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
«Проєктування алгоритмів»

„**Проєктування і аналіз алгоритмів для вирішення NP-складних задач ч.1**”

Виконав(ла) *ІП-35 Адаменко Арсен Богданович* _____
(шифр, прізвище, ім'я, по батькові)

Київ 2024

ЗМІСТ

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 (допускається самостійній вибір кроку та верхньої границі ітерацій) і побудувати графік залежності якості розв’язку від числа ітерацій.

Зробити узагальнений висновок.

Таблиця 2.1 – Варіанти алгоритмів

№	**Задача і алгоритм**
1	Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
2	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0{,}4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах).
3	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
4	Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити

	власний оператор локального покращення.
5	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм (α = 2, β = 3, ρ = 0,4, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах).
6	Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники).
7	Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
8	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 3, β = 2, ρ = 0,3, Lmin знайти жадібним алгоритмом, кількість мурах M = 45, починають маршрут в різних випадкових вершинах).
9	Задача розфарбовування графу (150 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 25 із них 3 розвідники).
10	Задача про рюкзак (місткість P=150, 100 предметів, цінність предметів від 2 до 10 (випадкова), вага від 1 до 5 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування рівномірний, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
11	Задача комівояжера (250 вершин, відстань між вершинами випадкова від 0(перехід заборонено) до 50), мурашиний алгоритм (α = 2, β = 4, ρ = 0,6, Lmin знайти жадібним алгоритмом, кількість мурах M = 45,

	починають маршрут в різних випадкових вершинах).
12	Задача розфарбовування графу (300 вершин, степінь вершини не більше 30, але не менше 1), бджолиний алгоритм ABC (число бджіл 60 із них 5 розвідники).
13	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 одноточковий 30% і 70%, мутація з ймовірністю 5% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
14	Задача комівояжера (250 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ($\alpha = 4$, $\beta = 2$, $\rho = 0,3$, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (10 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах).
15	Задача розфарбовування графу (100 вершин, степінь вершини не більше 20, але не менше 1), класичний бджолиний алгоритм (число бджіл 30 із них 3 розвідники).
16	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 двоточковий 30%, 40% і 30%, мутація з ймовірністю 10% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
17	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,7$, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (15 з них дикі, обирають випадкові напрямки), починають маршрут в різних випадкових вершинах).
18	Задача розфарбовування графу (300 вершин, степінь вершини не більше

	50, але не менше 1), класичний бджолиний алгоритм (число бджіл 60 із них 5 розвідники).
19	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% два випадкові гени міняються місцями). Розробити власний оператор локального покращення.
20	Задача комівояжера (200 вершин, відстань між вершинами випадкова від 1 до 40), мурашиний алгоритм ($\alpha = 3$, $\beta = 2$, $\rho = 0,7$, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (10 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах).
21	Задача розфарбовування графу (200 вершин, степінь вершини не більше 30, але не менше 1), класичний бджолиний алгоритм (число бджіл 40 із них 2 розвідники).
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.
23	Задача комівояжера (300 вершин, відстань між вершинами випадкова від 1 до 60), мурашиний алгоритм ($\alpha = 3$, $\beta = 2$, $\rho = 0,6$, Lmin знайти жадібним алгоритмом, кількість мурах M = 45 (15 з них елітні, подвійний феромон), починають маршрут в різних випадкових вершинах).
24	Задача розфарбовування графу (400 вершин, степінь вершини не більше 50, але не менше 1), класичний бджолиний алгоритм (число бджіл 70 із

	них 10 розвідники).
25	Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
26	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах).
27	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
28	Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
29	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах).
30	Задача розфарбовування графу (250 вершин, степінь вершини не більше 25, але не менше 2), бджолиний алгоритм ABC (число бджіл 35 із них 3 розвідники).
31	Задача про рюкзак (місткість P=250, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету,

