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 № 4 з дисципліни
«Проектування алгоритмів»

„Проектування і аналіз алгоритмів для вирішення NP-складних задач ч.2”

Виконав(ла) *ІП-02 Василенко Павло Олександрович* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Вечєрковська А.С.* _____
(прізвище, ім'я, по батькові)

Київ 2020

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.

Таблиця 2.2 – Варіанти алгоритмів і досліджувані параметри

№	**Алгоритми і досліджувані параметри**
3	**Бджолиний алгоритм:** – кількість ділянок; – кількість бджіл (фуражирів і розвідників).

Таблиця 2.3 – Варіанти задач і алгоритмів

№	**Задачі і алгоритми**
2	Задача про рюкзак + Бджолиний алгоритм

3 ВИКОНАННЯ

3.1 Покроковий алгоритм

1) Випадкова генерація ділянок (можливих заповнень рюкзака)
2) Оцінка корисності знайдених ділянок
3) Вибір ділянок для пошуку їх в околиці
    a. Best scouts обирають кращі ділянки
    b. Random scouts обирають випадкові ділянки серед тих ,що залишилися
4) Відправка фуражирів
5) Перевірка того, чи кількість фуражирів не більша за степінь вершини(кількість предметів у рюкзаку)
6) Фуражири здійснюють пошук в околицях
    a. Кожен фуражир прибирає один з випадкових предметів
    b. Серед всіх предметів обираються найкращі необрані предмети(кращий предмет той, у якого price/weight набуває найбільшого значення)
    c. Заповнюємо рюкзак цими предметами, поки він може вміщати їх.
    d. Порівнюємо отриману Цінність послідовності предметів з початковою і обираємо ту, яка краща, та заміняємо початкову, якщо потрібно.
7) Для новоотриманих ділянок перераховуємо цінності, знаходимо найвищу та оновлюємо дані про краще рішення.
8) Повертаємося до п.2, який уже використовуватиме нові ділянки, n разів (кількість ітерацій)
9) Кінець роботи алгоритму

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

Laba2.cpp

```
#include <iostream>
#include <vector>
```

```cpp
#include "Item.h"
#include "Filehelper.h"
#include "BeeColonyAlgorithm.h"
#include <ctime>
using namespace std;


int main()
{
    srand(time(NULL));
    BeeColonyAlgorithm algo(500);
    algo.solve();
}
```

Filehelper.h

```cpp
#pragma once
#include <vector>
#include "Item.h"
#include <string>
#include <fstream>
#include <iostream>
using namespace std;

struct Filehelper
{
	static vector<Item> readItemsFromFile(string);
};
```

Filehelper.cpp

```cpp
#include "Filehelper.h"

vector<Item> Filehelper::readItemsFromFile(string fname)
{
	ifstream f(fname);
	vector<Item> result;

	if (!f.is_open()) {
		cout << "No such file";
		return result;
	}

	string str;
	while (getline(f, str))
	{
		if (str[0] == '#') continue;
		string name = str.substr(0, str.find(';'));
		str.erase(0, str.find(';') + 1);
		int weight = stoi(str.substr(0, str.find(';')));
		str.erase(0, str.find(';') + 1);
		int price = stoi(str.substr(0, str.size()));
		Item temp(name,weight,price);

		result.push_back(temp);
	}
	return result;
}
```

Item.h

```cpp
#pragma once
#include <string>
using namespace std;
```

```
class Item
{
	string name;
	int weight;
	int price;
public:
	Item(string, int, int);
	Item() { name = ""; weight = 0; price = 0; };
	string getName();
	int getWeight();
	int getPrice();
};
```

Item.cpp

```
#include "Item.h"

Item::Item(string name, int weight, int price)
{
	this->name = name;
	this->weight = weight;
	this->price = price;
}

string Item::getName()
{
	return this->name;
}

int Item::getWeight()
{
	return this->weight;
}

int Item::getPrice()
{
	return this->price;
}
```

