|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **НУЛП, ІКНІ, САП** | | Тема | оцінка | підпис |
| КНС-13 | 5(номер лаб.) | **Задача комівояжера** |  |  |
| Писаренко В.Б. | |
|  | |
| **Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні** | | Викладач: | |
| асистент каф. САПР  Кривий Р.З. | |

**1.МЕТА**

Ознайомитися з основними теоретичними відомостями за темою роботи. Вивчити роботу функції ga пакету Matlab. Розробити за допомогою пакету Matlab програмне забезпечення, що реалізує реалізовує задачу комівояжера.

**2. ТЕОРЕТИЧНІ ВІДОМОСТІ**

**Зада́ча комівояже́ра** ([комівояжер](https://uk.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D1%96%D0%B2%D0%BE%D1%8F%D0%B6%D0%B5%D1%80" \o "Комівояжер) — бродячий торговець; [англ.](https://uk.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D1%96%D0%B9%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Travelling Salesman Problem*, TSP; [нім.](https://uk.wikipedia.org/wiki/%D0%9D%D1%96%D0%BC%D0%B5%D1%86%D1%8C%D0%BA%D0%B0_%D0%BC%D0%BE%D0%B2%D0%B0) *Problem des Handlungsreisenden*) полягає у знаходженні найвигіднішого [маршруту](https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82), 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](https://uk.wikipedia.org/wiki/%D0%93%D0%B0%D0%BC%D1%96%D0%BB%D1%8C%D1%82%D0%BE%D0%BD%D1%96%D0%B2_%D1%86%D0%B8%D0%BA%D0%BB).

Існує маса різновидів узагальненої постановки задачі, зокрема геометрична задача комівояжера (коли матриця відстаней відображає [відстані](https://uk.wikipedia.org/wiki/%D0%92%D1%96%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D1%8C) між точками на площині), трикутна задача комівояжера (коли на матриці вартостей виконується [нерівність трикутника](https://uk.wikipedia.org/wiki/%D0%9D%D0%B5%D1%80%D1%96%D0%B2%D0%BD%D1%96%D1%81%D1%82%D1%8C_%D1%82%D1%80%D0%B8%D0%BA%D1%83%D1%82%D0%BD%D0%B8%D0%BA%D0%B0)), симетрична та асиметрична задачі комівояжера.

Прості методи розв'язання задачі комівояжера: повний лексичний перебір, [жадібні алгоритми](https://uk.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D1%96%D0%B1%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) ([метод найближчого сусіда](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BD%D0%B0%D0%B9%D0%B1%D0%BB%D0%B8%D0%B6%D1%87%D0%BE%D0%B3%D0%BE_%D1%81%D1%83%D1%81%D1%96%D0%B4%D0%B0" \o "Метод найближчого сусіда)), метод включення найближчого міста, метод найдешевшого включення, метод [мінімального кістяка дерева](https://uk.wikipedia.org/wiki/%D0%9C%D1%96%D0%BD%D1%96%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%B5_%D0%BA%D1%96%D1%81%D1%82%D1%8F%D0%BA%D0%BE%D0%B2%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE). На практиці застосовують різні модифікації ефективніших методів: [метод гілок і меж](https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B3%D1%96%D0%BB%D0%BE%D0%BA_%D1%96_%D0%BC%D0%B5%D0%B6) і [метод генетичних алгоритмів](https://uk.wikipedia.org/wiki/%D0%93%D0%B5%D0%BD%D0%B5%D1%82%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC), а так само [алгоритм мурашиної колонії](https://uk.wikipedia.org/wiki/%D0%9C%D1%83%D1%80%D0%B0%D1%88%D0%B8%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC).

Всі ефективні (такі, що скорочують повний перебір) методи розв'язання задачі комівояжера — [евристичні](https://uk.wikipedia.org/wiki/%D0%95%D0%B2%D1%80%D0%B8%D1%81%D1%82%D0%B8%D1%87%D0%BD%D0%B8%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC). 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.

Задача комівояжера — [NP-повна](https://uk.wikipedia.org/wiki/NP-%D0%BF%D0%BE%D0%B2%D0%BD%D0%B0_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87%D0%B0). Часто на ній проводять випробування нових підходів до евристичного скорочення повного перебору.

