

**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 Пархомчук Ілля Вікторович \_\_\_\_\_  
(шифр, прізвище, ім'я, по батькові)

**Перевірив**

Сопов О.О. \_\_\_\_\_  
(прізвище, ім'я, по батькові)

Київ 2022

## ЗМІСТ

<b>1</b>	<b>МЕТА ЛАБОРАТОРНОЇ РОБОТИ.....</b>	<b>3</b>
<b>2</b>	<b>ЗАВДАННЯ .....</b>	<b>4</b>
<b>3</b>	<b>ВИКОНАННЯ .....</b>	<b>6</b>
3.1	ПОКРОКОВИЙ АЛГОРИТМ .....	6
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ.....	6
3.2.1	<i>Вихідний код.....</i>	<i>6</i>
3.2.2	<i>Приклади роботи.....</i>	<i>14</i>
3.3	ТЕСТУВАННЯ АЛГОРИТМУ.....	16
	<b>ВИСНОВОК .....</b>	<b>20</b>
	<b>КРИТЕРІЇ ОЦІНЮВАННЯ .....</b>	<b>22</b>

## 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 ступенем 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
6	<b>Задача про найкоротший шлях</b> (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

	<p>складають шлях.</p> <p>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.</p>
--	---

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	Алгоритми і досліджувані параметри
3	<p><b>Бджолиний алгоритм:</b></p> <ul style="list-style-type: none"> <li>– кількість ділянок;</li> <li>– кількість бджіл (фуражирів і розвідників).</li> </ul>

Таблиця 2.3 – Варіанти задач і алгоритмів

№	Задачі і алгоритми
22	Задача про найкоротший шлях + Бджолиний алгоритм

## 3 ВИКОНАННЯ

### 3.1 Покроковий алгоритм

1. Ініціалізувати задачу графом
2. Створюємо початковий маршрут локальним пошуком та ініціалізуємо ним усі області, к-сть областей дорівнює к-сті робочих бджіл, причому одній бджолі відповідає одна область( шлях) [2]
3. Робочі бджоли вилітають на ділянки ( намагаємося покращити поточний шлях, локальним пошуком, де поточний шлях змінюється від випадкової вершини до кінцевої[1] )
  - 3.1. Якщо покращення вдалося, записуємо новий шлях замість поточного
  - 3.2. Якщо к-сть невдач більша за певний ліміт, то робоча бджола стає бджолою розвідником і шукає нову область (генерує новий шлях). Цей шлях записується замість попереднього, всі показники обнуляються.
4. Повернувшись у вулик робочі бджоли діляться інформацією з спостерігачами. Спостерігачі за методом пропорційного вибору обирають ділянку на яку вони летять і намагаються покращити.
  - 4.1. Якщо покращення шляху невелике або його довжина більша за поточну, то к-сть невдач для цієї області (шляху) збільшується
    - 4.1.1. Інакше записуємо отриманий шлях замість поточного та обнуляємо показник невдач
5. Повторити пункт 3-4 до знаходження рішення або досягнення певної к-сті ітерацій

### 3.2 Програмна реалізація алгоритму

#### 3.2.1 Вихідний код

##### **main.h**

```
#pragma once

constexpr int VERTEX_AMOUNT = 300;
constexpr int LENGHT_LOWER_BOUND = 5;
constexpr int LENGHT_UPPER_BOUND = 150;
```

```
constexpr int VERT_DEGREE_LOWER_BOUND = 1;
constexpr int VERT_DEGREE_UPPER_BOUND = 10;

constexpr int FAIL_COUNT = VERTEX_AMOUNT-1;
constexpr int FAIL_SEARCH_COUNT = 10;

constexpr int NUMBER_OF_ITERATION = 30;
```

## Graph.h

```
#pragma once
struct TaskGraph
{
    int** adj_matrix;
    const size_t _size;
    TaskGraph(int size);
    ~TaskGraph();
    void PrintMatrix();
    void PrintMatrixToFile();
};
```

## Graph.cpp

```
#include "Graph.h"
#include "main.h"

#include <iostream>
#include <random>
#include <time.h>

#include <fstream>

TaskGraph::TaskGraph(int size) : _size(size)
{
    adj_matrix = new int* [_size];
    for (size_t i = 0; i < _size; i++)
    {
        adj_matrix[i] = new int[_size] {0};
    }

    int taken;
    int taken_y;
    int pos_value;
    int bound;
    for (int i = 0; i < _size - 1; i++)
    {
        taken = 0;
        for (int k = 0; k < i; k++)
        {
            taken += adj_matrix[i][k];
        }
        bound = rand() % (VERT_DEGREE_UPPER_BOUND - VERT_DEGREE_LOWER_BOUND) +
VERT_DEGREE_LOWER_BOUND - taken;
        for (int k = 0; k < bound ; k++)
        {
            taken_y = 0;
            pos_value = rand() % (_size - i - 1) + i + 1;

            for (int m = 0; m < i -1; m++)
            {
                taken_y += adj_matrix[pos_value][m];
            }
            if (taken_y >= VERT_DEGREE_UPPER_BOUND)
```

```

        {
            --k;
            continue;
        }
        adj_matrix[i][pos_value] = 1;
        adj_matrix[pos_value][i] = 1;
    }
}

int value;
for (int i = 0; i < _size; i++)
{
    for (int k = 0; k < i; k++)
    {
        if (adj_matrix[i][k])
        {
            value = rand() % (LENGHT_UPPER_BOUND - LENGHT_LOWER_BOUND + 1) +
LENGHT_LOWER_BOUND;
            adj_matrix[i][k] = adj_matrix[k][i] = value;
        }
    }
}

}

TaskGraph::~TaskGraph()
{
    for (size_t i = 0; i < _size; i++)
    {
        delete[] adj_matrix[i];
    }
    delete[] adj_matrix;
}

void TaskGraph::PrintMatrix()
{
    for (size_t i = 0; i < _size; i++)
    {
        for (size_t k = 0; k < _size; k++)
        {
            std::cout << adj_matrix[i][k] <<
                ((k < _size - 1) ? ", " : " ");
        }
        std::cout << '\n';
    }
}

void TaskGraph::PrintMatrixToFile()
{
    std::ofstream file("matrix.txt");
    for (size_t i = 0; i < _size; i++)
    {
        for (size_t k = 0; k < _size; k++)
        {
            file << adj_matrix[i][k] <<
                ((k < _size - 1) ? ", " : " ");
        }
        file << '\n';
    }
}

