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 Жмайло Дмитро О.* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Сопов О. О.* _____
(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;

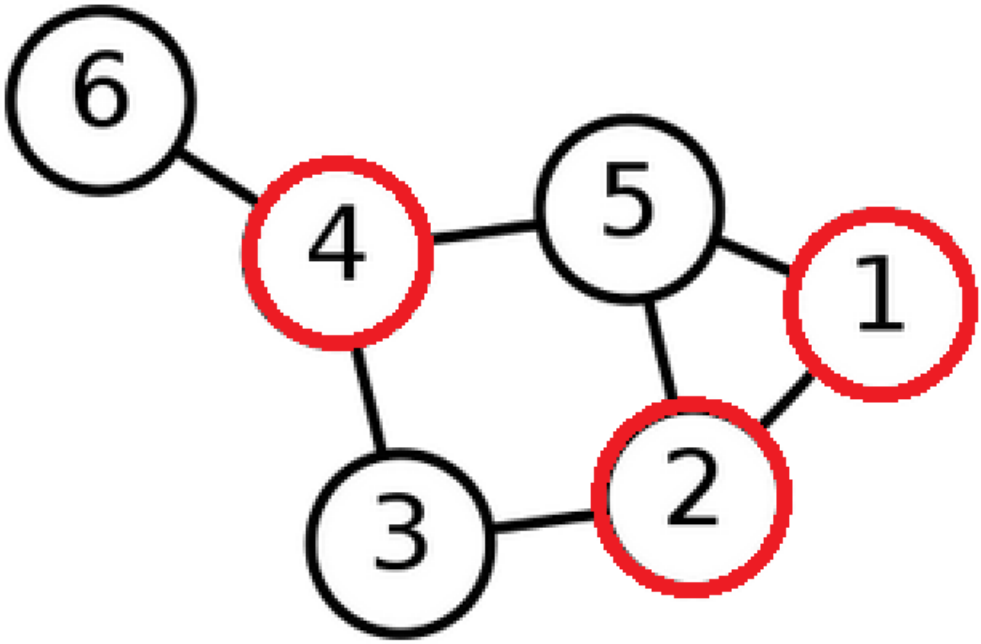
– після цього параметр фіксується і змінюються інші параметри;

– 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
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.

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).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	**Задачі і алгоритми**
9	Задача вершинного покриття + Генетичний алгоритм

3 ВИКОНАННЯ

3.1 Покроковий алгоритм

1. Генерація **початкової популяції**.

2. Сортування початкової популяції за функцією оцінки хромосом, встановлюємо **рекорд** оцінки.

3. Поки не наступить умова зупинки, **ПОВТОРИТИ**:

   3.1. Обираємо двох батьків з **найменшою** оцінкою **S1, S2**

   3.2. Застосовуємо до S1, S2 один з **кросоверів**

   3.3. Застосовуємо одне з **локальних покращень**

   3.4. З певною вірогідністю застосовуємо до нового гена одну з **мутацій**

   3.5. **Додаємо** отриману хромосому до популяції

   3.6. **Сортуємо** популяцію за функцією оцінки

   ВСЕ ПОВТОРИТИ

4. Обираємо **першу** з відсортованих хромосом

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

Program.cs:

```
using System.Diagnostics;

namespace Lab5_PA;

class Program
{
    public static void Main(string[] args)
    {
        Console.WriteLine("Welcome to my 5th lab! (9 variant)");
        Graph graph = new Graph();
        Console.Write("Is adjacency matrix validate? - ");

        if (graph.IsAdjMatrixValid())
            Console.WriteLine("Yes! :)");
        else
            Console.WriteLine("No :'(");

        graph.ShowMatrix();
```

```
        Console.WriteLine("Degrees of graph: ");
        graph.ShowArr(graph.GetDegreesArr());

        BalanceAlgorithm balanceAlgorithm = new BalanceAlgorithm(graph);
    }
      }
```

Graph.cs:

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

namespace Lab5_PA
{
    public class Graph
    {
        private bool[,] adjMatrix;

        const int vertexCount = 300;

        const int minVertexDegree = 2;
        const int maxVertexDegree = 30;

        public Graph(Graph g)
        {
            this.adjMatrix = new bool[g.adjMatrix.GetLength(0),
g.adjMatrix.GetLength(1)];
            Array.Copy(g.adjMatrix, this.adjMatrix, g.adjMatrix.Length);
        }

        public bool[,] GetMatrix()
        {
            return adjMatrix;
        }

        public Graph()
        {
            this.adjMatrix = new bool[vertexCount, vertexCount];
            Random rand = new Random();

            for (int currV = 0; currV < vertexCount; ++currV)
            {
                int finalVertexDegree = Math.Min(rand.Next(minVertexDegree,
maxVertexDegree + 1)
                    - GetVertexDegree(currV), vertexCount - currV - 1);
                for (int newConnections = 0; newConnections < finalVertexDegree;
++newConnections)
                {
                    bool isConnectedAlready = true;
                    for (int tryCount = 0, newVertex = rand.Next(currV + 1,
vertexCount);
                        isConnectedAlready && tryCount < vertexCount; ++tryCount,
newVertex = rand.Next(currV + 1, vertexCount))
                    {
                        if (this.adjMatrix[currV, newVertex] == false &&
GetVertexDegree(newVertex) < maxVertexDegree)
                        {
                            isConnectedAlready = false;
                            this.adjMatrix[currV, newVertex] = true;
                            this.adjMatrix[newVertex, currV] = true;
                        }
```

```
            }
        }
    }

}

public bool IsAdjMatrixValid()
{
    for (int vertex = 0; vertex < adjMatrix.GetLength(0); vertex++)
    {
        if (GetVertexDegree(vertex) > maxVertexDegree)
        {
            return false;
        }
    }
    return true;
}


public int[] GetDegreesArr()
{
    int[] degrees = new int[adjMatrix.GetLength(0)];
    for (int i = 0; i < degrees.Length; ++i)
    {
        degrees[i] = GetVertexDegree(i);
    }
    return degrees;
}

public int GetVertexDegree(int vertex)
{
    int sum = 0;
    for (int i = 0; i < adjMatrix.GetLength(0); i++)
    {
        if (adjMatrix[vertex, i])
        {
            sum += 1;
        }
    }
    return sum;
}


public void ShowMatrix()
{
    for (int i = 0; i < adjMatrix.GetLength(0); i++)
    {
        for (int j = 0; j < adjMatrix.GetLength(1); j++)
        {
            if (adjMatrix[i, j])
                Console.Write(1 + " ");
            else
                Console.Write(0 + " ");
        }
        Console.WriteLine();
    }
    Console.WriteLine();
}

public void ShowArr(int[] arr)
{
    const int maxRowLength = 10;
    for (int i = 0; i < arr.Length; i++)
    {
        Console.Write(arr[i] + "\t");
```