Bagpack.h

```
#pragma once
#include "Item.h"
#include <vector>
#include "Filehelper.h"
#include <algorithm>
using namespace std;
class Bagpack
{
	vector<Item> itemList;
	vector<Item> allItems;
	int P;
public:
	Bagpack() { P = 0; };
	Bagpack(int);
	void createItems();
	vector<vector<bool>> generatePlots(int);
	int totalWeight(vector<bool>, int);
	int totalPrice(vector<bool>);
	vector<bool> sendForagers(vector<bool>, int);
	vector<int> getSortedByPricePerWeight();
	void displayItem(int);
};
```

Bagpack.cpp

```cpp
#include "Bagpack.h"

Bagpack::Bagpack( int P)
{
	this->P = P;
}

void Bagpack::createItems()
{
	vector<Item> allItems = Filehelper::readItemsFromFile("data.csv");
	this->allItems = allItems;
}

vector<vector<bool>> Bagpack::generatePlots(int number_plots)
{
	vector<vector<bool>> result;
	for (int i = 0; i < number_plots; i++)
	{
		vector<bool> plot;
		for (int j = 0; j < allItems.size(); j++)
		{
			plot.push_back(false);
		}
		bool added = true;
		while (added){
			added = false;
			int random = rand() % this->allItems.size();
			while (random < allItems.size() && plot[random]) {
				if (random == allItems.size() - 1) {
					random = -1;
				}
				random++;
			}
			int totalW = this->totalWeight(plot, allItems[random].getWeight());
			if (totalW >= this->P) {
				result.push_back(plot);
				break;
			}
			plot[random] = true;
			added = true;
		} while (added);

	}
	return result;
}

int Bagpack::totalWeight(vector<bool> taken, int toAdd=0)
{
	int res = 0;
	for (size_t i = 0; i < allItems.size(); i++)
	{
		if (taken[i]) {
			res += allItems[i].getWeight();
		}
	}
	res += toAdd;
	return res;
}

int Bagpack::totalPrice(vector<bool> taken)
{
	int count=0;
	for (size_t i = 0; i < taken.size(); i++)
	{
		if (taken[i]) {
```

```
                        count += this->allItems[i].getPrice();
                }
        }
        return count;
}

vector<bool> Bagpack::sendForagers(vector<bool> before, int number_foragers)
{
        //number of foragers can't be larger than степень of a plot
        vector<bool> copy = before;

        int countItems = 0;
        for (int i = 0; i < before.size(); i++)
        {
                if (copy[i]) {
                        countItems++;
                }
        }
        if (number_foragers > countItems) {
                number_foragers = countItems;
        }


        vector<int> selected;
        for (int i = 0; i < copy.size(); i++)
        {
                if (copy[i]) {
                        selected.push_back(i);
                }
        }
        random_shuffle(selected.begin(), selected.end());
        for (int i = 0; i < number_foragers; i++)
        {
                copy[selected[i]] = false;
        }
        vector<int> sortedByPricePerWeight_ALL = this->getSortedByPricePerWeight();
        vector<int> sortedByPricePerWeight;
        for (int i = 0; i < sortedByPricePerWeight_ALL.size(); i++)
        {
                if (find(selected.begin(), selected.end(), sortedByPricePerWeight_ALL[i]) ==
selected.end()) {
                        sortedByPricePerWeight.push_back(sortedByPricePerWeight_ALL[i]);
                }
        }
        int i = 0;
        while (this->totalWeight(copy) < this->P) {
                if (this->totalWeight(copy, allItems[sortedByPricePerWeight[i]].getWeight()) >
this->P) break;
                copy[sortedByPricePerWeight[i]] = true;
                i++;
        }
        return this->totalPrice(before)<this->totalPrice(copy)?copy:before;
}

vector<int> Bagpack::getSortedByPricePerWeight()
{
        vector<int> result;
        for (int i = 0; i < this->allItems.size(); i++)
        {
                result.push_back(i);
        }

        for (int i = 0; i < this->allItems.size(); i++) {
                for (int j = i + 1; j < this->allItems.size(); j++)
                {
```

```
                    float price_per_weight1 = static_cast<float>
(allItems[result[j]].getPrice()) / static_cast<float>(allItems[result[j]].getWeight());
                    float price_per_weight2 = static_cast<float>
(allItems[result[i]].getPrice()) / static_cast<float> (allItems[result[i]].getWeight());

                    if (price_per_weight1 > price_per_weight2) {
                        int temp = result[i];
                        result[i] = result[j];
                        result[j] = temp;
                    }
                }
        }
        return result;
}

void Bagpack::displayItem(int n)
{
        cout << '{' << allItems[n].getName() << ',' << allItems[n].getWeight() << ',' <<
allItems[n].getPrice() <<'}' << ',';
}
```