```



## BeesAlgorithm.h

```
#pragma once
#include "Graph.h"
#include <vector>
#include <utility>

using std::vector;
using std::pair;

typedef int x_value;
typedef float f_value;
typedef int path_length;

typedef std::pair<vector<int>, path_length> path;
typedef pair<path*, int> path_err;

class BeesAlgorithm
{
private:
    int onlook_count;
    int empl_count;
    TaskGraph* task;
    vector<path_err> paths_mark;
    vector<path> paths;
    int best_ind;
    void CalcRouteLenght(path &v);
    vector<int> LocalSearch(int start_vert, int end_vert );
    x_value RouletteWheelChoice(vector < std::pair<x_value, f_value> > & values);
public:
    BeesAlgorithm(int o_count, int e_count);
    ~BeesAlgorithm();
    void Solve(int start_vert, int end_vert);
    void PrintBest();
};
```

## BeesAlgorithm.cpp

```
#include "BeesAlgorithm.h"
#include "main.h"

#include <iostream>
#include <random>
#include <time.h>
#include <fstream>
#include <string>
#include <assert.h>

BeesAlgorithm::BeesAlgorithm(int o_count, int e_count) : onlook_count(o_count),
empl_count(e_count)
{
    task = new TaskGraph(VERTEX_AMOUNT);
    srand(time(NULL));
    paths = vector<path>(empl_count);
    paths_mark = vector<path_err>(empl_count);

    for (size_t i = 0; i < empl_count; i++)
    {
        paths_mark[i].first = &paths[i];
        paths_mark[i].second = 0;
    }
}
```

```

}

BeesAlgorithm::~BeesAlgorithm()
{
    delete task;
}

x_value BeesAlgorithm::RouletteWheelChoice(vector<std::pair<x_value, f_value>>& values)
{
    float sum = 0;
    for (auto& v : values)
        sum += v.second;

    float x_choice = rand() / float(RAND_MAX);
    float x = 0;

    for (size_t i = 0; i < values.size(); i++)
    {
        x += values[i].second / sum;

        if (x >= x_choice)
            return values[i].first;
    }
    return values.back().first;
}

void BeesAlgorithm::CalcRouteLenght(path& v)
{
    v.second = 0;
    for (size_t i = 0; i < v.first.size()-1; i++)
    {
        v.second += task->adj_matrix[v.first[i]][v.first[i + 1]];
    }
}

vector<int> BeesAlgorithm::LocalSearch(int start_vert, int end_vert)
{
    vector<int> res = { start_vert };
    vector<std::pair<x_value, f_value>> x_f;
    int cur_vert = start_vert;
    int i = 0;
    do
    {
        x_f.clear();
        for (int i = 0; i < task->_size; ++i)
        {
            if (task->adj_matrix[cur_vert][i])
            {
                assert(cur_vert != i);
                x_f.push_back({ i, task->adj_matrix[cur_vert][i] });
            }
        }

        cur_vert = RouletteWheelChoice(x_f);

        res.push_back(cur_vert);
        ++i;
        if (i > FAIL_COUNT)
            return vector<int>({ -1 });
    } while (cur_vert != end_vert);
}

```

```

        for (size_t i = 0; i < res.size()-1; i++)
        {
            assert(res[i] != res[i + 1]);
        }

        return res;
    }

void BeesAlgorithm::Solve(int start_vert, int end_vert)
{
    path init_path;
    do
    {
        init_path.first = LocalSearch(start_vert, end_vert);
    } while (init_path.first[0] == -1);
    CalcRouteLenght(init_path);

    for (size_t i = 0; i < empl_count; i++)
    {
        paths[i] = init_path;
    }
    best_ind = 0;
    vector < std::pair<x_value, f_value> > f_x;

    std::ofstream file(std::to_string(onlook_count) + "_" + std::to_string(empl_count) +
".txt");
    for (size_t c = 0; c < NUMBER_OF_ITERATION; c++)
    {
        std::cout << "Best length = " << paths[best_ind].second << '\n';
        file << c << ", " << paths[best_ind].second << '\n';
        for (int i = 0; i < empl_count; i++)
        {
            int pivot = rand() % paths[i].first.size();
            vector<int> old_part_path = {paths[i].first.begin(),
paths[i].first.begin() + pivot};
            vector<int> new_part_path;
            path new_path;
            new_part_path = LocalSearch(
                paths[i].first[pivot], paths[i].first.back());
            if (new_part_path[0] == -1)
                continue;

            std::move(old_part_path.begin(), old_part_path.end(),
std::back_inserter(new_path.first));
            std::move(new_part_path.begin(), new_part_path.end(),
std::back_inserter(new_path.first));

            CalcRouteLenght(new_path);
            if (new_path.second < paths[i].second)
                paths[i] = new_path;

            if (paths_mark[i].second > FAIL_SEARCH_COUNT && best_ind != i)
            {
                do
                {
                    new_path.first = LocalSearch(start_vert, end_vert);
                } while (new_path.first[0] == -1);
                CalcRouteLenght(new_path);

                paths[i] = new_path;
                paths_mark[i].second = 0;
            }
        }
    }
}