```
                if (i + 1 % maxRowLength == 0)
                {
                    Console.WriteLine();
                }
            }
            Console.WriteLine();
            Console.WriteLine();
        }
    }
}
```

GeneticAlgorithm.cs:

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

namespace Lab5_PA
{
    public class GeneticAlgorithm
    {
        private readonly bool[,] _graphAdjMatrix;

        private List<KeyValuePair<List<bool>, int>> _population;

        private const int MutationProbability = 5; // %
        private const int MaxPopulationSize = 50;
        private const int MaxIterationsCount = 1000;
        private const int numberOfParts = 10;

        private readonly int _crossoverPartition;

        public enum CrossoverType
        {
            crossoverByParts, crossoverByEachGene, crossoverRandomGene
        }
        public enum MutationType
        {
            oneGeneMutation, chromosomeMutation
        }
        public enum LocalEnh
        {
            localEnhType1, localEnhType2
        }
```

```
        public GeneticAlgorithm(bool[,] graphAdjMatrix)
        {
            _graphAdjMatrix = graphAdjMatrix;
            Array.Copy(graphAdjMatrix, _graphAdjMatrix, graphAdjMatrix.Length);
            _crossoverPartition = graphAdjMatrix.Length / numberOfParts;
        }

        // Evaluation
        private int Evaluate(List<bool> chromosome)
        {
            return _graphAdjMatrix.GetLength(0) - CountOfSelectedEdges(chromosome) +
CountPaintedVertexes(chromosome);
        }

        public KeyValuePair<int, List<int>> Start(CrossoverType crosT, MutationType
mutationT, LocalEnh localEnhT)
        {
            _population = CreateInitialPopulation(mutationT);
            for (int i = 0; i < MaxIterationsCount; i++)
            {
                _population = MakeNewGeneration(_population, crosT, mutationT,
localEnhT);
            }

            _population.Sort(Comparer<KeyValuePair<List<bool>, int>>.Create((x, y) =>
x.Value.CompareTo(y.Value)));
            return new KeyValuePair<int,
List<int>>(CountPaintedVertexes(_population[0].Key),
ConvBoolListToInt(_population[0].Key));
        }

        // Create population
        private List<KeyValuePair<List<bool>, int>>
CreateInitialPopulation(MutationType mutationT)
        {
            List<KeyValuePair<List<bool>, int>> population = new
List<KeyValuePair<List<bool>, int>>();
            Random random = new Random();

            for (int i = 0; i < MaxPopulationSize; i++)
            {
                List<bool> genes = new List<bool>(_graphAdjMatrix.GetLength(0));
                for (int j = 0; j < _graphAdjMatrix.GetLength(0); j++)
                {
```

```
                    if (random.Next(0, 2) == 1)
                        genes.Add(true);
                    else
                        genes.Add(false);
                }

                population.Add(new        KeyValuePair<List<bool>,        int>(genes,
Evaluate(genes)));
            }
            return population;
        }

        // Create new Generation
        private                  List<KeyValuePair<List<bool>,                  int>>
MakeNewGeneration(List<KeyValuePair<List<bool>, int>> population, CrossoverType crosT,
MutationType mutationT, LocalEnh localEnhT)
        {
            List<KeyValuePair<List<bool>,     int>>     evaluationList     =     new
List<KeyValuePair<List<bool>, int>>(population.OrderBy(x => x.Value).ToList());

            evaluationList.RemoveAt(evaluationList.Count - 1);
            KeyValuePair<List<bool>,          int>          newChromosome          =
SetCrossover(population[0].Key, population[1].Key, crosT);

            newChromosome = SetMutation(newChromosome, mutationT);
            newChromosome = SetLocalEnhancement(newChromosome, localEnhT);
            evaluationList.Add(new   KeyValuePair<List<bool>,   int>(newChromosome.Key,
Evaluate(newChromosome.Key)));
            return evaluationList;
        }

        // Mutations
        private   KeyValuePair<List<bool>,   int>   SetMutation(KeyValuePair<List<bool>,
int> chromosome, MutationType mutationT)
        {
            Random random = new Random();
            bool mutationHappen = random.Next(101) < MutationProbability;
            var res = chromosome;
            if (mutationHappen)
            {
                if (mutationT == MutationType.oneGeneMutation)
                    res = MutateGen(chromosome.Key);

                else if (mutationT == MutationType.chromosomeMutation)
```

```
                res = MutateChromosome(chromosome.Key);
            }
            return res;
        }
        private KeyValuePair<List<bool>, int> MutateChromosome(List<bool> chromosome)
        {
            Random random = new Random();
            List<bool> mutatedChromosome = new List<bool>();
            foreach (bool gen in chromosome)
            {
                bool isMutated = random.Next(101) < MutationProbability;

                if (isMutated)
                    mutatedChromosome.Add(!gen);
                else
                    mutatedChromosome.Add(gen);
            }

            return new KeyValuePair<List<bool>, int>(mutatedChromosome,
Evaluate(mutatedChromosome));
        }

        private KeyValuePair<List<bool>, int> MutateGen(List<bool> chromosome)
        {
            Random random = new Random();
            int genIndex = random.Next(0, chromosome.Count);
            chromosome[genIndex] = !chromosome[genIndex];
            return new KeyValuePair<List<bool>, int>(chromosome,
Evaluate(chromosome));
        }

        // Crossovers
        private KeyValuePair<List<bool>, int> SetCrossover(List<bool> chromosome1,
