**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”**
Варіант 24

**Виконав(ла)** *ІП-12 Титаренко Данило Олегович* _____
(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.

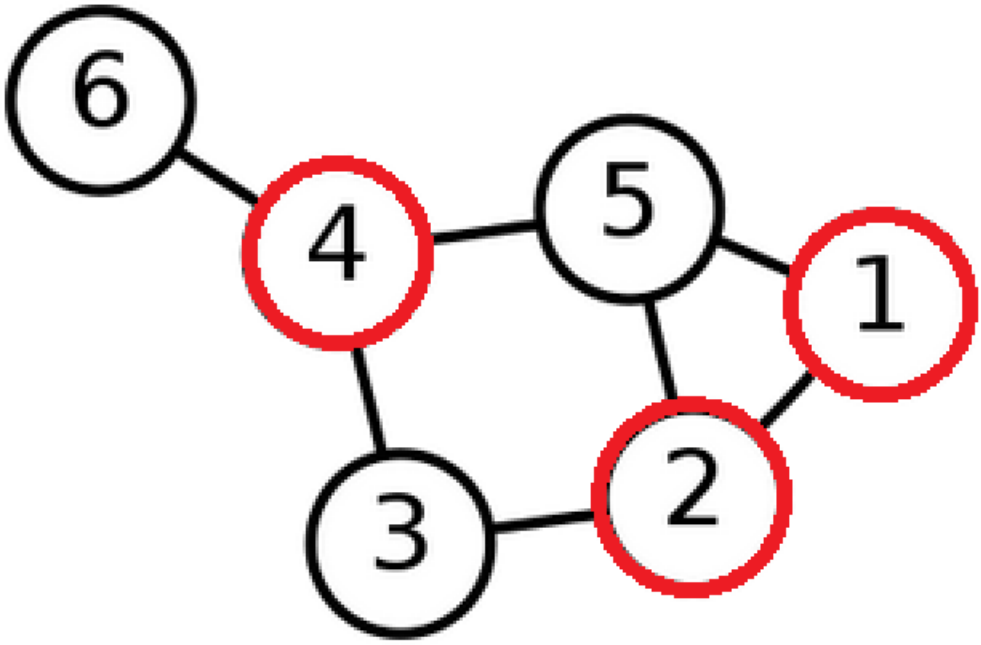
Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

| **№** | **Задача** |
|---|---|
| 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.<br>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.<br>**Розглядається симетричний, асиметричний та змішаний варіанти.**<br>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.<br>У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.<br>У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху.<br>Застосування:<br>– доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів);<br>– доставка води;<br>– моніторинг об'єктів; |

| | |
|---|---|
| | – поповнення банкоматів готівкою;<br>– збір співробітників для доставки вахтовим методом. |
| 3 | **Розфарбовування графа** (300 вершин, степінь вершини не більше 30, але не менше 2) – називають таке 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.<br>Застосування:<br>– розкладу для освітніх установ;<br>– розкладу в спорті;<br>– планування зустрічей, зборів, інтерв'ю;<br>– розклади транспорту, в тому числі - авіатранспорту;<br>– 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.<br>Задача вершинного покриття полягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).<br>На вході: Граф G = (V, E).<br>Результат: множина C ⊆ V - найменше вершинне покриття графа G. |

| | |
|---|---|
| |  <br>Застосування:<br>– розміщення пунктів обслуговування;<br>– призначення екіпажів на транспорт;<br>– 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.<br>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).<br>Застосування:<br>– біоінформатика;<br>– 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.<br>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:**<br>- оператор схрещування (мінімум 3);<br>- мутація (мінімум 2);<br>- оператор локального покращення (мінімум 2). |
| 2 | **Мурашиний алгоритм**:<br>– α;<br>– β;<br>– ρ;<br>– Lmin;<br>– кількість мурах M і їх типи (елітні, тощо…);<br>– маршрути з однієї чи різних вершин. |
| 3 | **Бджолиний алгоритм:**<br>– кількість ділянок;<br>– 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