```

```

        best_ind = std::min_element(paths.begin(), paths.end(),
        [] (const path &a, const path &b) { return (a.second < b.second);}) -
paths.begin();

f_x.clear();
for (size_t i = 0; i < empl_count; i++)
{
    f_x.push_back({ i, float(1)/paths[i].second });
}

for (size_t i = 0, pth; i < onlook_count; i++)
{
    pth = RouletteWheelChoice(f_x);

    int pivot = rand() % paths[pth].first.size();
    vector<int> old_part_path = { paths[pth].first.begin(),
paths[pth].first.begin() + pivot };
    vector<int> new_part_path;
    path new_path;
    new_part_path = LocalSearch(
        paths[pth].first[pivot], paths[pth].first.back());
    if (new_part_path[0] == -1)
    {
        paths_mark[pth].second++;
        continue;
    }

    std::move(old_part_path.begin(), old_part_path.end(),
std::back_inserter(new_path.first));
    std::move(new_part_path.begin(), new_part_path.end(),
std::back_inserter(new_path.first));

    CalcRouteLenght(new_path);
    if (new_path.second < paths[pth].second)
    {
        paths[pth] = new_path;
        paths_mark[pth].second = 0;
    }
}
}

void BeesAlgorithm::PrintBest()
{
    for (auto& v : paths[best_ind].first)
    {
        std::cout << v << '-';
    }
}

```

## main.cpp

```
#include <time.h>
#include <random>
#include <iostream>

#include "Graph.h"
#include "BeesAlgorithm.h"

int main()
{
    int onlook;
    int employ;
    std::cout << "Amount of onlookers: "; std::cin >> onlook;
    std::cout << "Amount of employers: "; std::cin >> employ;

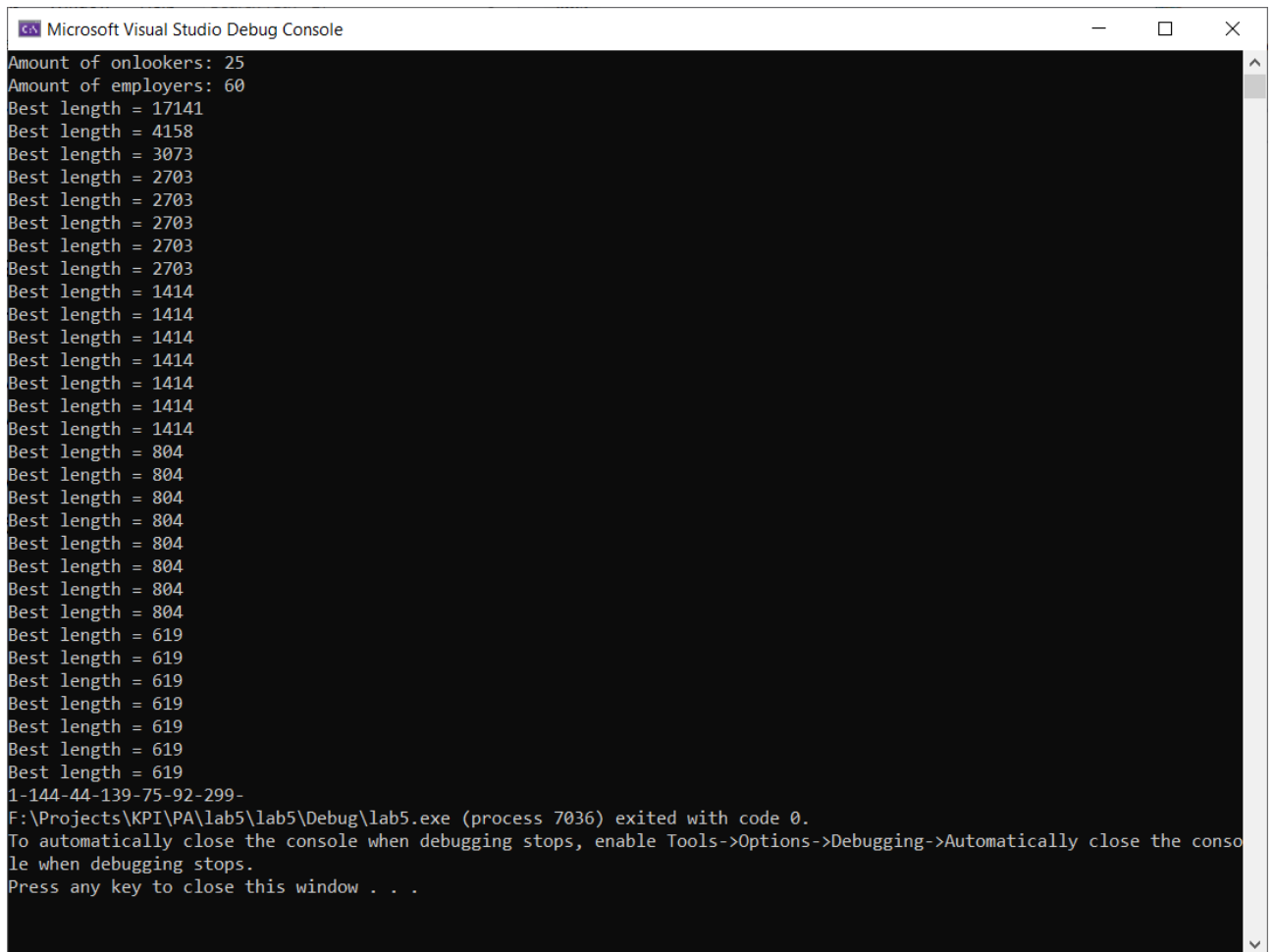
    BeesAlgorithm ba(onlook, employ);

    ba.Solve(1,299);
    ba.PrintBest();

    return 0;
}
```