List<bool> chromosome2, CrossoverType crosT)
        {
            List<bool> resChromosome = new List<bool>();
            if (crosT == CrossoverType.crossoverByParts)
                resChromosome = CrossoverByParts(chromosome1, chromosome2);

            else if (crosT == CrossoverType.crossoverByEachGene)
                resChromosome = CrossoverEachGen(chromosome1, chromosome2);

            else if (crosT == CrossoverType.crossoverRandomGene)
                resChromosome = CrossoverRandomGen(chromosome1, chromosome2);
```

```
            return       new       KeyValuePair<List<bool>,       int>(resChromosome,
Evaluate(resChromosome));
        }

        private   List<bool>   CrossoverByParts(List<bool>   chromosome1,   List<bool>
chromosome2)
        {
            List<bool> newChromosome = new List<bool>(_graphAdjMatrix.GetLength(0));
            for (int i = 0; i < _graphAdjMatrix.GetLength(0); i++)
            {
                if (i % _crossoverPartition != 0)
                    newChromosome.Add(chromosome1[i]);

                else
                    newChromosome.Add(chromosome2[i]);
            }
            return newChromosome;
        }

        private   List<bool>   CrossoverEachGen(List<bool>   chromosome1,   List<bool>
chromosome2)
        {
            List<bool> newChromosome = new List<bool>(_graphAdjMatrix.GetLength(0));
            for (int i = 0; i < _graphAdjMatrix.GetLength(0); i++)
            {
                if (i % 2 == 0)
                    newChromosome.Add(chromosome1[i]);
                else
                    newChromosome.Add(chromosome2[i]);
            }
            return newChromosome;
        }

        private   List<bool>   CrossoverRandomGen(List<bool>   chromosome1,   List<bool>
chromosome2)
        {
            Random random = new Random();
            List<bool> newChromosome = new List<bool>(_graphAdjMatrix.GetLength(0));
            for (int i = 0; i < _graphAdjMatrix.GetLength(0); i++)
            {
                if (random.Next(0, 2) == 1)
                    newChromosome.Add(chromosome1[i]);
                else
```

```
                newChromosome.Add(chromosome2[i]);
            }
            return newChromosome;
        }

        // Local Enhancement

        private                      KeyValuePair<List<bool>,                      int>
SetLocalEnhancement(KeyValuePair<List<bool>, int> chromosome, LocalEnh localEnhT)
        {
            KeyValuePair<List<bool>, int> enhanced = chromosome;
            if (localEnhT == LocalEnh.localEnhType1)
                return LocalEnh1(enhanced);

            else if (localEnhT == LocalEnh.localEnhType2)
                return LocalEnh2(enhanced);
            return enhanced;
        }
        private KeyValuePair<List<bool>, int> LocalEnh1(KeyValuePair<List<bool>, int>
chromosome)
        {
            var chromMaxEval = new List<bool>(chromosome.Key);
            int minEval = chromosome.Value;
            int elIndex = 0;

            for (int i = 0; i < chromosome.Key.Count; i++)
            {
                chromMaxEval[i] = !chromosome.Key[i];
                int evaluation = Evaluate(chromMaxEval);
                if (evaluation < minEval && CountOfSelectedEdges(chromosome.Key) ==
_graphAdjMatrix.Length)
                {
                    minEval = evaluation;
                    elIndex = i;
                }

                chromMaxEval[i] = !chromMaxEval[i];
            }
            chromMaxEval[elIndex] = !chromMaxEval[elIndex];
            return new KeyValuePair<List<bool>, int>(chromMaxEval, minEval);
        }

        private KeyValuePair<List<bool>, int> LocalEnh2(KeyValuePair<List<bool>, int>
chromosome)
```

```
{
    int maxConIndex = 0;
    int maxCon = GetVertexConnections(0).Count;

    int curVertexCon;
    for (int i = 1; i < chromosome.Key.Count; i++)
    {
        curVertexCon = GetVertexConnections(i).Count;
        if (!chromosome.Key[i])
        {
            if (maxCon < curVertexCon)
            {
                maxCon = curVertexCon;
                maxConIndex = i;
            }
        }
    }

    List<int> con = GetVertexConnections(maxConIndex);

    int bestEval = chromosome.Value;
    foreach (int conVertexIndex in con)
    {
        chromosome.Key[conVertexIndex] = false;
        if (CountOfSelectedEdges(chromosome.Key) != _graphAdjMatrix.Length)
            chromosome.Key[conVertexIndex] = true;
    }

    chromosome.Key[maxConIndex] = true;
    return chromosome;
}

private int CountOfSelectedEdges(List<bool> chromosome)
{
    HashSet<int> coloredHashSet = new HashSet<int>();

    for (int i = 0; i < chromosome.Count; i++)
    {
        if (chromosome[i])
        {
            coloredHashSet.Add(i);
            coloredHashSet.UnionWith(GetVertexConnections(i));
        }
    }
```

```
            return coloredHashSet.Count;
        }

        private List<int> GetVertexConnections(int vertexIndex)
        {
            List<int> connections = new List<int>();

            for (int i = 0; i < _graphAdjMatrix.GetLength(1); i++)
            {
                if (_graphAdjMatrix[vertexIndex, i])
                    connections.Add(i);
            }

            return connections;
        }

        private int CountPaintedVertexes(List<bool> chromosome)
        {
            return chromosome.Where(x => x).Count();
        }

        private List<int> ConvBoolListToInt(List<bool> lst)
        {
            var resList = new List<int>();
            for (int i = 0; i < lst.Count; i++)
            {
                if (lst[i])
                    resList.Add(i);
            }
            return resList;
        }
    }
}
```