## 3.1 Покроковий алгоритм

1. Згенеруємо 100 предметів
2. Згенеруємо пакування рюкзака допустимим але не найбільш оптимальним варіантом(наприклад жадібним алгоритмом)
3. Потім відправляємо бджіл-розвідників на ділянки(ділянка-одне пакування рюкзака)
4. Серед розвідників 80% - з найвищою цінністю і 20% - це випадкові ділянки з тих, що лишилися . Це потрібно, щоб уникнути локальних рішень
5. Бджола-розвідник буде оцінювати перспективність ділянки, тобто чим більша цінність рюкзака – тим більш перспективна ділянка
6. Потім бджоли-фуражири здійснюють заміни на своїх ділянках, тобто збільшити цінність рюкзаку, по черзі згідно з перспективністю ділянки
7. Бджола-фуражир пробує замінити всі предмети у рюкзаку відповідно до густини предмету, тобто цінність предмета поділена на вагу, а також пробує додати предмет до рюкзаку, якщо є ще вільне місце. Таким чином у рюкзаку залишаються найбільш цінні предмети

## 3.2 Програмна реалізація алгоритму

### 3.2.1 Вихідний код

```
import random

SCOUTS = 5
BEST_SCOUTS = 4
RANDOM_SCOUTS = SCOUTS - BEST_SCOUTS
FORAGERS = 45



class Knapsack:
```

```
    def __init__(self, items):
        self.items = items
        self.value = calc_value(self)
        self.amount=len(items)
        self.weight=calc_weight(self)

class Item:
    def __init__(self, value, weight):
        self.value = value
        self.weight = weight

    def __eq__(self, other):
        if not isinstance(other, Item):
            return NotImplemented

        return self.value == other.value and self.weight == other.weight

def generate_items(n):
    arr=[]
    for i in range(n):
        arr.append(Item(random.randint(2,30),random.randint(1,20)))
    return arr

def random_knapsack(W,arr):
    unpacked=True
    final=[]
    exist=[]
    finalvalue = 0.0
    while(unpacked):
        index=random.randint(1,len(arr)-1)
        if not(index in exist):
          W -= arr[index].weight
          if(W<0):
            W += arr[index].weight
            unpacked=False
          else:
            finalvalue += arr[index].value
            final.append(arr[index])
            exist.append(index)
    return Knapsack(final)

def greedy(W, arr):

    arr.sort(key=lambda x: (x.value/x.weight), reverse=True)
    begin=W
    finalvalue = 0.0
    final=[]

    for item in arr:
        final.append(item)
```

```
        if item.weight <= W:
            W -= item.weight
            finalvalue += item.value

        else:
            finalvalue += item.value * W / item.weight
            break

    return Knapsack(final)

def calc_value(knapsack):
    total=0
    for item in knapsack.items:
        total=total+item.value
    return total

def calc_weight(knapsack):
    total=0
    for item in knapsack.items:
        total=total+item.weight
    return total

def change_item(knapsack,arr,W,index):
    items=knapsack.items
    left_items=[]
    W_left=W-knapsack.weight
    max_val=knapsack.value

    for item in arr:
        if not(item in items):
            left_items.append(item)

    items.sort(key=lambda x: (x.value/x.weight), reverse=False)
    left_items.sort(key=lambda x: (x.value/x.weight), reverse=True)

    if((items[index].value/items[index].weight)<(left_items[0].value/left_items[0].
weight)):
        W_left=W_left+items[index].weight
        begin_val=max_val-items[index].value
        best_item_index=0
        find=False
        for i in range(len(left_items)):
            if(left_items[i].weight<W_left and
begin_val+left_items[i].value>=max_val):
                max_val=begin_val+left_items[i].value
                best_item_index=i
                find=True
        if(find):
            W_left=W_left-left_items[best_item_index].weight
            items[index]=left_items[best_item_index]
```

```
            left_items.remove(left_items[best_item_index])

    if(W_left>0):
        for item in left_items:
            if(item.weight<W_left):
                items.append(item)
                W_left=W_left-item.weight
                max_val=max_val+item.value
                left_items.remove(item)

    knapsack.value=max_val
    knapsack.weight=W-W_left
    knapsack.items=items
    return knapsack

def create_random_scouts(arr, W,scouts, n):
    while len(scouts) < n:
        scout=random_knapsack(W,arr)
        if scout not in scouts:
            scouts.append(scout)

def create_best_scouts(arr, W,scouts, n):
    while len(scouts) < n:
        scout=greedy(W,arr)
        if scout not in scouts:
            scouts.append(scout)

def local_search(arr,W,scouts,foragers):
    scout_result=[]
    scouts.sort(key=lambda x: (x.value), reverse=True)
    for scout in scouts:
        n=scout.amount
        if(foragers<n):
            n=foragers
        for i in range(n):
            scout=change_item(scout,arr,W,i)
            scout_result.append(scout)
    return scout_result

def get_best(scouts):
    scouts.sort(key=lambda x: (x.value), reverse=True)
    return scouts[0]


if __name__ == "__main__":
    W = 500
    arr = generate_items(100)
    scouts=[]
    best_result=Knapsack([Item(0,0)])
    create_best_scouts(arr,W,scouts,BEST_SCOUTS)
    create_random_scouts(arr,W,scouts,SCOUTS)
```

```
    for i in range(1000):
        if  (i % 20==0 and i!=0):
            print(f"i: {i}, max value: {best_result.value}")
        if(i==0):
            best=get_best(scouts)
            print(f"Begin Knapsack\n i: {i},  value: {best.value}")
        results = local_search(arr,W,scouts,FORAGERS)
        # print(get_best(results).value)
        if(get_best(results).value>best_result.value):
            best_result=get_best(results)
        scouts=[]
        create_best_scouts(arr,W,scouts,BEST_SCOUTS)
        create_random_scouts(arr,W,scouts,SCOUTS)
    print(f"Best knapsack:\n Weight: {best_result.weight} \n Value:
{best_result.value}")
```

