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 Макарчук Лідія Олександрівна* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов Олексій Олександрович* _____
(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.

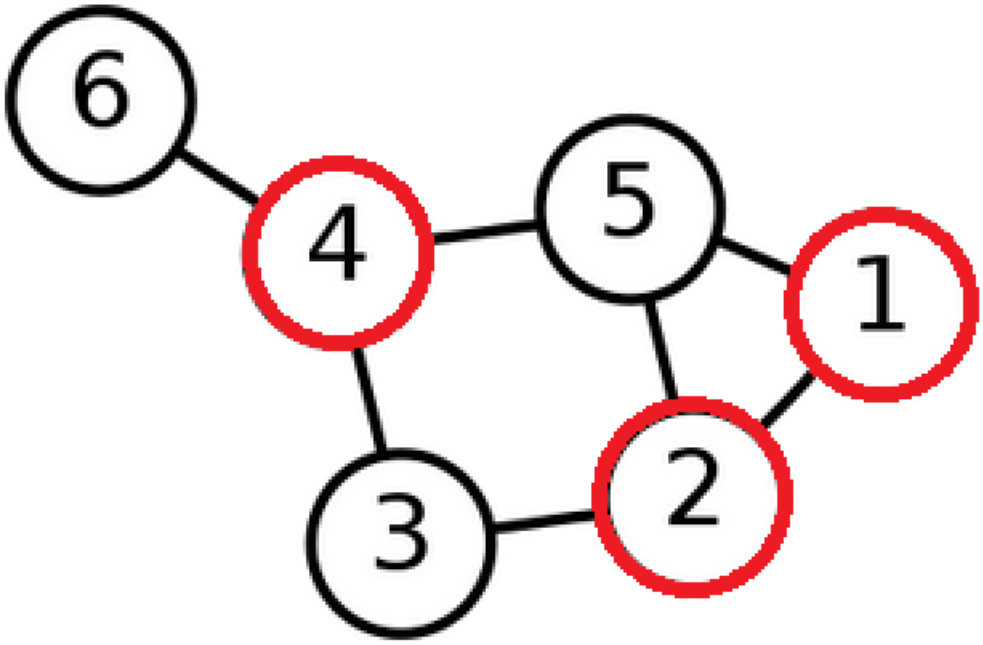
Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

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)– називаютьтаке 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

Варіант 16

№	**Задачі і алгоритми**
16	Задача про кліку (задача розпізнавання) + Генетичний алгоритм

3.1 Покроковий алгоритм

GA(G, k)

1. ПОЧАТОК
2. Створити початкову популяцію currPopulation
   2.1. Для кожної вершини у G створити Creature, що містить лише цю вершину та додати до currPopulation
   2.2. Для кожного Creature знайти за допомогою CliqueExtraction()
   2.3. Обчислити F як кількість вершин у maxClique.
3. bestCreature = currPopulation[0]
4. maxClique = currPopulation[0].clique
5. Вибрати terminationCondition (кількість ітерацій або stagnancy(коли найкращий результат не змінюється протягом певної кількості поколінь)
6. ПОКИ !hasAClique AND currIterationNumberOrStagnancy < terminationCondition
   6.1. parent1, parent2 = Selection(найкращий Creature та рандомний Creature)
   6.2. Crossover
      6.2.1. ЯКЩО обрано Crossover5 ТО виконати 5-точкове схрещування
      6.2.2. ЯКЩО обрано Crossover2 ТО виконати 2-точкове схрещування
      6.2.3. ЯКЩО обрано CrossoverDynamic ТО
         6.2.3.1. виконати m-точкове схрещування, де параметр $m \in [2, 10]$ має початкове значення 10, а через кожні 20 покоління зменшується на 1.
   6.3. Цикл для кожного child у списку нащадків children∈{child1, child2}

6.3.1. Mutation

6.3.1.1. ЯКЩО обрано changeMutation ТО з імовірністю p змінити випадковий ген на протилежний

6.3.1.2. ЯКЩО обрано exchangeMutation ТО з імовірністю p поміняти 2 випадкові гени місцями

6.3.2. cliqueOfChild = child.CliqueExtraction()

6.3.3. Local improvement

6.3.3.1. ЯКЩО обрано AddVerticesFromAdjList ТО

6.3.3.1.1. запустити child.ImprovFromAdjList()

6.3.3.2. ЯКЩО обрано AddVerticesToCliqueRandom ТО

6.3.3.2.1. запустити child.ImproveClique(рандомна позиція, chromosome.length)

6.3.3.2.2. запустити child.ImproveClique(0, рандомна позиція)

6.3.4. Оновити популяцію

6.3.4.1.1. ЯКЩО child.F() > currPopulation.WorstCreature.F() ТО

6.3.4.1.1.1. ЯКЩО child.F() > bestCreature.F() ТО bestCreature = child

6.3.4.1.1.2. Видалити currPopulation.WorstCreature

6.3.4.1.1.3. Додати child

6.4. Оновити currIterationNumberOrStagnancy

6.5. ЯКЩО bestCreature.F() >= k ТО hasAClique = true

7. ПОВЕРНУТИ hasAClique та bestCreature.maxClique
8. КІНЕЦЬ

функція CliqueExtraction()