BslsnceAlgorithm.cs:

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Lab5_PA
{
    internal class BalanceAlgorithm
    {
        public BalanceAlgorithm(Graph graph)
        {
            const int amountOfTests = 10;
```

```
            bool[,] matrix = graph.GetMatrix();
            GeneticAlgorithm geneticAlg = new GeneticAlgorithm(matrix);
            double bestScore = double.MaxValue;

            GeneticAlgorithm.CrossoverType[] crossOverTypes = {
GeneticAlgorithm.CrossoverType.crossoverByParts,
            GeneticAlgorithm.CrossoverType.crossoverByEachGene,
GeneticAlgorithm.CrossoverType.crossoverRandomGene };

            GeneticAlgorithm.MutationType[] mutationTypes = {
GeneticAlgorithm.MutationType.chromosomeMutation,
            GeneticAlgorithm.MutationType.oneGeneMutation };

            GeneticAlgorithm.LocalEnh[] localEnhancements = {
GeneticAlgorithm.LocalEnh.localEnhType1,
            GeneticAlgorithm.LocalEnh .localEnhType2};



            Console.WriteLine("*****-=Balance of params is started, stand by...=-
*****");
            Console.WriteLine();
            Console.WriteLine("#####-=Crossover balance=-#####");
            Console.WriteLine();
            GeneticAlgorithm.CrossoverType bestCrossoverType =
GeneticAlgorithm.CrossoverType.crossoverByParts;

            foreach (GeneticAlgorithm.CrossoverType crossoverT in crossOverTypes)
            {
                Console.WriteLine("-=" + crossoverT + "=-");

                double score = TestResults(geneticAlg, amountOfTests, crossoverT,
GeneticAlgorithm.MutationType.chromosomeMutation,
GeneticAlgorithm.LocalEnh.localEnhType2);
                Console.WriteLine("#avg " + crossoverT + " mark: " + score);
                if (score < bestScore)
                {
                    bestScore = score;
                    bestCrossoverType = crossoverT;
                }
                Console.WriteLine();
            }

            Console.WriteLine("#####################################");
            Console.WriteLine("#Best crossover type: " + bestCrossoverType);
            Console.WriteLine("#####################################");

            Console.WriteLine();
            Console.WriteLine("#####-=Mutation balance=-#####");
            GeneticAlgorithm.MutationType bestMutationType =
GeneticAlgorithm.MutationType.chromosomeMutation;

            bestScore = double.MaxValue;
            foreach (GeneticAlgorithm.MutationType mutationT in mutationTypes)
            {
                Console.WriteLine("-=" + mutationT + "=-");

                double mark = TestResults(geneticAlg, amountOfTests,
bestCrossoverType, mutationT, GeneticAlgorithm.LocalEnh.localEnhType2);
                Console.WriteLine("#avg " + mutationT + " mark: " + mark);
                if (mark < bestScore)
                {
                    bestScore = mark;
                    bestMutationType = mutationT;
```

```
                }
                Console.WriteLine();
            }
            Console.WriteLine("#####################################");
            Console.WriteLine("#Best mutation type: " + bestMutationType);
            Console.WriteLine("#####################################");

            Console.WriteLine();
            Console.WriteLine("#####-=Local enh balance=-#####");
            Console.WriteLine();
            GeneticAlgorithm.LocalEnh bestLocalEnhType =
GeneticAlgorithm.LocalEnh.localEnhType1;

            bestScore = double.MaxValue;
            foreach (GeneticAlgorithm.LocalEnh localEnhT in localEnhancements)
            {
                Console.WriteLine("-=" + localEnhT + "=-");

                double mark = TestResults(geneticAlg, amountOfTests,
bestCrossoverType, bestMutationType, localEnhT);
                Console.WriteLine("#avg " + localEnhT + " mark: " + mark);
                if (mark < bestScore)
                {
                    bestScore = mark;
                    bestLocalEnhType = localEnhT;
                }
                Console.WriteLine();
            }
            Console.WriteLine("#####################################");
            Console.WriteLine("#Best local enhancement type: " + bestLocalEnhType);
            Console.WriteLine("#####################################");
            Console.WriteLine("\n*****-=Balance results=-*****");
            Console.WriteLine("Best crossover: " + bestCrossoverType);
            Console.WriteLine("Best mutation: " + bestMutationType);
            Console.WriteLine("Best local enhancement: " + bestLocalEnhType);
            Console.WriteLine("#####################################");
        }

        public static double TestResults(GeneticAlgorithm ga, int amountOfTests,
GeneticAlgorithm.CrossoverType crossOverType, GeneticAlgorithm.MutationType
mutationType, GeneticAlgorithm.LocalEnh localEnhancement)
        {
            double evaluationsSum = 0;
            int testEval;
            for (int i = 1; i <= amountOfTests; i++)
            {
                Stopwatch stopwatch = Stopwatch.StartNew();
                testEval = ga.Start(crossOverType, mutationType,
localEnhancement).Key;
                evaluationsSum += testEval;
                stopwatch.Stop();
                Console.WriteLine("№" + i + " evaluation: " + testEval + ", time
taken: " + stopwatch.ElapsedMilliseconds/1000.0 + " s");
            }
            return evaluationsSum / amountOfTests;
        }
    }
}
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

