**Міністерство освіти і науки України**

**Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформатики та програмної інженерії**

**Звіт**

з лабораторної роботи № 4 з дисципліни

«Проектування алгоритмів»

„**Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.1**”





Київ 2022

Зміст

[**1**](#_heading=h.30j0zll) **Мета лабораторної роботи 3**

[**2**](#_heading=h.1fob9te) **Завдання 4**

[**3**](#_heading=h.3znysh7) **Виконання 10**

[3.1](#_heading=h.2et92p0) Програмна реалізація алгоритму 10

[*3.1.1*](#_heading=h.tyjcwt) *Вихідний код 10*

[*3.1.2*](#_heading=h.3dy6vkm) *Приклади роботи 10*

[3.2](#_heading=h.1t3h5sf) Тестування алгоритму 11

[*3.2.1*](#_heading=h.4d34og8) *Значення цільової функції зі збільшенням кількості ітерацій 11*

[*3.2.2*](#_heading=h.2s8eyo1) *Графіки залежності розв'язку від числа ітерацій 11*

[**Висновок 12**](#_heading=h.17dp8vu)

[**Критерії оцінювання 13**](#_heading=h.3rdcrjn)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи формалізації метаеврестичних алгоритмів і вирішення типових задач з їхньою допомогою.

# Завдання

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

| **№** | **Задача і алгоритм** |
| --- | --- |
| 1 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 2 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 3 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 4 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 5 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 6 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 7 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 8 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 9 | Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники). |
| 10 | Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 11 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45, починають маршрут в різних випадкових вершинах). |
| 12 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники). |
| 13 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 14 | Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 4, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 15 | Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники). |
| 16 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 17 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 2, β = 4, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах). |
| 18 | Задача розфарбовування графу (300 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники). |
| 19 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення. |
| 20 | Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм (α = 3, β = 2, ρ = 0,7, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 21 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники). |
| 22 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 23 | Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм (α = 3, β = 2, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах М = 45 (15 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах). |
| 24 | Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із них 10 розвідники). |
| 25 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 26 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 27 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 28 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 29 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |
| 30 | Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники). |
| 31 | Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 32 | Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 30, починають маршрут в різних випадкових вершинах). |
| 33 | Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники). |
| 34 | Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення. |
| 35 | Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах М = 35, починають маршрут в різних випадкових вершинах). |

# Виконання

## Програмна реалізація алгоритму

### Опис алгоритму

Перед початком роботи основного циклу алгоритму створюється матриця суміжності графа distanceMap (відстані між вершинами), в якому з кожної вершини можна потрапити в кожну іншу. З її використанням з допомогою жадібного алгоритму можна знайти Lmin. Також створюється список для збереження даних про кількість феромону на кожному з ребер графа pheromoneMap, у нього записується початкова кількість феромону. Далі створюється задана кількість об’єктів мурах Ant(), кожна з яких з’являється на випадковій вершині графа.

На початку основного циклу оновлюються значення в таблиці probabilityMap імовірностей переходу з однієї вершини в іншу.

Наступний крок циклу – мурахи одна за одною рухаються через усі вершини (метод move() у класі Ant: видаляється по одному рядку з таблиці ймовірностей та вершини по одній зі списку доступних переходять у список відвіданих, поки не будуть відвідані всі вершини). Останній шлях мурахи та його довжина зберігається.

Далі відбувається випарування феромону (множення таблиці з феромонами на константу)

Потім програма зберігає дані про найкращий шлях на поточній ітерації та знаходить 15-ту найменшу довжину шляху (за заданою кількістю елітних мурах).

Далі кожна мураха залишає феромон на своєму шляху. Якщо цей шлях є одним із 15 найкоротших, на цій ітерації ця мураха є елітною та залишає вдвічі більше феромону.

В кінці ітерації у список середніх та найкращих шляхів заноситься нове значення.

Після проходження циклу програма створює графік на основі отриманих значень довжин шляху.