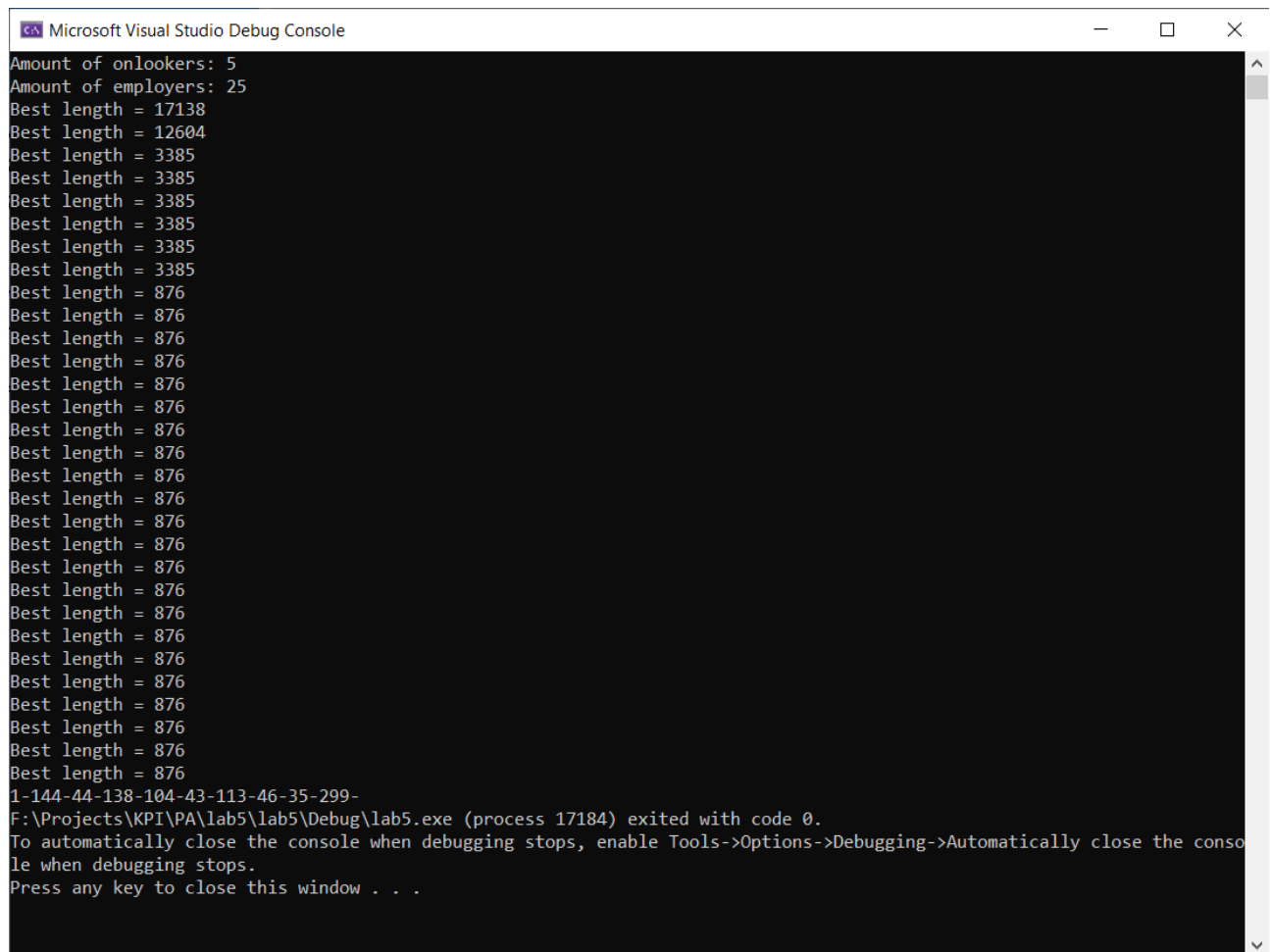
### 3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.



```
Microsoft Visual Studio Debug Console
Amount of onlookers: 25
Amount of employers: 60
Best length = 17141
Best length = 4158
Best length = 3073
Best length = 2703
Best length = 2703
Best length = 2703
Best length = 2703
Best length = 2703
Best length = 1414
Best length = 1414
Best length = 1414
Best length = 1414
Best length = 1414
Best length = 1414
Best length = 804
Best length = 804
Best length = 804
Best length = 804
Best length = 804
Best length = 804
Best length = 804
Best length = 804
Best length = 804
Best length = 619
Best length = 619
Best length = 619
Best length = 619
Best length = 619
Best length = 619
1-144-44-139-75-92-299-
F:\Projects\KPI\PA\lab5\lab5\Debug\lab5.exe (process 7036) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Рисунок 3.1 – Робота алгоритму з 25 спостерігачами та 60 робочими