### 3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

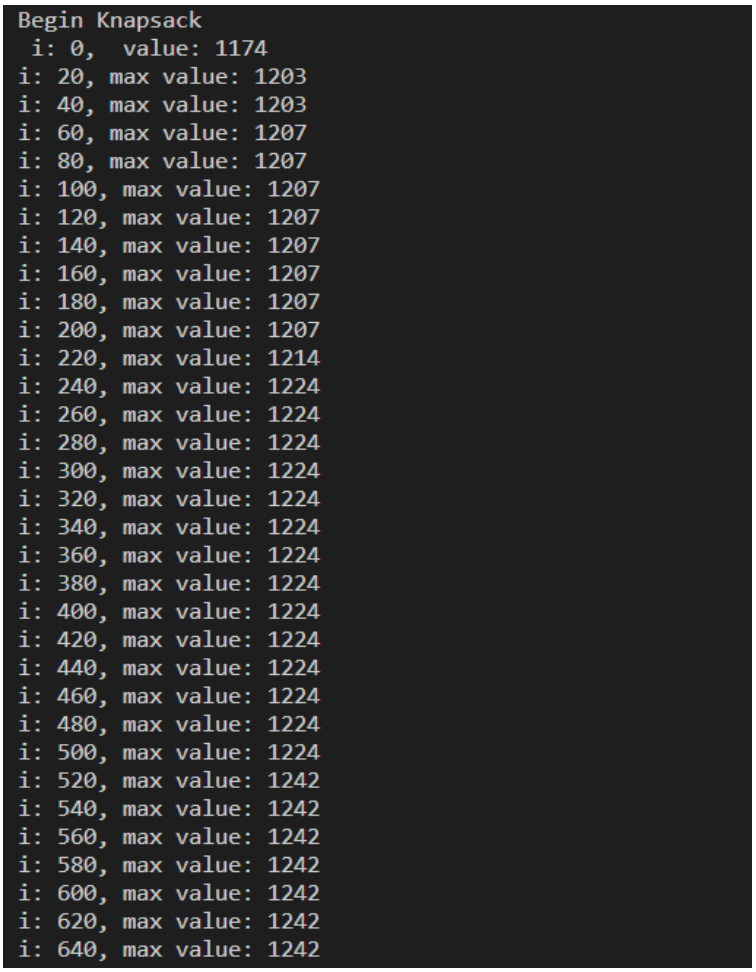



Рисунок 3.1 – Приклад роботи програми

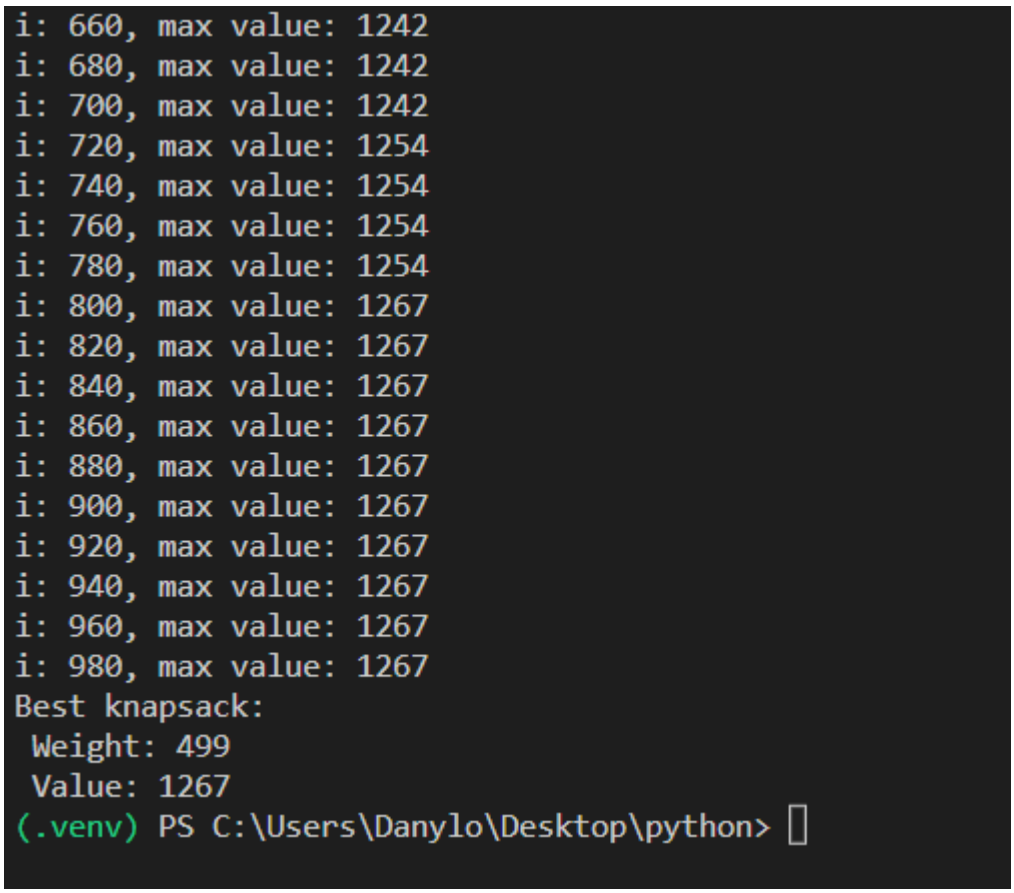



Рисунок 3.2 – Приклад роботи програми

## 3.3 Тестування алгоритму

- Таблиця залежності максимальної цінності рюкзака вікількості ітерацій

Початкова цінність предметів у рюкзаку - 1174

| Ітерація | Значення функції | Ітерація | Значення функції |
|---|---|---|---|
| 20 | 1203 | 520 | 1242 |
| 40 | 1203 | 540 | 1242 |
| 60 | 1207 | 560 | 1242 |
| 80 | 1207 | 580 | 1242 |
| 100 | 1207 | 600 | 1242 |
| 120 | 1207 | 620 | 1242 |
| 140 | 1207 | 640 | 1242 |
| 160 | 1207 | 660 | 1242 |
| 180 | 1207 | 680 | 1242 |
| 200 | 1207 | 700 | 1242 |
| 220 | 1214 | 720 | 1254 |
| 240 | 1224 | 740 | 1254 |
| 260 | 1224 | 760 | 1254 |
| 280 | 1224 | 780 | 1254 |
| 300 | 1224 | 800 | 1267 |
| 320 | 1224 | 820 | 1267 |
| 340 | 1224 | 840 | 1267 |
| 360 | 1224 | 860 | 1267 |
| 380 | 1224 | 880 | 1267 |
| 400 | 1224 | 900 | 1267 |
| 420 | 1224 | 920 | 1267 |
| 440 | 1224 | 940 | 1267 |
| 460 | 1224 | 960 | 1267 |
| 480 | 1224 | 980 | 1267 |

- Таблиця залежності цінності рюкзака від кількості бджіл. Серед бджіл 10% будуть бджолами-розвідниками