### Вихідний код

travelingSalesman.py

import Ant

import time

import func

import plotting

import random

def main():

startTime = time.time()

#main parameters:

alpha = 3 #weight of the pheromone

beta = 2 #weight of vertex visibility

evap = 0.6 #how much to evaporate

antCount = 45 #all ants

eliteAntCount = 15

citiesCount = ''

while not citiesCount.isnumeric():

citiesCount = input('Enter the number of cities: ')

citiesCount = int(citiesCount)

iterations = ''

while not iterations.isnumeric():

iterations = input('Enter the number of iterations: ')

iterations = int(iterations)

initialPheromone = ''

while not initialPheromone.replace('.','').isnumeric():

initialPheromone = input('Enter the initial pheromone value: ')

initialPheromone = float(initialPheromone)

distanceMap = func.createDistanceMap(citiesCount)

Lmin = func.findLmin(distanceMap)

pheromoneMap = [[initialPheromone for j in range(citiesCount)] for i in range(citiesCount)]

probabilityMap = []

ants = []

for j in range(antCount):

ants.append(Ant.Ant(citiesCount))

avgPathLengths = []

bestPathLengths = []

bestLength = citiesCount\*60

bestPath = []

for i in range(iterations):

print(f'-------------------------')

print(f'Starting iteration {i}...')

print(f'-------------------------')

print('Updating probabilities...')

probabilityMap = func.updateProbabilityMap(distanceMap, pheromoneMap, alpha, beta)

print(f'Average probability weight of a path between cities: {sum(sum(probabilityMap[j]) for j in range(citiesCount))/(citiesCount\*\*2)}')

print('Moving ants...')

for j in range(len(ants)):

ants[j].move(probabilityMap, distanceMap)

print(f'Evaporating pheromones...')

for j in range(citiesCount):

for k in range(citiesCount):

pheromoneMap[j][k] = pheromoneMap[j][k]\*(1-evap)

print('Gathering data...')

currLengths = []

currBestLength = citiesCount\*60

for j in range(len(ants)):

currLengths.append(ants[j].lastPathLength)

if ants[j].lastPathLength < currBestLength:

currBestLength = ants[j].lastPathLength

currBestPathInd = j

print(f'Adding new pheromones...')

top15minLength = sorted(currLengths)[15]

for ant in ants:

ant.addPheromone(Lmin, pheromoneMap, top15minLength)

avgPathLengths.append(sum(currLengths)/len(currLengths))

bestPathLengths.append(currBestLength)

if currBestLength < bestLength:

bestLength = currBestLength

bestPath = ants[currBestPathInd].lastPathLength

print(f'Shortest length found yet: {bestLength}')

endTime = time.time()

timeTaken = round((endTime - startTime)\*100)/100

print(f'Time taken: {timeTaken} seconds ({timeTaken/iterations} per iteration)')

print('Showing results... (close the plot window to continue)')

plotting.showResults(iterations, avgPathLengths, bestPathLengths, Lmin)

if \_\_name\_\_ == "\_\_main\_\_":

main()

func.py

import random

def createDistanceMap(size):

distanceMap = [[random.randint(1, 60) for j in range(i, -1, -1)] for i in range(size-1, -1, -1)] # half

for i in range(size):

temp = distanceMap[i]

distanceMap[i] = [0 for j in range(0, i)]

distanceMap[i].extend(temp)

for i in range(size):

for j in range(i):

distanceMap[i][j] = distanceMap[j][i]

for i in range(size):

distanceMap[i][i] = 0

return distanceMap

def findLmin(distanceMap):

size = len(distanceMap)

distance = 0

currVertex = 0

verticesToVisit = list(range(1,size))

for i in range(size-1):

nextVertexDistance = 61

for j in range(len(verticesToVisit)):

if distanceMap[currVertex][verticesToVisit[j]] < nextVertexDistance:

nextVertex = verticesToVisit[j]

nextVertexDistance = distanceMap[currVertex][verticesToVisit[j]]

nextVertexInd = j

distance += nextVertexDistance

currVertex = verticesToVisit.pop(nextVertexInd)

