**­­ 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 № 5 з дисципліни

«Проектування алгоритмів»

„**Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2**”

**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](#_Toc52291748)

[2 Завдання 4](#_Toc52291749)

[3 Виконання 10](#_Toc52291750)

[3.1 Покроковий алгоритм 10](#_Toc52291751)

[3.2 Програмна реалізація алгоритму 10](#_Toc52291752)

[3.2.1 Вихідний код 10](#_Toc52291753)

[3.2.2 Приклади роботи 10](#_Toc52291754)

[3.3 Тестування алгоритму 11](#_Toc52291755)

[Висновок 12](#_Toc52291756)

[Критерії оцінювання 13](#_Toc52291757)

# Мета лабораторної роботи

Мета роботи – вивчити основні підходи розробки метаеврестичних алгоритмів для типових прикладних задач. Опрацювати методологію підбору прийнятних параметрів алгоритму.

# Завдання

Згідно варіанту, формалізувати алгоритм вирішення задачі відповідно загальної методології.

Записати розроблений алгоритм у покроковому вигляді. З достатнім степенем деталізації.

Виконати його програмну реалізацію на будь-якій мові програмування.

Перелік задач наведено у таблиці 2.1.

Перелік алгоритмів і досліджуваних параметрів у таблиці 2.2.

Задача і алгоритм наведені в таблиці 2.3.

Змінюючи параметри алгоритму, визначити кращі вхідні параметри алгоритму. Для цього необхідно:

* обрати критерій зупинки алгоритму (кількість ітерацій або значення ЦФ);
* зафіксувати усі параметри крім одного і змінювати цей параметр, поки не буде досягнуто пікової ефективності;
* після цього параметр фіксується і змінюються інші параметри;
* далі повторюємо процедуру спочатку, з першого зафіксованого параметру;
* зупиняємось коли будуть знайдені оптимальні параметри для даної задачі або встановлена залежність одних параметрів від інших.

Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

ВАРІАНТ

Задача комівояжера (змішана мережа) + Генетичний алгоритм

# Виконання

## Покроковий алгоритм

let arr = []

generate\_random\_values(arr) //Перші батьки

for (…){

let child\_arr = arr\_crossing() //Схрещення

child\_arr = mutation(child\_arr) // Мутація

child\_arr = local\_better(child\_arr) //Локальне покращення

arr.push(child\_arr) //Додання елементу до популяції

arr.splice(…) //Природній відбір

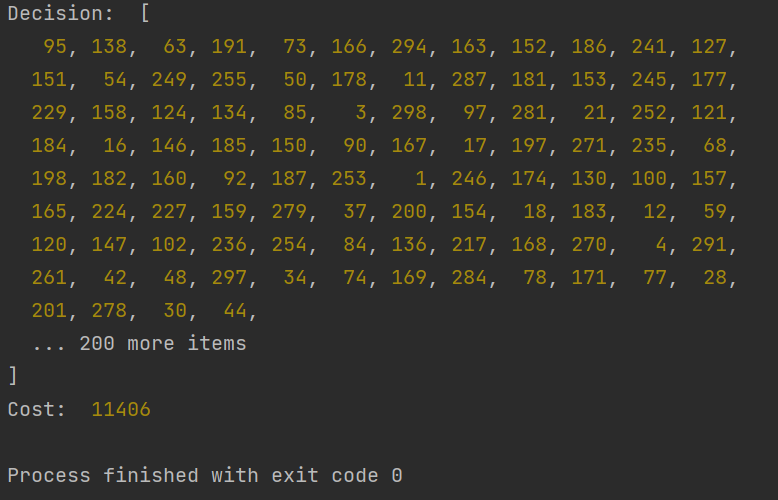
}

## Програмна реалізація алгоритму

### Вихідний код

const fs = ***require***('fs')  
  