```
Microsoft Visual Studio Debug Console
Amount of onlookers: 5
Amount of employers: 25
Best length = 17138
Best length = 12604
Best length = 3385
Best length = 3385
Best length = 3385
Best length = 3385
Best length = 3385
Best length = 3385
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
Best length = 876
1-144-44-138-104-43-113-46-35-299-
F:\Projects\KPI\PA\lab5\lab5\Debug\lab5.exe (process 17184) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

Рисунок 3.2 – Робота алгоритму з 5 спостерігачами та 25 робочими

### 3.3 Тестування алгоритму

Я тестував роботу алгоритму на 30-ти ітераціях.

Спочатку я зафіксував кількість спостерігачів та змінював кількість робочих.

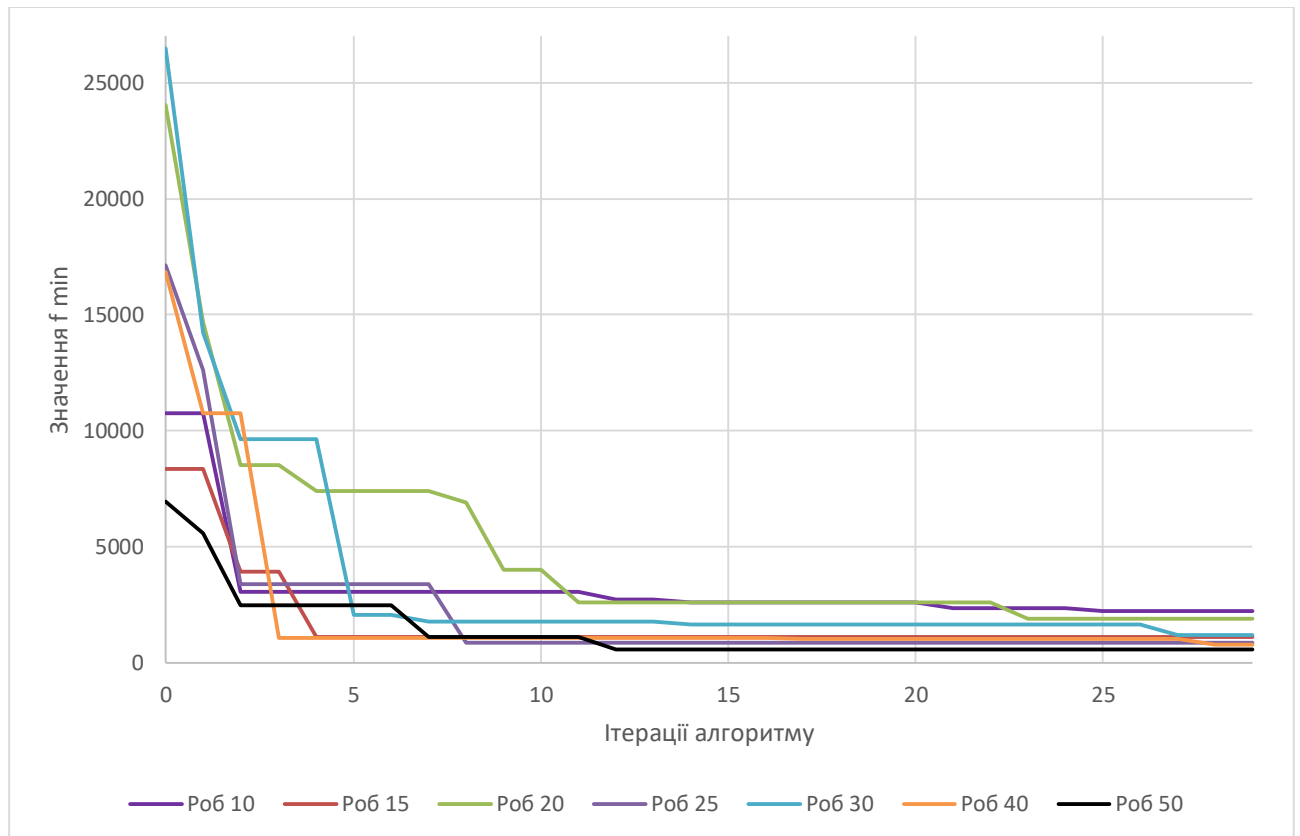


Рисунок 3.3 – Залежність ітерацій від к-сті робочих при 5 спостерігачах

Як бачимо, алгоритм збільшує свою ефективність при збільшенні робочих.

Проведемо схожу серію випробувань з 10 спостерігачами.