1. ПОЧАТОК
2. maxClique = всі вершини з child.chromosome
3. ПОКИ maxClique не є клікою (кожна вершина з’єднана з усіма іншими) ПОВТОРИТИ

3.1. Створити список вершин candidatesForDeleting з найменшими динамічними індексами (враховуються лише зв’язки між вершинами у child.chromosome)

3.2. Видалити рандомну вершину, що входить до candidatesForDeleting із maxClique

3.3. Оновити список суміжних вершин для кожної вершини, що була пов’язана із видаленою

4. Повернути maxClique
5. КІНЕЦЬ

процедура ImproveClique(start, end)

1. ПОЧАТОК
2. ЦИКЛ для і від start до end ПОВТОРИТИ

2.1. ЯКЩО chromosome[i]=0 AND вершина chromosome[i] з’єднана з усіма вершинами maxClique, ТО додати chromosome[i] до maxClique

3. КІНЕЦЬ

процедура ImprovFromAdjList()

1. ПОЧАТОК
2. vertices = ∅
3. Додати до vertices всі вершини, які є суміжними для першої вершини у кліці та яких немає у хромосомі (chromosome[i]=false);
4. ЦИКЛ проходу по вершинах у кліці maxClique[i], i =2,…, maxClique.lenght:

4.1. ЦИКЛ проходу по всіх вершинах vertices[j], j=1,…vertices.lenght

4.1.1. ЯКЩО vertices[j] немає у списку суміжності вершини maxClique[i] ТО видалити vertices[j] зі списку vertices

5. ЯКЩО vertices.lenght > 0 ТО додати першу вершину vertices[0] до maxClique
6. КІНЕЦЬ

функція F()

7. ПОЧАТОК
8. Повернути суму динамічних степенів вершин (враховувати лише зв’язки з тими вершинами, що наявні у хромосомі)
9. КІНЕЦЬ