//===================CONSTANTS==========================  
const GRAF\_LENGTH = 300 //Розмір графа  
const CROSSING\_OPERATOR = 3 //Оператор схрещування  
const MUTATION = 2 //Мутація  
const LOCAL\_IMPROVE = 2 //Оператор локального покращення  
const FIRST\_PARENTS = 30 //Перші батьки  
const ROUNDS = 10000 //Кількість кругів алгоритму  
  
//===================FUNCTIONS==========================  
//Building an oriented graf = [ [1,1,..], [1,1,..], .. ]  
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***.round(***Math***.random()\*145+5))  
 }  
 }  
 }  
 fs.writeFileSync('last\_graf.txt', ***JSON***.stringify(graf))  
 }  
 return graf  
}  
//Simple greedy search  
function find\_optimal\_way(graf){  
 let full\_cost = 0  
 let full\_way = [0]  
 for(let i=0, n=0; i<graf.length-1; i++){  
 let current\_best = {  
 value: 200,  
 index: null  
 }  
 for(let j=0; j<graf[i].length; j++){  
 if(graf[n][j]<current\_best.value&&!full\_way.includes(j)){  
 current\_best.value = graf[n][j]  
 current\_best.index = j  
 }  
 }  
 full\_cost+=current\_best.value  
 full\_way.push(current\_best.index)  
 n=current\_best.index  
 }  
 ***console***.log('Optimal Way', full\_way)  
 ***console***.log('Optimal Cost', full\_cost)  
 return [full\_way, full\_cost]  
}  
  
function generate\_random\_way(length){  
 let way = [0]  
 for(let i = 0; i<length-1; i++){  
 let new\_num = ***Math***.ceil(***Math***.random()\*(length-1))  
 while(way.includes(new\_num)){  
 new\_num = ***Math***.ceil(***Math***.random()\*(length-1))  
 }  
 way.push(new\_num)  
 }  
 return way  
}  
function calculate\_way(way, graf){  
 let full\_cost = 0  
 for(let i=1; i<way.length; i++){  
 full\_cost+=graf[way[i-1]][way[i]]  
 }  
 return full\_cost  
}  
function get\_best\_and\_random(arr, graf){  
 let temp\_arr = arr.map(one=>calculate\_way(one, graf))  
 let index = temp\_arr.indexOf(***Math***.min(...temp\_arr))  
 let random\_index  
 do{  
 random\_index = ***Math***.round(***Math***.random()\*(arr.length-1))  
 }while (random\_index===index)  
  
 return [arr[index], arr[random\_index]]  
}  
function arr\_crossing([arr1, arr2]){  
 let final\_arr = []  
  
 let switchOperator = false  
 for(let i=0; i<arr1.length; i++){  
 if(i%(GRAF\_LENGTH/CROSSING\_OPERATOR)===0){  
 switchOperator=!switchOperator  
 }  
  
 if(!switchOperator&&!final\_arr.includes(arr1[i])){  
 final\_arr.push(arr1[i])  
 }else {  
 if(!final\_arr.includes(arr2[i])){  
 final\_arr.push(arr2[i])  
 }  
 }  
 }  
 while(final\_arr.length!==arr1.length){  
 for(let one of arr1){  
 if(!final\_arr.includes(one)){  
 final\_arr.push(one)  
 }  
 }  
 }  
 return final\_arr  
}  
function mutation(arr){  
 let percent = ***Math***.round(***Math***.random()\*100)  
 if(percent>20){  
 return arr  
 }  
  
 let n1 = ***Math***.floor(***Math***.random()\*arr.length)  
 let n2  
 do{  
 n2 = ***Math***.floor(***Math***.random()\*arr.length)  
 } while (n2 === n1)  
  
 let temp\_arr = arr.slice(0)  
  
 let temp = temp\_arr[n1]  
 temp\_arr[n1] = temp\_arr[n2]  
 temp\_arr[n2] = temp  
  