BeeColonyAlgorithm.h

```
#pragma once
#include <vector>
#include "Item.h"
#include "Bagpack.h"
#include <set>
class BeeColonyAlgorithm
{
        int number_of_scouts = 1,
                number_random_scouts = 0,
                number_best_scouts = number_of_scouts - number_random_scouts,
                number_iter = 100,
                number_plots = 100,
                number_items = 100,
                n_foragers = 10;

        vector<vector<bool>> plots;
        vector<int> prices;

        vector<bool> isTaken;
        Bagpack bagpack;

        int best_solution_price=0,
                best_solution_weight=0;
        vector<bool> best_solution;


        void sortPlotsByPrice();
public:
        BeeColonyAlgorithm(int);
        void solve();
};
```

BeeColonyAlgorithm.cpp

```
#include "BeeColonyAlgorithm.h"

void BeeColonyAlgorithm::sortPlotsByPrice()
{
        for (int i = 0; i < this->number_plots; i++) {
                for (int j = i + 1; j < this->number_plots; j++)
                {
                        if (this->prices[j] < this->prices[i]) {
```

```
                        vector<bool> temp = plots[i];
                        int temp_price = prices[i];
                        plots[i] = plots[j];
                        prices[i] = prices[j];
                        plots[j] = temp;
                        prices[j] = temp_price;
                    }
                }
            }
}

BeeColonyAlgorithm::BeeColonyAlgorithm( int P)
{
        this->bagpack = Bagpack(P);
        this->bagpack.createItems();

}

void BeeColonyAlgorithm::solve()
{
        this->plots = this->bagpack.generatePlots(this->number_plots);
        for (int i = 0; i < number_plots; i++)
        {
                prices.push_back(this->bagpack.totalPrice(plots[i]));
        }

        for (int iter = 0; iter < this->number_iter; iter++)
        {
                this->sortPlotsByPrice();
                vector<int> chosenPlots;
                for (int i = 0; i < this->number_best_scouts; i++)
                {
                        chosenPlots.push_back(i); //best scouts choose best solutions
                }
                set<int> randomSet;
                for (int i = 0; i < this->number_random_scouts; i++)
                {
                        int counter = 0;
                        int random = (rand() % (this->number_plots - this->number_best_scouts))
+ this->number_best_scouts;
                        while (randomSet.find(random) != randomSet.end()) {
                                counter++;
                                random = (rand() % (this->number_plots - this-
>number_best_scouts)) + this->number_best_scouts;
                                if (counter >= 100) return;
                        }
                        randomSet.insert(random);
                        chosenPlots.push_back(random);
                }


                for (int i = 0; i < chosenPlots.size(); i++)
                {
                        this->plots[i] = this->bagpack.sendForagers(this-
>plots[chosenPlots[i]], this->n_foragers);
                }
                for (int i = 0; i < chosenPlots.size(); i++)
                {
                        prices[i]=(this->bagpack.totalPrice(plots[i]));
                        if (prices[i] > this->best_solution_price) {
                                this->best_solution_price = prices[i];
                                this->best_solution_weight = this-
>bagpack.totalWeight(plots[i],0);
                                this->best_solution = plots[i];
                        }
```

```
			}

			//cout << "iteration: " << iter<<endl;

		}
		cout << "Best price: " << this->best_solution_price << endl;
		cout << "Weight for this price: " << this->best_solution_weight << endl;
		cout << "Items taken: " << endl;
		cout << "#Name,weight,price" << endl;
		for (int i = 0; i < this->best_solution.size(); i++)
		{
			if (this->best_solution[i]) {
				this->bagpack.displayItem(i);
			}
		}
		cout << endl;
}
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
Best price: 1019
Weight for this price: 498
Items taken:
#Name,weight,price
{2Item,9,29},{4Item,8,18},{7Item,9,25},{8Item,5,22},{10Item,10,25},{12Item,18,
10},{16Item,14,28},{18Item,20,24},{20Item,5,13},{26Item,9,21},{27Item,14,11},{
28Item,9,25},{29Item,12,3},{31Item,6,24},{32Item,12,29},{33Item,10,29},{34Item
,13,29},{38Item,3,13},{39Item,11,6},{40Item,12,21},{41Item,3,20},{42Item,6,26}
,{44Item,1,12},{47Item,7,5},{49Item,6,28},{51Item,10,26},{52Item,14,27},{53Ite
m,6,27},{59Item,8,27},{60Item,5,12},{61Item,9,21},{62Item,15,13},{63Item,16,26
},{64Item,13,14},{65Item,10,16},{67Item,8,12},{68Item,1,7},{69Item,4,22},{70It
em,3,18},{72Item,12,27},{74Item,15,8},{75Item,15,29},{76Item,9,19},{77Item,2,2
5},{79Item,8,14},{81Item,18,10},{89Item,10,25},{90Item,8,21},{92Item,2,29},{93
Item,11,9},{95Item,13,3},{98Item,8,14},{99Item,13,22},
```

