

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.1”

Виконав(ла)

ІІІ-13 Пархомчук Ілля Вікторович
(шифр, прізвище, ім'я, по батькові)

Перевірив

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

Київ 2022

ЗМІСТ

1	МЕТА ЛАБОРАТОРНОЇ РОБОТИ.....	3
2	ЗАВДАННЯ	4
3	ВИКОНАННЯ	5
3.1	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ.....	5
3.1.1	<i>Вихідний код.....</i>	<i>5</i>
3.1.2	<i>Приклади роботи.....</i>	<i>14</i>
3.2	ТЕСТУВАННЯ АЛГОРИТМУ.....	15
3.2.1	<i>Значення цільової функції зі збільшенням кількості ітерацій..</i>	<i>15</i>
3.2.2	<i>Графіки залежності розв'язку від числа ітерацій.....</i>	<i>17</i>
	ВИСНОВОК	18
	КРИТЕРІЇ ОЦІНЮВАННЯ	ОШИБКА! ЗАКЛАДКА НЕ ОПРЕДЕЛЕНА.

1 МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Мета роботи – вивчити основні підходи формалізації метаевристичних алгоритмів і вирішення типових задач з їхньою допомогою.

2 ЗАВДАННЯ

Згідно варіанту, розробити алгоритм вирішення задачі і виконати його програмну реалізацію на будь-якій мові програмування.

Задача, алгоритм і його параметри наведені в таблиці 2.1.

Зафіксувати якість отриманого розв'язку (значення цільової функції) після кожних 20 ітерацій до 1000 і побудувати графік залежності якості розв'язку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

№	Задача і алгоритм
22	Задача про рюкзак (місткість $P=250$, 100 предметів, цінність предметів від 2 до 30 (випадкова), вага від 1 до 25 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування триточковий 25%, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

main.cpp

```
#include <iostream>
#include <array>
#include <time.h>
#include <random>
#include "main.h"

using std::cin;
using std::cout;

int main()
{
    srand(time(NULL));

    Population p1;

    for (size_t i = 0; i < 1000; i++)
    {
        if (!(i % 20))
            cout << p1.GetMaxWorth() << '\n';
        p1.Iterate();
    }
    p1.PrintMax();
    char key;
    int num;
    do
    {
        cin >> key;
        switch (key)
        {
            case 'i':
            case 'I': p1.Iterate(); p1.PrintMax(); break;
            case 'o':
            case 'O': cout << "Enter object number:"; cin >> num;
            p1.PrintObject(num); break;
            case 'p':
```

```

        case 'P': cout << "Enter individ number:"; cin >> num;
p1.PrintIndivid(num); break;
        case 'a':
        case 'A': p1.PrintWorthValue(); break;
        default:
            break;
    }
} while (key != 'q' && key != 'Q');
return 0;
}

```

main.h

```

#pragma once

#include <array>
#include <utility>

using std::array;
using std::pair;

constexpr int BAG_CAPACITY = 250;
constexpr int OBJJS_AMOUNT = 100;
constexpr int INDS_AMOUNT = 100;

constexpr int GENS_AMOUNT = OBJJS_AMOUNT;

constexpr int LOW_WORTH_BOUND = 2;
constexpr int UP_WORTH_BOUND = 30;

constexpr int LOW_WEIGHT_BOUND = 1;
constexpr int UP_WEIGHT_BOUND = 25;

constexpr int MUT_CHANCE = 5;
constexpr float OP_PERCENTAGE = 0.25;

struct OBJJS_WORTH
{
    const array<int, OBJJS_AMOUNT> value;
    OBJJS_WORTH();
};

