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) *ІП-13 Нещерет В. О.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О. О.* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ 3

2 ЗАВДАННЯ 4

3 ВИКОНАННЯ 11

3.1 Покроковий алгоритм 11

3.2 Програмна реалізація алгоритму 11

3.2.1 Вихідний код 11

3.2.2 Приклади роботи 13

3.3 Тестування алгоритму 15

ВИСНОВОК 17

КРИТЕРІЇ ОЦІНЮВАННЯ 18

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

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

2 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.

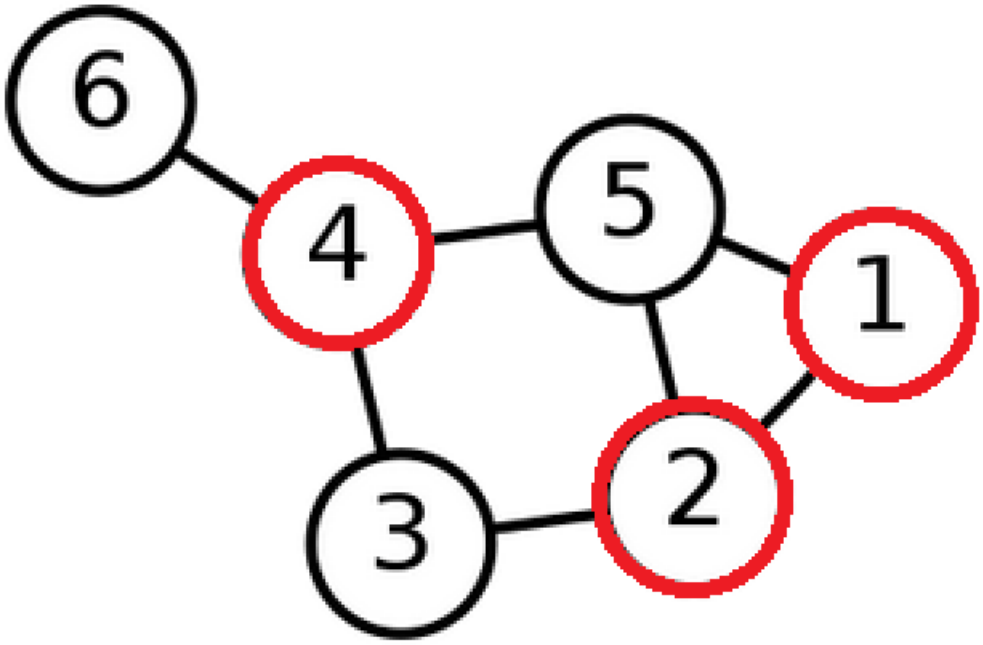
Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

№	**Задача**
1	**Задача про рюкзак** (місткість P=500, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 20 (випадкова)). Для заданої множини предметів, кожен з яких має вагу і цінність, визначити яку кількість кожного з предметів слід взяти, так, щоб

	сумарна вага не перевищувала задану, а сумарна цінність була максимальною. Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика.
2	**Задача комівояжера** (300 вершин, відстань між вершинами випадкова від 5 до 150) полягає 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. **Розглядається симетричний, асиметричний та змішаний варіанти.** В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напряму, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра. У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу. У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху. Застосування: – доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); – доставка води;

	– моніторинг об'єктів; – поповнення банкоматів готівкою; – збір співробітників для доставки вахтовим методом.
3	**Розфарбовування графа** (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке приписування кольорів (або натуральних чисел) його вершинам, що ніякі дві суміжні вершини не набувають однакового кольору. Найменшу можливу кількість кольорів у розфарбуванні називають хроматичне число. Застосування: – розкладу для освітніх установ; – розкладу в спорті; – планування зустрічей, зборів, інтерв'ю; – розклади транспорту, в тому числі - авіатранспорту; – розкладу для комунальних служб;
4	**Задача вершинного покриття** (300 вершин, степінь вершини не більше 30, але не менше 2)**.** Вершинне покриття для неорієнтованого графа G = (V, E) - це множина його вершин S, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з S. Задача вершинного покриття полягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа). На вході: Граф G = (V, E). Результат: множина C ⊆ V - найменше вершинне покриття графа G.

