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-15 Мочалов Дмитро Юрійович* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Ахаладзе І.Е.* _____
(прізвище, ім'я, по батькові)

Київ 2022

ЗМІСТ

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.

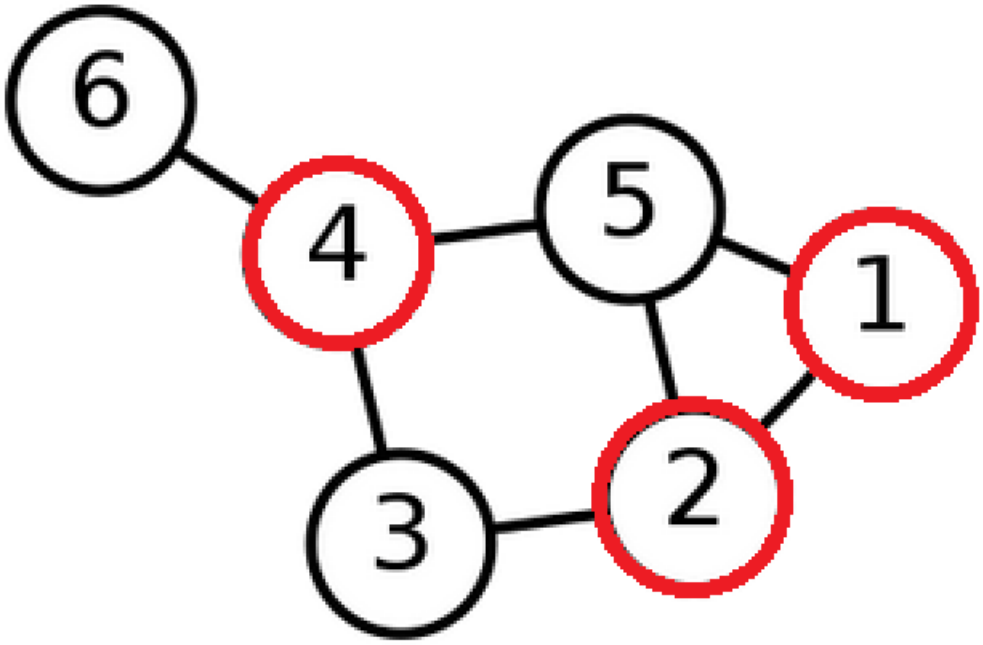
Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

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. Ініціалізувати бджіл-розвідників і бджіл-фуражирів
2. Для заданої кількості ітерацій виконувати
    2.1. Для кожної бджоли розвідника згенерувати рандомне рішення
    2.2. Відібрати найкращі рішення серед бджіл розвідників і бджіл фуражирів
    2.3. Для кожного фуражира дослідити окіл найкращих рішень
    2.4. Замінити найгірше рішення в популяції найкращим досліджених фуражирами
3. Повернути найкраще рішення

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
public class BeeAlgorithm
{
    private int ForagersCount { get; }
    private int ScoutsCount { get; }
    private int MaxIterations = 1000;
    private int EliteBeesCount { get; set; }
    private Graph Graph { get; }

    private List<Bee> ForagerBees { get; }
    private List<Bee> ScoutBees { get; }

    public BeeAlgorithm(int foragers, int scouts, Graph graph)
    {
        ForagersCount = foragers;
        ScoutsCount = scouts;
```

```
        Graph = graph;
        EliteBeesCount = (ForagersCount + ScoutsCount) / 10;
        ForagerBees = new List<Bee>(ForagersCount);
        ScoutBees = new List<Bee>(ScoutsCount);

        InitializeBees();
    }

    public List<Bee> Solve()
    {
        List<Bee> BestSolutions = new List<Bee>();

        for (int i = 0; i < MaxIterations; i++)
        {
            foreach (var scout in ScoutBees)
            {
                scout.GenerateRandomSolution();
            }

            var EliteBees = GetBestSolutions();

            for (int j = 0; j < ForagersCount; j++)
            {
                ForagerBees[j].ExploreNeighbourhood(EliteBees[j %
EliteBeesCount].Nodes);
            }
            ReplaceWorstSolution(EliteBees);
            var bestSolution = GetBestSolutions().First();
            BestSolutions.Add(bestSolution);
        }
```