```

```

struct OBJS_WEIGHT
{
    const array<int, OBJS_AMOUNT> value;
    OBJS_WEIGHT();
};

struct Task
{
    OBJS_WORTH worth;
    OBJS_WEIGHT weight;
};

class Population
{
private:
    struct Individ
    {
        int sum_value = 0;
        int weight = 0;
        bool genome[OBJS_AMOUNT] = {};

        inline bool IsAlive();
        void UpdateWeight(const OBJS_WEIGHT&);
        void UpdateWorth(const OBJS_WORTH&);
    };

    Individ individs[INDS_AMOUNT];
    int iteration = 0;
    int max_sum = 0;
    Individ* best;
    Task* task;
    int attraction[OBJS_AMOUNT];
public:
    Population();
    int GetIteration() const;

    void Iterate();

    void PrintWorthValue();
    void PrintIndivid(int n);
    void PrintObject(int n);
    void PrintMax();

```

```

        int GetMaxWorth();
        ~Population();
};

    Algs.cpp

#include <time.h>
#include <random>
#include "main.h"
#include <algorithm>
#include <iostream>
#include <iomanip>

OBJS_WORTH::OBJS_WORTH() : value { []{
    array<int, OBJS_AMOUNT> v;
    for (size_t i = 0; i < OBJS_AMOUNT; i++)
    {
        v[i] = rand() % UP_WORTH_BOUND + LOW_WORTH_BOUND;
    }

    return std::move(v); }() }
{
}

OBJS_WEIGHT::OBJS_WEIGHT() : value { []{
    array<int, OBJS_AMOUNT> v;
    for (size_t i = 0; i < OBJS_AMOUNT; i++)
    {
        v[i] = rand() % UP_WEIGHT_BOUND + LOW_WEIGHT_BOUND;
    }

    return std::move(v); }() }
{
}

bool Population::Individ::IsAlive()
{
    return weight <= BAG_CAPACITY;
}

void Population::Individ::UpdateWeight(const OBJS_WEIGHT& WEIGHT)
{
    weight = 0;

```



```

    for (size_t i = 0; i < OBJS_AMOUNT; ++i)
    {
        weight += genome[i] * WEIGHT.value[i];
    }
}

void Population::Individ::UpdateWorth(const OBJS_WORTH& WORTH)
{
    sum_value = 0;
    for (size_t i = 0; i < OBJS_AMOUNT; ++i)
    {
        sum_value += genome[i] * WORTH.value[i];
    }
}

Population::Population() : task( new Task())
{
    for (size_t i = 0; i < INDS_AMOUNT; i++)
    {
        individs[i].genome[i] = true;
        individs[i].sum_value += task->worth.value[i];
        individs[i].weight += task->weight.value[i];
    }

    best = individs + 0;
    max_sum = individs[0].sum_value;

    pair<int, float> attr[OBJS_AMOUNT];
    for (size_t i = 0; i < OBJS_AMOUNT; i++)
    {
        attr[i].first = i;
        attr[i].second = (float)task->worth.value[i] / task->weight.value[i];
    }
    std::sort(attr + 0, attr + OBJS_AMOUNT,
        [](const pair<int, float>& a, const pair<int, float>& b)
        {return a.second > b.second; });

    for (size_t i = 0; i < OBJS_AMOUNT; i++)
        attraction[i] = attr[i].first;
}

Population::~~Population()

```

```

{
    delete task;
}

int Population::GetIteration() const
{
    return iteration;
}

void Population::Iterate()
{
    Individ* parents[2];
    Individ offspring;

    auto cmp = [](const Individ& a, const Individ& b)
    { return a.sum_value < b.sum_value; };

    {
        int pivot = rand() % INDS_AMOUNT;

        parents[0] = std::max_element(individs + 0, individs + pivot, cmp);

        parents[1] = std::max_element(individs + pivot, individs + INDS_AMOUNT,
cmp);
    }

    {
        int pivot1 = OBJS_AMOUNT * OP_PERCENTAGE;
        int pivot2 = OBJS_AMOUNT * 2 * OP_PERCENTAGE;
        int pivot3 = OBJS_AMOUNT * 3 * OP_PERCENTAGE;

        memcpy(offspring.genome, parents[0]->genome, pivot1);
        memcpy(offspring.genome + pivot1, parents[1]->genome + pivot1, pivot2 -
pivot1);
        memcpy(offspring.genome + pivot2, parents[0]->genome + pivot2, pivot3 -
pivot2);
        memcpy(offspring.genome + pivot3, parents[1]->genome + pivot3,
GENS_AMOUNT - pivot3);

        offspring.UpdateWeight(task->weight);

        if (!offspring.IsAlive())
        {