Протестуємо алгоритм для конкретного набору предметів

| К-ть бджіл | Цінність рюкзака |
|---|---|
| 12 | 968 |
| 22 | 1007 |
| 33 | 1020 |
| 44 | 1054 |
| 50 | 1075 |
| 55 | 1033 |
| 61 | 1034 |

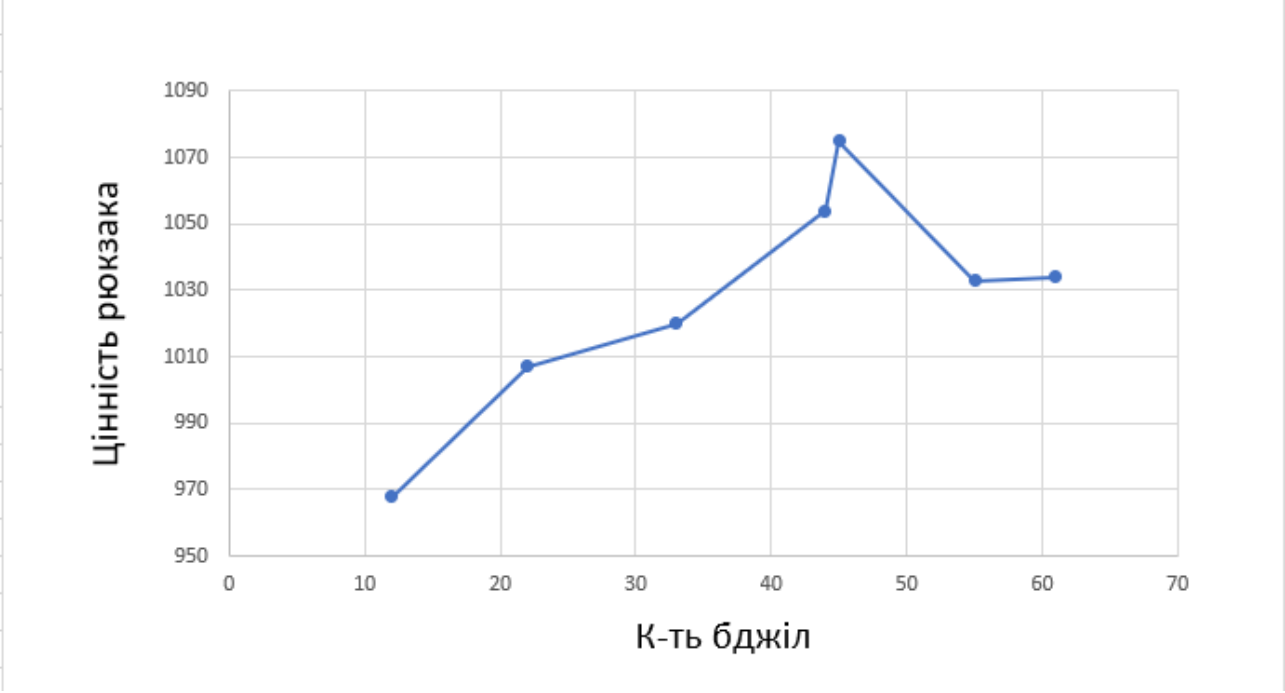



Рисунок 3.3 – Графік залежності цінності рюкзака від кількості бджіл

# ВИСНОВОК

В рамках даної лабораторної роботи було розроблено та виконано програмну реалізацію бджолиного алгоритму для задачі про рюкзак. Даний алгоритм було детально протестовано за різною кількістю бджіл-розвідників та бджіл-фуражирів, а також за кількістю ділянок. В даній задачі було досягнуто покращення цінності рюкзака від 1174 до 1267, що є непоганим результатом. Також було виявлено, що при застосуванні жадібного алгоритму у якості початкового пакування рюкзака ми знайдемо найбільш оптимальне рішення за меншу кількість ділянок. Було досліджено, що при бджолиному алгоритмі цінність рюкзака буде збільшуватися, якщо буде збільшуватися к-ть ділянок, але при умові якщо цінність рюкзака не є локальним рішення.

Для даної задачі алгоритм найкраще працює за такими параметрами:

Бджоли-розвідники – 5

Бджоли-фуражири – 45

## 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%.