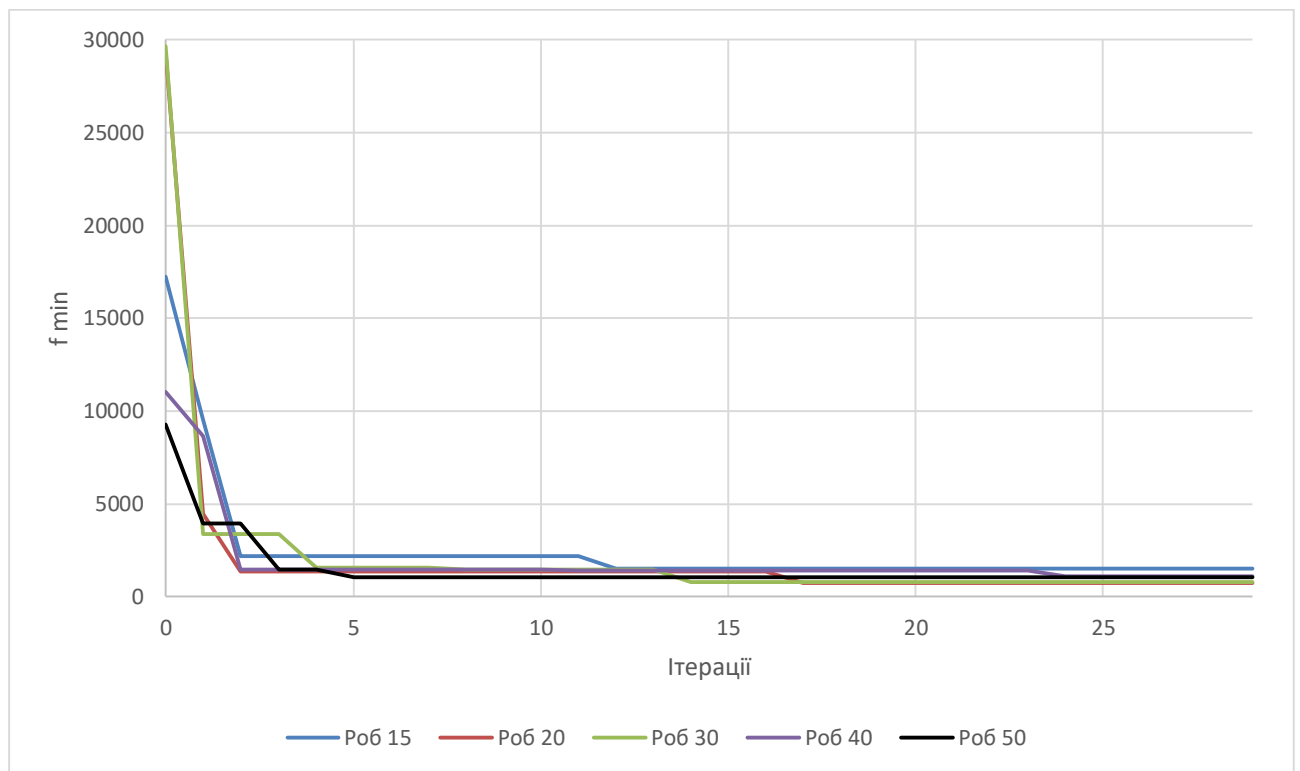


Рисунок 3.4 – Залежність ітерацій від к-сті робочих при 10 спостерігачах

Бачимо, що тепер пік ефективності на значеннях 20 та 30

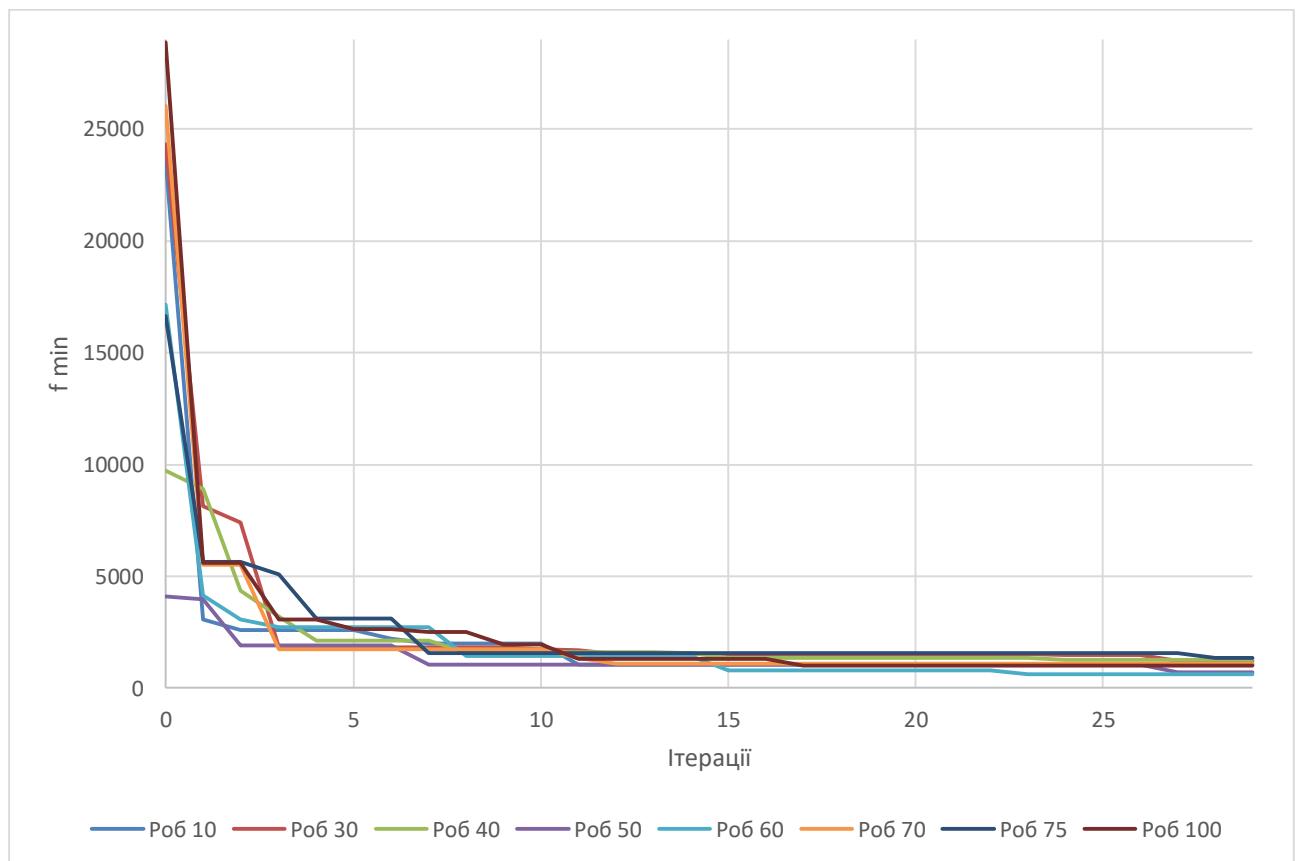


Рисунок 3.5 – Залежність ітерацій від к-сті робочих при 25 спостерігачах

Як бачимо, пік ефективності припадає на 50 та 60.

