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-14 Хільчук А.В.* _____
(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.

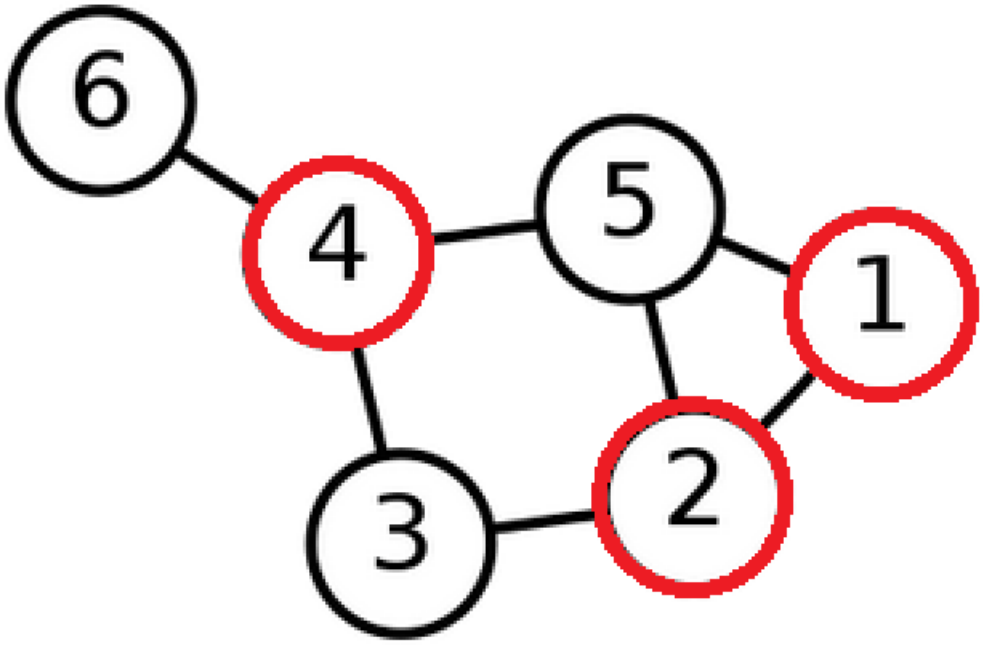
Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

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. Бджоли-розвідники формують випадковий підграф, переміщуючись між суміжними вузлами, базуючись на їхніх ступенями, і, обрізаючи його до кліки, діляться отриманим підграфом з вуликом
2. Вулик запам’ятовує отримані розв’язки, встановлюючи вагу їх нектару, базуючись на квадратичній кореляції з кількістю членів підграфа та лінійній кореляції з кількістю нащадків графа-кількістю таких гарфів, що містять у собі даний та теж становлять кліки
3. Бджоли-фуражири, базуючись на наявних значеннях нектару ділянок, відправляються до найбільш перспективних та розвідують один із їх нащадків, видаляючи його з множини нащадків батьківського вузла
4. Вулик знову ж таки отримує розвідані розв’язки та запам’ятовує їх, встановлюючи вагу їх нектару, базуючись на наведеному принципу
5. Вулик ітерує по кожному розв’язку в пам’яті та перевіряє, чи не рівна його величина шуканому розміру кліки. Якщо такий знайдений- повернути його та завершити роботу алгоритму
6. Вулик аналізує отримані розв’язки, видаляє ті, що не мають нащадків, та обирає певну кориговану частку від них для підтримання принципу забування виснажених ділянок
7. Повторювати кроки 1-6