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

//Модуль GeneticAlgorithm

```
using laba5.GraphModel;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace laba5.GA
{
   public class GeneticAlgorithm
   {
      public enum CrossoverMethod
      {
         TwoPoints,
         FivePoints,
         Dynamic
      }
      public enum MutationMethod
      {
         ChangeToOpposite,
         Exchange
      }
      public enum LocalImprMethod
      {
         AddVerticesToCliqueStraight,
         AddVerticesToCliqueRandom,
         AddVerticesFromAdjList
      }
      public enum TerminationCondition
      {
         Iterations,
         Stagnancy,
         FullGraph
      }
      int v;
      private SortedList<int, Creature> _currPopulation;
      private Creature bestCreature;
      private CrossoverMethod crossMethod;
      private MutationMethod mutMethod;
      private LocalImprMethod imprMethod;
      private TerminationCondition terminationCondition;
      private int iterations;
      private int currPointNumber;//for dynamic crossover
```

```
        private double mutationPossibility;
        private int terminationNumber;
        private bool hasAClique;

        public bool HasAClique => hasAClique;
        public Creature BestCreature => bestCreature;
        public int Iterations => iterations;

        public GeneticAlgorithm(Graph graph, int crossMethod, int mutMethod, double mutationPossibl, int imprMethod, int
termCondition, int terminationNumber)
        {
            Creature.adjList = graph.adjList;
            v = graph.adjList.Length;
            CreateInitialPopulation();
            bestCreature = _currPopulation.Last().Value;
            this.crossMethod = (CrossoverMethod)crossMethod;
            this.mutMethod = (MutationMethod)mutMethod;
            this.imprMethod = (LocalImprMethod)imprMethod;
            this.terminationCondition = (TerminationCondition)termCondition;
            iterations = 0;
            if (v > 50)
                currPointNumber = 10;
            else if (v > 18)
                currPointNumber = 5;
            else
                currPointNumber = 3;
            mutationPossibility = mutationPossibl;
            this.terminationNumber = terminationNumber;
        }
        private void CreateInitialPopulation()
        {
            _currPopulation = new SortedList<int, Creature>(new DuplicateKeyComparer<int>());
            bool[] chromosome;
            for (int i=0; i<v; i++)
            {
                chromosome = new bool[v];
                chromosome[i] = true;
                //for (int j = 0; j < Creature.adjList[i].Count; j++)
                //    chromosome[Creature.adjList[i][j]] = true;
                Creature creature = new Creature(chromosome);
                creature.ExtractCliqueAndSetF();
                _currPopulation.Add(creature.F, creature);
            }
        }
        public void Start(int k)
        {
            iterations = 0;
            hasAClique = false;
            int lastBestF;

            int currIterationNumberOrStagnancy = 0;//iteration number or stagnancy
                    while (!hasAClique && currIterationNumberOrStagnancy < terminationNumber)

    {
                lastBestF = bestCreature.F;//for stagnancy condition

                Creature parent1, parent2;
                Creature child1, child2;
                Selection(out parent1, out parent2);
                Crossover(parent1, parent2, out child1, out child2);
                Mutation(ref child1);
                Mutation(ref child2);
                child1.ExtractCliqueAndSetF();
                child2.ExtractCliqueAndSetF() ;
                LocalImprovement(child1);
                LocalImprovement(child2);
```

```
            AddChildToPopulation(child1);
            AddChildToPopulation(child2);

            if (terminationCondition == TerminationCondition.Stagnancy)
            {
                if (lastBestF != bestCreature.F)
                    currIterationNumberOrStagnancy = -1;
                currIterationNumberOrStagnancy++;
            }
            else if(terminationCondition== TerminationCondition.Iterations)
                currIterationNumberOrStagnancy++;
            else
            {
                if (child1.IsCompleteGraph() || child2.IsCompleteGraph())
                    currIterationNumberOrStagnancy = terminationNumber;
            }

            if (bestCreature.CliqueSize >= k)
                hasAClique = true;

            iterations++;
        }
    }
    private void Selection(out Creature parent1, out Creature parent2)
    {
        Random rnd = new Random();
        parent1 = bestCreature;
        do
        {
            parent2 = _currPopulation.ElementAt(rnd.Next(0, _currPopulation.Count)).Value;
        } while (parent1 == parent2);
    }
    private void Crossover(Creature parent1, Creature parent2, out Creature child1, out Creature child2)
    {
        switch (crossMethod)
        {
            case CrossoverMethod.TwoPoints:
                C_kPoints(parent1, parent2, out child1, out child2, 2);
                break;
            case CrossoverMethod.FivePoints:
                C_kPoints(parent1, parent2, out child1, out child2, 5);
                break;
            case CrossoverMethod.Dynamic:
                C_Dynamic(parent1, parent2, out child1, out child2);
                break;
            default:
                C_Dynamic(parent1, parent2, out child1, out child2);
                break;
        }
    }
    private void C_kPoints(Creature parent1, Creature parent2, out Creature child1, out Creature child2, int pointNumber)
    {
        bool[] childChromosome1 = new bool[v];
        bool[] childChromosome2 = new bool[v];
        List<int> indices = GenerateDifferentNumbers(0, v, pointNumber);
        int start = 0, genNum;
        int i = 0;
        do
        {
            genNum = indices[i] - start;
            if (i%2 == 0)
            {
                Array.Copy(parent1.Chromosome, start, childChromosome1, start, genNum);
                Array.Copy(parent2.Chromosome, start, childChromosome2, start, genNum);
            }
            else
```

```
        {
            Array.Copy(parent2.Chromosome, start, childChromosome1, start, genNum);
            Array.Copy(parent1.Chromosome, start, childChromosome2, start, genNum);
        }
        start = indices[i];
        i++;
    }
    while (i < pointNumber);
    //copy the rest
    genNum = v - start;
    if (i % 2 == 0)
    {
        Array.Copy(parent1.Chromosome, start, childChromosome1, start, genNum);
        Array.Copy(parent2.Chromosome, start, childChromosome2, start, genNum);
    }
    else
    {
        Array.Copy(parent2.Chromosome, start, childChromosome1, start, genNum);
        Array.Copy(parent1.Chromosome, start, childChromosome2, start, genNum);
    }

    child1 = new Creature(childChromosome1);
    child2 = new Creature(childChromosome2);
}
private void C_Dynamic(Creature parent1, Creature parent2, out Creature child1, out Creature child2)
{
    C_kPoints(parent1, parent2, out child1, out child2, currPointNumber);
    if (currPointNumber>2 && iterations%20 == 0)
        currPointNumber--;
}
private static List<int> GenerateDifferentNumbers(int start,int end, int number)
{
    List<int> numbers = new List<int>();
    Random rnd = new Random();
    int index;
    while (numbers.Count < number)
    {
        index = rnd.Next(start, end);
        if (!numbers.Contains(index))
            numbers.Add(index);
    }
    numbers.Sort();
    return numbers;
}
private void Mutation(ref Creature child)
{
    switch (mutMethod)
    {
        case MutationMethod.ChangeToOpposite:
            M_ChangeToOpposite(child);
            break;
        case MutationMethod.Exchange:
            M_Exchange(child);
            break;
        default: throw new NotImplementedException();
    }
}
private void M_ChangeToOpposite(Creature child)
{
    Random rnd = new Random();
    if (rnd.NextDouble() <= mutationPossibility)
    {
        int gen;
        gen = rnd.Next(0, v);
        child.Chromosome[gen] = !child.Chromosome[gen];
    }
```

```
        }
        private void M_Exchange(Creature child)
        {
            Random rnd = new Random();
            if (rnd.NextDouble() <= mutationPossibility)
            {
                int gen1, gen2;
                gen1 = rnd.Next(0, v);
                do
                {
                    gen2 = rnd.Next(0, v);
                } while (gen1 == gen2);
                bool temp = child.Chromosome[gen1];
                child.Chromosome[gen1] = child.Chromosome[gen2];
                child.Chromosome[gen2] = temp;
            }
        }
        private void LocalImprovement(Creature child)
        {
            switch (imprMethod)
            {
                case LocalImprMethod.AddVerticesToCliqueStraight:
                    LI_Straight(child); break;
                case LocalImprMethod.AddVerticesToCliqueRandom:
                    LI_Random(child); break;
                case LocalImprMethod.AddVerticesFromAdjList:
                    LI_Adj(child); break;
                default: break;
            }
        }
        private void LI_Straight(Creature child)
        {
            child.ImproveClique();
        }
        private void LI_Random(Creature child)
        {
            Random rnd = new Random();
            int number = rnd.Next(0, v);
            child.ImproveClique(number, v);
            child.ImproveClique(0, number + 1);
        }
        private void LI_Adj(Creature child)
        {
            child.AddVerticesFromAdjList();
        }
        private void AddChildToPopulation(Creature child)   //add child and remove the worst
        {
            if (bestCreature.F < child.F)
                bestCreature = child;
            if (child.F >= _currPopulation.ElementAt(0).Value.F)
            {
                _currPopulation.Add(child.F, child);
                _currPopulation.RemoveAt(0);
            }
        }
    }
}
```