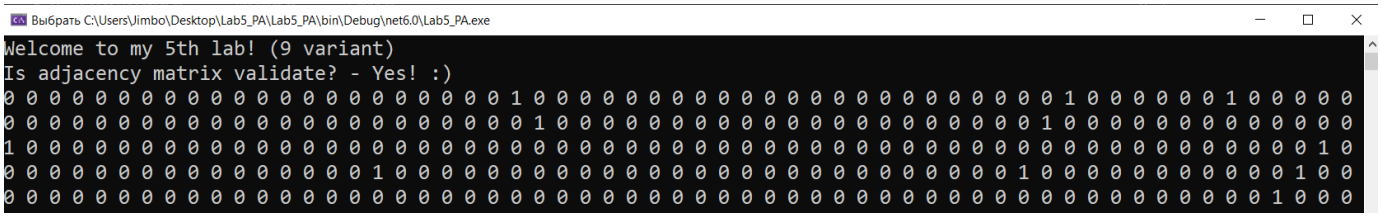


Рисунок 3.1 – Запуск програми

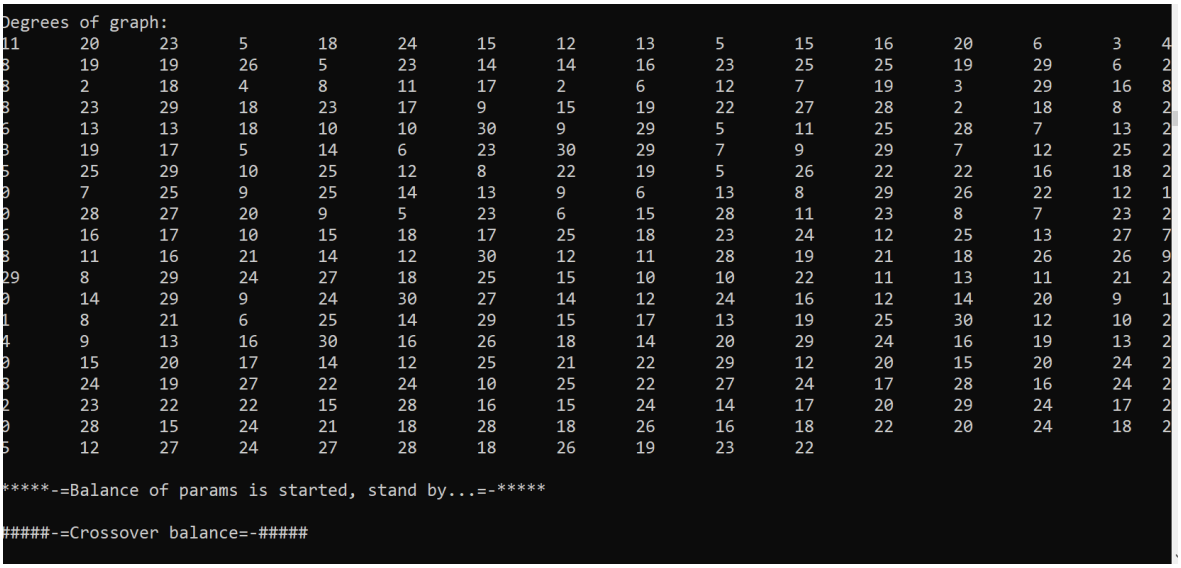


Рисунок 3.2 – Початок балансування

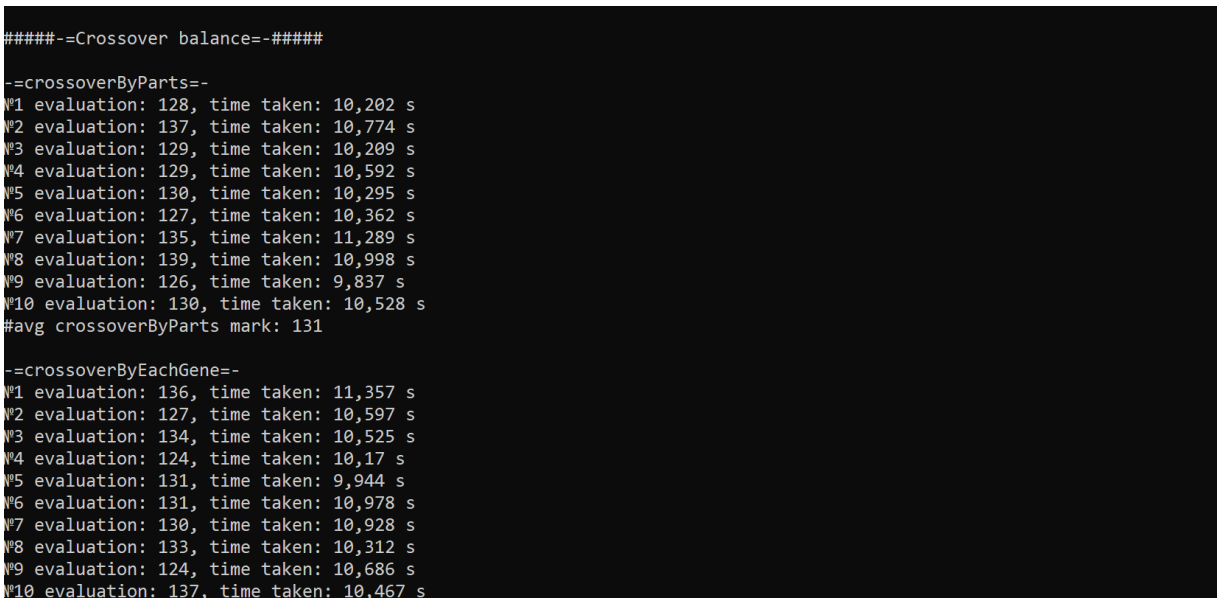


Рисунок 3.3 –Процес балансування

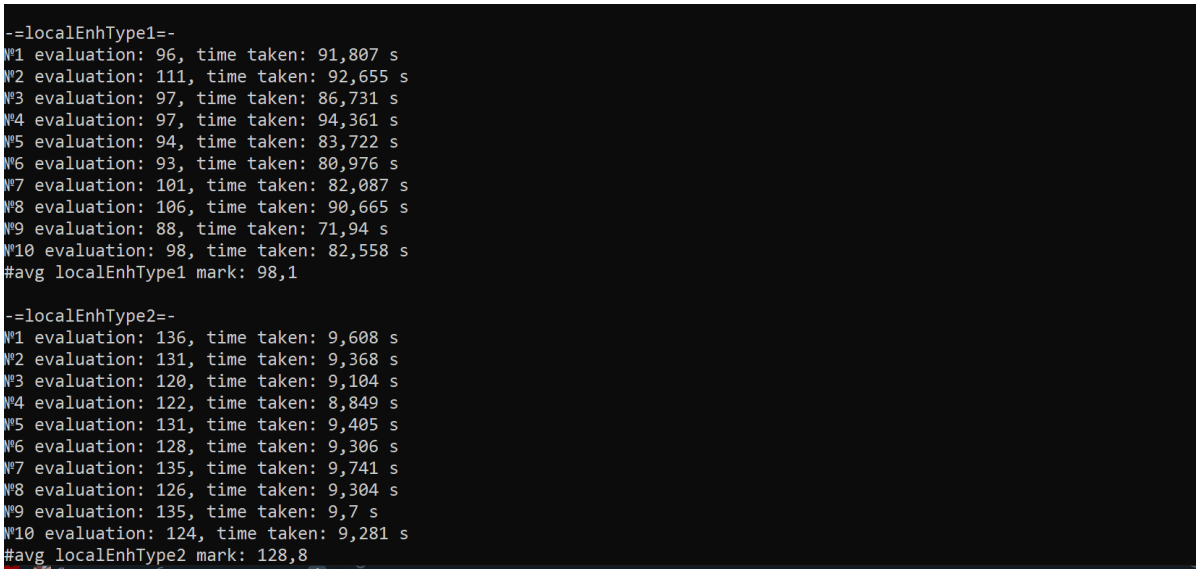


Рисунок 3.4 – Результат балансування локальних покращень

3.3 Тестування алгоритму

Через те, що застосовується ***мета-евристичний*** алгоритм, нам потрібно знайти баланс між швидкістю знаходження та якістю отриманого рішення.