distance += distanceMap[currVertex][0]

return distance

def updateProbabilityMap(distanceMap, pheromoneMap, alpha, beta):

probabilityMap = []

size = len(distanceMap)

for i in range(size):

probabilityMap.append([])

for j in range(i):

probabilityMap[i].append( (pheromoneMap[i][j]\*\*alpha) \* ((1/distanceMap[i][j])\*\*beta) )

probabilityMap[i].append(0)

for j in range(i+1, size):

probabilityMap[i].append( (pheromoneMap[i][j]\*\*alpha) \* ((1/distanceMap[i][j])\*\*beta) )

return probabilityMap

def displayArray(array, name): #display the values of an array -- debug function

print('--------------------')

print(name)

print('--------------------')

size = len(array)

for i in range(size):

for j in range(size):

print(format(array[i][j], '.6f'), end = '\t')

print()

print('--------------------')

Ant.py

import random

import func

class Ant():

def \_\_init\_\_(self, maxPos):

self.position = random.randrange(0, maxPos)

def move(self, probabilityMap, distanceMap):

"""Ant makes a circle around all the cities"""

size = len(distanceMap)

availableCities = list(range(size))

probabilities = [x[:] for x in probabilityMap]

availableCities.pop(self.position)

probabilities.pop(self.position)

self.lastPath = [self.position]

self.lastPathLength = 0

currCity = self.position

for i in range(size-1):

weights = [probabilities[j][currCity] for j in range(size-1-i)]

indChoices = range(size-1-i)

nextCityInd = random.choices(indChoices, weights)[0]

nextCity = availableCities.pop(nextCityInd)

probabilities.pop(nextCityInd)

self.lastPath.append(nextCity)

self.lastPathLength += distanceMap[currCity][nextCity]

currCity = nextCity

self.lastPath.append(self.position)

self.lastPathLength + distanceMap[currCity][self.position]

def addPheromone(self, Lmin, pheromoneMap, top15minLength):

for i in range(len(pheromoneMap)):

start = self.lastPath[i]

end = self.lastPath[i+1]

toAdd = Lmin/self.lastPathLength

if self.lastPathLength <= top15minLength:

toAdd \*= 2

pheromoneMap[start][end] += toAdd

pheromoneMap[end][start] += toAdd

plotting.py

import matplotlib.pyplot as plot

def showResults(iterations, avgPathLengths, bestPathLengths, Lmin):

"""show results of all iterations"""

X = list(range(iterations))

LminConst = [Lmin for i in range(iterations)]

plot.plot(X, avgPathLengths, color = 'r', label = 'Avg')

plot.plot(X, bestPathLengths, color = 'b', label = 'Min')

plot.plot(X, LminConst, color = 'g', label = 'Greedy')

plot.xlabel('Iterations')

plot.ylabel('Path length')

plot.title(f'results of {iterations} iterations')

plot.legend()

plot.show()