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
#include "stdafx.h"
```

BeeColony.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;
using PA_Lab5.NodeStructures;
using PA_Lab5.Bees;

namespace PA_Lab5.BeeColonies
{
    class BeeColony
    {
        private List<EmployedBee> _employedBees;
        private List<OnlookerBee> _onlookerBees;
        private List<Tuple<SubGraph, int>> _foodSourcesAndSizes;
        private int _cliqueSize;
        private Graph _graph;
        private double _coefficient;

        public BeeColony(Graph graph, int cliqueSize, double coefficient)
        {
            _foodSourcesAndSizes = new List<Tuple<SubGraph, int>>();
            _graph = graph;
            _cliqueSize = cliqueSize;
            _coefficient = coefficient;
        }

        public void SendBees()
        {
            var discoveredAreas = new List<SubGraph>();
            foreach (var bee in _employedBees)
            {
                bee.PickArea(_cliqueSize);
                discoveredAreas.Add(MakeAClique(bee.GetLocation()));
            }
            ShareDiscoveredAreas(discoveredAreas);
            _foodSourcesAndSizes = _foodSourcesAndSizes.Where(a =>
a.Item1.GetDescendants().Count != 0).ToList();
            discoveredAreas = new List<SubGraph>();
            foreach (var bee in _onlookerBees)
            {
                bee.PickArea(_cliqueSize);
                bee.DiscoverAdjacentArea(_cliqueSize);
                discoveredAreas.Add(MakeAClique(bee.GetLocation()));
            }
            ShareDiscoveredAreas(discoveredAreas);
        }

        public void ShareDiscoveredAreas(List<SubGraph> discoveredAreas)
        {
            foreach (var area in discoveredAreas)
            {
                _foodSourcesAndSizes.Add(Tuple.Create(area, AnalyzeNectar(area)));
            }
        }

        public void SetBees(List<EmployedBee> employedBees, List<OnlookerBee> onlookerBees)
        {
            _employedBees = employedBees;
            _onlookerBees = onlookerBees;
        }

        public void UpdateFoodSourcesData()
        {
            _foodSourcesAndSizes = _foodSourcesAndSizes
                .Where(a=> a.Item1.GetDescendants().Count!=0)
```

```
                .OrderByDescending(a =>a.Item2)
                .Take((int)(_foodSourcesAndSizes.Count*_coefficient)).ToList();
        }

        public static int AnalyzeNectar(SubGraph subGraph)
        {
            return subGraph.GetMembers().Count *
subGraph.GetMembers().Count*(1+subGraph.GetDescendants().Count);
        }

        public Graph GetSearchArea()
        {
            return _graph;
        }

        public List<Tuple<SubGraph, int>> GetDiscoveredFoodSources()
        {
            return _foodSourcesAndSizes;
        }

        public SubGraph FindClique()
        {
            for (int i = 0; i < int.MaxValue; i++)
            {
                SendBees();
                foreach (var source in _foodSourcesAndSizes)
                {
                    if (source.Item1.GetMembers().Count>=_cliqueSize) return source.Item1;
                }
                UpdateFoodSourcesData();
                if (i % 100 == 0) Console.WriteLine("Biggest clique in memory so far:{3}",
_foodSourcesAndSizes.Max(a => a.Item2),
_foodSourcesAndSizes.Count,_foodSourcesAndSizes.Average(a=>a.Item1.GetMembers().Count),_food
SourcesAndSizes.Max(a=>a.Item1.GetMembers().Count));
            }
            return null;
        }

        public SubGraph MakeAClique(SubGraph original)
        {
            List<Node> traversed= new List<Node>();
            List<Node> toRemove = new List<Node>();

            foreach (var member in original.GetMembers())
            {
                foreach(var traversedOne in traversed)
                {
                    if (!traversedOne.GetAdjacents().Contains(member)) {
toRemove.Add(member); break; }
                }
                if (!toRemove.Contains(member)&&member.GetAdjacents().Count>=_cliqueSize)
traversed.Add(member);
            }
            return new SubGraph(traversed);
        }
    }
}
```

Ibee.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using PA_Lab5.NodeStructures;

namespace PA_Lab5.Bees
```

```
{
    interface IBee
    {
            public void PickArea(int areaSize);
            public SubGraph GetLocation();