	оператор схрещування одноточковий по 50 генів, мутація з ймовірністю 5% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
32	Задача комівояжера (100 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 4$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 30, починають маршрут в різних випадкових вершинах).
33	Задача розфарбовування графу (200 вершин, степінь вершини не більше 20, але не менше 1), бджолиний алгоритм ABC (число бджіл 30 із них 2 розвідники).
34	Задача про рюкзак (місткість P=200, 100 предметів, цінність предметів від 2 до 20 (випадкова), вага від 1 до 10 (випадкова)), генетичний алгоритм (початкова популяція 100 осіб кожна по 1 різному предмету, оператор схрещування двоточковий порівну генів, мутація з ймовірністю 10% змінюємо тільки 1 випадковий ген). Розробити власний оператор локального покращення.
35	Задача комівояжера (150 вершин, відстань між вершинами випадкова від 5 до 50), мурашиний алгоритм ($\alpha = 2$, $\beta = 3$, $\rho = 0,4$, Lmin знайти жадібним алгоритмом, кількість мурах M = 35, починають маршрут в різних випадкових вершинах).

3 ВИКОНАННЯ

3.1 Програмна реалізація алгоритму

3.1.1 Вихідний код

```
import random
import matplotlib.pyplot as plt

CAPACITY = 250 # 250
VALUE_RANGE = {'min': 2, 'max': 20}
WEIGHT_RANGE = {'min': 1, 'max': 10}

ITEMS_COUNT = 100
POPULATION_COUNT = 100
CROSSOVER_VALUE = 50
MUTATION_CHANCE = 0.05
GENERATIONS_COUNT = 2400

def GeneratePopulation() -> list[list[int]]:
    population = []

    for item_id in range(POPULATION_COUNT):
        population += [
            [
                int(item_id == gen_id)
                for gen_id in range(ITEMS_COUNT)
            ]
        ]

    return population

def Crossover(p1: list[int], p2: list[int]) -> list[int]:
    point = CROSSOVER_VALUE

    c = p1[:point] + p2[point:]

    return c

def Weight(ind: list[int]) -> int:
    global items

    weight = sum(
        items[i]['weight'] for i in range(ITEMS_COUNT) if ind[i]
    )

    return weight

def Value(ind):
    return sum(items[i]['value'] for i in range(ITEMS_COUNT) if ind[i])

def Mutate(ind: list[int]) -> tuple[list[int], bool]:
    ind = ind[:]

    if random.random() < MUTATION_CHANCE:
        idx = random.randint(0, ITEMS_COUNT - 1)
        ind[idx] = 1 - ind[idx]

        return ind, True
```

```
    return ind, False

def LocallyOptimize(ind: list[int]) -> None:
    old_weight = Weight(ind)

    worst_indexer = len(items_end_worst) - 1
    while not (old_weight < CAPACITY):
        if worst_indexer < 0:
            break

        item_data = items_end_worst[worst_indexer]
        index = item_data['index']

        if ind[index] == 1:
            old_weight -= item_data['weight']
            ind[index] = 0

        worst_indexer -= 1

    for _ in range(random.randint(0, 2)):
        ind[random.randint(0, ITEMS_COUNT - 1)] = 0

    appends = 0
    for item in items_from_best:
        if appends > 2:
            break

        index = item['index']

        if old_weight + item['weight'] <= CAPACITY and ind[index] == 0:
            ind[index] = 1
            old_weight += item['weight']
            appends += 1

def SolveKnapsack() -> dict:
    average_values = []
    worst_values = []
    best_values = []

    best_ind = {'value': -1, 'ind': None}

    population = GeneratePopulation()

    for _ in range(GENERATIONS_COUNT):
        # mate
        match random.choice([0, 1, 2]):
            case 0:
                p1 = sorted(population, key=lambda x: Value(x))[-1]
                p2 = random.choice(population)
            case 1:
                p1 = sorted(population, key=lambda x: Value(x))[0]
                p2 = random.choice(population)
            case 2:
                p1 = random.choice(population)
                p2 = random.choice(population)

        c = Crossover(p1, p2)
        c_mut, is_mut = Mutate(c)

        best_c = c
        if Value(c_mut) > Value(c):
            best_c = c_mut
        else:
```

```
            is_mut = False

        if not is_mut:
            LocallyOptimize(best_c)

        population += [best_c]

        population.sort(key=lambda x: Value(x), reverse=True)
        population.pop()

        values = list(map(Value, population))

        average_values += [sum(values) / len(population)]
        worst_values += [min(values)]
        best_values += [max(values)]

        may_new_best_ind = (sorted(population, key=lambda x: Value(x)))[-1]
        may_new_best_ind_value = Value(may_new_best_ind)

        if may_new_best_ind_value > best_ind['value']:
            best_ind['value'] = may_new_best_ind_value
            best_ind['ind'] = may_new_best_ind[:]

    return {
        'best_value': best_ind['value'],
        'best_gens': best_ind['ind'],
        'average_values': average_values,
        'worst_values': worst_values,
        'best_values': best_values
    }

items = [
    {
        'value': random.randint(
            VALUE_RANGE['min'],
            VALUE_RANGE['max']
        ),
        'weight': random.randint(
            WEIGHT_RANGE['min'],
            WEIGHT_RANGE['max']
        )
    }
    for _ in range(ITEMS_COUNT)
]

items_indexes = [
    {
        'value': item['value'] / item['weight'],
        'index': i,
        'weight': item['weight']
    }
    for i, item in enumerate(items)
]

items_from_best = items_indexes[:]
items_from_best.sort(key=lambda x: x['value'], reverse=True)

items_end_worst = items_indexes[:]
items_end_worst.sort(key=lambda x: x['value'], reverse=True)

# generate result
results = SolveKnapsack()
```

```
print(f'Best value: {results["best_value"]}')
print(f'Best gens:  {results["best_gens"]}')
print(f'Values ranges:')

for i in range(0, GENERATIONS_COUNT, 100):
    print(f'{i},{results["worst_values"][i]},{results["average_values"][i]},
{results["best_values"][i]}')

# output results
fig, axs = plt.subplots(1, 1, figsize=(10, 8))

ax = axs
ax.grid(color='#777', which='major')
ax.minorticks_on()
ax.grid(color='#aaa', which='minor')

ax.plot(
    range(GENERATIONS_COUNT),
    results['best_values'],
    color = '#077',
    label = 'Best value'
)
ax.plot(
    range(GENERATIONS_COUNT),
    results['average_values'],
    color = '#770',
    label = 'Average value'
)
ax.plot(
    range(GENERATIONS_COUNT),
    results['worst_values'],
    color = '#700',
    label = 'Worst value'
)

ax.legend()

plt.show()
```