Рисунок 3.1

‘

```
Best price: 1086
Weight for this price: 491
Items taken:
#Name,weight,price
{2Item,9,29},{4Item,8,18},{7Item,9,25},{8Item,5,22},{10Item,10,25},{11Item,7,8},{13Item,13,23},{14Item,9,22},{15Item,20,
18},{16Item,14,28},{17Item,17,20},{18Item,20,24},{19Item,15,18},{24Item,6,21},{25Item,15,16},{26Item,9,21},{28Item,9,25}
,{31Item,6,24},{32Item,12,29},{34Item,13,29},{38Item,3,13},{40Item,12,21},{41Item,3,20},{42Item,6,26},{44Item,1,12},{49I
tem,6,28},{52Item,14,27},{53Item,6,27},{59Item,8,27},{60Item,5,12},{61Item,9,21},{64Item,13,14},{65Item,10,16},{68Item,1
,7},{69Item,4,22},{70Item,3,18},{72Item,12,27},{75Item,15,29},{76Item,9,19},{77Item,2,25},{79Item,8,14},{82Item,20,25},{
87Item,16,26},{88Item,17,19},{89Item,10,25},{90Item,8,21},{92Item,2,29},{94Item,11,16},{97Item,10,19},{98Item,8,14},{99I
tem,13,22},
```