```
        return BestSolutions;
    }


    private List<Bee> GetBestSolutions()
    {
        var UnitedBees = ForagerBees.Concat(ScoutBees).ToList();
        return UnitedBees.OrderByDescending(u =>
u.Fitness).Take(EliteBeesCount).ToList();
    }

    private void ReplaceWorstSolution(List<Bee> eliteBees)
    {
        var orderedForagers = ForagerBees.OrderBy(u => u.Fitness).ToList();
        ForagerBees.Clear();
        ForagerBees.AddRange(orderedForagers);
        for (int i = 0; i < EliteBeesCount; i++)
        {
            ForagerBees[i] = eliteBees[i];
        }

    }

    private void InitializeBees()
    {
        for (int i = 0; i < ForagersCount; i++)
        {
            ForagerBees.Add(new Bee(Graph));
        }
```

```
        for (int i = 0; i < ScoutsCount; i++)
        {
            ScoutBees.Add(new Bee(Graph));
        }
    }

}
        public class Bee
{
    private Graph Graph { get; }

    public HashSet<Node> Nodes = new();

    public int Fitness => IsClique(Nodes) ? Nodes.Count : 0;

    public Bee(Graph graph)
    {
        Graph = graph;
    }

    public void GenerateRandomSolution()
    {
        Nodes.Clear();
        var random = new Random();
        var nodesCount = Graph.Nodes.Count;
        var randomSolutionNodesCount = random.Next(2,Graph.NodeDegree+1);
        for (int i = 0; i < randomSolutionNodesCount; i++)
        {
            var randomNode = Graph.Nodes.ElementAt(random.Next(0, nodesCount));
            while (Nodes.Contains(randomNode))
```

```
            {
                randomNode = Graph.Nodes.ElementAt(random.Next(0, nodesCount));
            }
            Nodes.Add(randomNode);
        }
    }

    public void ExploreNeighbourhood(HashSet<Node> newNodes)
    {
        Nodes.Clear();
        foreach (var node in newNodes)
        {
            Nodes.Add(node);
        }

        if (newNodes.Count != 0)
        {
            var neighbourNodes = newNodes.Select(node => new HashSet<Node>
(node.Connections )).ToList();
            HashSet<Node> neighbourNodesMutual = neighbourNodes.Aggregate((x, y)
=> x.Intersect(y).ToHashSet());
            if (neighbourNodesMutual.Count != 0)
            {
                var listNeighbourNodesMutual = neighbourNodesMutual.ToList();
                var randomMutualNode = listNeighbourNodesMutual[new
Random().Next(0, listNeighbourNodesMutual.Count)];
                Nodes.Add(randomMutualNode);
            }
        }
```

```
    }
    private bool IsClique(HashSet<Node> nodes)
    {
        foreach (var node in nodes)
        {
            foreach (var otherNode in nodes)
            {
                if(node != otherNode && !node.IsConnected(otherNode))
                {
                    return false;
                }
            }
        }
        return true;
    }
}
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
The best solution has 4 fitness and nodes = [Node: 224, Node: 11, Node: 82, Node: 92]
```

Рисунок 3.1 – Приклад роботи алгоритму для графа з 300 вершинами та степенем вершин в межах [2-30]