    }
}
```

EmployedBee.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using PA_Lab5.NodeStructures;
using System.Linq;
using PA_Lab5.BeeColonies;

namespace PA_Lab5.Bees
{
    class EmployedBee : IBee
    {
        private BeeColony _colony;
        private SubGraph _subGraph;
        public EmployedBee(BeeColony colony)
        {
            _colony = colony;
        }

        public SubGraph GetLocation()
        {
            return _subGraph;
        }

        public void PickArea(int size)
        {
            List<Node> tempSolution = new List<Node>();
            tempSolution.Add(PickNode(_colony.GetSearchArea().GetNodes()));

            while (tempSolution.Count != size)
            {
                int i = 0;
                Node goal;
                do
                {
                    goal = null;
                    i++;
                    var tempNode = tempSolution[tempSolution.Count - i];
                    goal = PickNode(tempNode.GetAdjacents().Where(a =>
!tempSolution.Contains(a)).ToList());
                } while (tempSolution.Contains(goal));
                tempSolution.Add(goal);
            }

            int counter = 0;
            foreach (var a in tempSolution)
            {
                foreach (var b in tempSolution)
                {
                    if (a.Identificator == b.Identificator) counter++;
                }
            }
            if (counter > size) throw new Exception();
```

```
            _subGraph = new SubGraph(tempSolution);
            _subGraph.AddDescendants(_subGraph.AnalyzeAdjacentCommonNodes().ToList());
        }

        public Node PickNode(List<Node> nodes)
        {
            var cumulativeNectar = 0;
            foreach (var node in nodes)
            {
                cumulativeNectar += node.GetAdjacents().Count;
            }

            Dictionary<Node, double> chances = new Dictionary<Node, double>();
            foreach (var node in nodes)
            {
                chances.Add(node, (double)node.GetAdjacents().Count /
(double)cumulativeNectar);
            }

            chances.OrderBy(a => a.Value);
            var cumulativeChance = 0D;
            Random rand = new Random();

            var chance = rand.NextDouble();

            foreach (var probability in chances)
            {
                if (chance > cumulativeChance && chance < (cumulativeChance +
probability.Value))
                {
                    return probability.Key;
                }
                cumulativeChance += probability.Value;
            }

            return null;
        }
    }
}
```

OnlookerBee.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using PA_Lab5.NodeStructures;
using System.Linq;
using PA_Lab5.BeeColonies;

namespace PA_Lab5.Bees
{
    class OnlookerBee:IBee
    {
        private BeeColony _colony;
        private SubGraph _subGraph;

        public OnlookerBee(BeeColony colony)
        {
            _colony = colony;
        }

        public SubGraph GetLocation()
        {
```

```
            return _subGraph;
        }

        public void PickArea(int areaSize)
        {
            var sources = _colony.GetDiscoveredFoodSources().OrderBy(a => a.Item2).ToList();

            int cumulativeNectar = 0;
            int i = 0;
            foreach (var source in sources)
            {
                cumulativeNectar += source.Item2*(sources.Count-i);
                i++;
            }

            var subGraphsAndChances = new Dictionary<SubGraph, double>();
            i = 0;
            foreach (var source in sources)
            {
                subGraphsAndChances.Add(source.Item1, (double)source.Item2*(sources.Count -
i) / (double)cumulativeNectar);
            }

            var rand = new Random();

            var chance = rand.NextDouble();
            double cumulativeChance = 0;
            foreach (var probability in subGraphsAndChances)
            {
                if (chance > cumulativeChance && chance < (cumulativeChance +
probability.Value))
                {
                    _subGraph = probability.Key;
                    return;
                }
                cumulativeChance += probability.Value;
            }
        }

        public void DiscoverAdjacentArea(int areaSize)
        {
            if (_subGraph.GetDescendants().Count == 0) return;
            var subgraph =_subGraph.GetDescendants().First();

            _subGraph.PopDescendant(subgraph);
            _subGraph = subgraph;

            _subGraph.AddDescendants(_subGraph.AnalyzeAdjacentCommonNodes().ToList());
        }
    }
}
```

Graph.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;

namespace PA_Lab5.NodeStructures
{
    class Graph
    {
```

```
        List<Node> _nodes;

        public Graph(List<Node> nodes)
        {
            _nodes = nodes;

            ReInitNodes();
        }

        public void ReInitNodes()
        {
            Random rand = new Random();

            foreach (var node in _nodes)
            {
                node.SetAdjacents(new List<Node>());
            }

            foreach (var node in _nodes)
            {
                var limit = rand.Next(2, 31);
                if (node.GetAdjacents().Count + limit > 31) continue;
                var tempAdjacents = _nodes.Where(a =>
                a.Identificator != node.Identificator&&
                a.GetAdjacents().Count<31
                &&!a.GetAdjacents().Contains(node)).OrderBy(_ =>
rand.Next()).Take(limit).ToList();

                foreach(var a in tempAdjacents)
                {
                    node.AddAdjecent(a);
                    a.AddAdjecent(node);
                }
            }
        }
        public List<Node> GetNodes()
        {
            return _nodes;
        }
    }
}
```

