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-25Карпов Л В* _____
(шифр, прізвище, ім'я, по батькові)

Перевірив *Головченко М.Н.* _____
(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.

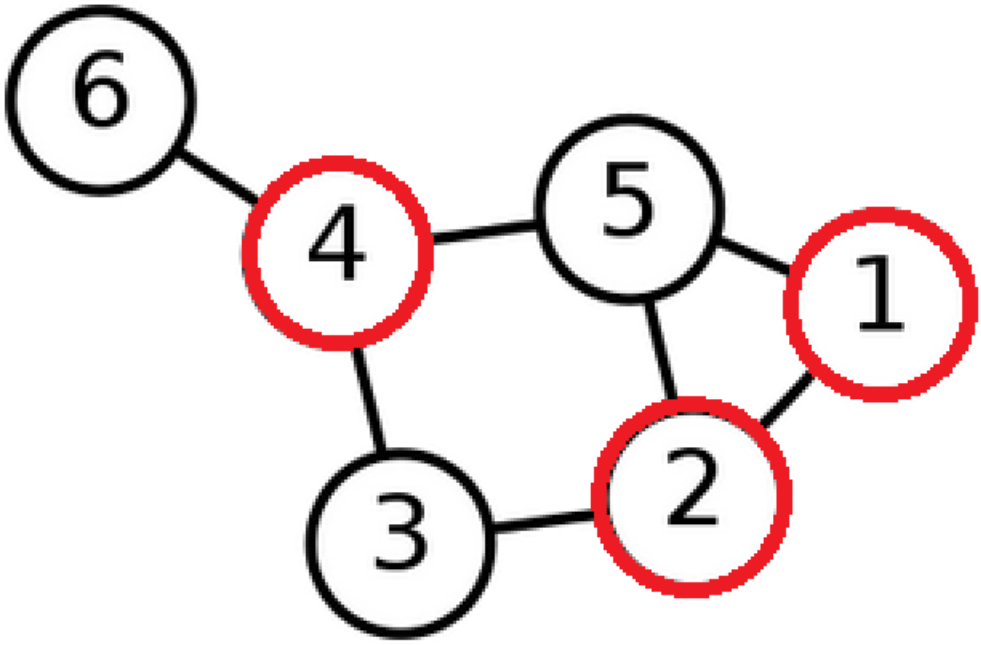
Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

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. 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. **Розглядається симетричний, асиметричний та змішаний варіанти.** 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. У випадку симетричної задачі всі пари ребер між одними й тими самими вершинами мають однакову вагу. У випадку реальних міст може бути як симетричною, так і асиметричною в залежності від тривалості або довжини маршрутів і напряму руху. Застосування: – доставка товарів (в цьому випадку може бути більш доречна постановка транспортної задачі - доставка в кілька магазинів з декількох складів); – доставка води; – моніторинг об'єктів;

	– поповнення банкоматів готівкою; – 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. Застосування: – розкладу для освітніх установ; – розкладу в спорті; – планування зустрічей, зборів, інтерв'ю; – розклади транспорту, в тому числі - авіатранспорту; – 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. Задача вершинного покриття полягає в пошуку вершинного покриття найменшого розміру для заданого графа (цей розмір називається числом вершинного покриття графа). На вході: Граф G = (V, E). Результат: множина C ⊆ V - найменше вершинне покриття графа G.

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

Генерація випадкового графу

Створення початкової популяції

Початок ітеративного процесу

Відбір батьків наступного покоління

Схрещування (кількість наслідників половина від популяції)

Мутація наслідників

Локальне покращення наслідників

Відбір найкращих особин (розмір популяції залишається сталим)

