Прізвище: Тищенко

Ім’я: Данило

Група: КНМ-14

Дата прийняття роботи

у системі Git: ХХ.04.2017

Дисципліна: Методи нечіткої логіки та еволюційні алгоритми при автоматизованому проектуванні

Перевірив: Кривий Р.З.

**Звіт до лабораторної роботи № 5**

**«Рішення задачі комівояжера за допомогою генетичного алгоритму»**

**МЕТА РОБОТИ**

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.

**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. 

**ІНДИВІДУАЛЬНЕ ЗАВДАННЯ**

За допомогою засобів мови програмування Python розробити програмне забезпечення для розв’язку задачі комівожера.

**Фрагмент коду програми:**

**class TSPSolution(OrganismClass):**

**genome = genome**

**mutateOneOnly = mutateOneOnly**

**crossoverRate = crossoverRate**

**numMutants = 0.5**

**def fitness(self):**

**"""**

**return distance**

**"""**

**distance = 0.0**

**sortedCities = self.getCitiesInOrder()**

**for i in range(cityCount - 1):**

**distance += sortedCities[i] - sortedCities[i+1]**

**distance += sortedCities[0] - sortedCities[-1]**

**return distance**

**def getCitiesInOrder(self):**

**"""**

**return cities list**

**"""**

**sorter = [(self[name], cityDict[name]) for name in cityNames]**

**sorter.sort()**

**sortedCities = [tup[1] for tup in sorter]**

**return sortedCities**

**def normalise(self):**

**"""**

**modifies genes**

**"""**

**genes = self.genes**

**for i in range(2):**

**sorter = [(genes[name][i], name) for name in cityNames]**

**sorter.sort()**

**sortedGenes = [tup[1] for tup in