З отриманих даних можна зробити висновок, що к-сть робочих повинна бути приблизно у 2 рази більша, ніж спостерігачів.

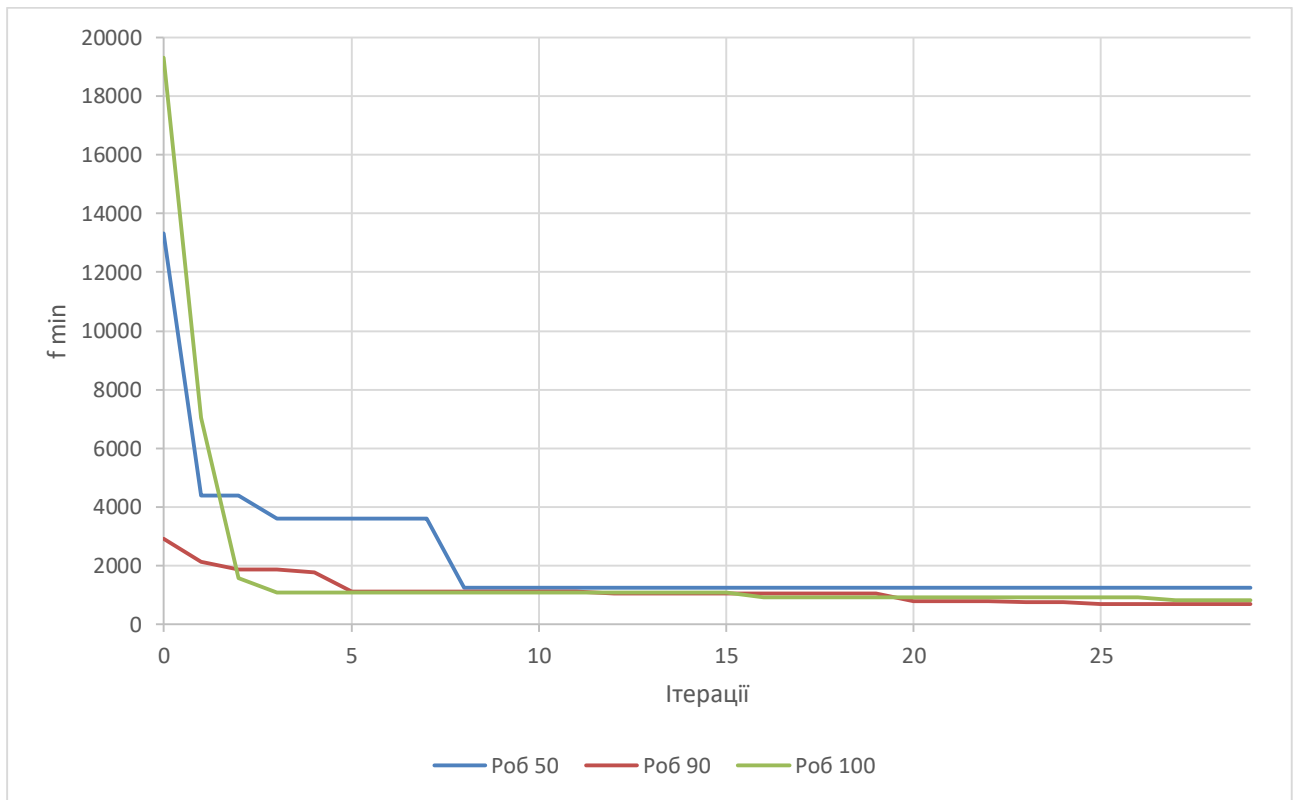


Рисунок 3.5 – Залежність ітерацій від к-сті робочих при 40 спостерігачах

Дійсно, коли робочих у 2 рази більше алгоритм працює краще.