```
The best solution has 6 fitness and nodes = [Node: 278, Node: 151, Node: 164, Node: 185, Node: 50, Node: 94]
```

Рисунок 3.2 – Приклад роботи алгоритму для графа з 300 вершинами та степенем вершин в межах [2-100]

3.3 Тестування алгоритму

Критерій зупинки 1000 ітарецій

Протестуємо алгоритм з різними параметрами графа беручи 10 бджіл розвідників і 100 фуражирів.

Таблиця 3.1 – Таблиця результатів роботи алгоритму в залежності від параметрів графа

Кількість вершин	Максимальна степінь вершин	Кількість ітерацій	Розмір максимальної кліки
50	5	1000	3
100	25	1000	5
200	100	1000	8

Протестуємо якість розвязку в залежності в кількості взятих фуражирів і розвідників. Максимальна кількість ітерацій така ж. Граф на 300 вершин з макс степенем 100. Нехай загальна сума бджіл буде 200. Будемо змінювати відсоткове відношення.

Кількість розвідників	Кількість фуражирів	Розмір максимальної кліки
20	180	5
40	160	6
100	100	8
150	50	4

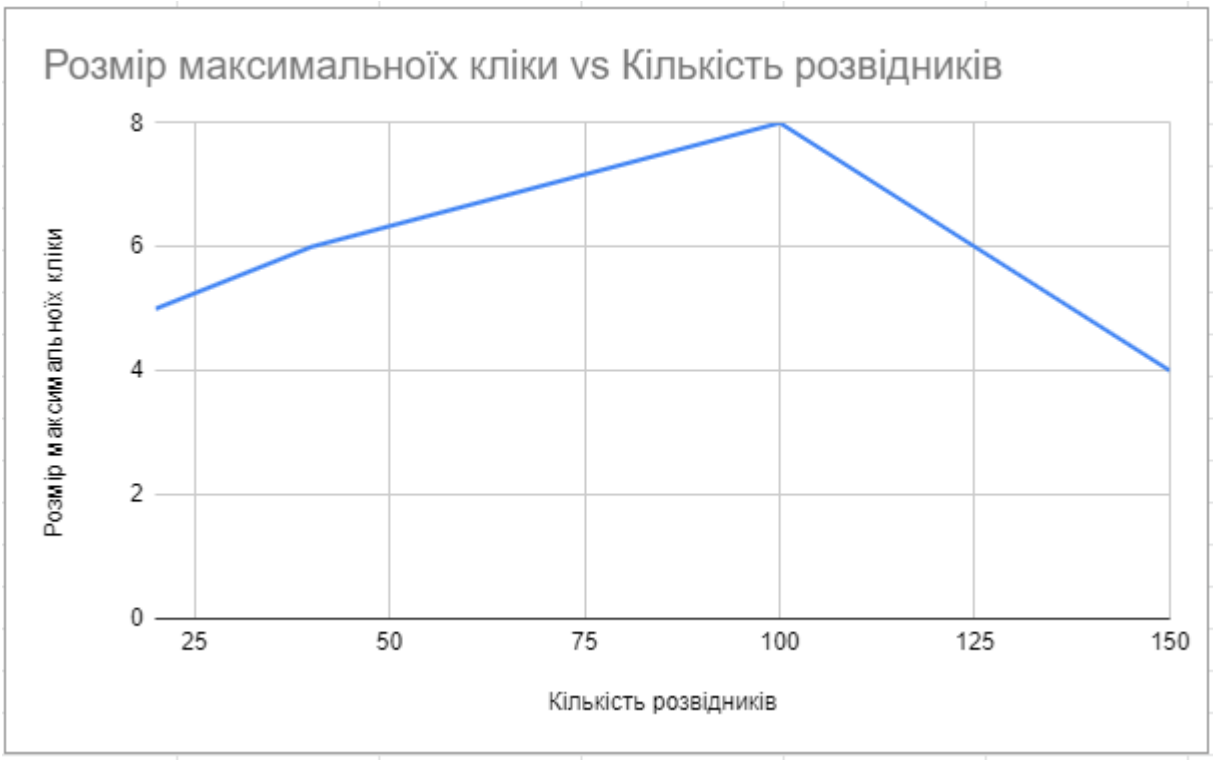


Рисунок 3.1 – Графік залежності кількості бджіл розвідників до розміру макс кліки

ВИСНОВОК

В рамках даної лабораторної роботи я розробив псевдокод для бджолиного алгоритму, виконав програмну реалізацію цього алгоритму, та використав його для розв’язання задачі про максимальну кліку графа.

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%.