

**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 Бабашев О.Д.  
(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>11</b>
3.1	ПОКРОКОВИЙ АЛГОРИТМ .....	11
3.2	ПРОГРАМНА РЕАЛІЗАЦІЯ АЛГОРИТМУ .....	11
3.2.1	<i>Вихідний код.....</i>	<i>11</i>
3.2.2	<i>Приклади роботи .....</i>	<i>14</i>
3.3	ТЕСТУВАННЯ АЛГОРИТМУ .....	16
	<b>ВИСНОВОК .....</b>	<b>21</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.

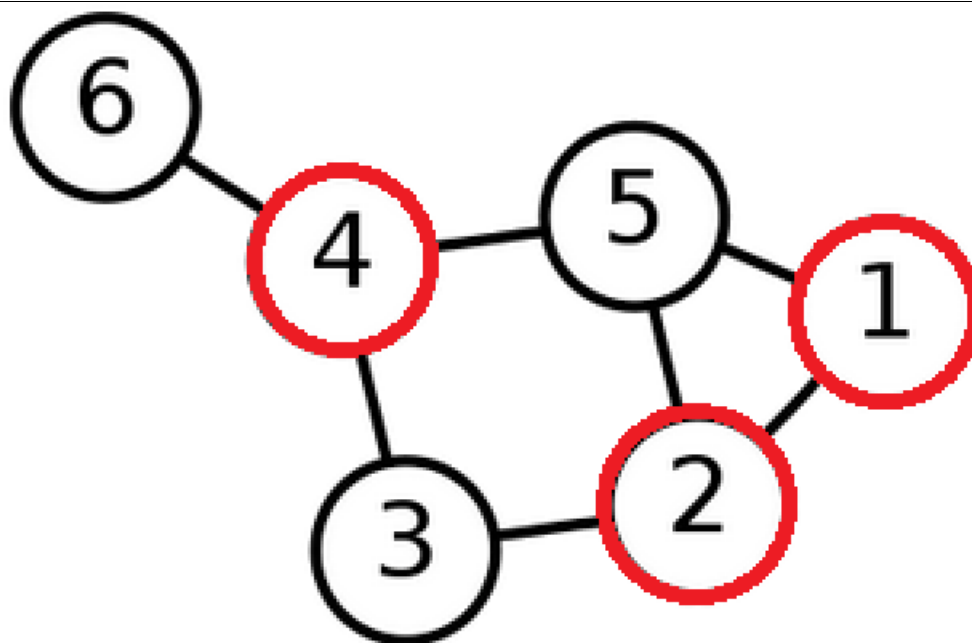
Зробити узагальнений висновок в якому обов'язково описати залежність якості розв'язку від вхідних параметрів.

Таблиця 2.1 – Прикладні задачі

№	Задача
1	<b>Задача про рюкзак</b> (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

	<p>перевищувала задану, а сумарна цінність була максимальною.</p> <p>Задача часто виникає при розподілі ресурсів, коли наявні фінансові обмеження, і вивчається в таких областях, як комбінаторика, інформатика, теорія складності, криптографія, прикладна математика.</p>
2	<p><b>Задача комівояжера</b> (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.</p> <p><b>Розглядається симетричний, асиметричний та змішаний варіанти.</b></p> <p>В загальному випадку, асиметрична задача комівояжера відрізняється тим, що ребра між вершинами можуть мати різну вагу в залежності від напрямку, тобто, задача моделюється орієнтованим графом. Таким чином, окрім ваги ребер графа, слід також зважати і на те, в якому напрямку знаходяться ребра.</p> <p>У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу.</p> <p>У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напрямку руху.</p> <p>Застосування:</p> <ul style="list-style-type: none"> <li>– доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів);</li> <li>– доставка води;</li> <li>– моніторинг об'єктів;</li> </ul>