```

```

        return;
    }
}

{
    if (rand() % 100 < MUT_CHANCE)
    {
        int point = rand() % INDS_AMOUNT;
        offspring.weight += (offspring.genome[point] =
!offspring.genome[point] ? 1 : -1) * task->weight.value[point];
        if (!offspring.IsAlive())
        {
            offspring.genome[point] = 0;
            offspring.weight -= task->weight.value[point];
        }
    }
}

{
    int num;

    for (int i = OBJS_AMOUNT - 1; i >= 0; --i)
    {
        num = attraction[i];
        if (offspring.genome[num])
        {
            offspring.genome[num] = false;
            offspring.weight -= task->weight.value[num];
            break;
        }
    }
    int free_space = BAG_CAPACITY - offspring.weight;
    for (size_t i = 0; i < OBJS_AMOUNT; i++)
    {
        num = attraction[i];
        if (!offspring.genome[num] && task->weight.value[num] <=
free_space)
        {
            offspring.genome[num] = true;
            offspring.weight += task->weight.value[num];
            break;
        }
    }
}

```

```

    }

    {
        offspring.UpdateWorth(task->worth);

        size_t min_index = std::min_element(individs + 0, individs +
OBJJS_AMOUNT, cmp) - (individs + 0);
        individs[min_index] = offspring;

        if (offspring.sum_value > max_sum)
        {
            max_sum = offspring.sum_value;
            best = individs + min_index;
        }
    }
}

void Population::PrintWorthValue()
{
    using namespace std;

    for (size_t i = 0; i < OBJJS_AMOUNT; i++)
    {
        cout << "No\t" << setw(3) << i << '\n';
        cout << "Weight\t" << setw(3) << task->weight.value[i] << '\n';
        cout << "Worth\t" << setw(3) << task->worth.value[i] << '\n';
        cout << "=====\n";
    }
}

void Population::PrintIndivid(int n)
{
    using namespace std;
    if (n < 0 || n >= OBJJS_AMOUNT)
    {
        return;
    }
    cout << "No\t" << setw(3) << n << '\n';
    cout << "Weight\t" << setw(3) << individs[n].weight << '\n';
    cout << "Worth\t" << setw(3) << individs[n].sum_value << '\n';
    cout << "=====\n";
}