3.1.2 Вихідний код розв’язання задачі рюкзака з використанням динамічного програмування.

```
from random import randint

# Генеруємо випадкові предмети
items = [{'value': randint(2, 20), 'weight': randint(1, 10)} for _ in
range(100)]

# Ємність рюкзака
capacity = 250

# Ініціалізуємо таблицю DP
dp = [0] * (capacity + 1)

# Алгоритм динамічного програмування
for item in items:
    value, weight = item['value'], item['weight']
    for c in range(capacity, weight - 1, -1):  # Перебір від кінця, щоб уникнути
перезаписів
        dp[c] = max(dp[c], dp[c - weight] + value)
```

```
# Максимальна цінність
print("Максимальна цінність:", dp[capacity])
```

3.1.3 Псевдокод алгоритму:

Функція Схрещування(Тато1, Тато2) *{ O(|i|) }*

Повернути ПершаПоловина(Тато1) **З'єднати** ДругаПоловина(Тато2) *{ O(|i|) }*

Кінець

Функція ЛокальнийПошукДітей(Діти) *{ min = O(1); max = O(|p|) }*

ВидалитиПаруВипадковихПредметів(Діти) *{ min = O(1); max = O(|p|) }*

ДодатиПаруПредметівЯкщоЄМісце(Діти) *{ min = O(1); max = O(|p|) }*

Кінець

Функція ВирішитиПроблемуЗЦимРюкзаком *{ O(GC * (|p| log |p|) + |i|) }*

Популяція = СтворитиНовуПопуляцію() *{ O(PC) }*

Найкращий = **Нема**

Повторити КількістьІтераційЕволюції **Разів**

Тато1 = НайбільшийЗаЦінністю(Популяція) *{ O(|p| log |p|) }*

Тато2 = ВибратиВипадково(Популяція) *{ O(1) }*

Діти = Схрещування(Тато1, Тато2) *{ O(|p|) }*

МутогенДітей = Мутація(Діти)

Якщо МутаціяВдала **То**

ОстаточніДіти = МутогенДітей

Інакше

ОстаточніДіти = ЛокальнийПошукДітей(Діти)

Кінець

Якщо ОстаточніДіти.Цінність > Найкращий.Цінність **То**

Найкращий = ОстаточніДіти

Кінець

Популяція.Додати(ОстаточніДіти)

Популяція.ВидалитиНайменимЗаЦінністю() *{ O(|p|) }*

Кінець

Повернути Накращий

Кінець

3.1.4 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми. Параметрами алогритму є розмір рюкзака 250, кількість предметів 100, розмір популяції 100, одноточкове схрещування, шанс 1 мутації 5%, кількість кроків еволюції 2400.