	 Застосування: – розміщення пунктів обслуговування; – призначення екіпажів на транспорт; – проектування інтегральних схем і конвеєрних ліній.
5	**Задача про кліку** (300 вершин, степінь вершини не більше 30, але не менше 2)**.** Клікою в неорієнтованому графі називається підмножина вершин, кожні дві з яких з'єднані ребром графа. Іншими словами, це повний підграф первісного графа. Розмір кліки визначається як число вершин в ній. Задача про кліку існує у двох варіантах: у **задачі розпізнавання** потрібно визначити, чи існує в заданому графі G кліка розміру k, тоді як в **обчислювальному варіанті** потрібно знайти в заданому графі G кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити). Застосування: – біоінформатика; – електротехніка;
6	**Задача про найкоротший шлях** (300 вершин, відстань між вершинами випадкова від 5 до 150, степінь вершини не більше 10, але

	не менше 1) - задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що складають шлях. Важливість задачі визначається її різними практичними застосуваннями. Наприклад, в GPS-навігаторах здійснюється пошук найкоротшого шляху між точкою відправлення і точкою призначення. Як вершин виступають перехрестя, а дороги є ребрами, які лежать між ними. Якщо сума довжин доріг між перехрестями мінімальна, тоді знайдений шлях найкоротший.

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	**Алгоритми і досліджувані параметри**
1	**Генетичний алгоритм:** - оператор схрещування (мінімум 3); - мутація (мінімум 2); - оператор локального покращення (мінімум 2).
2	**Мурашиний алгоритм**: – α; – β; – ρ; – Lmin; – кількість мурах M і їх типи (елітні, тощо…); – маршрути з однієї чи різних вершин.
3	**Бджолиний алгоритм:** – кількість ділянок; – кількість бджіл (фуражирів і розвідників).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
1	Задача про рюкзак + Генетичний алгоритм
2	Задача про рюкзак + Бджолиний алгоритм
3	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
4	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
5	Задача комівояжера (змішана мережа) + Генетичний алгоритм
6	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
7	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
8	Задача комівояжера (змішана мережа) + Мурашиний алгоритм
9	Задача вершинного покриття + Генетичний алгоритм
10	Задача вершинного покриття + Бджолиний алгоритм
11	Задача комівояжера (асиметрична мережа) + Бджолиний алгоритм
12	Задача комівояжера (симетрична мережа) + Бджолиний алгоритм
13	Задача комівояжера (змішана мережа) + Бджолиний алгоритм
14	Розфарбовування графа + Генетичний алгоритм
15	Розфарбовування графа + Бджолиний алгоритм
16	Задача про кліку (задача розпізнавання) + Генетичний алгоритм
17	Задача про кліку (задача розпізнавання) + Бджолиний алгоритм
18	Задача про кліку (обчислювальна задача) + Генетичний алгоритм
19	Задача про кліку (обчислювальна задача) + Бджолиний алгоритм
20	Задача про найкоротший шлях + Генетичний алгоритм
21	Задача про найкоротший шлях + Мурашиний алгоритм
22	Задача про найкоротший шлях + Бджолиний алгоритм
23	Задача про рюкзак + Генетичний алгоритм
24	Задача про рюкзак + Бджолиний алгоритм
25	Задача комівояжера (асиметрична мережа) + Генетичний алгоритм
26	Задача комівояжера (симетрична мережа) + Генетичний алгоритм
27	Задача комівояжера (змішана мережа) + Генетичний алгоритм

28	Задача комівояжера (асиметрична мережа) + Мурашиний алгоритм
29	Задача комівояжера (симетрична мережа) + Мурашиний алгоритм
30	Задача комівояжера (змішана мережа) + Мурашиний алгоритм

3 ВИКОНАННЯ

3.1 Покроковий алгоритм

1. Для заданої кількості ітерацій:
   1.1. Для кожної бджоли-розвідника згенерувати випадковий розв’язок.
   1.2. Відібрати найкращих бджіл серед розвідників та фуражирів.
   1.3. Для кожного фуражира знайти можливий розв’язок поруч із одним із списку найкращих наявних.
   1.4. Замінити найгірші згенеровані фуражирами розв’язки на найкращі, що були знайдені на кроці 1.2.
2. Повернути найкраще знайдене рішення.