```

```

    for (int i = 0; i < GENS_AMOUNT; i++)
    {
        cout << i << " : " << individs[n].genome[i] << "; ";
    }
    cout << endl;
}

void Population::PrintObject(int n)
{
    using namespace std;
    if (n < 0 || n >= OBJS_AMOUNT)
    {
        return;
    }
    cout << "No\t" << setw(3) << n << '\n';
    cout << "Weight\t" << setw(3) << task->weight.value[n] << '\n';
    cout << "Worth\t" << setw(3) << task->worth.value[n] << '\n';
    cout << "=====\n";
}

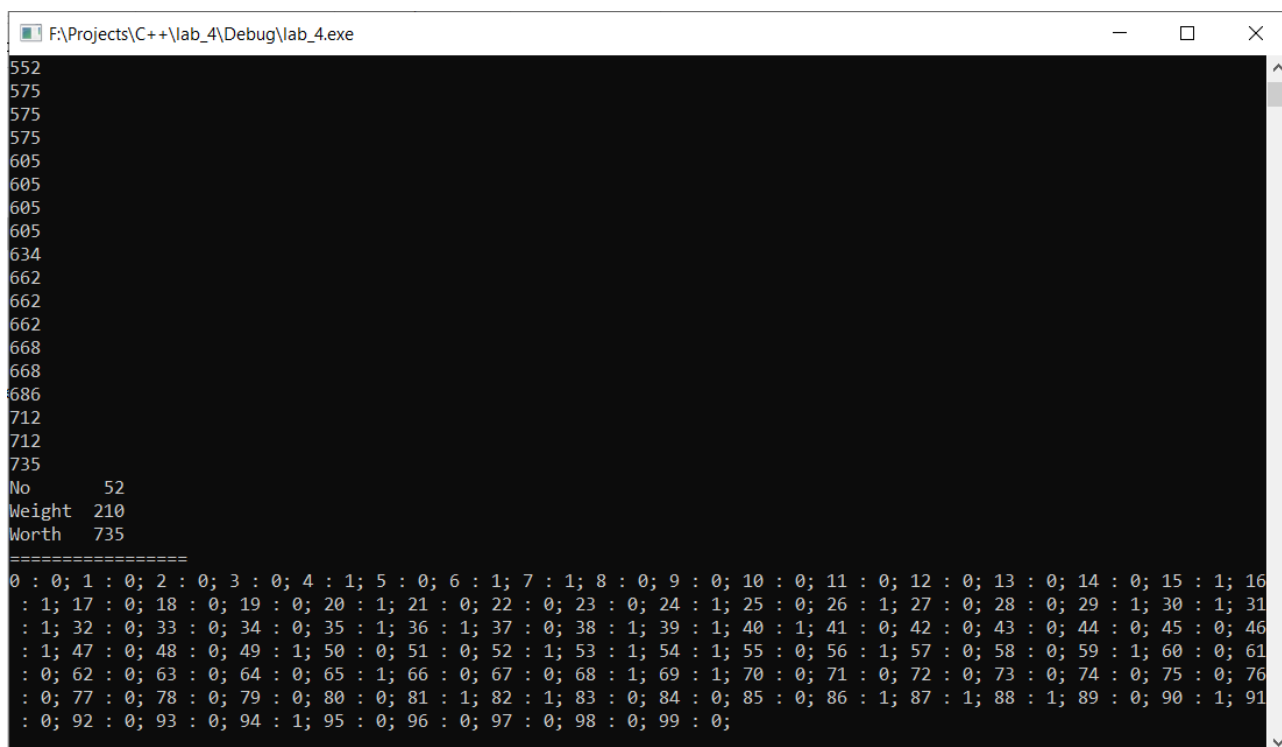
void Population::PrintMax()
{
    PrintIndivid(best - individs);
}

int Population::GetMaxWorth()
{
    return max_sum;
}