 return temp\_arr  
}  
function local\_better(arr){  
 let calced\_way\_arr = calculate\_way(arr, graf)  
 let temp\_arr\_calc = []  
 for(let i=1; i<arr.length; i++){  
 temp\_arr\_calc.push(graf[arr[i-1]][arr[i]])  
 }  
 while (temp\_arr\_calc.length>0){  
 let temp\_arr = arr.slice(0)  
 let index = temp\_arr\_calc.indexOf(***Math***.max(...temp\_arr\_calc))  
 let min\_way\_graf = ***Math***.min.apply(null, graf[index].filter(e => e!==null))  
  
 let min\_way\_graf\_index = graf[index].indexOf(min\_way\_graf)  
 let temp\_arr\_min\_graf\_index = temp\_arr.indexOf(min\_way\_graf\_index)  
 let temp = temp\_arr[index+1]  
 temp\_arr[index+1] = min\_way\_graf\_index  
 temp\_arr[temp\_arr\_min\_graf\_index] = temp  
  
 if(calculate\_way(temp\_arr, graf)<=calced\_way\_arr){  
 return temp\_arr  
 }  
 temp\_arr\_calc = temp\_arr\_calc.slice(index, 1)  
 }  
 return arr  
}  
  
let graf = buildGraf(GRAF\_LENGTH, true)  
//console.log(graf)  
//console.log('=============================')  
  
function main(){  
 let [optimal\_way, optimal\_cost] = find\_optimal\_way(graf)  
  
 //First steps  
 let arr = []  
 for(let i=0; i<FIRST\_PARENTS; i++){  
 arr.push(generate\_random\_way(GRAF\_LENGTH))  
 }  
 //console.log('S1:', calculate\_way(arr[0], graf))  
 //console.log('S2:', calculate\_way(arr[1], graf))  
 //console.log('S3:', calculate\_way(arr[2], graf))  
 //console.log('S3:', calculate\_way(arr[3], graf))  
  
 for(let rounds = 0; rounds<ROUNDS; rounds++){  
 let child\_arr = arr\_crossing(get\_best\_and\_random(arr, graf))  
  
 for(let i=0; i<MUTATION; i++){  
 child\_arr = mutation(child\_arr)  
 }  
  
 for(let i=0; i<LOCAL\_IMPROVE; i++){  
 child\_arr = local\_better(child\_arr)  
 }  
  
 arr.push(child\_arr)  
 let temp\_calc\_arr = arr.map(one=>calculate\_way(one, graf))  
 arr.splice(temp\_calc\_arr.indexOf(***Math***.max(...temp\_calc\_arr)), 1)  
 }  
 let temp\_calc\_arr = arr.map(one=>calculate\_way(one, graf))  
 let final\_arr = arr[temp\_calc\_arr.indexOf(***Math***.min(...temp\_calc\_arr))]  
 ***console***.log('Decision: ', final\_arr,'\nCost: ', calculate\_way(final\_arr, graf))  
  
}  
main()

### Приклади роботи



На рисунках 3.1 і 3.2 показані приклади роботи програми.

## Тестування алгоритму

|  |  |  |
| --- | --- | --- |
| Ітерації | Параметри | Результат (ціна) |
| 1 | CO=3 | M=2 | LI=2 | FP=30 | 21822 |
| 10 | CO=3 | M=2 | LI=2 | FP=30 | 21166 |
| 100 | CO=3 | M=2 | LI=2 | FP=30 | 20063 |
| 1000 | CO=3 | M=2 | LI=2 | FP=30 | 18166 |
| 1000 | CO=30 | M=2 | LI=2 | FP=30 | 18399 |
| 1000 | CO=3 | M=2 | LI=100 | FP=30 | 18112 |
| 1000 | CO=3 | M=100 | LI=10 | FP=30 | 17993 |
| 10000 | CO=3 | M=2 | LI=2 | FP=30 | 11277 |

Висновок

При виконанні даної роботи був побудований генетичний алгоритм у задачі комівояжера. Після тестування різних параметрів можна прийти до висновку, що найбільш потужним параметром є саме кількість циклів (тобто етапів народження). Параметр локального покращення показав себе також корисним, параметр мутації та схрещування також є дієвими, але не стабільними.