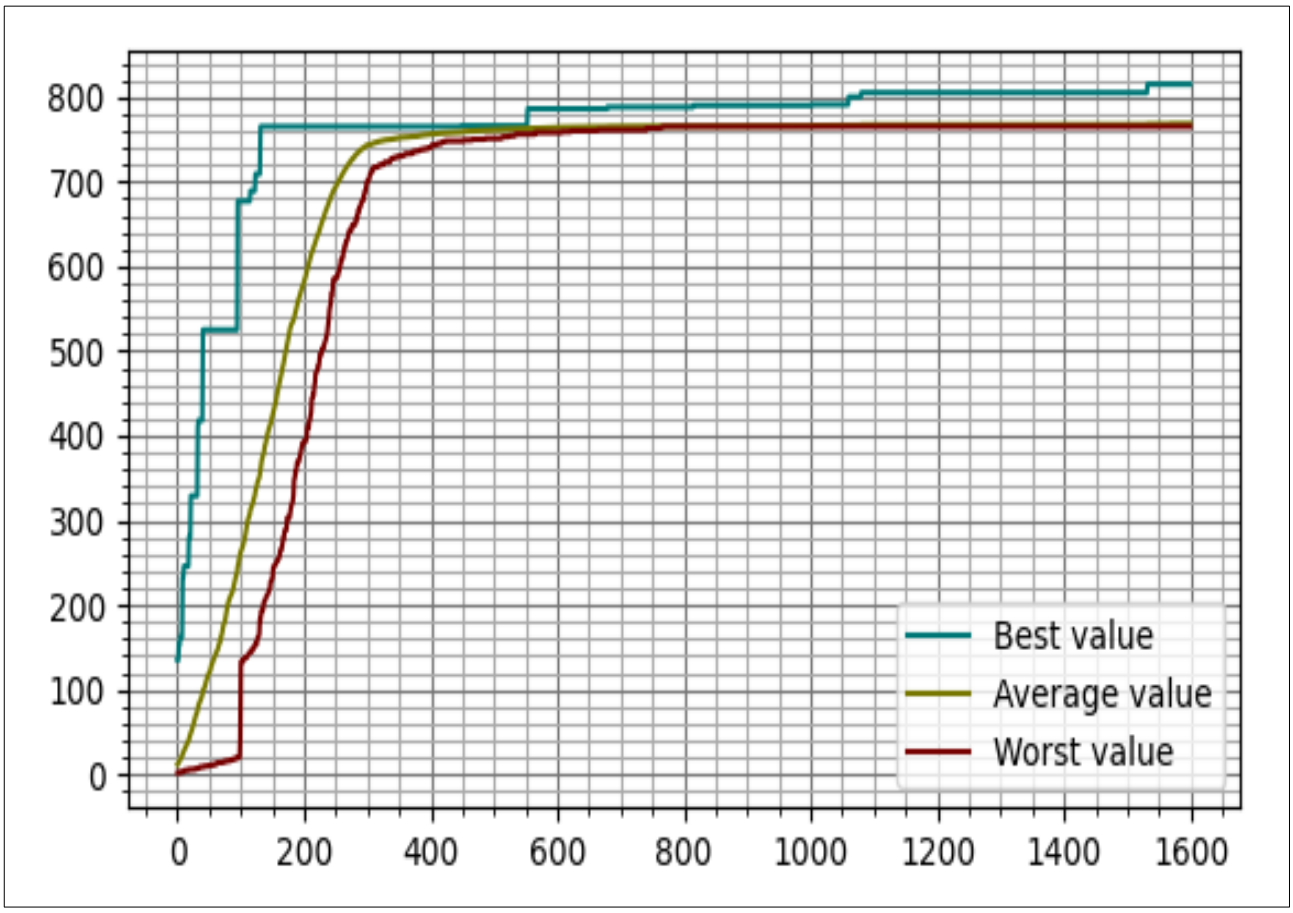

Рисунок 3.1 – використання функції локального покращення «видали пару найгірших, встав пару найкрфщих»