//Модуль Creature

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace laba5.GA
{
    public class Creature
    {
        public static List<int>[] adjList;
        private bool[] _chromosome;
        private int _cliqeSize;
        private Dictionary<int, List<int>> maxClique;
        private int _F;//dynamicDegree
        public int CliqueSize
        {
            get { return _cliqeSize; }
            set { _cliqeSize = value; }
        }
        public int F => _F;
        public bool[] Chromosome => _chromosome;
        public Creature(bool[] chromosome)
        {
            _chromosome = new bool[chromosome.Length];
            Array.Copy(chromosome, _chromosome,chromosome.Length);
            CalcF();
        }
        public void ExtractCliqueAndSetF()
        {
            ExtractMaxClique();
            _cliqeSize = maxClique.Count;
        }
        private void CalcF()
        {
            _F = 0;
            Dictionary<int, List<int>> tempDictio = FromChromosomeToVertices();
            foreach (int key in tempDictio.Keys)
            {
                _F += tempDictio[key].Count;
            }
        }
        private Dictionary<int, List<int>> FromChromosomeToVertices()
        {
            Dictionary<int, List<int>> vertices = new Dictionary<int, List<int>>();
            // add vertices from chromosome to dictionary
            for (int i = 0; i < _chromosome.Length; i++)
            {
                if (_chromosome[i])
                    vertices[i] = new List<int>();
            }
            //connect vertices
            foreach (int i in vertices.Keys)
            {
                for (int j = 0; j < adjList[i].Count; j++)
                {
                    int index = adjList[i][j];
                    if (vertices.ContainsKey(index) && !vertices[i].Contains(index) && i != index)
                    {
                        vertices[index].Add(i);
                        vertices[i].Add(index);
                    }
                }
            }
            return vertices;
        }
        private void ExtractMaxClique()
        {
            maxClique = FromChromosomeToVertices();
            Random rnd= new Random();
            while (!IsClique())
            {
```

```
            List<int> candidatesForDeleting = new List<int>();
            int minDegree = int.MaxValue;
            //forming a candidatesForDeleting list
            foreach (int i in maxClique.Keys)
            {
                if (maxClique[i].Count < minDegree)
                {
                    candidatesForDeleting.Clear();
                    candidatesForDeleting.Add(i);
                    minDegree = maxClique[i].Count;
                }
                else if (maxClique[i].Count == minDegree)
                    candidatesForDeleting.Add(i);
            }
            int index = candidatesForDeleting[rnd.Next(candidatesForDeleting.Count)];//get vertex index to remove
            for (int i=0; i < maxClique[index].Count; i++)
            {
                int connectedVertex = maxClique[index][i];
                maxClique[connectedVertex].Remove(index);
            }
            maxClique.Remove(index);
        }
                }
private bool IsClique()
    {
        foreach (int i in maxClique.Keys)
        {
            foreach (int j in maxClique.Keys)
            {
                if (i == j) continue;
                if (!maxClique[j].Contains(i))
                    return false;
            }
        }
        return true;
    }
    public void ImproveClique(int start = 0, int number = -1)
    {
        if (number == -1)
            number = _chromosome.Length;
        for(int i=start;i < number;i++)
        {
            if (!_chromosome[i])
            {
                bool sholdAdd = true;
                foreach (int key in maxClique.Keys)
                {
                    if (!Creature.adjList[i].Contains(key))
                    {
                        sholdAdd= false;
                        break;
                    }
                }
                if (sholdAdd)
                    AddVertex(i);
            }
        }
    }
    private void AddVertex(int i)
    {
        foreach (int key in maxClique.Keys)
            maxClique[key].Add(i);
        List<int> adjListForI = new List<int>(maxClique.Keys);
        maxClique[i] = adjListForI;
        _chromosome[i] = true;
```

```
            _cliqeSize++;
            _F += maxClique.Count*2;
        }
        public void AddVerticesFromAdjList()
        {
            //create candidates to add to clique as vertices that are common for all vertices in the click but are not added yet
            if (maxClique.Count > 1)
            {
                //start with first vertex
                int firstVertexKey = maxClique.Keys.First();
                List<int> commonVerticesList = new List<int>();
                for (int i = 0; i < Creature.adjList[firstVertexKey].Count; i++)
                {
                    if (!_chromosome[Creature.adjList[firstVertexKey][i]])
                        commonVerticesList.Add(Creature.adjList[firstVertexKey][i]);
                }
                foreach (int key in maxClique.Keys)
                {
                    for (int j = 0; j < commonVerticesList.Count; j++)
                    {
                        if (!Creature.adjList[key].Contains(commonVerticesList[j]))
                            commonVerticesList.Remove(commonVerticesList[j]);
                    }
                    if (commonVerticesList.Count == 0) break;
                }
                if (commonVerticesList.Count > 0)
                    AddVertex(commonVerticesList[0]);
            }
            else
                ImproveClique();
        }
        public void DisplayMaxClique()
        {
            foreach (int key in maxClique.Keys)
            {
                Console.Write(Convert.ToString(key).PadLeft(3) + ":");
                for (int i = 0; i < maxClique[key].Count; i++)
                    Console.Write(" "+maxClique[key][i]);
                Console.WriteLine();
            }
        }
    }
}
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