```

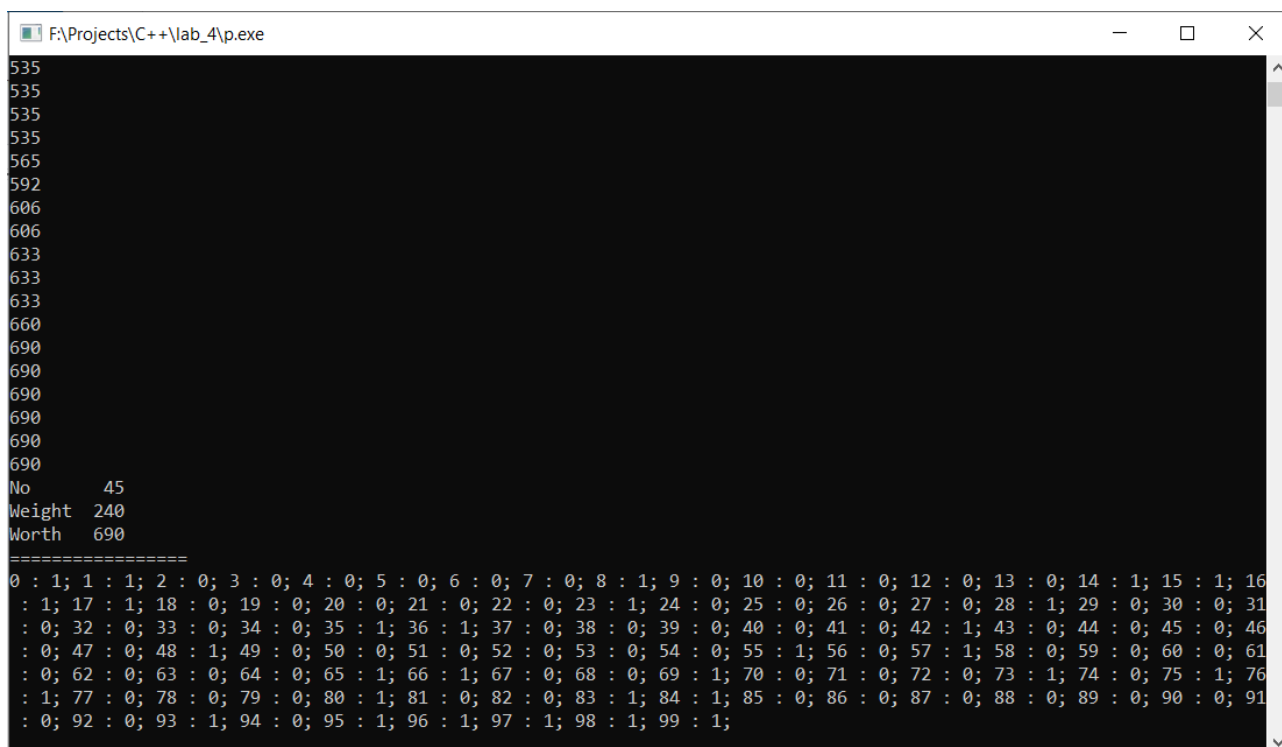
3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.



```
F:\Projects\C++\lab_4\Debug\lab_4.exe
552
575
575
575
605
605
605
605
634
662
662
662
668
668
686
712
712
735
No      52
Weight 210
Worth  735
=====
0 : 0; 1 : 0; 2 : 0; 3 : 0; 4 : 1; 5 : 0; 6 : 1; 7 : 1; 8 : 0; 9 : 0; 10 : 0; 11 : 0; 12 : 0; 13 : 0; 14 : 0; 15 : 1; 16 : 1; 17 : 0; 18 : 0; 19 : 0; 20 : 1; 21 : 0; 22 : 0; 23 : 0; 24 : 1; 25 : 0; 26 : 1; 27 : 0; 28 : 0; 29 : 1; 30 : 1; 31 : 1; 32 : 0; 33 : 0; 34 : 0; 35 : 1; 36 : 1; 37 : 0; 38 : 1; 39 : 1; 40 : 1; 41 : 0; 42 : 0; 43 : 0; 44 : 0; 45 : 0; 46 : 1; 47 : 0; 48 : 0; 49 : 1; 50 : 0; 51 : 0; 52 : 1; 53 : 1; 54 : 1; 55 : 0; 56 : 1; 57 : 0; 58 : 0; 59 : 1; 60 : 0; 61 : 0; 62 : 0; 63 : 0; 64 : 0; 65 : 1; 66 : 0; 67 : 0; 68 : 1; 69 : 1; 70 : 0; 71 : 0; 72 : 0; 73 : 0; 74 : 0; 75 : 0; 76 : 0; 77 : 0; 78 : 0; 79 : 0; 80 : 0; 81 : 1; 82 : 1; 83 : 0; 84 : 0; 85 : 0; 86 : 1; 87 : 1; 88 : 1; 89 : 0; 90 : 1; 91 : 0; 92 : 0; 93 : 0; 94 : 1; 95 : 0; 96 : 0; 97 : 0; 98 : 0; 99 : 0;
```

