**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**”

**Виконав(ла)**

(шифр, прізвище, ім'я, по батькові)

*ІП-12 Боборикін Артем*

**Перевірив**

(прізвище, ім'я, по батькові)

*Головченко М.Н.*

Київ 2022

Зміст

[1 Мета лабораторної роботи 3](#_Toc51260917)

[2 Завдання 4](#_Toc51260918)

[3 Виконання 10](#_Toc51260919)

[3.1 Програмна реалізація алгоритму 10](#_Toc51260920)

[3.1.1 Вихідний код 10](#_Toc51260921)

[3.1.2 Приклади роботи 10](#_Toc51260922)

[3.2 Тестування алгоритму 11](#_Toc51260923)

[3.2.1 Значення цільової функції зі збільшенням кількості ітерацій 11](#_Toc51260924)

[3.2.2 Графіки залежності розв'язку від числа ітерацій 11](#_Toc51260925)

[Висновок 12](#_Toc51260926)

[Критерії оцінювання 13](#_Toc51260927)

# 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.

# Виконання

## Програмна реалізація алгоритму

### Вихідний код

const fs = ***require***('fs')  
  
function buildGraf(num, load=false){  
 let graf = []  
 if(load){  
 graf = ***JSON***.parse(fs.readFileSync('last\_graf.txt').toString())  
 }else{  
 for(let i=0; i<num; i++){  
 graf.push([])  
 for(let j=0; j<num; j++){  
 if(i===j){  
 graf[i].push(null)  
 }else{  
 graf[i].push(***Math***.floor(***Math***.random()\*46+5))  
 }  
 }  
 }  
 fs.writeFileSync('last\_graf.txt', ***JSON***.stringify(graf))  
 }  
  
 return graf  
}  
function build\_d\_graf(graf){  
 let d\_graf = []  
  
 for(let one of graf){  
 d\_graf.push(one.slice(0))  
 }  
  
 for(let i=0; i<graf.length; i++){  
 for(let j=0; j<graf.length; j++){  
 if(d\_graf[i][j]){  
 d\_graf[i][j] = 1/d\_graf[i][j]  
 }  
 }  
 }  
  
 return d\_graf  
}  
function build\_feromon\_graf(num, load=false){  
 let graf = []  
  
 if(load){  
 graf = ***JSON***.parse(fs.readFileSync('last\_feromon\_graf.txt').toString())  
 }else{  
 for(let i=0; i<num; i++){  
 graf.push([])  
 for(let j=0; j<num; j++){  
 if(i===j){  
 graf[i].push(null)  
 }else{  
 graf[i].push(***Math***.floor(***Math***.random()\*3+1)/10)  
 }  
 }  
 }  
 fs.writeFileSync('last\_feromon\_graf.txt', ***JSON***.stringify(graf))  
 }  
  
 return graf  
}  
  
function find\_Lmin(graf){  
 let l = 0  
 let visited = [0]  
  
 for(let i=0; i<graf.length; i++){  
 let current = {  
 weight: 100,  
 num: visited[visited.length-1]  
 }  
 let prev\_mum = visited[visited.length-1]  
  
 for(let j=0; j<graf.length; j++){  
 if(graf[prev\_mum][j]&&!visited.includes(j)&&graf[prev\_mum][j]<current.weight){  
 current.weight = graf[prev\_mum][j]  
 current.num = j  
 }  
 }  
  
 if(visited.length===graf.length){  
 visited.push(0)  
 l+=graf[current.num][0]  
 }else{  
 visited.push(current.num)  
 l+=current.weight  
 }  
 }  
  
 return [visited, l]  
}  
//Функция, которая выводит результаты по основному массиву феромона  
function show\_way(){  
 let current\_feromon = 0  
 let str = '[ 0 '  
 let arr = []  
 for(let i=0; i<feromon\_graf.length; i++){  
 let index = feromon\_graf[current\_feromon].indexOf(***Math***.max.apply(null, feromon\_graf[current\_feromon]))  
 str+= index + ' '  
 arr.push(index)  
 current\_feromon = index  
 }  
 let sum = 0  
 let next = 0  
 for(let one of arr){  
 sum+=graf[next][one]  
 next = one  
 }  
 ***console***.log(str, ']')  
 ***console***.log('Final l: ', sum)  
}  
  
//Константы из задачи  
const alfa\_const = 2  
const beta\_const = 3  
const ro\_const = 0.4  
const M\_const = 35  
const graf\_length = 150  
  
//Строим граф  
let graf = buildGraf(graf\_length)  
let d\_graf = build\_d\_graf(graf)  
//Строим граф феромонов  
let feromon\_graf = build\_feromon\_graf(graf\_length)  
//Находим Lmin жадным алгоритмом  
let [arr, Lmin] = find\_Lmin(graf)  
  
***console***.log('Lmin: ', Lmin)  
  
//Второй массив феромона, который аккамулирует новые значения и потом перезаписывает основной в конце итерации  
let feromon\_graf\_temp = []  
for(let one of feromon\_graf){  
 feromon\_graf\_temp.push(one.slice(0))  
}  
  
