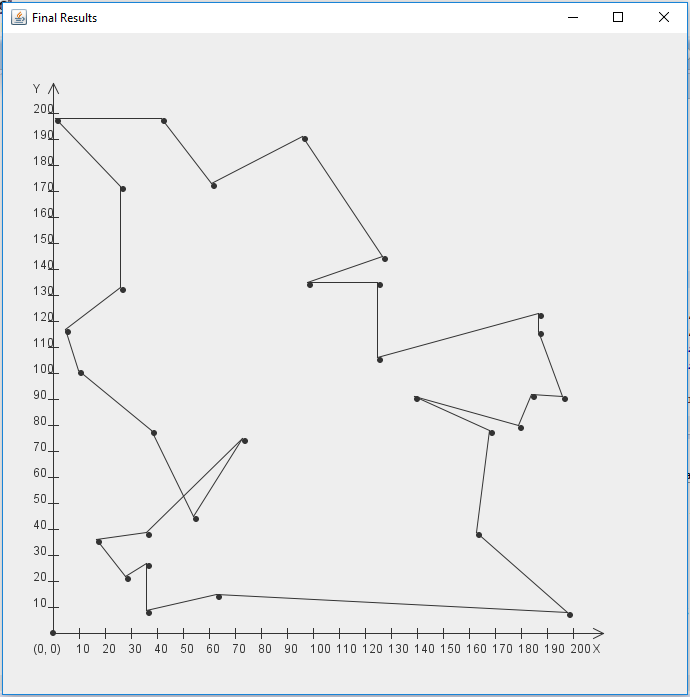
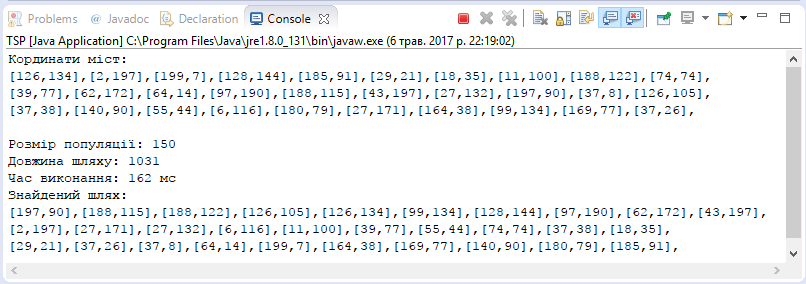
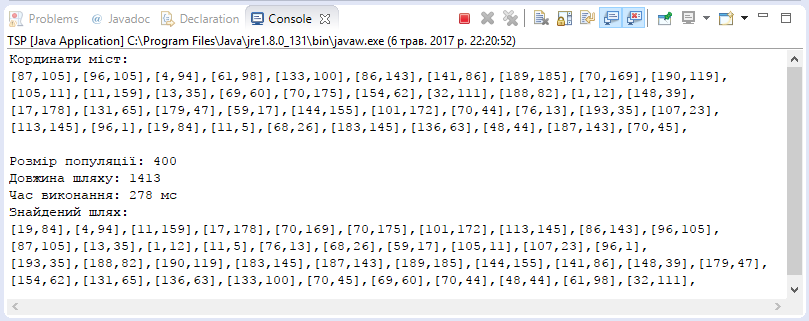


Рис. 3 Результати при запуску програми з даними для Тесту №3



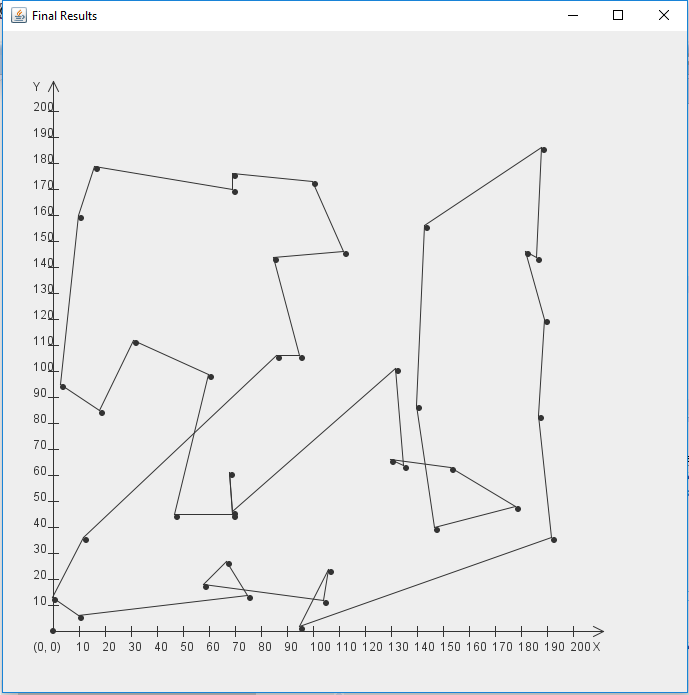
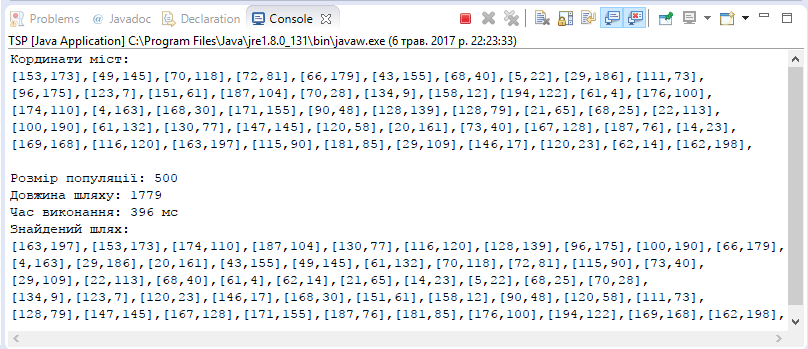


Рис. 4 Результати при запуску програми з даними для Тесту №4



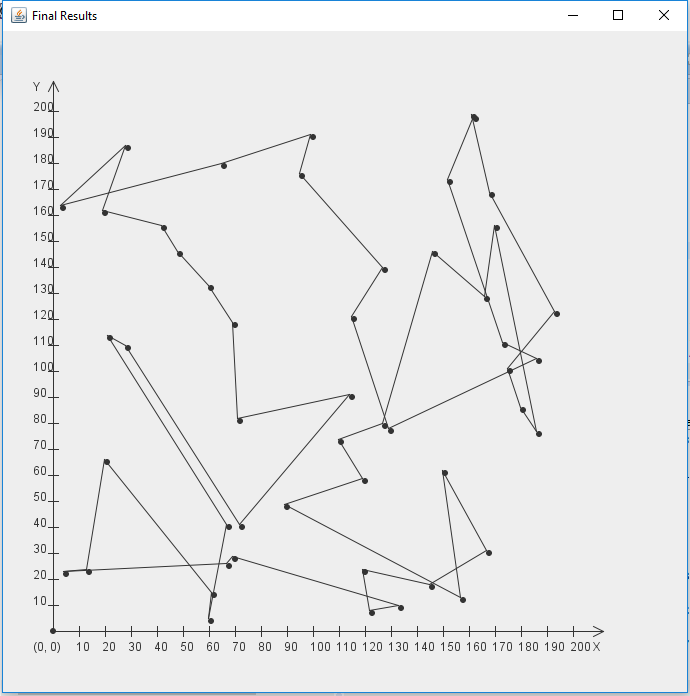


Рис. 5 Результати при запуску програми з даними для Тесту №5

1. **Висновки**

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