Рисунок 3.2

3.3 Тестування алгоритму

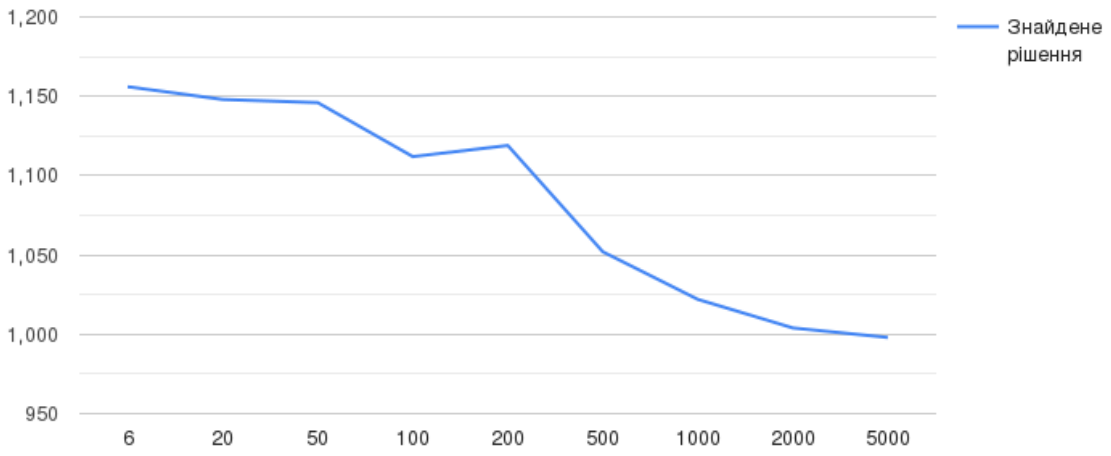

Рисунок 3.3 – Залежність якості рішення від кількості ділянок для 6 розвідників і 30 фуражирів

Рисунок 3.4 – Залежність якості рішення від кількості ділянок для 20 розвідників і 300 фуражирів

З першого графіка можемо зробити висновок: велика кількість ділянок руйнує якість результату. З другого графіку також очевидно, що коли кількість

розвідників дорівнює кількості ділянок результат є гіршим, адже «кращі» розвідники не мають з чого обирати, і сенсу в «випадкових» розвідниках нема. Гарні результати в обох випадках співпали на позначці в 50 ділянок, тому можна припустити, що відношення кількості ділянок до кількості предметів має бути 0,5:1 або 1:1 або 1,5:1.

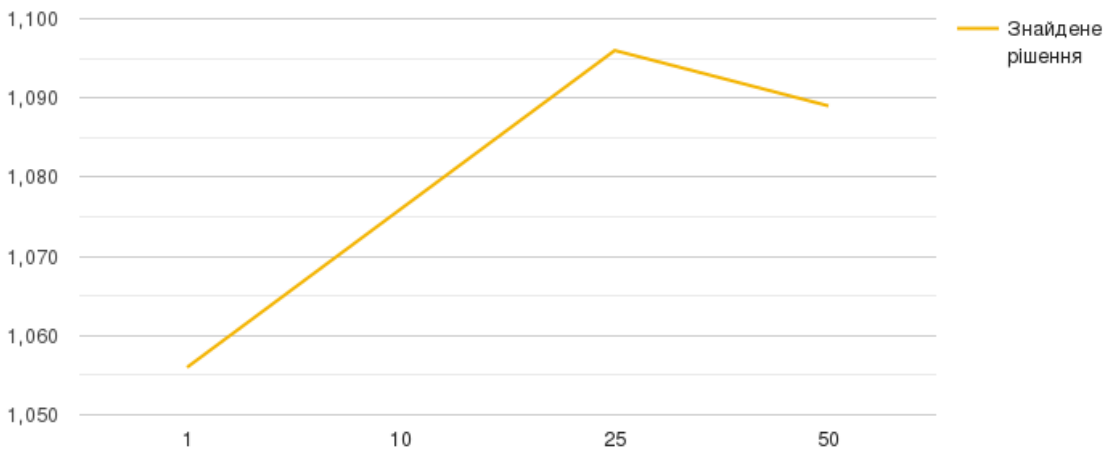

Рисунок 3.5 – Залежність якості рішення від кількості розвідників для 50 ділянок та (15*n, де n=кількість розвідників) фуражирів. Графік без випадкових розвідників