	<ul style="list-style-type: none"> <li>– поповнення банкоматів готівкою;</li> <li>– збір співробітників для доставки вахтовим методом.</li> </ul>
3	<p><b>Розфарбовування графа</b> (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.</p> <p>Застосування:</p> <ul style="list-style-type: none"> <li>– розкладу для освітніх установ;</li> <li>– розкладу в спорті;</li> <li>– планування зустрічей, зборів, інтерв'ю;</li> <li>– розклади транспорту, в тому числі - авіатранспорту;</li> <li>– розкладу для комунальних служб;</li> </ul>
4	<p><b>Задача вершинного покриття</b> (300 вершин, степінь вершини не більше 30, але не менше 2). Вершинне покриття для неорієнтованого графа <math>G = (V, E)</math> - це множина його вершин <math>S</math>, така, що, у кожного ребра графа хоча б один з кінців входить в вершину з <math>S</math>.</p> <p>Задача вершинного покриття полягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа).</p> <p>На вході: Граф <math>G = (V, E)</math>.</p> <p>Результат: множина <math>C \subseteq V</math> - найменше вершинне покриття графа <math>G</math>.</p>



Застосування:

- розміщення пунктів обслуговування;
- призначення екіпажів на транспорт;
- проектування інтегральних схем і конвеєрних ліній.

5	<p><b>Задача про кліку</b> (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.</p> <p>Задача про кліку існує у двох варіантах: у <b>задачі розпізнавання</b> потрібно визначити, чи існує в заданому графі <math>G</math> кліка розміру <math>k</math>, тоді як в <b>обчислювальному варіанті</b> потрібно знайти в заданому графі <math>G</math> кліку максимального розміру або всі максимальні кліки (такі, що не можна збільшити).</p> <p>Застосування:</p> <ul style="list-style-type: none"> <li>– біоінформатика;</li> <li>– електротехніка;</li> </ul>
6	<p><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) -</p>

	<p>задача пошуку найкоротшого шляху (ланцюга) між двома точками (вершинами) на графі, в якій мінімізується сума ваг ребер, що 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
1	<p><b>Генетичний алгоритм:</b></p> <ul style="list-style-type: none"> <li>- оператор схрещування (мінімум 3);</li> <li>- мутація (мінімум 2);</li> <li>- оператор локального покращення (мінімум 2).</li> </ul>
2	<p><b>Мурашиний алгоритм:</b></p> <ul style="list-style-type: none"> <li>– <math>\alpha</math>;</li> <li>– <math>\beta</math>;</li> <li>– <math>\rho</math>;</li> <li>– <math>L_{min}</math>;</li> <li>– кількість мурах <math>M</math> і їх типи (елітні, тощо...);</li> <li>– маршрути з однієї чи різних вершин.</li> </ul>
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
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

- The function "generate\_items()" is used to create a list of 100 randomly generated "Item" objects, each with a random value and weight.
- The function "generate\_explorer\_bees(items)" is used to create a specified number of "ExplorerBee" objects, each with a randomly selected position and corresponding "Item" object from the list of items.
- The function "generate\_forager\_bees()" is used to create a specified number of "ForagerBee" objects, each with a starting position of 0.
- The function "collect\_items\_info(explorers)" is used to gather information about the items from the explorer bees and return it as a list.
- The function "find\_best\_item(explorers)" uses the information collected from the explorers to guide the forager bees to the best item, and returns the position of the most visited item.
- The function "most\_visited\_item(foragers)" is used to determine the position of the most visited item by counting how many forager bees visited each item's position and returning the position with the highest count.
- The function "foragers\_goes(foragers, explorers, information)" is used to guide the forager bees to the best item using the information from the explorer bees.
- The function "solve\_knapsack\_bee(generated\_items)" is the main function that ties all the above together to solve the knapsack problem using the bee algorithm. It repeatedly fills a "Backpack" object with items, using the information gathered by the explorer and forager bees to determine the best item to add, until the backpack's value exceeds a specified threshold or a specified number of iterations are reached.

