*дата :*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| КНC-13 | Лабораторна робота №5 | Рішення задачі комівояжера за допомогою генетичного алгоритму |  |  |
| Дідик Д.С. | |
| 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 6**

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**

namespace Tsp

{

class Population : List<Tour>

{

private Tour bestTour = null;

public Tour BestTour

{ set

{bestTour = value; }

get

{ return bestTour; }}

public void CreateRandomPopulation(int populationSize, Cities cityList, Random rand, int chanceToUseCloseCity)

{

int firstCity, lastCity, nextCity;

for (int tourCount = 0; tourCount < populationSize; tourCount++)

{

Tour tour = new Tour(cityList.Count);

firstCity = rand.Next(cityList.Count);

lastCity = firstCity;

for (int city = 0; city < cityList.Count - 1; city++)

{

do

{

if ((rand.Next(100) < chanceToUseCloseCity) && ( cityList[city].CloseCities.Count > 0 ))

{

nextCity = cityList[city].CloseCities[rand.Next(cityList[city].CloseCities.Count)];

}

else

{

nextCity = rand.Next(cityList.Count);

}

} while ((tour[nextCity].Connection2 != -1) || (nextCity == lastCity));

tour[lastCity].Connection2 = nextCity;

tour[nextCity].Connection1 = lastCity;

lastCity = nextCity;

}

tour[lastCity].Connection2 = firstCity;

tour[firstCity].Connection1 = lastCity;

tour.DetermineFitness(cityList);

Add(tour);

if ((bestTour == null) || (tour.Fitness < bestTour.Fitness))

{

BestTour = tour;

}}}}}

public void DrawTour(object sender, TspEventArgs e)

{

this.lastFitnessValue.Text = Math.Round(e.BestTour.Fitness, 2).ToString(CultureInfo.CurrentCulture);

this.lastIterationValue.Text = e.Generation.ToString(CultureInfo.CurrentCulture);

if (cityImage == null)

{

cityImage = new Bitmap(tourDiagram.Width, tourDiagram.Height);

cityGraphics = Graphics.FromImage(cityImage);

}

int lastCity = 0;

int nextCity = e.BestTour[0].Connection1;

cityGraphics.FillRectangle(Brushes.White, 0, 0, cityImage.Width, cityImage.Height);

foreach( City city in e.CityList )

{

cityGraphics.DrawEllipse(Pens.Black, city.Location.X - 2, city.Location.Y - 2, 5, 5);

cityGraphics.DrawLine(Pens.Black, cityList[lastCity].Location, cityList[nextCity].Location);

if (lastCity != e.BestTour[nextCity].Connection1)

{

lastCity = nextCity;

nextCity = e.BestTour[nextCity].Connection1;

}

else

{

lastCity = nextCity;

nextCity = e.BestTour[nextCity].Connection2;

}

}

this.tourDiagram.Image = cityImage;

if (e.Complete)

{

StartButton.Text = "Begin";

StatusLabel.Text = "Open a City List or click the map to place cities.";

StatusLabel.ForeColor = Color.Black;

}

}

**Результат виконання**

Для однакової кількості міст = 31, проводжу серію експериментів, щоб відобразити зміну шляху для різних значень популяції та кількості ітерацій

**№1.** Кількість міст = 31. Розмір популяції =500. Кількість ітерацій =10000 . Загальна довжина шляху = 2276,38

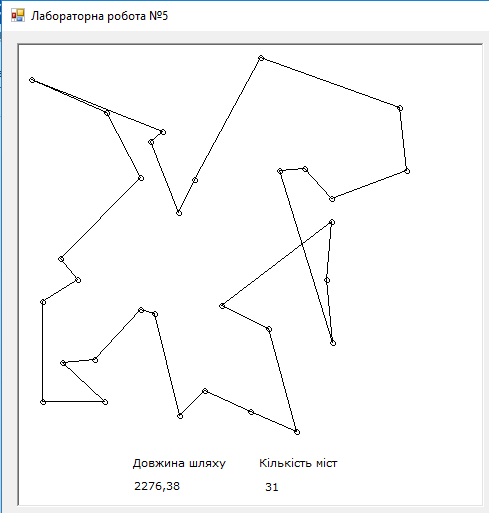


Рис.1. Результат виконання програми №1.