Отже, окрім вимірювання швидкості виконання та цільової функці, ми додатково вводимо ще один допоміжній параметр – ***оцінка***.

Спираючись на оцінку, робиться висновок щодо якості підібраних нами параметрів. Оцінка дорівнює сумі середнього часу, витраченого на пошук відповіді та середньої довжини знайденого маршруту.

Для оцінки балансування використаємо граф, з кількістю вершин = **300**.

Таблиця 3.1 Результати тесту Кросовера **частинами**

Кросовер частинами			
№ тесту	Оцінка	Затрачений час	Значення цільової ф-ії
1	141,298	10,298	131
2	140,832	9,832	131
3	137,728	9,728	128
4	146	10	136
5	139,809	9,809	130
6	141,818	9,818	132
7	148,295	10,295	138
8	133,372	9,372	124
9	138,036	10,036	128
10	140,493	10,49	130
Результат (avg)	140,7681	9,9681	130,8

Таблиця 3.2 Результати тесту Кросовера **кожного гену**

Кросовер кожного гена			
№ тесту	Оцінка	Затрачений час	Значення цільової ф-ії
1	142,6	10,6	132
2	149,244	14,244	135
3	145,565	10,565	135
4	142,184	10,184	132
5	145,154	10,154	135
6	145,438	10,438	135
7	141,448	9,448	132
8	137,311	9,311	128
9	144,376	10,376	134
10	141,323	9,323	132
Результат (avg)	143,4643	10,4643	133

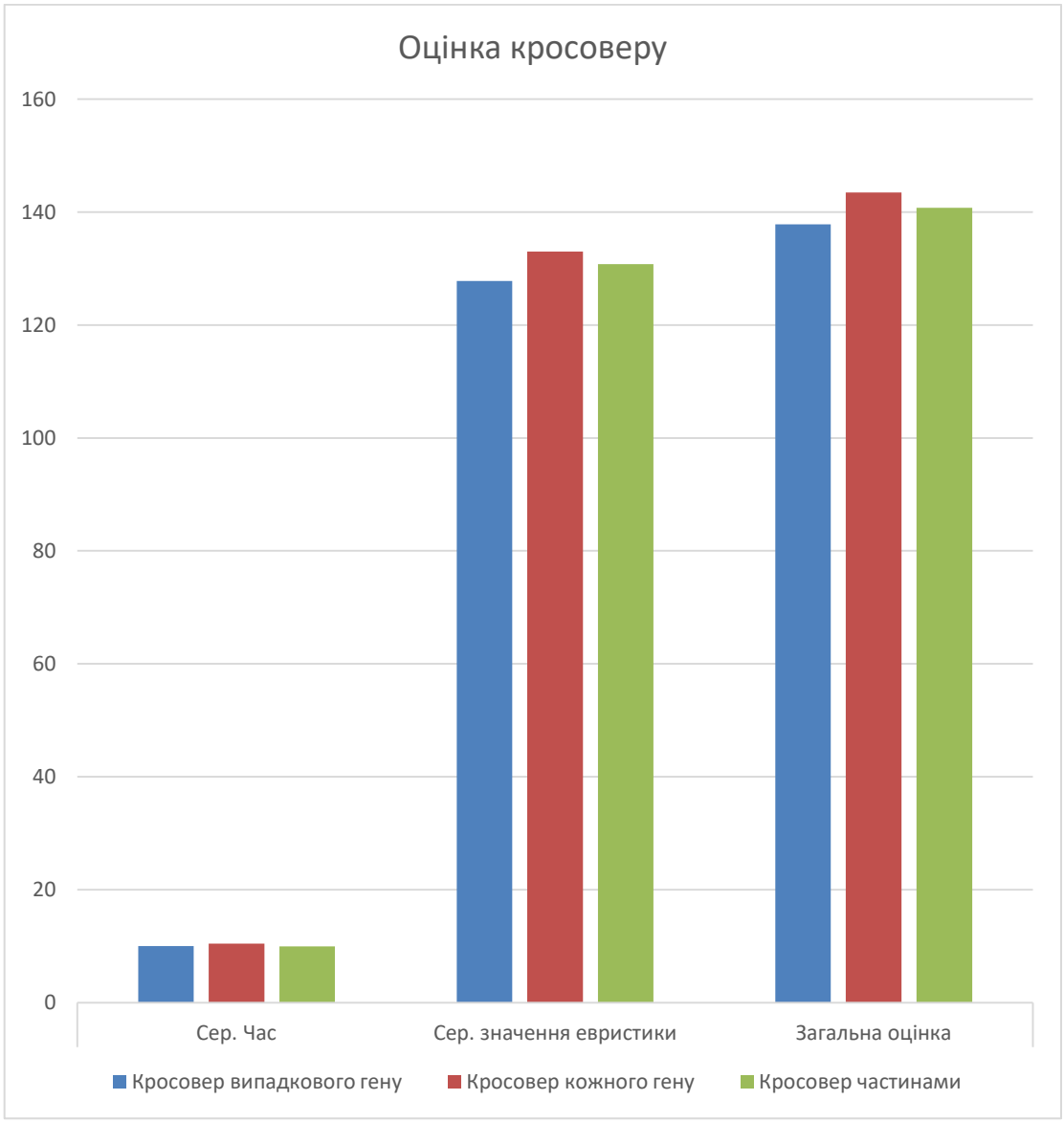
Таблиця 3.3 Результати тесту Кросовера **випадкового гену**