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

main.kt

```
fun main() {
    val nodesNumber = 300
    val maxDegree = 150

    val graph = Graph(nodesNumber, maxDegree)

    val solution = BeeAlgorithm(graph).run()

    println("Max clique size: ${solution?.fitness}\n" +
            "Max clique: ${solution?.nodes}")
}
```

Graph.kt

```
class Graph(nodesNum : Int, val maxDegree : Int) {
    val nodes : Set<Node> =
        (0 until nodesNum).map { Node(it) }.toSet()

    init {
        nodes.forEach { node ->
            val degree = (2..maxDegree).random()
            val neighbors = nodes.filter{ it != node }
                .shuffled()
                .take(degree)
            neighbors.forEach { node.addNeighbor(it) }
        }
    }
}
```

Node.kt

```
class Node(val id : Int, private val adjacentNodes : MutableSet<Node> =
mutableSetOf()) {
    fun addNeighbor(node : Node) {
        adjacentNodes.add(node)
    }

    fun areNeighbors(node : Node) : Boolean {
        return adjacentNodes.contains(node)
    }

    override fun toString() : String{
        return "Node $id"
    }
}
```

Bee.kt

```
class Bee(private val graph : Graph) {
    var nodes = mutableSetOf<Node>()
    val fitness : Int
        get() = if (isClique(nodes)) nodes.size else 0

    fun generateRandomSolution() {
        nodes = graph.nodes.shuffled()
            .take((2..8).random())
            .toMutableSet()
    }

    fun modifySolution(newNodes : MutableSet<Node>) {
        nodes = mutableSetOf()
        nodes.addAll(newNodes)
        var counter = 0
        do {
            var isModified = false
            val newNode = graph.nodes.shuffled().first()
            if (!nodes.contains(newNode)) {
                nodes.add(newNode)
                isModified = true
            }
            counter++
        } while (counter < nodes.size && !isModified)
    }

    private fun isClique(nodes : MutableSet<Node>) : Boolean {
        for (node in nodes) {
            for (otherNode in nodes) {
                if (node != otherNode && !node.areNeighbors(otherNode)) {
                    return false
                }
            }
        }
        return true
    }
}
```

BeeAlgorithm.kt

```
class BeeAlgorithm(graph: Graph) {
    val foragerBeesNumber = 100
    val scoutBeesNumber = 100
    val maxIterations = 1000
    val eliteBeesNumber = 1

    val scoutBees = (0 until scoutBeesNumber).map { Bee(graph) }.toMutableList()
    val foragerBees = (0 until foragerBeesNumber).map { Bee(graph)
}.toMutableList()

    fun run() : Bee? {
        var bestSolution : Bee? = null
        for (iteration in 0..maxIterations) {
            scoutBees.forEach { it.generateRandomSolution() }

            val eliteBees = selectEliteBees()
            for (i in foragerBees.indices) {
                foragerBees[i].modifySolution(eliteBees[i %
eliteBeesNumber].nodes)
            }
            replaceWorstSolutions(eliteBees)
            bestSolution = (foragerBees + scoutBees).maxBy { it.fitness }
            println("Iteration: $iteration, Solution: ${bestSolution.fitness}")
        }

        return bestSolution
    }

    fun selectEliteBees() : List<Bee> {
        return (foragerBees + scoutBees).sortedByDescending { it.fitness
}.take(eliteBeesNumber)
    }

    fun replaceWorstSolutions(eliteBees : List<Bee>) {
        foragerBees.sortBy { it.fitness }
        for (i in 0 until eliteBeesNumber) {
            foragerBees[i] = eliteBees[i]
        }
    }
}
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
Iteration: 1000, Solution: 4
Max clique size: 4
Max clique: [Node 43, Node 155, Node 238, Node 102]

Process finished with exit code 0
```

Рисунок 3.1 – Робота програми з розміром графу 300 та степенем вершин в межах (2..30).

```
Iteration: 1000, Solution: 7
Max clique size: 7
Max clique: [Node 182, Node 6, Node 121, Node 265, Node 130, Node 4, Node 274]

Process finished with exit code 0
```

Рисунок 3.2 – Робота програми з розміром графу 300 та степенем вершин в межах (2..150).

3.3 Тестування алгоритму

Як критерій зупинки алгоритму я обрав кількість ітерацій. Зазвичай значення в 1000 ітерацій дозволяло отримати прийнятний розв’язок.