**3. ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

**Варіант №4**

**Пропорційний відбір**  
  
Код програми на мові C# :

using System;

using System.Collections.Generic;

using System.Windows.Forms;

using System.Text;

using System.Data;

using System.Drawing;

namespace Tsp

{

class Tsp

{

public delegate void NewBestTourEventHandler(Object sender, TspEventArgs e);

public event NewBestTourEventHandler foundNewBestTour;

Random rand;

Cities cityList;

Population population;

private bool halt = false;

public bool Halt

{

get

{

return halt;

}

set

{

halt = value;

}

}

public Tsp()

{

}

public void Begin(int populationSize, int maxGenerations, int groupSize, int mutation, int seed, int chanceToUseCloseCity, Cities cityList)

{

rand = new Random(seed);

this.cityList = cityList;

population = new Population();

population.CreateRandomPopulation(populationSize, cityList, rand, chanceToUseCloseCity);

displayTour(population.BestTour, 0, false);

bool foundNewBestTour = false;

int generation;

for (generation = 0; generation < maxGenerations; generation++)

{

if (Halt)

{

break;

}

foundNewBestTour = makeChildren(groupSize, mutation);

if (foundNewBestTour)

{

displayTour(population.BestTour, generation, false);

}

}

displayTour(population.BestTour, generation, true);

}

bool makeChildren(int groupSize, int mutation)

{

int[] tourGroup = new int[groupSize];

int tourCount, i, topTour, childPosition, tempTour;

for (tourCount = 0; tourCount < groupSize; tourCount++)

{

tourGroup[tourCount] = rand.Next(population.Count);

}

for (tourCount = 0; tourCount < groupSize - 1; tourCount++)

{

topTour = tourCount;

for (i = topTour + 1; i < groupSize; i++)

{

if (population[tourGroup[i]].Fitness < population[tourGroup[topTour]].Fitness)

{

topTour = i;

}

}

if (topTour != tourCount)

{

tempTour = tourGroup[tourCount];

tourGroup[tourCount] = tourGroup[topTour];

tourGroup[topTour] = tempTour;

}

}

bool foundNewBestTour = false;

childPosition = tourGroup[groupSize - 1];

population[childPosition] = Tour.Crossover(population[tourGroup[0]], population[tourGroup[1]], cityList, rand);

if (rand.Next(100) < mutation)

{

population[childPosition].Mutate(rand);

}

population[childPosition].DetermineFitness(cityList);

if (population[childPosition].Fitness < population.BestTour.Fitness)

{

population.BestTour = population[childPosition];

foundNewBestTour = true;

}

childPosition = tourGroup[groupSize - 2];

population[childPosition] = Tour.Crossover(population[tourGroup[1]], population[tourGroup[0]], cityList, rand);

if (rand.Next(100) < mutation)

{

population[childPosition].Mutate(rand);

}

population[childPosition].DetermineFitness(cityList);

if (population[childPosition].Fitness < population.BestTour.Fitness)

{

population.BestTour = population[childPosition];

foundNewBestTour = true;

}

return foundNewBestTour;

}

void displayTour(Tour bestTour, int generationNumber, bool complete)

{

if (foundNewBestTour != null)

{

this.foundNewBestTour(this, new TspEventArgs(cityList, bestTour, generationNumber, complete));

}

}

}

}

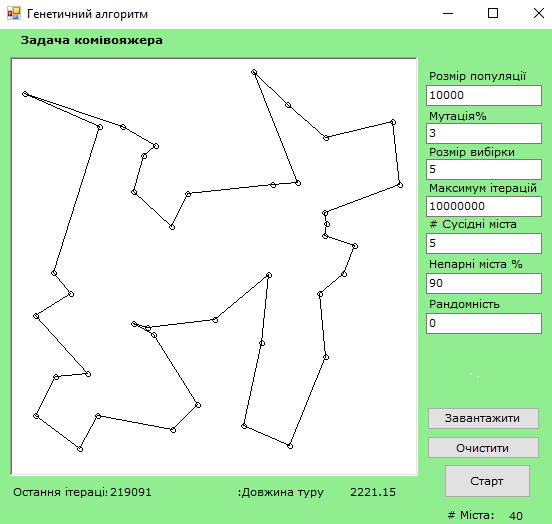


Рис 1 : Результат виконання програми

**4.ВИСНОВОК.**

У результаті проведення лабораторної роботи я розробив за допомогою пакету Visual Studio програмне забезпечення, що реалізує реалізовує алгоритм комівояжера.