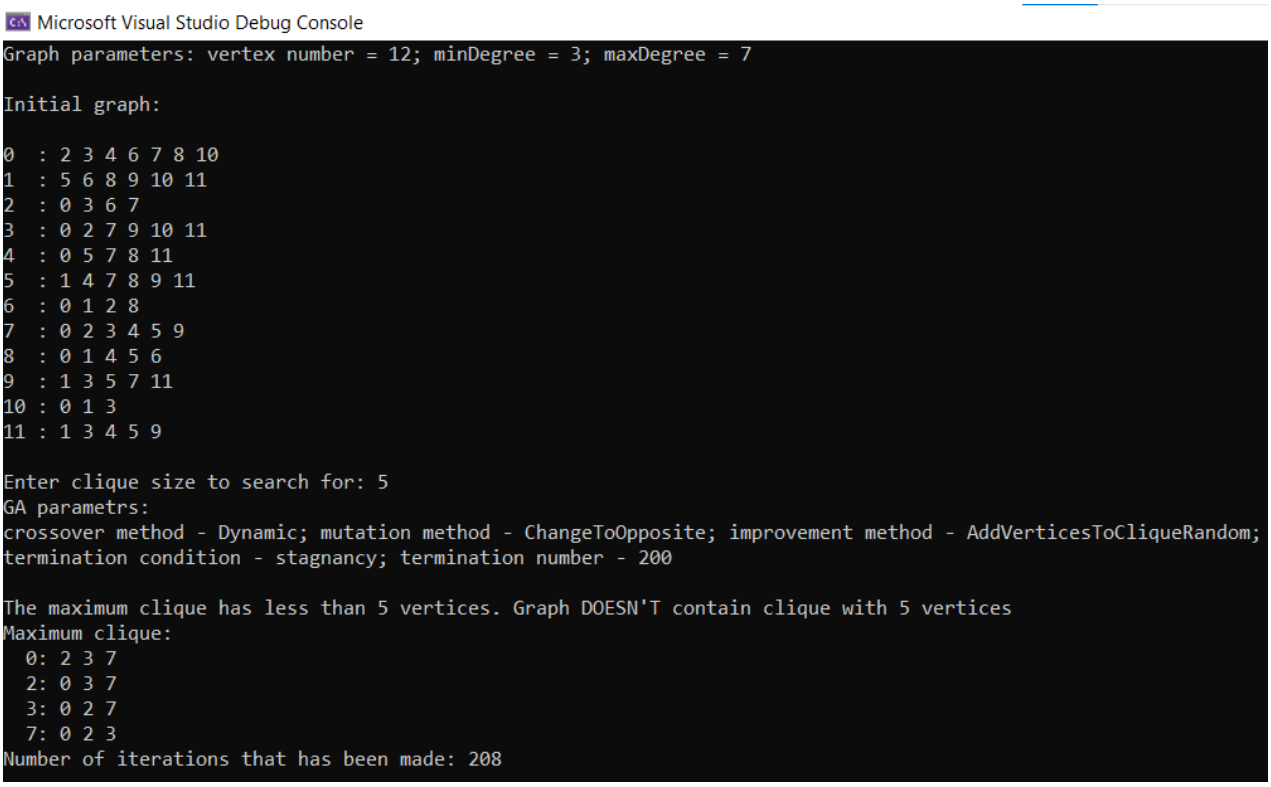


Рисунок 3.1 – Пошук кліки розміру 5 у графі з 2-ма вершинами, мінімальним степенем вершини 3 та максимальним 7