//Главная функция алгоритма  
function main\_func(){  
 let l = 0  
 let visited = [***Math***.floor(***Math***.random()\*graf\_length)]  
  
 for(let i=0; i<graf.length; i++){  
 let current = {  
 weight: 100,  
 num: visited[visited.length-1],  
 p: 0  
 }  
 let this\_num = visited[visited.length-1]  
  
 let i\_sum = 0  
 for(let c=0; c<graf.length; c++){  
 i\_sum+=***Math***.pow(d\_graf[this\_num][c], beta\_const)\****Math***.pow(feromon\_graf[this\_num][c], alfa\_const)  
 }  
  
 for(let j=0; j<graf.length; j++){  
 if(graf[this\_num][j]&&!visited.includes(j)){  
 let next\_p = ***Math***.pow(d\_graf[this\_num][j], beta\_const)\****Math***.pow(feromon\_graf[this\_num][j], alfa\_const)/i\_sum  
 if(next\_p>current.p){  
 current.weight = graf[this\_num][j]  
 current.num = j  
 current.p = next\_p  
 }  
 }  
 }  
  
 if(visited.length===graf.length){  
 visited.push(visited[0])  
 l+=graf[current.num][visited[0]]  
 }else{  
 visited.push(current.num)  
 l+=current.weight  
 }  
 }  
  
 let feromon\_current = visited[0]  
 for(let c=0; c<graf.length; c++){  
 for(let c2 = 0; c2<graf.length; c2++){  
 if(feromon\_graf\_temp[feromon\_current][c2]&&c2===visited[c+1]){  
 feromon\_graf\_temp[feromon\_current][c2] +=Lmin/l  
 }  
  
 }  
 feromon\_current = visited[c+1]  
 }  
  
 return [visited, l]  
}  
  
function iteration(){  
 //Первая часть работы с феромоном (перемножаем все на коофициэнт)  
 //Сейчас видоизменяется дополнительный массив с феромонами, который не участвует в вычислениях  
 for(let i=0; i<feromon\_graf\_temp.length; i++){  
 for(let j=0; j<feromon\_graf\_temp; j++){  
 feromon\_graf\_temp[i][j] = (1-ro\_const)\*feromon\_graf\_temp[i][j]  
 }  
 }  
 //Главная функция для М муравьёв  
 for(let circle = 0; circle<M\_const; circle++){  
 main\_func()  
 }  
 //Перезаписываем основное значение феромона (таким образом, чтобы не было ссылок)  
 feromon\_graf = []  
 for(let one of feromon\_graf\_temp){  
 feromon\_graf.push(one.slice(0))  
 }  
}  
  
//Цикл итераций  
for(let one = 0; one<1; one++){  
 iteration()  
}  
  
show\_way()  
  
// for(let one of feromon\_graf){  
// let str = ''  
// for(let two of one){  
// if(two){  
// str+=Math.floor(two\*100)/100+' '  
// }else{  
// str+=two+' '  
// }  
// }  
// console.log(str)  
// }

### Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

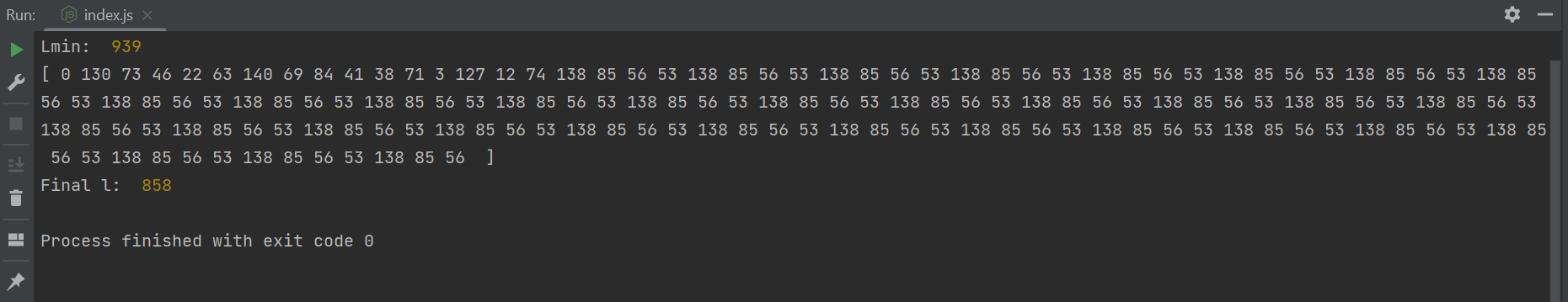


Рисунок 3.1 –

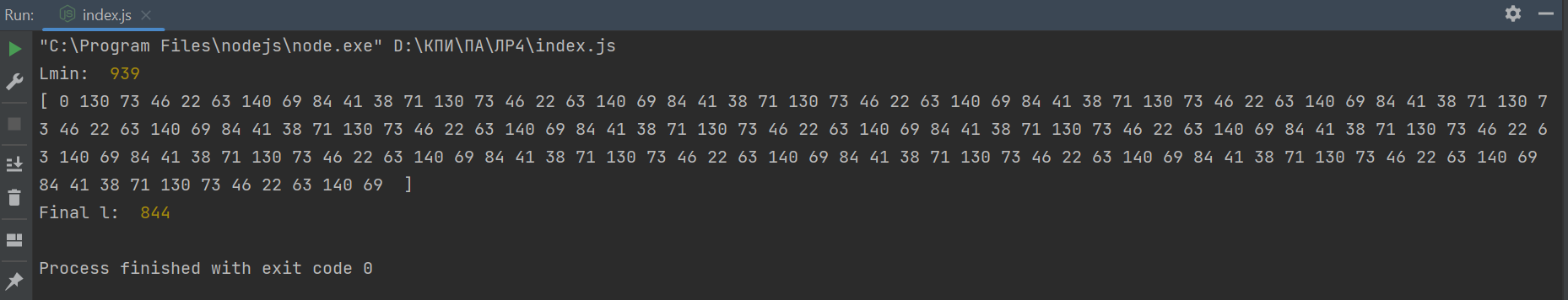


Рисунок 3.2 –

## Тестування алгоритму

### Значення цільової функції зі збільшенням кількості ітерацій

У таблиці 3.1 наведено значення цільової функції зі збільшенням кількості ітерацій.

|  |  |  |
| --- | --- | --- |
| Кількість ітерацій | Середнє значення цільової функціїї | Lmin |
| 1 | 863 | 939 |
| 5 | 847 | 939 |
| 10 | 830 | 939 |
| 50 | 826 | 939 |
| 100 | 822 | 939 |
| 1000 | 816 | 939 |

### Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

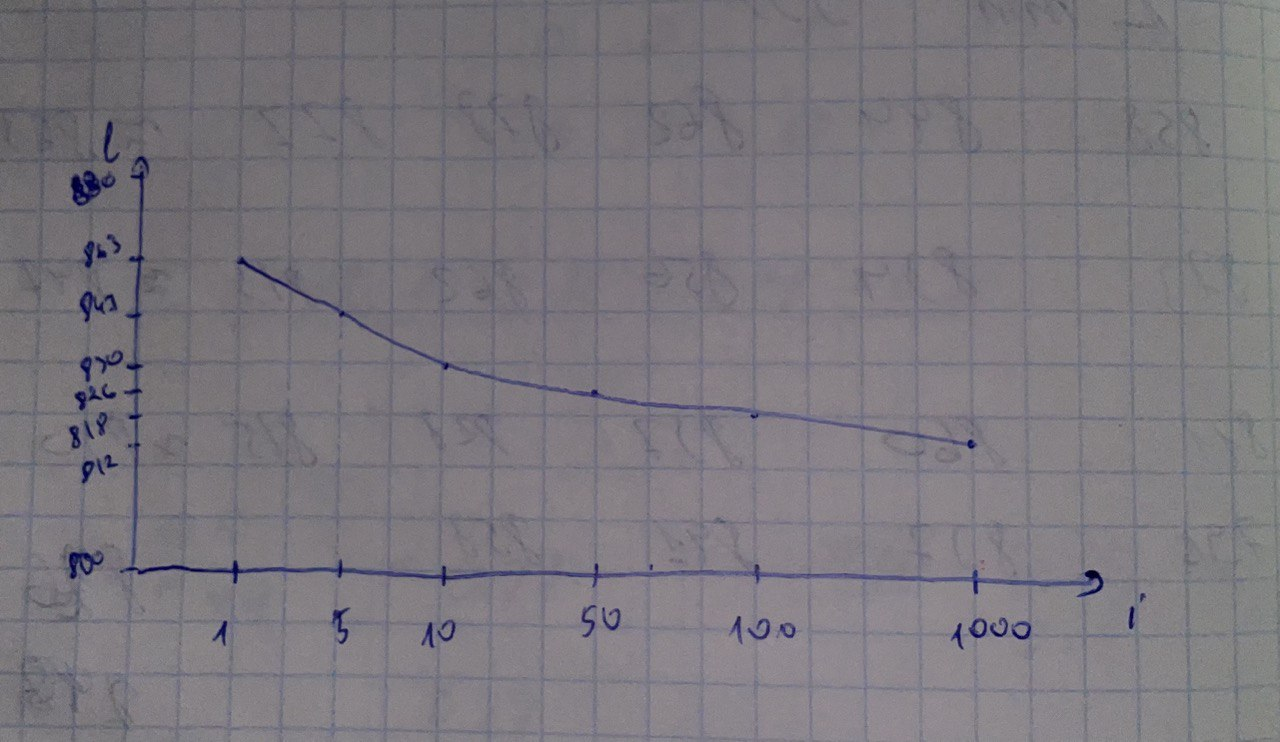


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

Висновок

В рамках даної лабораторної роботи був розроблений мишиний алгоритм для задачі з комівояжером, який показав себе ефективним способом розв’язку задачі (більш ефективний, ніж звичайний жадібний алгоритм). Після проведення експериментів можна побачити, що збільшення ітерацій (а також збільшення мишей) дають більш оптимальні розв’язки у середньому. Але чим більше збільшувати ітерації тим менше стає рости ефективність, тобто графік збільшення ефективності можна уявити у виді графіку

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%.