Node.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace PA_Lab5.NodeStructures
{
    class Node
    {
        public int Identificator
        {
            get;
            private set;
        }
        private List<Node> _adjacetns;

        public Node(int identificator)
        {
            Identificator = identificator;
        }
```

```
        public void SetAdjacents(List<Node> adjacents)
        {
            _adjacetns = adjacents;
        }

        public List<Node> GetAdjacents()
        {
            return _adjacetns;
        }

        public void AddAdjecent(Node node)
        {
            _adjacetns.Add(node);
        }
    }
}
```

SubGraph.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Linq;

namespace PA_Lab5.NodeStructures
{
    class SubGraph
    {
        private List<Node> _members;
        private List<SubGraph> descendants= new List<SubGraph>();

        public int Common;
        public SubGraph()
        {
            _members = new List<Node>();
            Common = AnalyzeAdjacentCommonNodes().Count;
            AddDescendants(AnalyzeAdjacentCommonNodes().ToList());
        }

        public SubGraph(List<Node> members)
        {
            _members = members;
            Common = AnalyzeAdjacentCommonNodes().Count;
            AddDescendants(AnalyzeAdjacentCommonNodes().ToList());
        }
        public SubGraph(List<Node> members,SubGraph parent)
        {
            _members = members;
            Common= AnalyzeAdjacentCommonNodes().Count;
        }

        public bool IsClique()
        {
            foreach (var node in _members)
            {
                foreach (var cliqueMember in _members.Where(a => a != node))
                {
                    if (!cliqueMember.GetAdjacents().Contains(node))
                    {
                        return false;
                    }
                }
```

```
            }
            return true;
        }



        public List<Node> GetMembers()
        {
            return _members;
        }

        public int GetTotalAdjacentsCount()
        {
            List<Node> adjacent = new List<Node>();

            foreach(var node in _members)
            {
                adjacent.AddRange(node.GetAdjacents());
            }
            return adjacent.Distinct().ToList().Count;
        }
        public List<SubGraph> GetDescendants()
        {
            return descendants;
        }

        public void AddDescendants(List<Node> ddescendants)
        {
            foreach(var node in ddescendants)
            {
                List<Node> tempList = new List<Node>();
                foreach (var member in _members)
                {
                    tempList.Add(member);
                }

                tempList.Add(node);
                var tempGraph = new SubGraph(tempList, this);
                descendants.Add(tempGraph);
            }
        }

        public HashSet<Node> AnalyzeAdjacentCommonNodes()
        {
            HashSet<Node> adjacents = new HashSet<Node>();
            foreach (var a in GetMembers()[0].GetAdjacents())
            {
                adjacents.Add(a);
            }

            foreach (var b in GetMembers())
            {
                adjacents.IntersectWith(b.GetAdjacents());
            }
            return adjacents;
        }

        public void PopDescendant(SubGraph descendant)
        {
            descendants.Remove(descendant);
        }
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using PA_Lab5.Bees;
using PA_Lab5.BeeColonies;
using PA_Lab5.NodeStructures;
using System.Diagnostics;
using System.Linq;

namespace BetterOne
{
    class Program
    {
        static void Main(string[] args)
        {
            Graph graph = new Graph(GenerateNodes(200));

            Console.WriteLine("Welcome to bee algorithm for clique-searching");

            Console.WriteLine("Please, enter the bee number (for instance, 100)\n");

            int beeCount = -1;
            var tempValue = Console.ReadLine();

            while (!Int32.TryParse(tempValue, out beeCount) || beeCount < 10)
            {
                Console.WriteLine("Please, enter the correct value");
                tempValue = Console.ReadLine();
            }


            Console.WriteLine("Value accepted. Now enter clique size:\n(Warning, occurence
of clique sized more than 5 is unlikely)\n");

            int cliqueSize = -1;
            tempValue = Console.ReadLine();

            while (!Int32.TryParse(tempValue, out cliqueSize) || cliqueSize <1)
            {
                Console.WriteLine("Please, enter the correct value");
                tempValue = Console.ReadLine();
            }

            double dilyankyCoefficient = -1;

            Console.WriteLine("Now enter the portion of areas in the memory of colony to be
saved:");

            while (!Double.TryParse(tempValue, out
dilyankyCoefficient)||(dilyankyCoefficient <= 0 || dilyankyCoefficient > 1))
            {
                Console.WriteLine("Please, enter the correct value (for instance 0,4 ))\n");
                tempValue = Console.ReadLine();
            }

            BeeColony beeColony = new BeeColony(graph, cliqueSize,dilyankyCoefficient);

            var employedBees = GenerateEmployedBees((int)(beeCount*0.1), beeColony);
            var onlookerBees = GenerateOnlookerBees(beeCount - employedBees.Count,
beeColony);


            beeColony.SetBees(employedBees,onlookerBees);
```

```
            Console.WriteLine("Great! Everything is ready\nExecuting order 66....");

            beeColony.SetBees(employedBees, onlookerBees);

            Stopwatch sw = new Stopwatch();
            sw.Start();
            var clique = beeColony.FindClique();
            sw.Stop();
            var ts = sw.Elapsed;
            Console.WriteLine("Success; Time to find: {0:00}:{1:00}:{2:00}.{3:000}",
ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10);
            Console.WriteLine("Nodes and their adjacents:");
            var temp2 = clique.GetMembers();
            foreach (var member in temp2)
            {
                Console.WriteLine("Member identificator:{0}", member.Identificator);
                Console.Write("Adjacents");
                foreach (var a in member.GetAdjacents())
                {
                    Console.Write("{0} ", a.Identificator);
                }
                Console.WriteLine('\n');
            }


        }
        public static List<Node> GenerateNodes(int size)
        {
            var tempList = new List<Node>();
            for (int i = 0; i < size; i++)
            {
                tempList.Add(new Node(i));
            }
            return tempList;
        }

        public static List<EmployedBee> GenerateEmployedBees(int count, BeeColony colony)
        {
            var bees = new List<EmployedBee>();

            for (int i = 0; i < count; i++)
            {
                bees.Add(new EmployedBee(colony));
            }

            return bees;
        }

        public static List<OnlookerBee> GenerateOnlookerBees(int count, BeeColony colony)
        {
            var bees = new List<OnlookerBee>();

            for (int i = 0; i < count; i++)
            {
                bees.Add(new OnlookerBee(colony));
            }
            return bees;
        }
    }
}
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
Welcome to bee algorithm for clique-searching
Please, enter the bee number (for instance, 100)

100
Value accepted. Now enter clique size:
(Warning, occurence of clique sized more than 5 is unlikely)

5
Now enter the portion of areas in the memory of colony to be saved:
Please, enter the correct value (for instance 0,4 ))

0,4
Great! Everything is ready
Executing order 66....
Biggest clique in memory so far:2
Success; Time to find: 00:00:00.033
Nodes and their adjacents:
Member identificator:0
Adjacents: 146 97 100 148 47 92 108 90 134 107 79 198 174 120 139 30 10 13 21 28 45 56 78 140 175 184

Member identificator:78
Adjacents: 13 23 28 36 45 70 76 152 146 182 40 124 83 140 90 128 6 84 89 50 114 161 66 0 73 138 118 185 97 86 189

Member identificator:45
Adjacents: 5 13 25 39 68 111 114 78 98 149 73 0 32 134 159 88 72 87 97 144 150 160

Member identificator:97
Adjacents: 0 2 8 13 41 49 78 17 123 34 112 74 69 100 179 45 64 110 67 162 163 143 194 125 16 81 133 180 26 119 120

Member identificator:13
Adjacents: 7 47 174 89 186 69 116 78 190 45 29 31 0 15 137 97 108 70 12 139 27 40 93 104 114 123 192
```