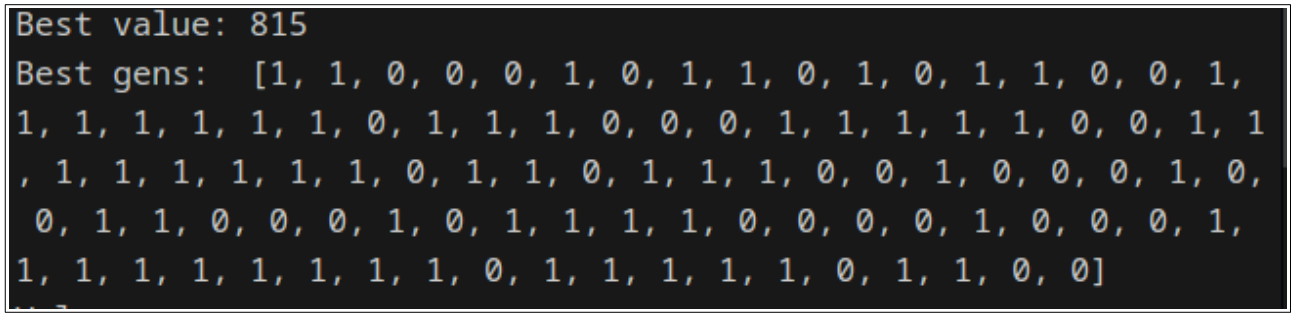

Рисунок 3.2 – найкрще рішення або його шлях досягнення та абсолютно найкраще числове рішення

На рисунку 3.1 та 3.2 видно, що найкращим рішенням задачі про рюкзак є 815, використовуючи генетичний алгоритм. Гени, або рішення чи шлях досягнення рішення, зображені на рисунку 3.2 для огляду кінцевого результату виконня цього алгоритму для більш повної картини розуміння цього алгоритму.

Його ефективним застосуванням є знаходження напівоптимального рішення з великої множини всіх рішень. При знаходженні рішення, найбільш близького до оптимального, він починає дуже повільно працювати через те, що він намагається лише покращити поточну популяцію для створення нової на її основі, опираючись на рішення, яке може бути локальним, а не глобальним максимумом до найкрахого рішення такої складної задачі як задача про рюкзак.

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.

Номер ітерації	Найгірше	Середнє	Найкраще
0	2	11.96	135
100	133	263.29	677
200	393	583.44	765
300	700	742.89	765
400	741	756.11	765
500	751	761.09	766
600	758	763.87	786
700	761	765.21	788
800	765	765.53	788
900	765	765.78	790
1000	765	765.84	790
1100	765	766.94	805
1200	765	766.96	805
1300	765	767.06	805
1400	765	767.21	805
1500	765	767.27	805

3.2.2 Графіки залежності розв’язку від числа ітерацій

На рисунку 3.3 наведений графік, який показує якість отриманого розв’язку. Чим ближче знаходиться графік до значення по ординаті 1, тим більш оптимальним є розв’язок генетичним алгоритмом.