**№2.** Кількість міст = 31. Розмір популяції =1000. Кількість ітерацій =50000 . Загальна довжина шляху = 2052,62

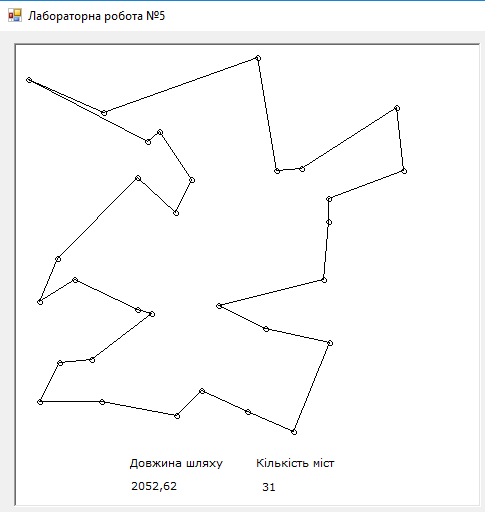


Рис.2. Результат виконання програми №2.

**№3.** Кількість міст = 31. Розмір популяції = 1500. Кількість ітерацій = 80000. Загальна довжина шляху = 1988,47

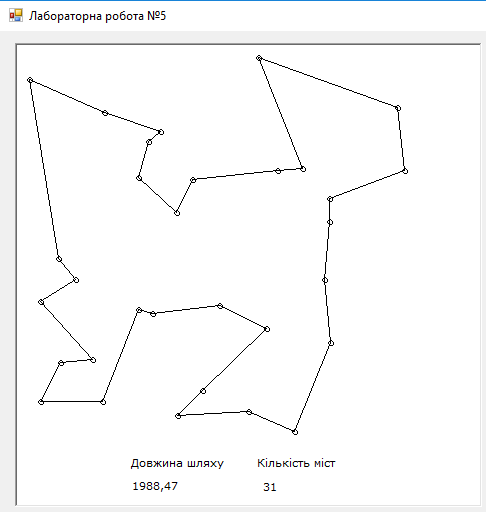


Рис.3. Результат виконання програми №3.

**№4.** Кількість міст = 31. Розмір популяції = 2000. Кількість ітерацій = 200000 . Загальна довжина шляху = 1980,52

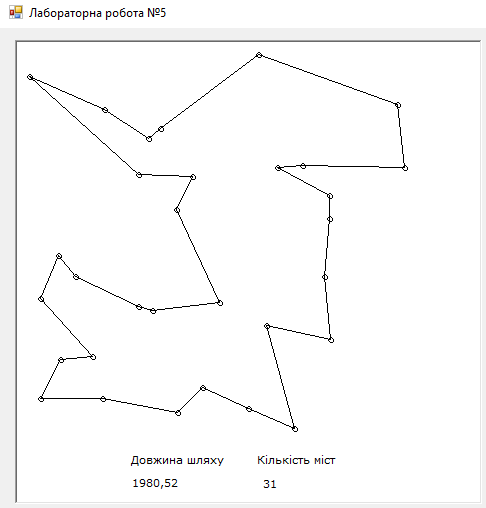


Рис.4. Результат виконання програми №4.

**Висновок**

Під час виконання лабораторної роботи я ознайомився із поняттям задачі комівояжера, та реалізував програму для розв’язку задачі комівояжера за допомогою генетичного алгоритму. З рисунків можна побачити, що зі збільшенням кількості ітерацій та розмірів популяції довжина шляху зменшується, отже чим більші дані показники, тим точніший результат буде отримано.