Кросовер випадкового гену			
№ тесту	Оцінка	Затрачений час	Значення цільової ф-ії
1	143,878	9,878	134
2	133,517	9,517	124
3	137,919	9,919	128
4	139,824	9,824	130
5	136,639	9,639	127
6	134,902	9,902	125
7	134,035	10,035	124
8	139,716	10,716	129
9	137,332	10,332	127
10	140,358	10,358	130
Результат (avg)	137,812	10,012	127,8

Таблиця 3.4 Загальні результати тестів оцінки **кросовера**

Балансування кросовера			
Тип	Сер. Час	Сер. Значення цільової ф-ії	Загальна оцінка
Кросовер випадкового гену	10,012	127,8	137,812
Кросовер кожного гену	10,4643	133	143,4643
Кросовер частинами	9,9681	130,8	140,7681

З таблиці 3.4 видно, що найкращим типом кросоверу є кросовер **випадкового гену**



Діаграма 3.1 – балансування кросоверу

Переходимо до тестування мутацій:

Таблиця 3.5 Результати тесту Мутації **одного гену**

Мутація одного гену			
№ тесту	Оцінка	Затрачений час	Значення цільової ф-ії
1	143,644	10,644	133
2	141,788	10,788	131
3	146,123	11,123	135
4	141,533	10,533	131
5	142,016	11,016	131
6	143,617	10,617	133
7	140,618	10,618	130
8	141,788	10,788	131
9	141,645	10,645	131
10	143,95	10,95	133
Результат (avg)	142,6722	10,7722	131,9

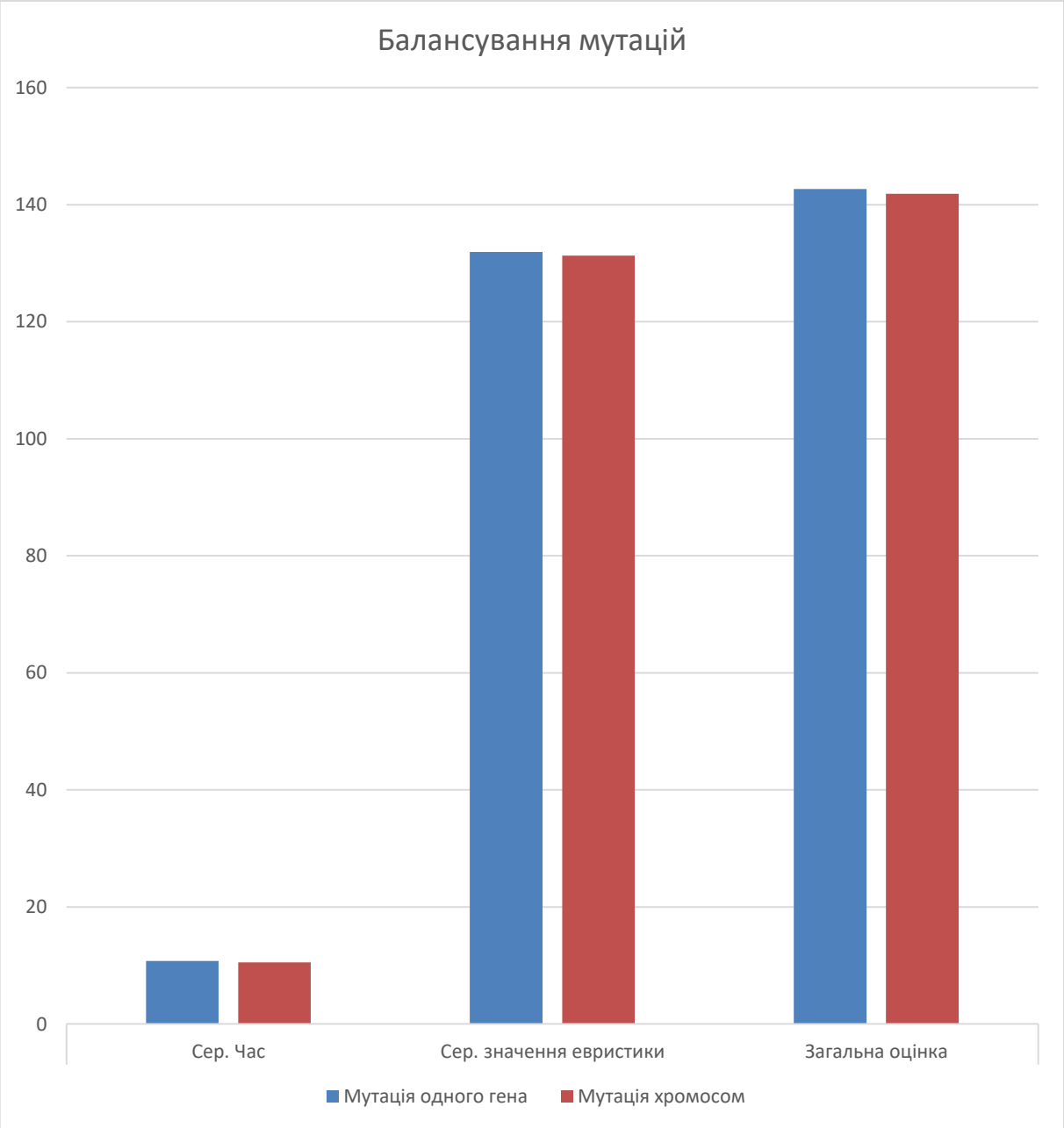
Таблиця 3.6 Результати тесту Мутації **хромосом**

Мутація хромосом			
№ тесту	Оцінка	Затрачений час	Значення цільової ф-ії
1	138,481	10,481	128
2	139,095	10,095	129
3	139,303	10,303	129
4	144,815	10,815	134
5	142,35	10,35	132
6	143,319	10,319	133
7	141,685	10,685	131
8	144,006	11,006	133
9	142,585	10,585	132
10	142,733	10,733	132
Результат (avg)	141,8372	10,5372	131,3

Таблиця 3.7 Загальні результати оцінювання **мутацій**

Балансування мутацій			
Тип	Сер. Час	Сер. Значення цільової ф-ії	Загальна оцінка
Мутацій одного гена	10,7722	131,9	142,6722
Мутація хромосом	10,5372	131,3	141,8372

З таблиці 3.7 видно, що найкращим типом мутацій є мутація **хромосом**



Діаграма 3.2 – балансування мутацій

Переходимо до тестування локальних покращень:

Таблиця 3.8 Результати тесту локального покращення **№1**

Локальне покращення №1			
№ тесту	Оцінка	Затрачений час	Значення цільової ф-ії
1	187,807	91,807	96
2	203,655	92,655	111
3	183,731	86,731	97
4	191,361	94,361	97
5	177,722	83,722	94
6	173,976	80,976	93
7	183,087	82,087	101
8	196,665	90,665	106
9	159,94	71,94	88
10	180,558	82,558	98
Результат (avg)	183,8502	85,7502	98,1

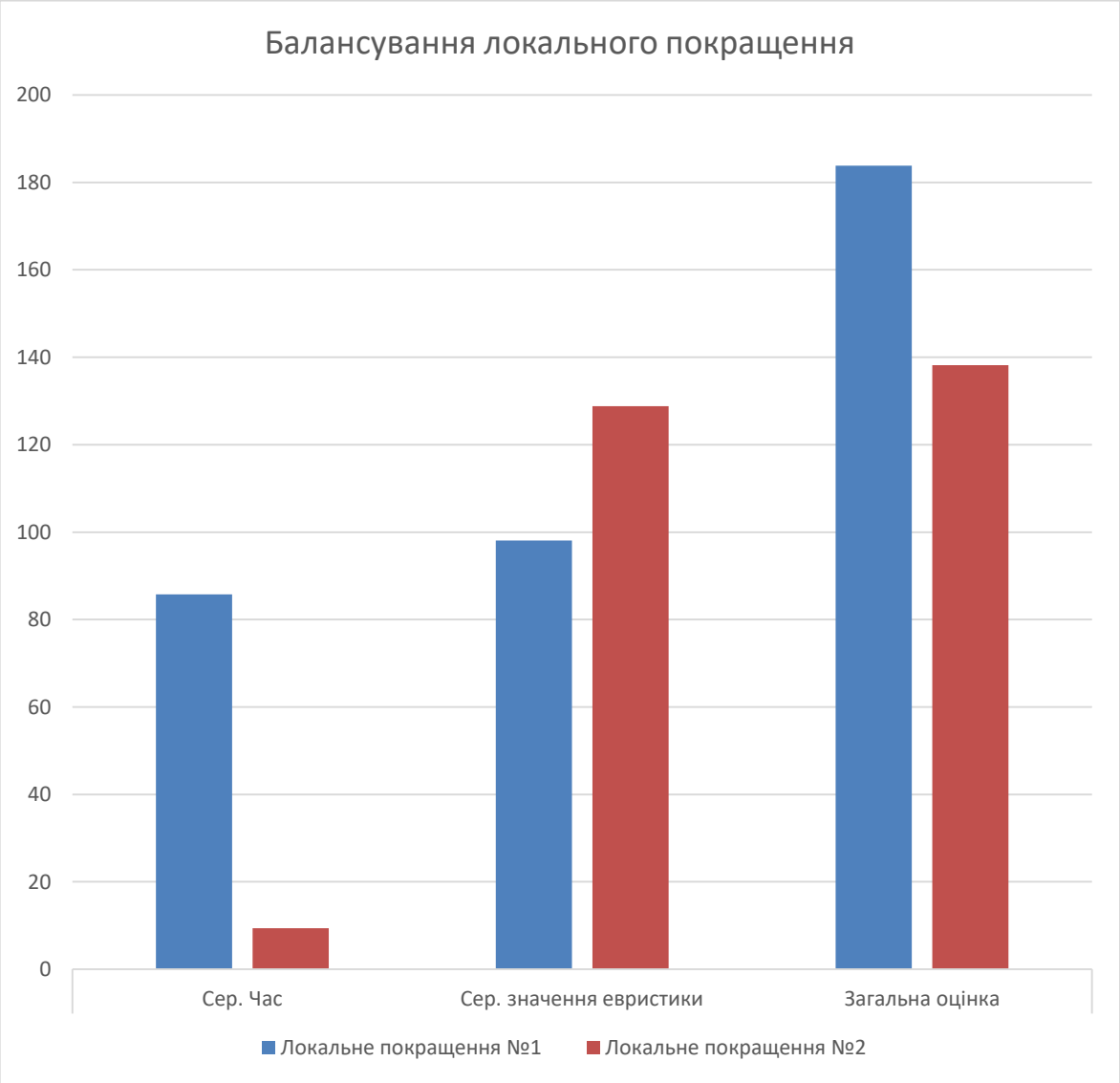
Таблиця 3.9 Результати тесту локального покращення **№2**

Локальне покращення №2			
№ тесту	Оцінка	Затрачений час	Значення цільової ф-ії
1	145,608	9,608	136
2	140,368	9,368	131
3	129,104	9,104	120
4	130,849	8,849	122
5	140,405	9,405	131
6	137,306	9,306	128
7	144,741	9,741	135
8	135,304	9,304	126
9	144,7	9,7	135
10	133,281	9,281	124
Результат (avg)	138,1666	9,3666	128,8

Таблиця 3.10 Загальні результати оцінювання **локальних покращень**

Балансування локального покращення			
Тип	Сер. Час	Сер. Значення цільової ф-ії	Загальна оцінка
Локальне покращення №1	85,7502	98,1	183,8502
Локальне покращення №2	9,3666	128,8	138,1666

З таблиці 3.10 видно, що найкращим типом локального покращення є локальне покращення **№2** завдяки його швидкодії



Діаграма 3.3 – балансування локального покращення

ВИСНОВОК

В рамках даної лабораторної роботи, реалізував задачу вершинного покриття, використовуючи генетичний алгоритм та виконав підбір та аналіз даних найефективніших параметрів для вирішення цієї задачі.

Взагалом було розроблено три варіанти кросоверу, два варіанти мутацій та два варіанти локального покращення. Для порівняння розраховувалася середня оцінка ефективності алгоритму на базі евристичної функції.

В результаті тестування всіх варіантів ***Генетичного алгоритму,*** ми можемо зробити висновок, що найкращий тип кросовера – **кросовер випадкового гена**, найкращий тип мутації – **мутація хромосом** та найкращий тип локального покращення – локальне покращення **№2**.

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%.