### 3.2 Програмна реалізація алгоритму

#### 3.2.1 Вихідний код

*main.py*

```
from algorithm import *

def generate_items():
    items = []
    for i in range(100):
        value = random.randint(2, 30)
        weight = random.randint(1, 20)
        item = Item(value, weight)
        items.append(item)
```

```

        return items

def main():
    items = generate_items()
    solve_knapsack_bee(items)

if __name__ == "__main__":
    main()

```

### ***algorithm.py***

```

from Bee import *

EXPLORERS = 30
FORAGERS = 3

def generate_explorer_bees(items):
    engaged_positions = []
    explorers = []

    min_value = len(items)
    if EXPLORERS + 1 < len(items):
        min_value = EXPLORERS + 1

    for i in range(min_value - 1):
        position = random.randint(0, len(items) - 1)
        while position in engaged_positions:
            position = random.randint(0, len(items) - 1)

        explorer = ExplorerBee(position, items[position])
        explorers.append(explorer)
        engaged_positions.append(position)
    return explorers

def generate_forager_bees():
    foragers = []
    for i in range(FORAGERS):
        forager = ForagerBee(0)
        foragers.append(forager)
    return foragers

def collect_items_info(explorers):
    information = []
    for explorer in explorers:
        information.append(explorer.collect_info())
    return information

def find_best_item(explorers):
    foragers = generate_forager_bees()
    information = collect_items_info(explorers)
    foragers_goes(foragers, explorers, information)
    foragers_accumulation = most_visited_item(foragers)
    return foragers_accumulation

def most_visited_item(foragers):

```

```

    accumulation = {}
    for forager in foragers:
        if forager.position not in accumulation:
            accumulation[forager.position] = 1
        else:
            accumulation[forager.position] += 1
    return max(accumulation)

def foragers_goes(foragers, explorers, information):
    for forager in foragers:
        forager.find_the_way(information, explorers)

def solve_knapsack_bee(generated_items):
    current_goal_function_value = 0
    iterations = 0
    while current_goal_function_value < 1100:
        items = generated_items.copy()
        backpack = Backpack()

        while True:
            explorers = generate_explorer_bees(items)
            foragers_accumulation = find_best_item(explorers)
            if backpack.weight + items[foragers_accumulation].weight >
backpack.max_weight:
                break
            backpack.put_in(items[foragers_accumulation])
            items.pop(foragers_accumulation)

            current_goal_function_value = backpack.value
            iterations += 1

    print(f"items:")
    count = 0
    for item in backpack.items:
        count += 1
        print(f"{count}) value = {item.value} weight = {item.weight}")
    print(f"\nfinal backpack value = {backpack.value}\n"
          f"final backpack weight = {backpack.weight}\n"
          f"avg value = {round(backpack.value / len(backpack.items), 2)}\n"
          f"avg weight = {round(backpack.weight / len(backpack.items), 2)}\n"
          f"amount of iterations = {iterations}\n")

```

### ***Backpack.py***

```

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

    def get_nectar(self):
        return self.value / self.weight

class Backpack:
    def __init__(self):
        self.max_weight = 500
        self.items = []
        self.weight = 0
        self.value = 0

    def put_in(self, item):
        if self.weight + item.weight <= self.max_weight:

```

```
self.items.append(item)
self.value += item.value
self.weight += item.weight
```

### ***Bee.py***

```
import random
from Backpack import *

class Bee:
    def __init__(self, position):
        self.position = position

class ForagerBee(Bee):
    def find_the_way(self, information, explorers):
        discrete_quantity = []
        discrete_quantity_sum = 0
        for i in range(len(information)):
            discrete_quantity_sum += information[i]
            discrete_quantity.append(discrete_quantity_sum)
        random_number = random.uniform(0,
discrete_quantity[len(discrete_quantity) - 1])

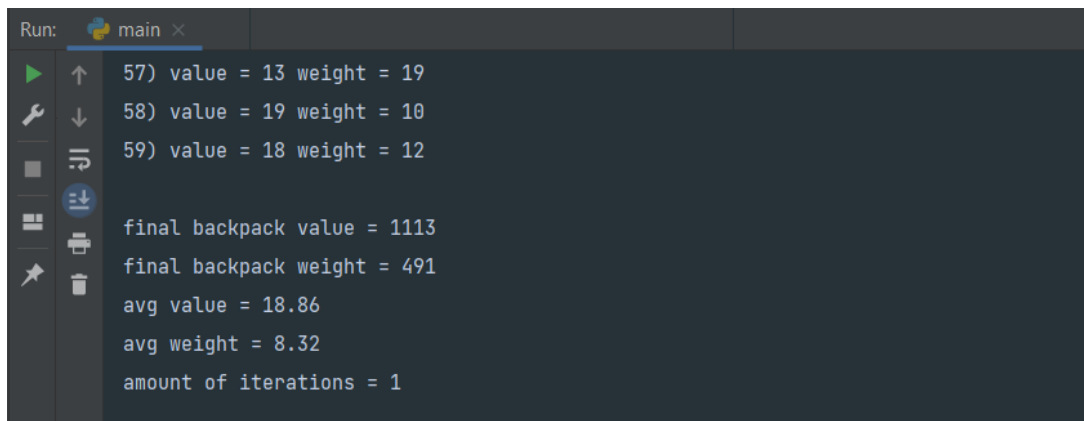
        for i in range(len(discrete_quantity)):
            if i == 0:
                if random_number < discrete_quantity[i]:
                    self.position = explorers[i].position
                    break
            else:
                if discrete_quantity[i - 1] < random_number <=
discrete_quantity[i]:
                    self.position = explorers[i].position
                    break

class ExplorerBee(Bee):
    def __init__(self, position, item):
        super().__init__(position)
        self.explored_item = item

    def collect_info(self):
        return self.explored_item.get_nectar()
```