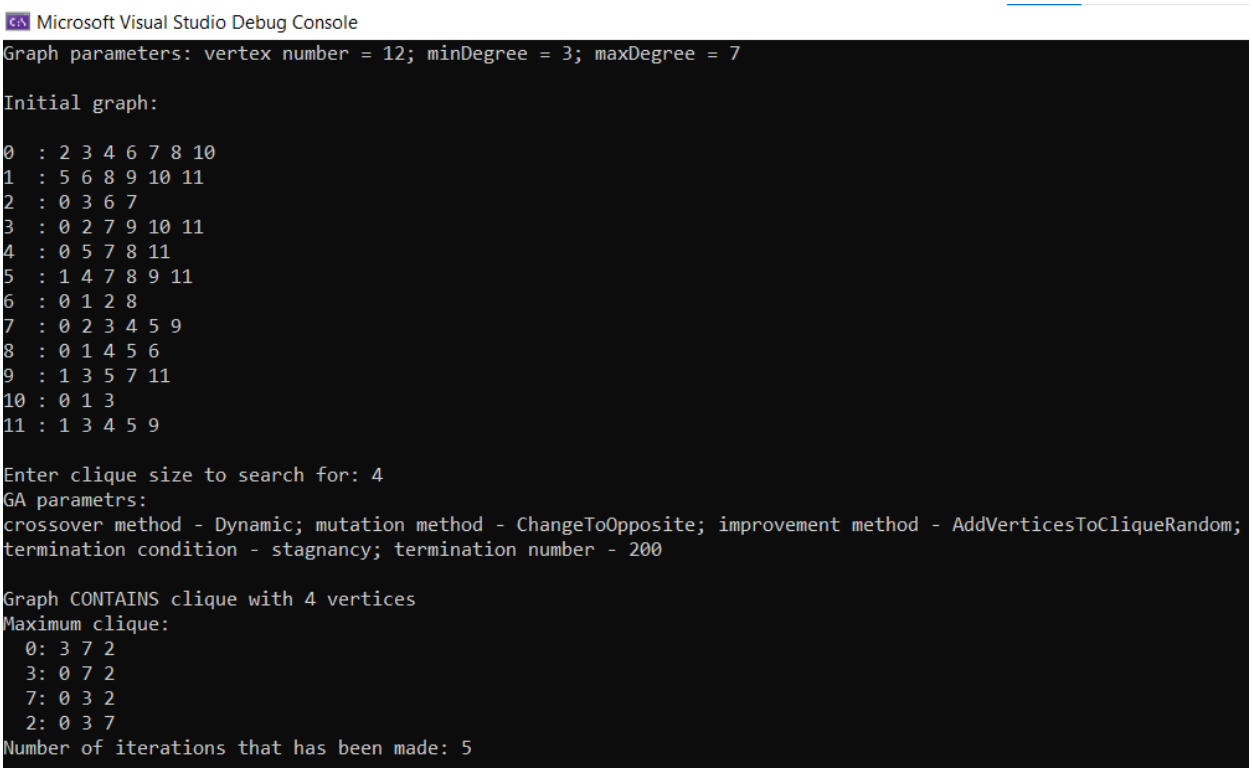


Рисунок 3.2 – Пошук кліки розміру 4 у графі з 2-ма вершинами, мінімальним степенем вершини 3 та максимальним 7

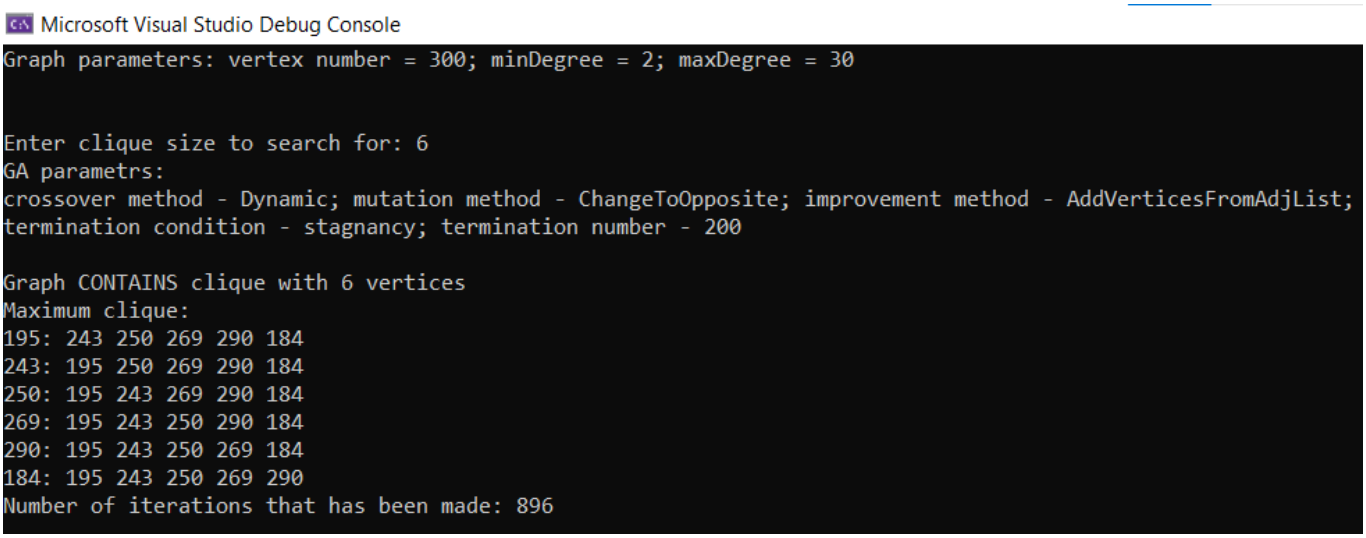


Рисунок 3.3 – Пошук кліки розміру 6 на графі з 300 вершинами, мінімальні степені для яких 2, а максимальні - 30

3.3 Тестування алгоритму

Для тестування створимо 10 графів, на яких і будемо перевіряти роботу алгоритмів. Зробимо усі можливі комбінування параметрів алгоритму: параметр схрещування + параметр мутації + параметр локального покращення. У відповідні таблиці занесемо кількість зроблених ітерацій для знаходження кліки.

Критерій зупинки: досягнення значення stagnancy = 300 (коли цільова функція кращого нащадка не змінюється протягом 300 поколінь).