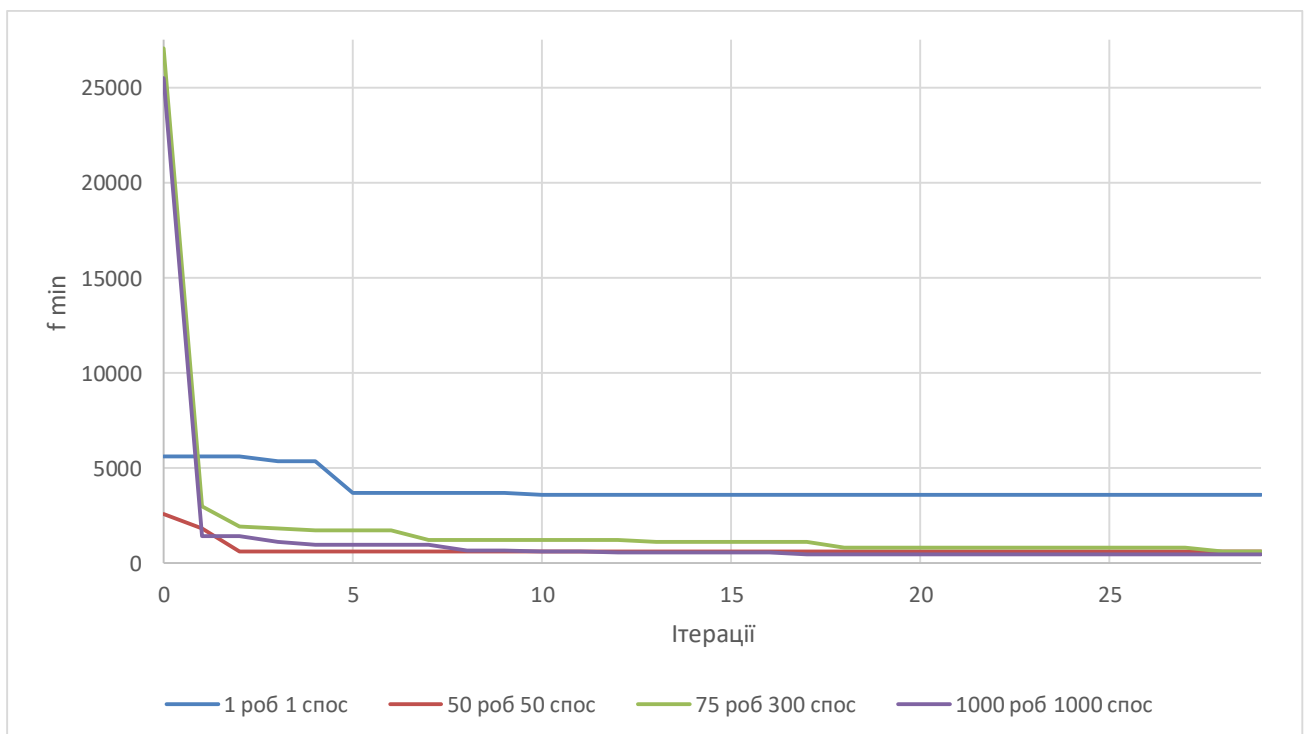


Рисунок 3.6 – Декілька цікавих випадків

## ВИСНОВОК

В рамках даної лабораторної роботи я зробив програмну реалізацію бджолиного алгоритму для пошуку найкратшого шляху в неорієнтованому графі. Мною було проведено серію тестувань, та встановлено, що найкращу ефективність алгоритм показує при кількості робочих, що вдічі більше за кількість спостерігачів та значень спостерігачів більших 15.

Я релізував локальний пошук, як рандомізований, причому ймовірність вибрати ребро обернено пропорційна його вазі. Ще в моєму локальному пошуку наявні обмеження щодо довжини побудованого шляху, який обмежено кількістю вершин – 1. Це обмеження враховує неможливість шляху бути довше ніж шлях у графі, що має усі вершини з'єднані підряд.

Іноді кінцевий результат містить цикл, або навіть декілька. Особливо при малих кількостях спостерігачів. Це пов'язано з тим, що локальний пошук не опирається на отримані раніше дані. Але при збільшенні кількості бджіл шлях з циклами або пропадає, або цикли «розгладжуються». Оскільки частіше обирається кращий шлях, навіть якщо він містить цикл, то рано чи пізно цикл пропадає.

Також я обмежую кількість невдалих пошуків у спостерігачів. У джерелі[2] вказано число 8, але я брав 10, аби розраховуючи на неточність свого алгоритму.

Якщо розглянути отримані графіки, то можна помітити, що графіки не завжди починаються однаково, ба більше початки дуже різні. Я вважаю, що це не сильно впливає на оцінку ефективності, адже уже з 2-3 ітераціями алгоритми виходять на майже однаковий рівень. Варто сказати, що у всіх випадках початковий граф однаковий, але вже рішення та початковий шлях завжди різні.

Якщо поставити кількість ітерацій 100, то алгоритм майже завжди знаходить оптимальне рішення, але це займає доволі багато часу. Так само, якщо поставити малу кількість ітерацій, алгоритм не встигатиме знайти рішення навіть при великій кількості бджіл. Цікавий випадок, коли

спостерігачів і робочих по 1000 штук кожних (рис. 3.6). Тут так само, як і у випадку великої кількості ітерацій: майже завжди знаходить оптимальне рішення, але займає дуже багато часу.

## СПИСОК ЛІТЕРАТУРИ

1. Modified artificial bee colony algorithm for solving mixed interval-valued fuzzy shortest path problem [Електронний документ]. URL: <https://d-nb.info/1230951229/34> (дата звернення 15.01.2023).
2. A novel artificial bee colony algorithm for shortest path problems with fuzzy arc weights [Електронний документ]. URL: <https://fardapaper.ir/mohavaha/uploads/2018/08/Fardapaper-A-novel-artificial-bee-colony-algorithm-for-shortest-path-problems-with-fuzzy-arc-weights.pdf> (дата звернення 15.01.2023).