### 3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

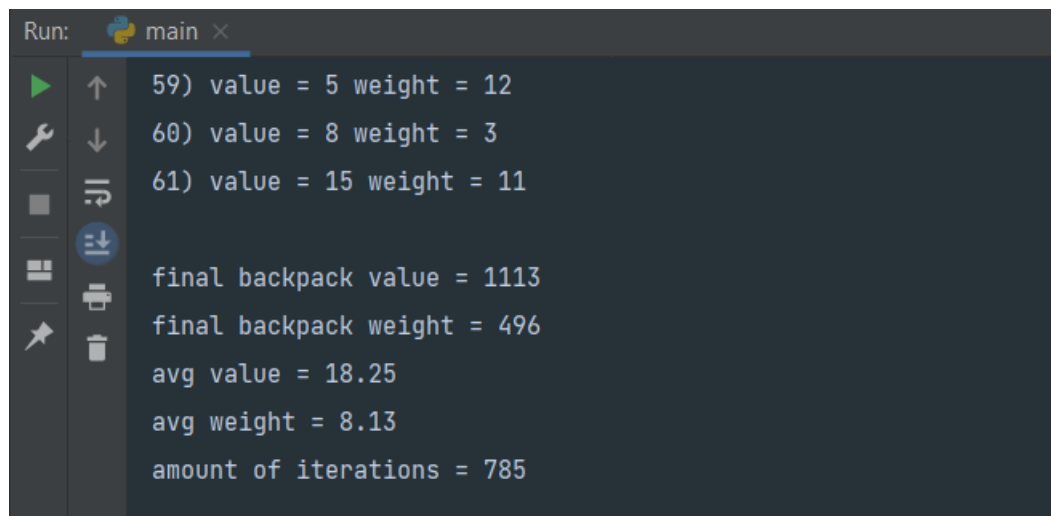


The screenshot shows a Jupyter Notebook console with the following output:

```
Run: main x
57) value = 13 weight = 19
58) value = 19 weight = 10
59) value = 18 weight = 12

final backpack value = 1113
final backpack weight = 491
avg value = 18.86
avg weight = 8.32
amount of iterations = 1
```

Рисунок 3.1 – Приклад роботи програми



The screenshot shows a Jupyter Notebook console with the following output:

```
Run: main x
59) value = 5 weight = 12
60) value = 8 weight = 3
61) value = 15 weight = 11

final backpack value = 1113
final backpack weight = 496
avg value = 18.25
avg weight = 8.13
amount of iterations = 785
```

Рисунок 3.2 – Приклад роботи програми

### 3.3 Тестування алгоритму

Якщо розвідників < максимально предметів,

То ділянки для розвідки = розвідники

Якщо розвідники  $\geq$  максимально предметів,

То ділянки для розвідки = максимально предметів

Маємо 2 параметри. Змінюючи фіксуючи один потім другий зможемо дослідити оптимальну кількість параметрів за значеннями цільової функції. Кількість розвідників та кількість фуражирів.і8

Почнемо з фіксованих значень фуражирів.