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

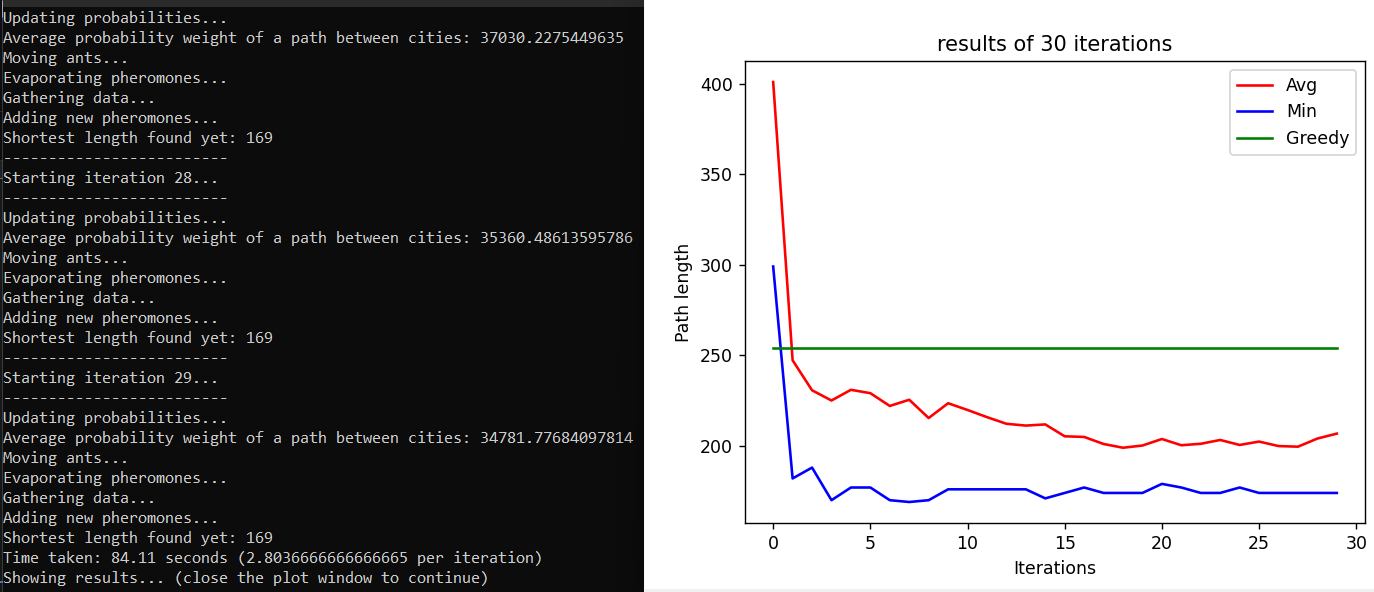


Рисунок 3.1 – Результати роботи програми (40 вершин, 30 ітерацій, початкова кількість феромону – 1)