Рисунок 3.1 – приклад роботи програми

```
Welcome to bee algorithm for clique-searching
Please, enter the bee number (for instance, 100)

100
Value accepted. Now enter clique size:
(Warning, occurence of clique sized more than 5 is unlikely)

5
Now enter the portion of areas in the memory of colony to be saved:
Please, enter the correct value (for instance 0,4 ))

0,4
Great! Everything is ready
Executing order 66....
Biggest clique in memory so far:3
Biggest clique in memory so far:3
Success; Time to find: 00:00:00.058
Nodes and their adjacents:
Member identificator:152
Adjacents: 7 13 26 27 32 60 61 64 76 120 128 132 155 82 77 65 66 84 168 160 166 187

Member identificator:77
Adjacents: 13 61 71 80 107 134 38 183 165 108 187 91 162 139 31 130 8 76 174 156 140 12 81 82 122 152 166 168 169

Member identificator:61
Adjacents: 6 18 32 39 54 19 81 121 56 80 163 62 77 49 152 93 60 180 168 47 130 166 177 181 185 187

Member identificator:187
Adjacents: 17 33 34 77 82 145 166 181 140 52 75 129 186 61 169 102 42 74 197 56 97 105 182 162 131 86 126 192 152 89

Member identificator:166
Adjacents: 26 38 66 74 133 145 146 153 163 130 171 114 187 152 139 77 57 150 61 67 117 184
```