sorter]**

**class TSPSolutionPopulation(Population):**

**initPopulation = popInitSize**

**species = TSPSolution**

**childCull = popChildCull**

**childCount = popChildCount**

**incest = popIncest**

**mutants = popNumMutants**

**numNewOrganisms = popNumRandomOrganisms**

**mutateAfterMating = mutateAfterMating**

**Отримані результати:**

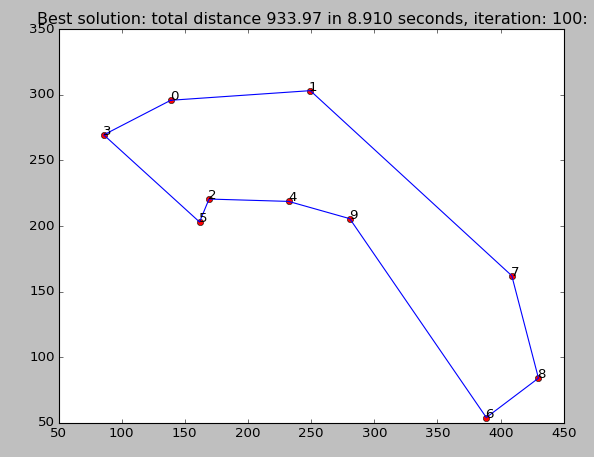


Рис 1. Розв`язок задачі при наявності десяти міст

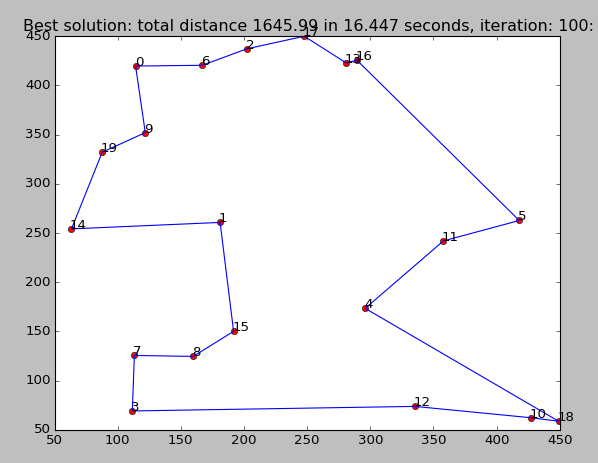


Рис 2. Розв`язок задачі при наявності двадцяти міст

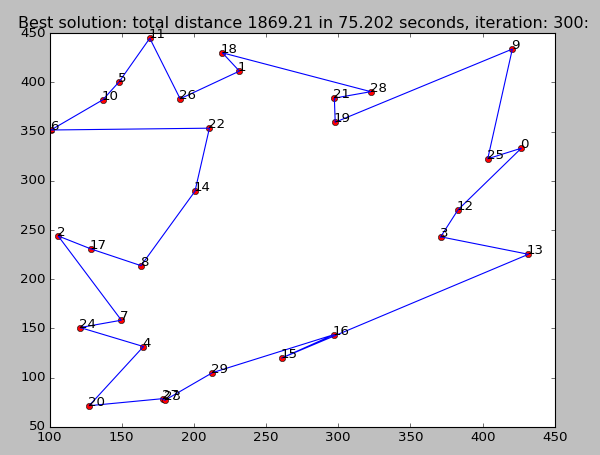


Рис 3. Розв`язок задачі при наявності тридцяти міст.

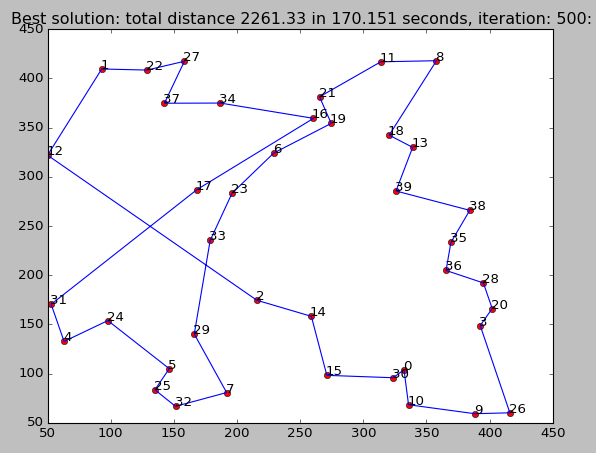


Рис 4. Розв`язок задачі при наявності сорока міст.

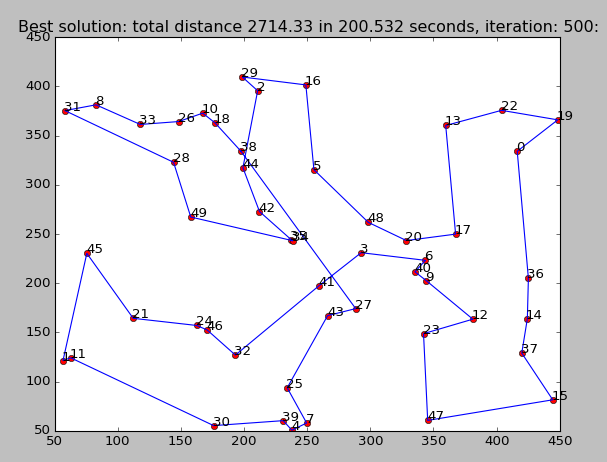


Рис 5. Розв`язок задачі при наявності п’ятдесяти міст.

**Висновок:** виконуючи дану лабораторну роботу я ознайомився із теоретичними відомостями, а також, використовуючи засоби мови програмування Python. Розробив програмне забезпечення для розв’язку задачі комівояжера за допомогою генетичного алгоритму.

Для великих обсягів даних наприклад, як для задачі комівояжера, коли є кілька десятків міст, алгоритм стає доволі повільним, та за малої кількості поколінь дає не найоптимальніший результат. В даній роботі результат вимагає покращення уже для 40-50 міст, однак час проведений на обчислення шляху уже є доволі великим, і росте не лінійно в залежності від кількості міст: ~9 секунд для 10 міст і ~200 для 50, при тому що результат не є оптимальними.