Протестуємо алгоритм на різних вхідних параметрах розміру графа, степенів його вершин та кількості ітерацій.

Таблиця 3.1 – Результат роботи алгоритму в залежності від параметрів графа.

Кількість вершин	**Максимальна степінь вершин**	**Кількість ітерацій**	**Розмір знайденої кліки**
5	5	100	3
50	25	100	5
100	50	1000	7
200	50	1000	5
300	100	1000	8

Щоб знайти оптимальну кількість бджіл-фуражирів та бджіл-розвідників, я вирішив не змінювати загальну кількість бджіл, а лиш пропорцію їх розподілу на групи. За результатими тестування я отримав середні значення (на основі 5 тестувань) розміру знайденої кліки для кожного розподілу. Розмір графа дорівнював 300, а максимальна степінь вершин – 150.

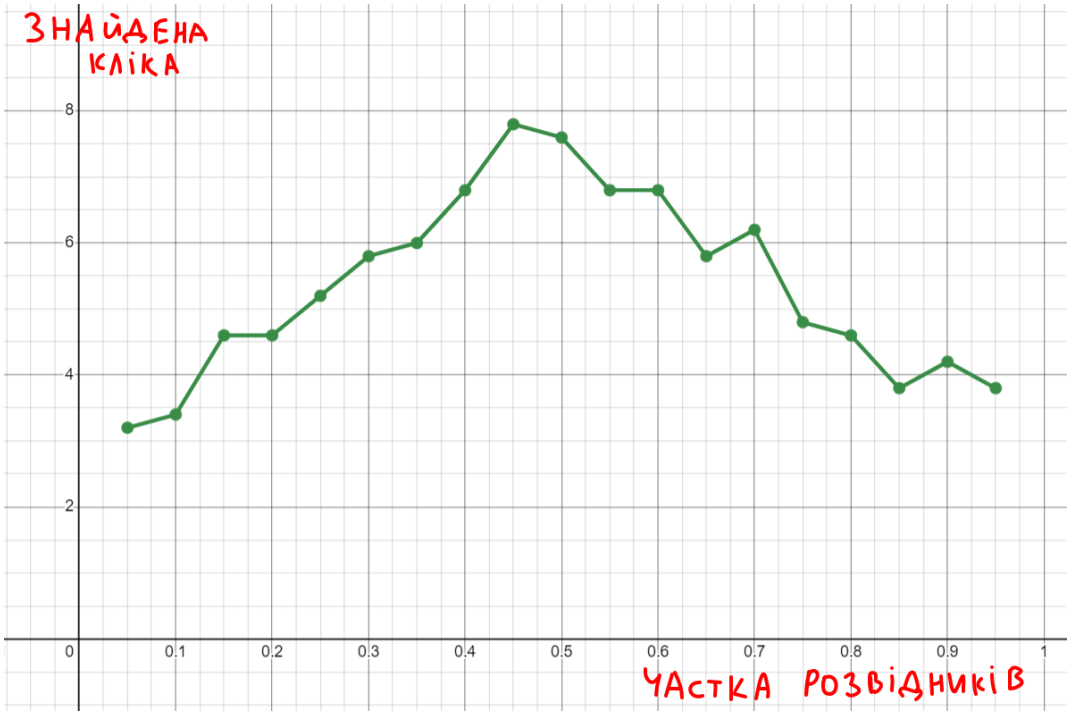


Рисунок 3.3 – Залежність розв’язку від частки бджіл-розвідників

З графіку можна зробити висновок, що оптимальна кількість розвідників – 45%.

ВИСНОВОК

В рамках даної лабораторної роботи я дізнався як працює бджолиний алгоритм, описав послідовність його роботи та використав його для вирішення задачі про кліку. Для цього довелось представити можливий розв’язок задачі як набір вершин, бджоли розвідники підбирали випадкові набори вершин, а фуражири – намагалися покращити розв’язок шляхом додавання нової вершини до набору. В результаті я розробив програмну реалізацію алгоритму та дослідив її для визначення залежності розв’язку від вхідних параметрів алгоритму.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 11.12.2022 включно максимальний бал дорівнює – 5. Після 11.12.2022 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 15%;
- програмна реалізація алгоритму – 50%;
- тестування алгоритму– 30%;
- висновок – 5%.