Рисунок 3.2 – приклад роботи програми

3.3 Тестування алгоритму

Перш за все, необхідно встановити, що кількість ділянок, що підлягають запам’ятовуванню найліпше не є константною, а мусить бути скоригована відповідно наявної кількості ділянок. Тому буде запропоновано поняття коефіцієнту запам’ятовування- частки ділянок, що будуть запам’ятовуватися вуликом після кожної ітерації.

Спочатку встановимо найоптимальнішу кількість бджіл. Нехай коефіцієнт запам’ятовування становить 0,5- половина. Тоді при використанні різних значень числа бджіл алгоритм буде працювати наведені числа часу:

Табл. 1 час необхідний для знаходження кліки розміром 5 для значень від 100 до 800

	Час виконання, мс							
Номер випроб.	BeeCount=100	BeeCount=200	BeeCount=300	BeeCount=400	BeeCount=500	BeeCount=600	BeeCount=700	BeeCount=
1	441	1212	1042	2088	3007	inf	6095	6045
2	160	inf	1019	inf	5001	5039	4019	7094
3	417	891	2055	2017	2099	inf	6009	inf
4	322	138	2029	3098	4075	3057	7020	7058
5	745	1081	1074	2069	3089	3086	5012	6030
6	inf	inf	inf	1090	inf	3081	inf	inf
7	416	1482	1095	1082	3048	3022	7008	10084
8	inf	1095	1017	2031	inf	4086	inf	inf
9	791	721	1062	2029	2081	5062	5084	6003
10	660	1081	1082	2023	inf	5005	4058	4097
Середнє значення	494	962,625	1275	1947,444444	3200	3929,75	5538,125	6630,142
Число незнайдених рішень	2	2	1	1	3	2	2	3