**Фуражирів = 10**

Розвідники	Цільова функція
1	1003
2	982
5	950
10	952
20	1011
50	990
100	960

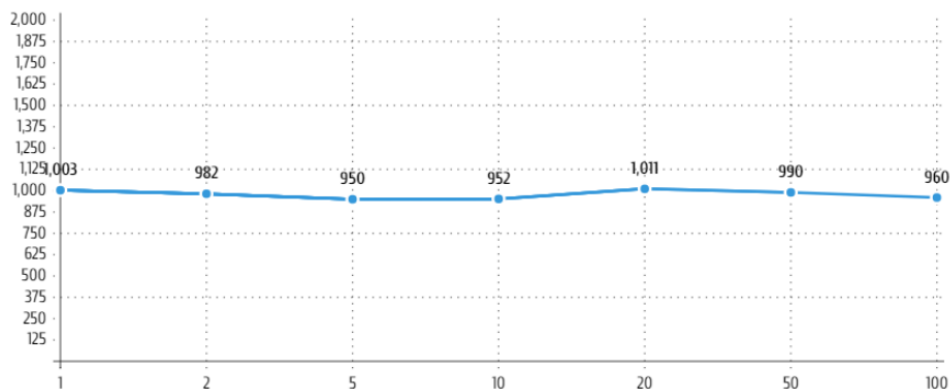


Рисунок 3.3 – Графік залежності



**Фуражирів = 50**

Розвідники	Цільова функція
1	810
2	868
5	825
10	936
20	933
50	864
100	898

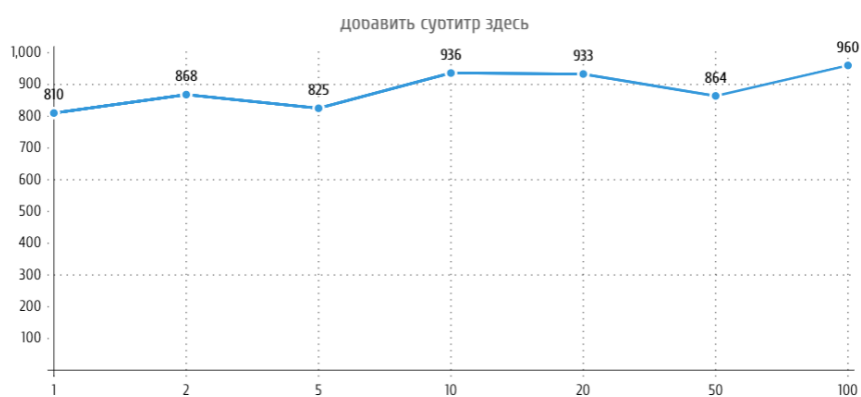


Рисунок 3.4 – Графік залежності

**Фуражирів = 100**

Розвідники	Цільова функція
1	908
2	980
5	925
10	888
20	853
50	870
100	924

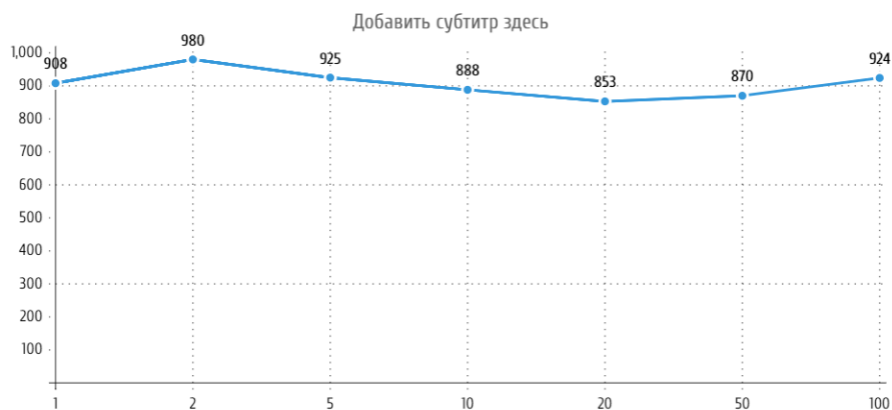


Рисунок 3.5 – Графік залежності

Розвідників = 10

Фуражирів	Цільова функція
1	1241
2	1150
5	1045
10	1005
20	988
50	985
100	969

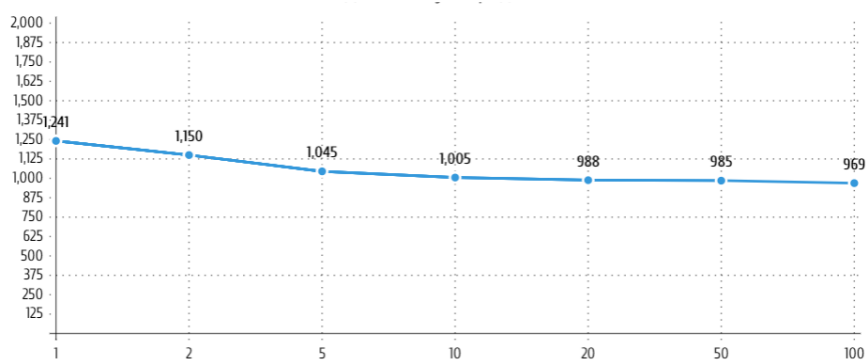


Рисунок 3.6 – Графік залежності