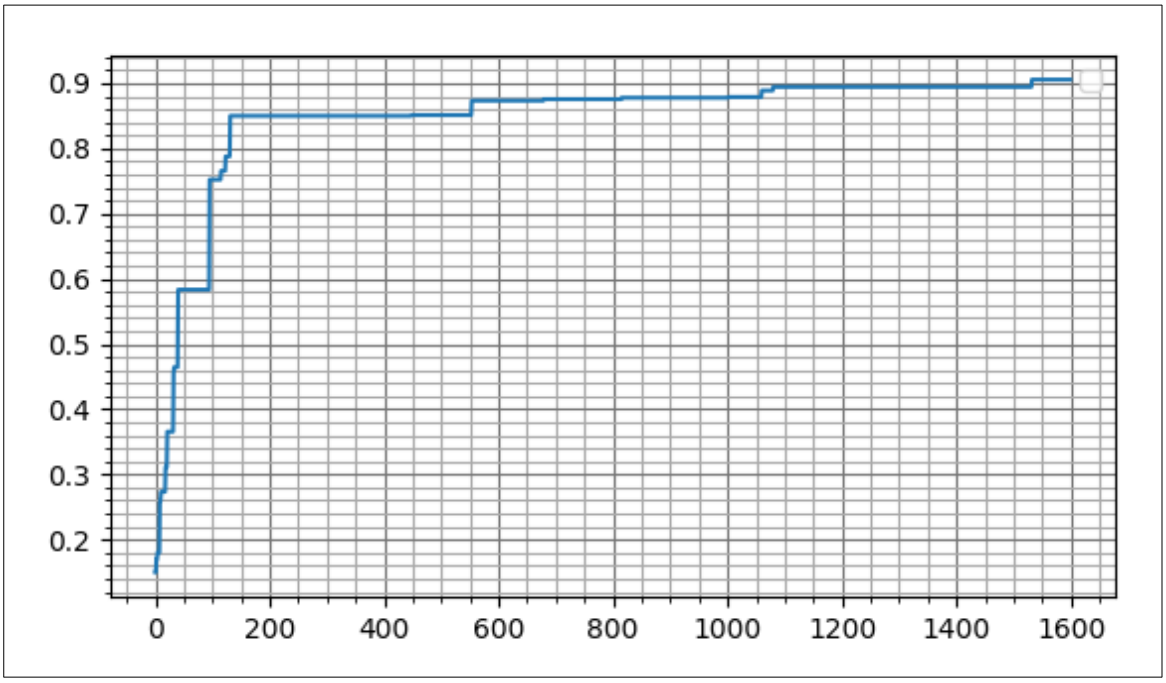

Рисунок 3.3 – Графіки залежності розв’язку від числа ітерацій

В даному випадку генетичний алгоритм непогано справляється з задачею про рюкзак, але він довго добирається до плато, тому він трохи неефективний в даному плані для задачі про рюкзак.

Судячи по кривій якості розв’язку, він добре підходить для знаходження напів- або практично опимальних розв’язків задач через його природу.

Під час реалізації алгоритму використовуючи свою стратегію відбору[1] батьків та створення нового покоління було помічено, що воно було більш ефективним для виходу на плато, а ніж варіація дана лектором під час лекції з описом цього алогритму та застосування його в цій задачі на практичному прикладі.

ВИСНОВОК

В рамках даної лабораторної роботи я нарешті зміг проаналізувати, що таке генетичні алгоритми та як їх можна використовувати для вирішення складних задач, як задача про рюкзак, та як він працює в теорії та на практичном прикладі та досвіді.

Було виконано програмну реалізацію генетичного алгоритма для задачі про рюкзак з використанням певних налаштувань програмного забезпечення. Під час тестування було помічено, що його легко перевірити на коректність та послідовність виконання. Функція локального пошуку індивідів є однією з найбільш цікавих та складних в плані комплексності та багатогранності функції цього метаевристичного алгориму.

Результат викання програмної реалізації цього алгоритму є досить приємним та швидким. Він дозволяє отримати все більш оптимальні рішення з плином виконання головного циклу виконння цго алгоритму (найкраще значення розв’язку для рюкзака постійно збільшується).

Абсолютне найкраще рішення задачі про рюкзак з кожною ітераціює становиться все кращим, але воно досягає плато після стрімкого початкового збільшення реального значення цінності розв’язку задачі при випадкових вхідних параметрах. В порівнянні з еталонним алгоритмом (той, що використовує динамічне програмування) він дає практично найкраще рішення перші 200 ітерацій, але потім він знаходить їх все більш повільніше при спробі досягти найкращого реального рішення задачі.

Цей алгоритм, як і інші метаевристичні, дає змогу зрозуміти, що задача може бути вирішена навіть частково та використовуючи нестандартні техніки та стратегії досягення найкращого результату для обраної задачі.

ПОСИЛАННЯ

1. Своя реалізація стратегії відбору батьків та статегія створення нового покоління URL: https://arpitbhayani.me/blogs/genetic-knapsack/

КРИТЕРІЇ ОЦІНЮВАННЯ

Критерії оцінювання у відсотках від максимального балу:

- програмна реалізація алгоритму – 75%;
- тестування алгоритму– 20%;
- висновок – 5%.