Вхідні дані: 300 вершин, степені вершин від 2 до 30, пошук кліки розміром 5, генетичний алгоритм (варіанти параметрів):

- Схрещування
  - Двоточкове
  - П’ятиточкове
  - Динамічне
- Мутація
  - Зміна гена на протилежний
  - Зміна генів місцями
- Локальне покращення
  - Додавання вершин з хромосоми (перевірка чи вершини, які не включені у хромосому утворюють кліку з вершинами, що вже у ній)
  - Додавання вершин з списку суміжності (пошук вершин, що ще не були додані до кліки, проте є у списках суміжності кожної вершини кліки)

Таблиця 1 – Двоточкове схрещування + мутація зі зміною гена на протилежний + локальне покращення: додавання вершин з хромосоми

Номер графа	Кількість ітерацій	Знайдено кліку
1	97	+
2	329	-

3	5	+
4	6732	+
5	301	-
6	11846	-
7	23	+
8	68	+
9	75	+
10	2707	+

Рисунок 3.4 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 1

Таблиця 2 – П’ятиточкове схрещування + мутація зі зміною гена на протилежний + локальне покращення: додавання вершин з хромосоми

Номер графа	Кількість ітерацій	Знайдено кліку
1	2563	+
2	1181	+
3	1521	+
4	178	+
5	3589	+
6	9132	-
7	1	+
8	11	+
9	1140	+
10	7709	+

Рисунок 3.5 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 2

Таблиця 3 – Динамічне схрещування + мутація зі зміною гена на протилежний + локальне покращення: додавання вершин з хромосоми

Номер графа	Кількість ітерацій	Знайдено кліку
1	953	-
2	418	+
3	14	+
4	461	-
5	77	+
6	710	-
7	4543	+
8	141	+
9	79	+
10	6	+

Рисунок 3.6 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 3

Таблиця 4 – Двоточкове схрещування + мутація з обміном генів місцями + локальне покращення: додавання вершин з хромосоми

Номер графа	Кількість ітерацій	Знайдено кліку
1	472	-
2	933	-
3	159	+
4	812	-
5	28	+
6	558	-
7	41	+
8	148	+
9	930	-
10	799	-

Рисунок 3.7 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 4

Таблиця 5 – П’ятиточкове схрещування + мутація з обміном генів місцями + локальне покращення: додавання вершин з хромосоми

Номер графа	Кількість ітерацій	Знайдено кліку
1	933	+
2	586	-
3	848	-
4	820	-
5	96	+
6	462	-
7	17	+
8	1	+
9	33	+
10	84	+

Рисунок 3.8 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 5

Таблиця 6 – Динамічне схрещування + мутація з обміном генів місцями + локальне покращення: додавання вершин з хромосоми

Номер графа	Кількість ітерацій	Знайдено кліку
1	16	+
2	19	+
3	737	-
4	786	-
5	57	+
6	760	-
7	22	+
8	10	+
9	36	+
10	549	-

Рисунок 3.9 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 6

Таблиця 7 – Двоточкове схрещування + мутація зі зміною гена на протилежний + локальне покращення: додавання вершин з списку суміжності

Номер графа	Кількість ітерацій	Знайдено кліку
1	1068	+
2	594	+
3	4	+
4	11	+
5	747	+
6	140	+
7	705	+
8	975	+
9	17	+
10	103	+

Рисунок 3.10 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 7

Таблиця 8 – П’ятиточкове схрещування + мутація зі зміною гена на протилежний + локальне покращення: додавання вершин з списку суміжності

Номер графа	Кількість ітерацій	Знайдено кліку
1	942	+
2	811	+
3	14	+
4	1090	+
5	9	+
6	108	+
7	815	+
8	1092	-
9	404	+
10	444	+

Рисунок 3.11 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 8

Таблиця 9 – Динамічне схрещування + мутація зі зміною гена на протилежний + локальне покращення: додавання вершин з списку суміжності

Номер графа	Кількість ітерацій	Знайдено кліку
1	569	+
2	180	+
3	12	+
4	607	+
5	621	+
6	137	+
7	1110	+
8	6	+
9	16	+
10	191	+

Рисунок 3.12 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 9

Таблиця 10 – Двоточкове схрещування + мутація з обміном генів місцями + локальне покращення: додавання вершин з списку суміжності

Номер графа	Кількість ітерацій	Знайдено кліку
1	1094	+
2	573	+
3	747	-
4	794	-
5	721	+
6	50	+
7	622	+
8	530	-
9	372	+
10	479	+

Рисунок 3.13 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 10

Таблиця 11 – П’ятиточкове схрещування + мутація з обміном генів місцями + локальне покращення: додавання вершин з списку суміжності

Номер графа	Кількість ітерацій	Знайдено кліку
1	1298	+
2	806	+
3	1030	+
4	759	+
5	920	+
6	96	+
7	5	+
8	1117	-
9	42	+
10	12	+

Рисунок 3.14 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 11

Таблиця 12 – Динамічне схрещування + мутація з обміном генів місцями + локальне покращення: додавання вершин з списку суміжності

Номер графа	Кількість ітерацій	Знайдено кліку
1	1171	+
2	179	+
3	338	+
4	5	+
5	326	+
6	16	+
7	330	+
8	996	+
9	481	-
10	549	-

Рисунок 3.15 – Графік залежності кількості ітерацій від конкретного графу для алгоритму 12

На рисунку 3.1 порівняно кількість ітерацій, зроблена кожним алгоритмом для кожного з 10 графів.