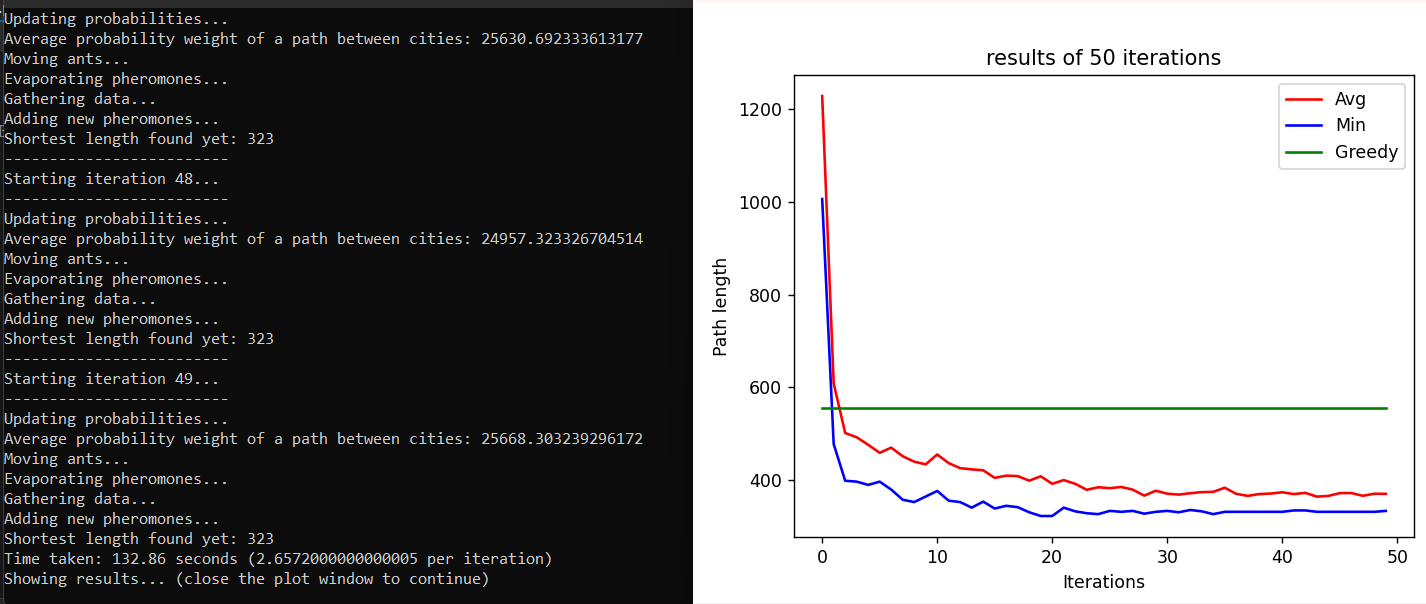


Рисунок 3.2 – Результати роботи програми (300 вершин графа, 50 ітерацій, початкова кількість феромону – 5)

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

У таблиці 3.1 наведено значення цільової функції (мінімальна знайдена до цієї ітерації довжина шляху) зі збільшенням кількості ітерацій (початкова кількість феромону = 5).

| Ітерації | Мінімальна знайдена довжина |
| --- | --- |
| 1 | 956 |
| 2 | 502 |
| 3 | 426 |
| 4 | 419 |
| 5 | 414 |
| 10 | 368 |
| 20 | 356 |
| 50 | 352 |
| 100 | 352 |
| 200 | 352 |

### 

### Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку, показаного в пункті 3.2.1.

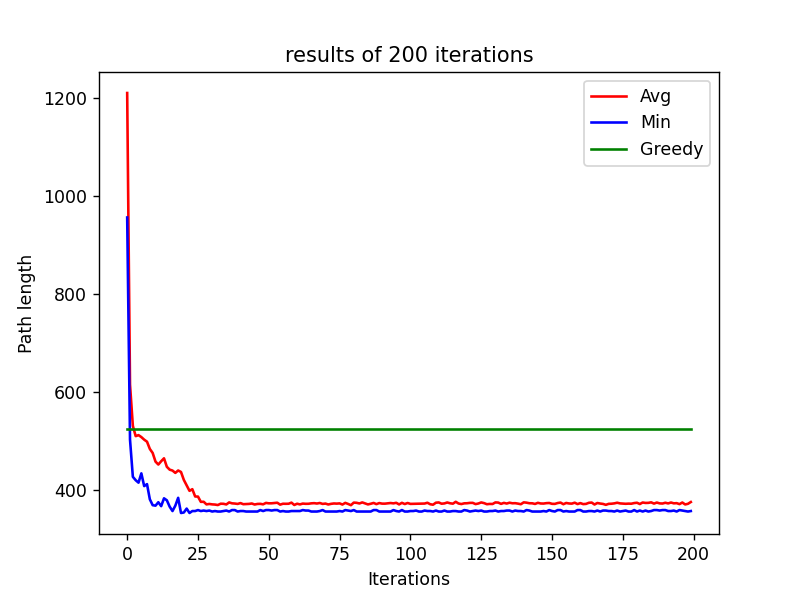


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Як можна побачити з графіку, проводити 1000 ітерацій недоцільно. Основні зміни маршруту відбуваються в перші кілька ітерацій, та після 20-30 ітерацій зміни результатів стають мінімальними.

Висновок

В рамках даної лабораторної роботи було розглянуто роботу мурашиного алгоритму (ACO) на прикладі задачі комівояжера. Використана варіація алгоритму – зі звичайними та елітними мурахами. Досліджено переваги та недоліки алгоритму.

Розроблено програмне забезпечення для роботи мурашиного алгоритму з задачею комівояжера з різними параметрами – кількістю вершин графа та початковою кількістю феромону на кожній вершині.

Проведено тестування роботи алгоритму за різних параметрів.

Критерії оцінювання

При здачі лабораторної роботи до 27.11.2021 включно максимальний бал дорівнює – 5. Після 27.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

* програмна реалізація алгоритму – 75%;
* тестування алгоритму– 20%;
* висновок – 5%.