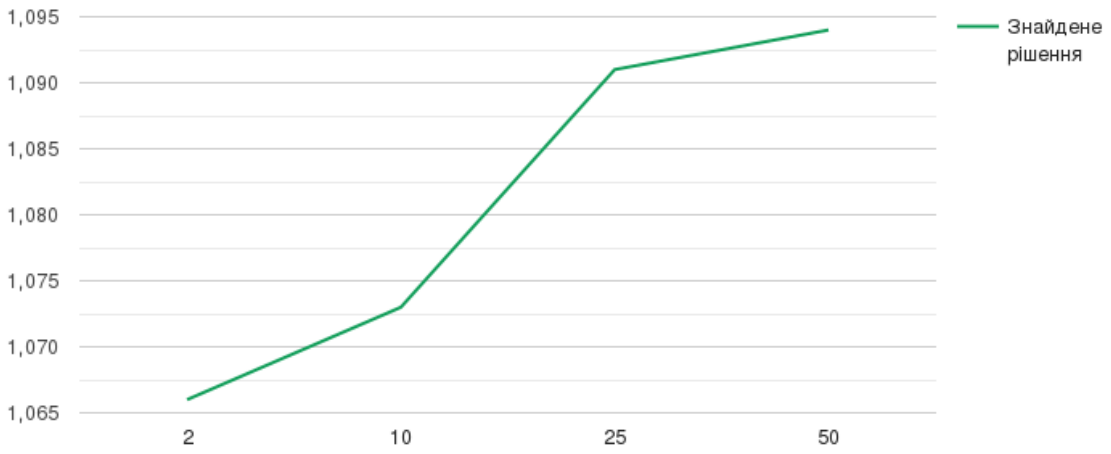

Рисунок 3.6 – Залежність якості рішення від кількості розвідників для 50 ділянок та (15*n, де n=кількість розвідників) фуражирів. Графік з випадковими розвідниками у відношенні 1:1

З першого графіку можемо побачити, що при збільшенні кількості розвідників у середньому якість результату збільшується. Проте через те, що у ньому не було «випадкових» розвідників, пошук нових, неочевидних ділянок був значно сповільнений, і надалі збільшення кількості розвідників спричинило погіршення результату. На другому ж графіку, де використовувалося відношення «кращих» розвідників до «випадкових» можемо бачити стабільний ріст якості рішення. Звісно, кількість розвідників не може перевищувати кількість ділянок.

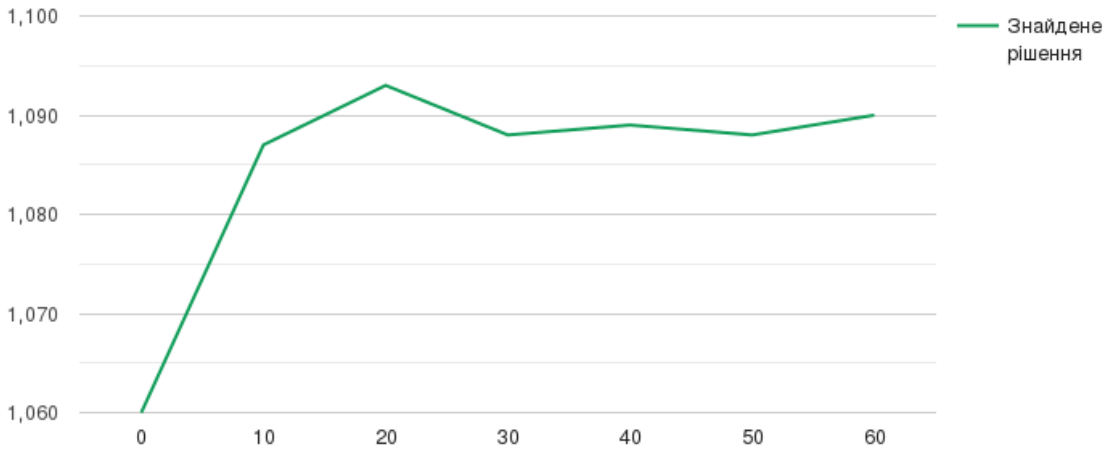

Рисунок 3.7 – Залежність якості рішення від кількості «випадкових» розвідників для 100 ділянок та (15*n, де n=кількість розвідників) фуражирів. Усього розвідників 60.