Рисунок 3.1 –



```
F:\Projects\C++\lab_4\p.exe
535
535
535
535
565
592
606
606
633
633
633
660
690
690
690
690
690
No      45
Weight 240
Worth  690
=====
0 : 1; 1 : 1; 2 : 0; 3 : 0; 4 : 0; 5 : 0; 6 : 0; 7 : 0; 8 : 1; 9 : 0; 10 : 0; 11 : 0; 12 : 0; 13 : 0; 14 : 1; 15 : 1; 16 : 1; 17 : 1; 18 : 0; 19 : 0; 20 : 0; 21 : 0; 22 : 0; 23 : 1; 24 : 0; 25 : 0; 26 : 0; 27 : 0; 28 : 1; 29 : 0; 30 : 0; 31 : 0; 32 : 0; 33 : 0; 34 : 0; 35 : 1; 36 : 1; 37 : 0; 38 : 0; 39 : 0; 40 : 0; 41 : 0; 42 : 1; 43 : 0; 44 : 0; 45 : 0; 46 : 0; 47 : 0; 48 : 1; 49 : 0; 50 : 0; 51 : 0; 52 : 0; 53 : 0; 54 : 0; 55 : 1; 56 : 0; 57 : 1; 58 : 0; 59 : 0; 60 : 0; 61 : 0; 62 : 0; 63 : 0; 64 : 0; 65 : 1; 66 : 1; 67 : 0; 68 : 0; 69 : 1; 70 : 0; 71 : 0; 72 : 0; 73 : 1; 74 : 0; 75 : 1; 76 : 1; 77 : 0; 78 : 0; 79 : 0; 80 : 1; 81 : 0; 82 : 0; 83 : 1; 84 : 1; 85 : 0; 86 : 0; 87 : 0; 88 : 0; 89 : 0; 90 : 0; 91 : 0; 92 : 0; 93 : 1; 94 : 0; 95 : 1; 96 : 1; 97 : 1; 98 : 1; 99 : 1;
```

Рисунок 3.2 –

3.2 Тестування алгоритму

3.2.1 Значення цільової функції зі збільшенням кількості ітерацій

У таблиці 3.1 наведено значення цільової функції зі збільшенням кількості ітерацій.

Iteration	Max F Value
0	30
20	89
40	134
60	134
80	134
100	134
120	182
140	197
160	225
180	251
200	262
220	267
240	267
260	318
280	348
300	348
320	348
340	378
360	378
380	429
400	458
420	458
440	458

460	487
480	487
500	487
520	487
540	487
560	505
580	505
600	505
620	505
640	553
660	553
680	553
700	583
720	583
740	613
760	613
780	644
800	672
820	672
840	672
860	672
880	701
900	701
920	729
940	729
960	745
980	765
1000	765

3.2.2 Графіки залежності розв'язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв'язку.