### Розвідників = 50

Фуражирів	Цільова функція
1	1127
2	1158
5	1115
10	1034
20	989
50	868
100	827

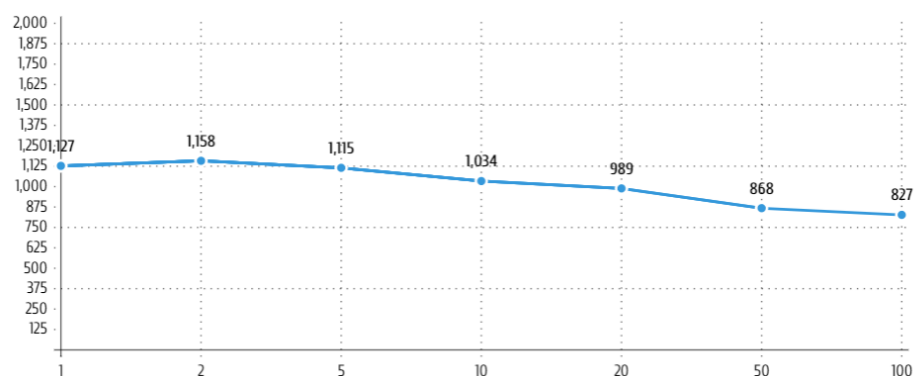


Рисунок 3.7 – Графік залежності

### Розвідників = 100

Фуражирів	Цільова функція
1	1186
2	1119
5	1050
10	993
20	969
50	931
100	878

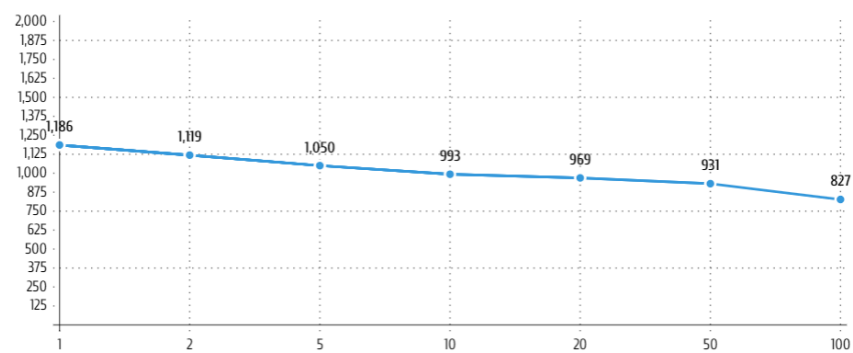


Рисунок 3.8 – Графік залежності

## ВИСНОВОК

В рамках даної лабораторної роботи вивчив основні підходи розробки метаевристичних алгоритмів для типових прикладних задач. Реалізував на пайтоні типову задачу про рюкзак алгоритмом бджолиного рою. Опрацювати методологію підбору прийнятних параметрів алгоритму на практиці.

## 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%.