З графіку видно, що рішення є майже ідентичним у межах [1087 - 1095], незалежно від кількості «випадкових розвідників». Таке відбувається через те, що коли є «кращі» розвідники вони знайдуть краще рішення і фуражири будуть його покращувати. Існує ймовірність, що «випадкові» розвідники перевершать їх результат. Коли ж «кращих» розвідників взагалі нема — тоді маємо 100

ділянок, серед яких 60 оберуться випадково. Ймовірність попасти на хорошу ділянку все ще є високою. Також на графіку ще раз доведено той факт, що наявність «випадкових» розвідників є бажаним. Усе ж, краще рішення було знайдено у відношенні «випадкових» до «кращих» 1:2.

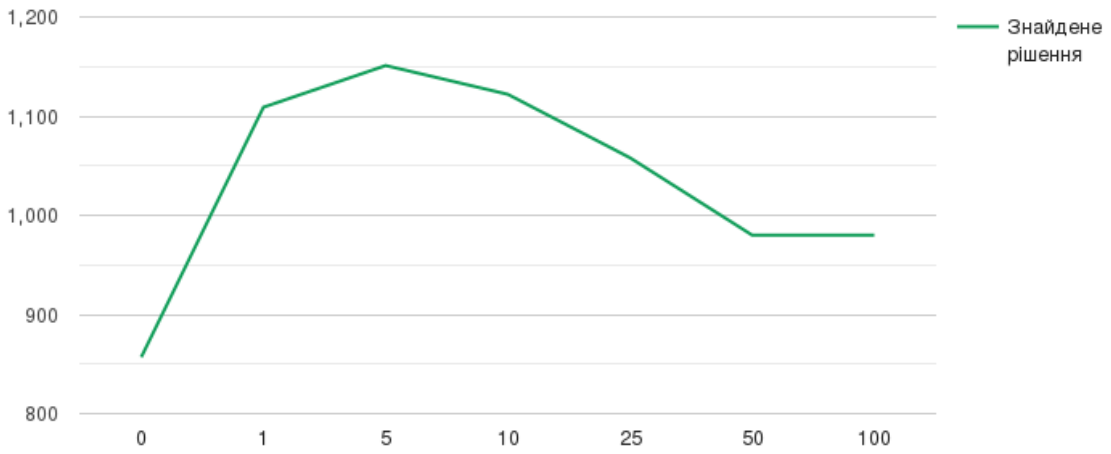

Рисунок 3.8 – Залежність якості рішення від кількості фуражирів на кожного розвідника для 100 ділянок та 24 розвідників(серед яких 8 випадкових).

Коли в алгоритмі беруть участь 0 фуражирів, це означає, що рішення не буде покращено шляхом локального пошуку. Вибравши від кількість фуражирів від 1 до 1/3 від кількості предметів маємо гарні результати роботи алгоритму, адже здійснюється якісний пошук кращих варіантів. Коли кількість фуражирів перевищує 1/3 з великою ймовірністю ця кількість буде перевищувати кількість предметів в рюкзаку на певній ділянці, через що фуражири просто виберуть предмети жадібним алгоритмом.

ВИСНОВОК

В рамках даної лабораторної роботи я навчився розв’язувати відому задачу про рюкзак. Розв’язано її було за допомогою бджолиного алгоритму. Було проведено тестування алгоритму.

Підведу підсумки тестування: для 100 предметів, цінність яких від 2 до 3, а вага від 1 до 20, рюкзак вміщує P = 500, 100 ітерацій. Гарним рішенням буде зробити відношення кількості ділянок до кількості предметів як 0,5:1 чи 1:1 чи 1,5:1. Відсоток розвідників до кількості ділянок – 10-30%. Відношення «випадкових» розвідників до «кращих» 1:2. Наявність «випадкових» розвідників є бажаним. Кількість фуражирів на одного розвідника – від 1 до їх кількості, це число не має перевищувати середню кількість предметів в рюкзаку.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 26.11.2021 включно максимальний бал дорівнює – 5. Після 26.11.2021 максимальний бал дорівнює – 1.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 15%;
- програмна реалізація алгоритму – 50%;
- тестування алгоритму– 30%;
- висновок – 5%.