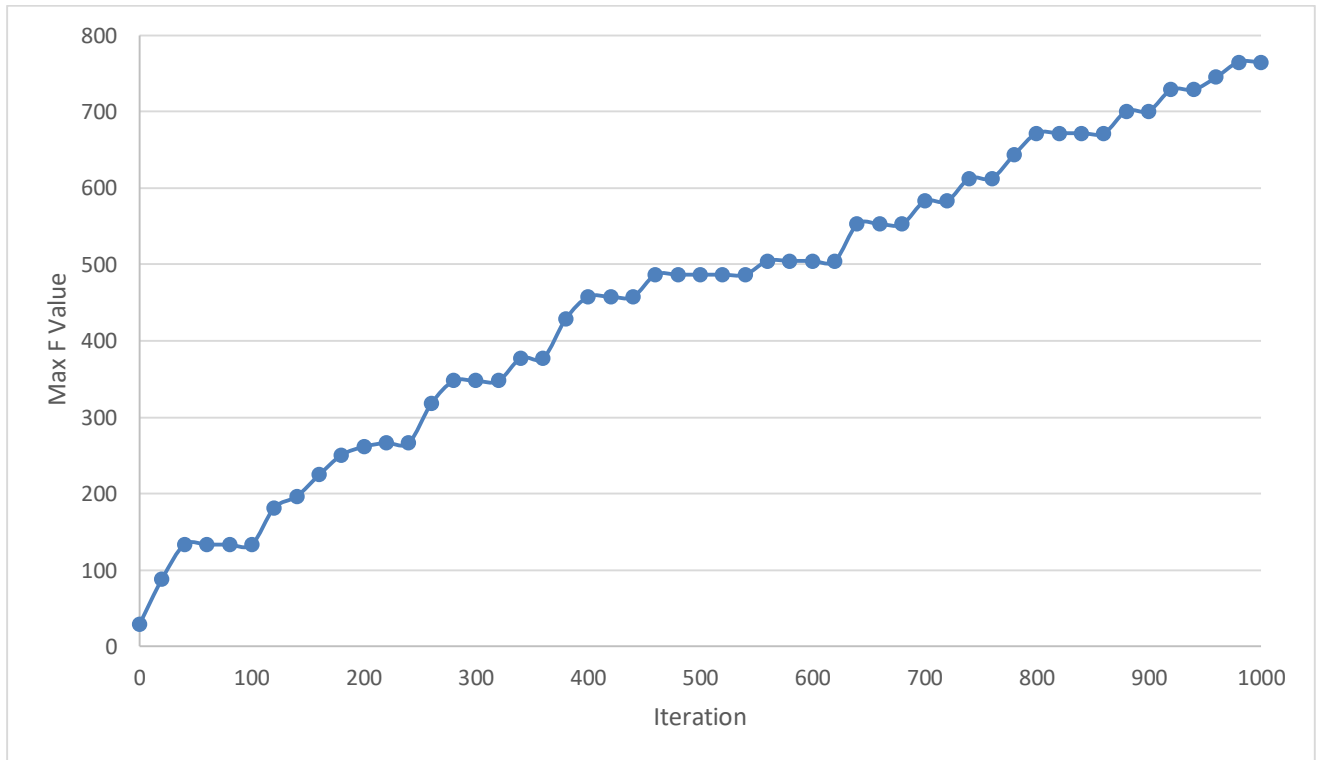


Рисунок 3.3 – Графіки залежності розв'язку від числа ітерацій

ВИСНОВОК

В рамках даної лабораторної роботи я познайомився з метаевристичними алгоритмами. Мною була написана реалізація генетичного алгоритму для задачі про рюкзак з місткістю 250 та кількістю предметів 100. Я використав турнірний відбір: розбивав популяцію на дві групи та з цих груп відбирав найкращих. Розроблене мною локальне покращення полягало у видаленні найнепривабливішого предмету з рюкзака та додаванні найпривабливішого, що поміщається. Привабливість визначалась за формулою $\text{привабливість} = \frac{\text{вартість}}{\text{вага}}$. Тобто предмет, що його вартість набагато перевищує вагу є привабливим. В ході тестування я переконався, що мутації дійсно важливі, адже без них алгоритм заходив у глухий кут. Мною було зіставлено таблицю ітерацій та максимального значення функції (цінності рюкзака) та на основі неї побудовано графік залежності.

З графіка видно, що еволюція іде приблизно лінійно, але з'являються плато і стрибки – ділянки з глухим кутом та мутацією, що вивела з глухого кута відповідно.

Також я помітив, що якщо зробити схрещування не по 25%, а випадковими ділянками, то процес еволюції досягав оптимального значення значно швидше (к-сть ітерацій < 1000). З 25% процес еволюції також доходить до оптимального рішення, але воно лежить за 1000 ітерацією.