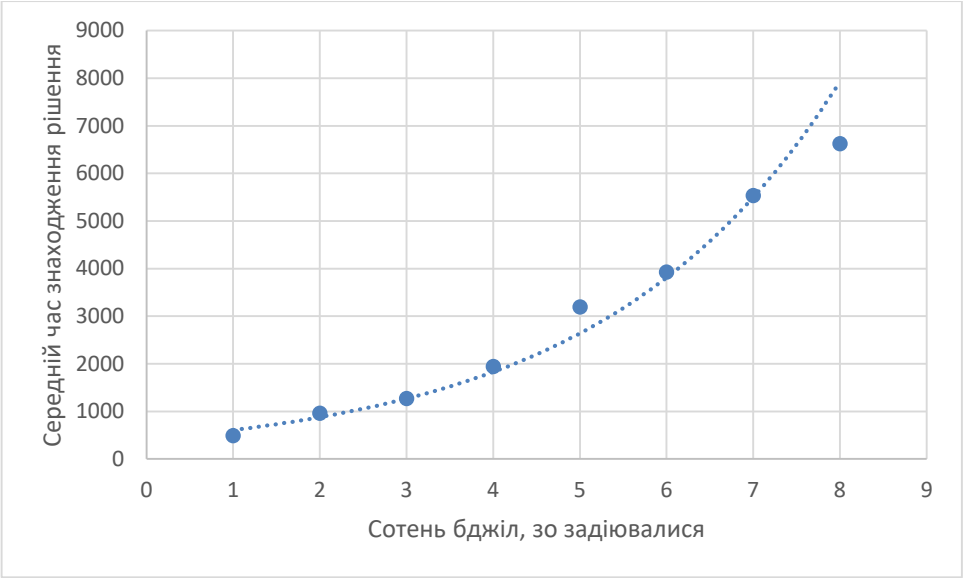


Рисунок 3.3 – залежність середнього часу знаходження розв’язку від числа бджіл

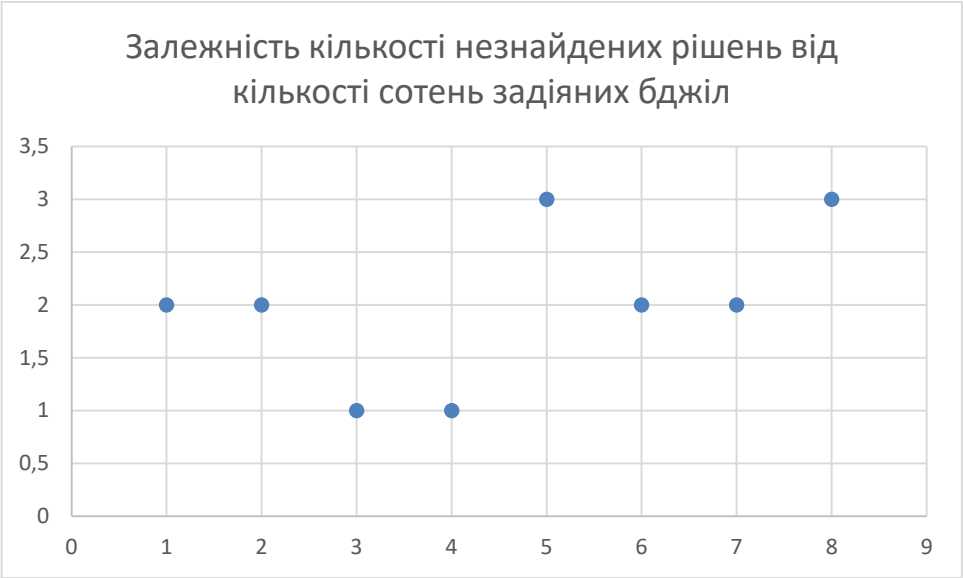


Рисунок 3.4 – залежність числа незнайдених розв’язків від числа бджіл

Як бачимо, час необхідний для знаходження розв’язку експоненційно корелює з числом бджіл, починаючи зі 100 шт.

Кореляція числа незнайдених розв’язків із числом бджіл, як показує графік, відсутня

Дослідімо числа популяції бджіл менше 100:

Табл. 3.2- час необхідний для знаходження кліки розміром 5 для значень від 80 до 100

	Час виконання, мс		
Номер випробування	BeeCount=80	BeeCount=90	BeeCount=100
1	1022	85	441
2	5039	1086	160
3	96	1059	417
4	inf	69	322
5	1045	inf	745
6	70	76	inf
7	1075	1013	416
8	74	inf	inf
9	98	22	791
10	2000	inf	660
Середнє значення для знайдених	1343	555,6667	494
Число незнайдених рішень	1	3	2

Де inf означає, що розв’язок не було знайдено за більш як 10 хвилин

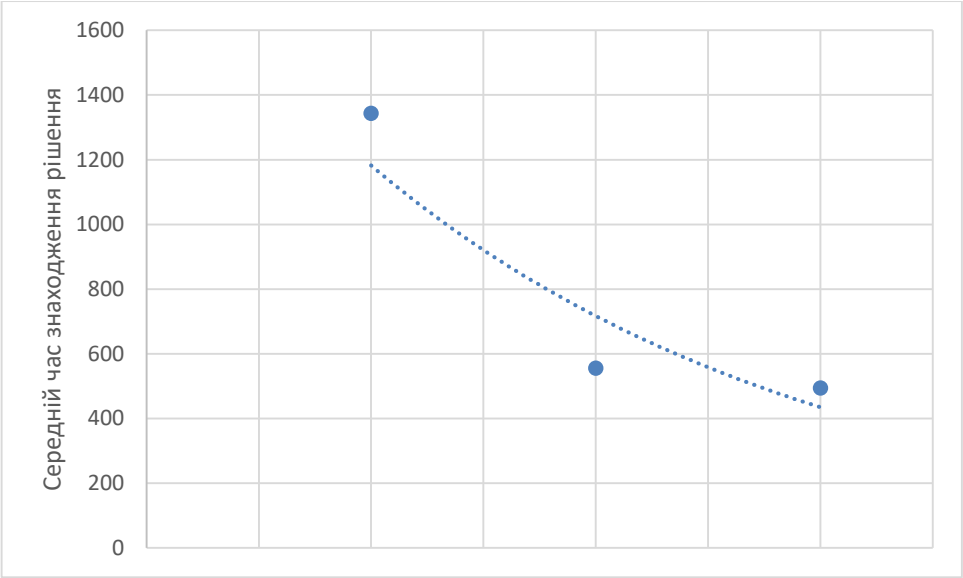


Рисунок 3.5 – залежність середнього часу знаходження розв’язку від числа бджіл

Як бачимо, найменше середнє значення часу знаходження розв’язку спостерігається при популяції бджіл в 100 одиниць.