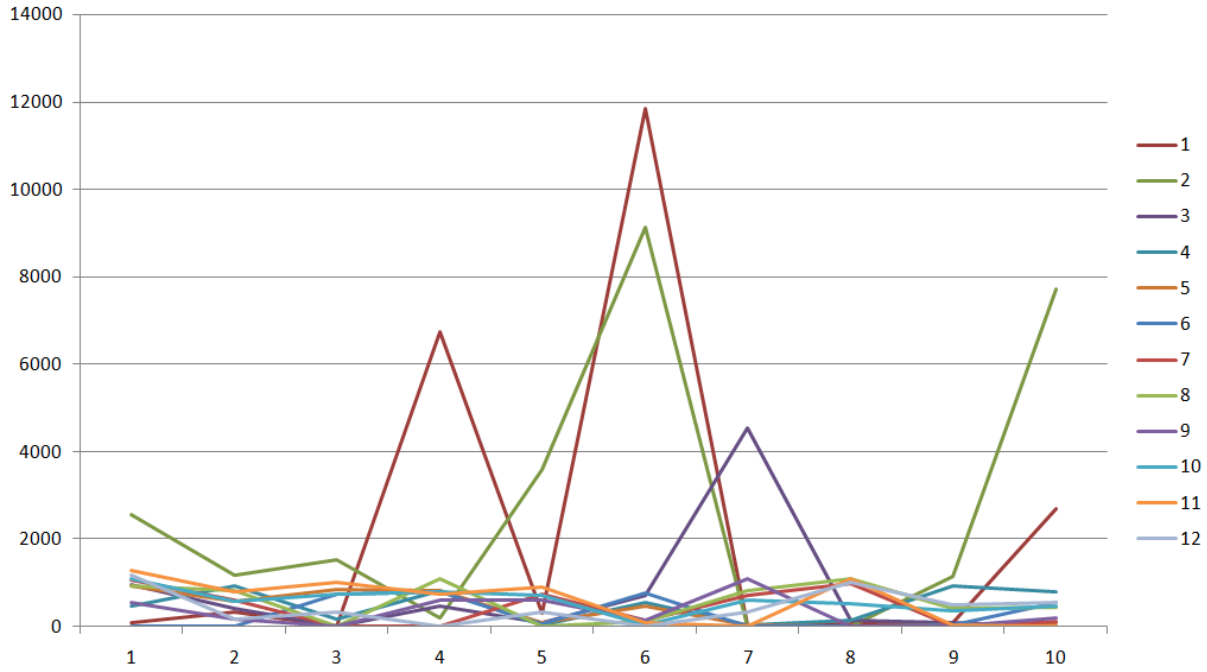


Рисунок 3.16 – Порівняння результатів різних алгоритмів для 10 графів

Рисунок 3.17 – Кількість знайдених клік у 10 графах кожним алгоритмом

ВИСНОВОК

В рамках даної лабораторної роботи я розробила генетичний алгоритм для знаходження кліки певного розміру у графі. Для даного алгоритму був зроблений покроковий опис дій, а також виконана програмна реалізація.

Алгоритм був протестований на 10 графах, кожен з яких мав 300 вершин зі степенями від 2 до 30. У рамках тестування було визначено кількість ітерацій циклу алгоритму під час пошуку кліки розміру 5. Також було зазначено, чи знайшов алгоритм кліку даного розміру у графі. При тестуванні були досліджені всі можливі комбінації параметрів схрещування, мутації та локального покращення

На основі отриманих результатів можна зробити певні висновки стосовно ефективності роботи алгоритму при різних параметрах.

Найгіршими параметрами за критерієм кількості ітерацій виявилися:

- мутація зі зміною гена на протилежний + локальне покращення з додаванням вершин з хромосоми.

За точністю знайденого рішення найгіршими параметрами є:

- мутація з обміном генів місцями + локальне покращення з додаванням вершин з хромосоми.

Найкращими комбінаціями із врахуванням ітерацій та точності рішення були:

- двоточкове схрещування + мутація зі зміною гена на протилежний + локальне покращення з додаванням вершин з списку суміжності;
- динамічне схрещування + мутація зі зміною гена на протилежний + локальне покращення з додаванням вершин з списку суміжності;

Також можна зазначити, що використання мутації з обміном генів місцями завжди призводило до отримання менш точного рішення. Із параметрів схрещування не можна виділити ті, які суттєво погіршують чи покращують роботу алгоритму. Найбільш значимим фактором виявилося локальне покращення: при зміні локального покращення з додаванням вершин із

хромосоми на локальне покращення з додаванням вершин зі списку суміжності алгоритм завжди працював ефективніше (рішення було максимально точним, а кількість ітерацій відносно невеликою).

Таким чином, використання алгоритму з наступними параметрами

- двоточкове схрещування + мутація зі зміною гена на протилежний + локальне покращення з додаванням вершин з списку суміжності
- динамічне схрещування + мутація зі зміною гена на протилежний + локальне покращення з додаванням вершин з списку суміжності

та підбір правильного критерію завершення роботи алгоритму є підставою для отримання оптимального рішення.

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%.