Тепер проаналізуймо аналогічним чином коефіцієнт запам’ятовування при популяції бджіл у 100 одиниць:

табл. 3.3 – залежність часу знаходження кліки розміром 5 для різних коефіцієнтів запам’ятовування

	Час виконання, мс				
Номер випробування	0,2	0,3	0,4	0,5	0,6
1	4067	76	52	116	108
2	2015	inf	64	178	192
3	10018	89	48	65	inf
4	415	76	inf	114	167
5	inf	inf	78	inf	67
6	inf	35	87	94	24
7	154	inf	67	231	inf
8	189	92	inf	inf	117
9	201	inf	117	43	274
10	inf	230	78	117	174
Середнє значення для знайдених	594,8	117,75	72	119,75	140,375
Число незнайдених рішень	3	4	2	2	2

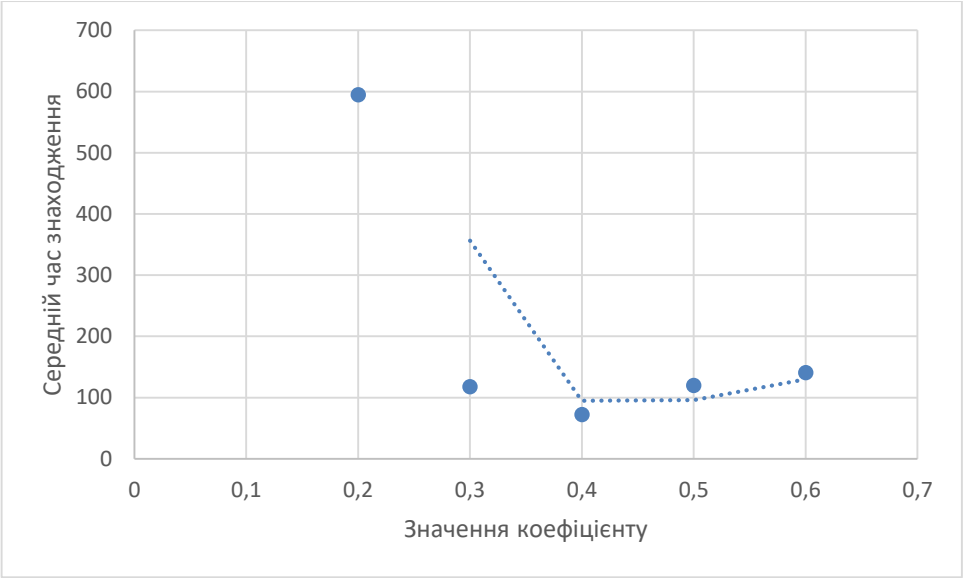


Рисунок 3.6 – залежність часу знаходження розв’язку від коефіцієнту запам’ятовування

Як бачимо, наймеший час спостерігається за коефіцієнта в 0,4

ВИСНОВОК

Отож, у ході виконання лабораторної роботи було проведено поглиблене дослідження метаевристичного пробаблістичного алгоритму бджолиної колонії. Було проаналізовано його особилвості та атрибутику й спроектовано на завдання пошуку кліки в ненаправленому графі, реалізувавши його компільованою мовою C#. Було проведено проведено аналіз його роботи при модифікації різноманітних показників, зокрема числа бджіл та кількості розв’язків у пам’яті бджолиної колонії. Проводилася поступова фіксація одного показника та зміна іншого з метою встановлення найбільш продуктивних значень цих показників. Зокрема практичним чином було встановлено, що найоптимальнішим числом популяції бджіл для даної реалізації алгоритму є 100, а найкращою часткою запам’ятовуваних розв’язків після кожної ітерації є 0,4.

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%.