Кінець ітеративного процесу

Відображення результатів

3.2 Програмна реалізація алгоритму

3.2.1 Вихідний код

```
import random

import numpy as np
import networkx as nx
import matplotlib.pyplot as plt

GRAPH_SIZE = 300
EDGE_PROBABILITY = 0.05

POPULATION_SIZE = 10
MUTATION_PROBABILITY = 0.025


class Solution:
    def __init__(self, graph=None, genes=None):
        self.graph = graph or nx.gnp_random_graph(GRAPH_SIZE, EDGE_PROBABILITY)
        self.genes = genes or [False for _ in range(GRAPH_SIZE)]

    @property
    def cover_len(self):
        return sum(self.genes)

    def __repr__(self):
        return f'Sol({self.cover_len})'

    def draw(self):
        plt.figure(figsize=(32, 24))
        nx.draw_networkx(self.graph, node_color=['blue' if gene else 'white' for
gene in self.genes], with_labels=True)
```

```
        plt.show()

    def solve(self) -> 'Solution':
        edges = list(self.graph.edges)

        for i in range(GRAPH_SIZE):
            if self.genes[i]:
                for j in range(len(edges) - 1, -1, -1):
                    if i in edges[j]:
                        edges.pop(j)

        if len(edges) == 0:
            return self

        edges_to_cover: list = random.sample(sorted(edges), len(edges))

        while len(edges_to_cover):
            a, b = edges_to_cover[0]
            self.genes[a] = True
            for j in range(len(edges_to_cover) - 1, -1, -1):
                if a in edges_to_cover[j]:
                    edges_to_cover.pop(j)

        return self

    def local_improvement(self):
        edges = list(self.graph.edges)
        covered_edges = set()
        for i in random.sample(range(GRAPH_SIZE), GRAPH_SIZE):
            if not self.genes[i]:
                continue
            not_covering = True
            for edge in edges:
                if i in edge and edge not in covered_edges:
                    covered_edges.add(edge)
                    not_covering = False

            if not_covering:
                self.genes[i] = False

        assert len(edges) == len(covered_edges)

    def mutate(self):
        for i in range(GRAPH_SIZE):
            self.genes[i] = bool(self.genes[i] - (random.randint(0, 1000) < 1000
* MUTATION_PROBABILITY))

        return self.solve()

    @classmethod
    def crossover_prb(cls, first: 'Solution', second: 'Solution'):
        new_genes = [random.choice((first.genes[i], second.genes[i])) for i in
range(GRAPH_SIZE)]

        return cls(first.graph, new_genes)


class Population:
    def __init__(self, graph=None):
        self.graph = graph or nx.fast_gnp_random_graph(GRAPH_SIZE,
EDGE_PROBABILITY)

        self.population = [Solution(self.graph).solve() for _ in
range(POPULATION_SIZE)]
```

```
        self.population.sort(key=lambda x: x.cover_len)

    def nex_gen(self):
        parrent1, parrent2 = random.choices(
            self.population, [1 / solution.cover_len for solution in
self.population], k=2
        )
        children: list[Solution] = [Solution.crossover_prb(parrent1, parrent2)
for _ in range(POPULATION_SIZE // 2)]

        for child in children:
            child.mutate()
            child.local_improvement()

        self.population = children + self.population

        self.population.sort(key=lambda x: x.cover_len)

        self.population = self.population[:POPULATION_SIZE]

    @property
    def best(self):
        return self.population[0]

    def draw(self):
        plt.figure(figsize=(32, 24))
        nx.draw_networkx(self.graph, with_labels=True)
        plt.show()


ITERATION_COUNT = 1000


def _main():
    pop = Population()
    pop.draw()
    y = []
    for _ in range(ITERATION_COUNT):
        y.append(pop.best.cover_len)
        pop.nex_gen()
    x = np.array(range(ITERATION_COUNT))
    plt.plot(x, y)
    plt.show()
    print(y[0] - y[-1])
    pop.best.draw()


if __name__ == '__main__':
    _main()
```

3.2.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

Рисунок 3.1 –

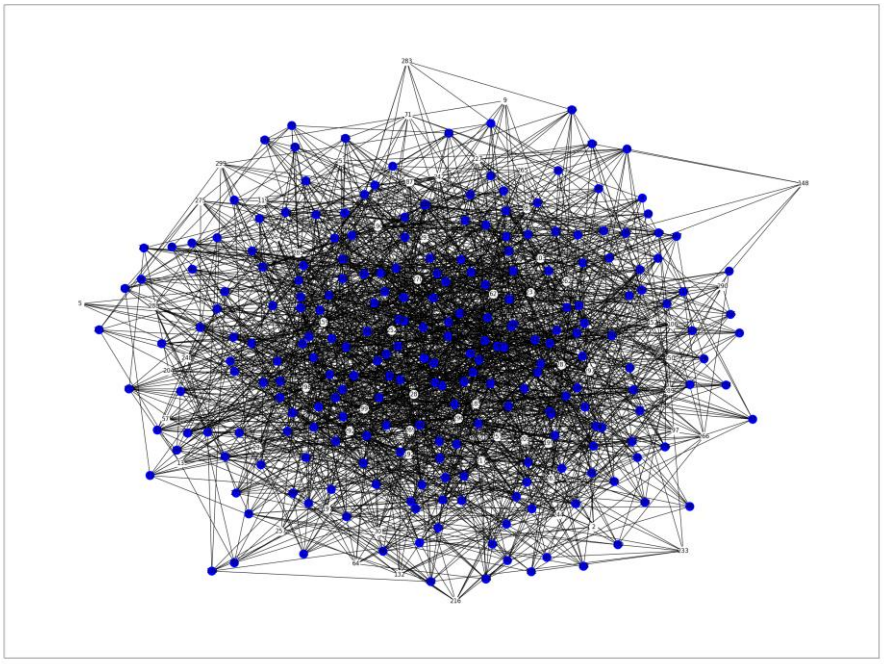
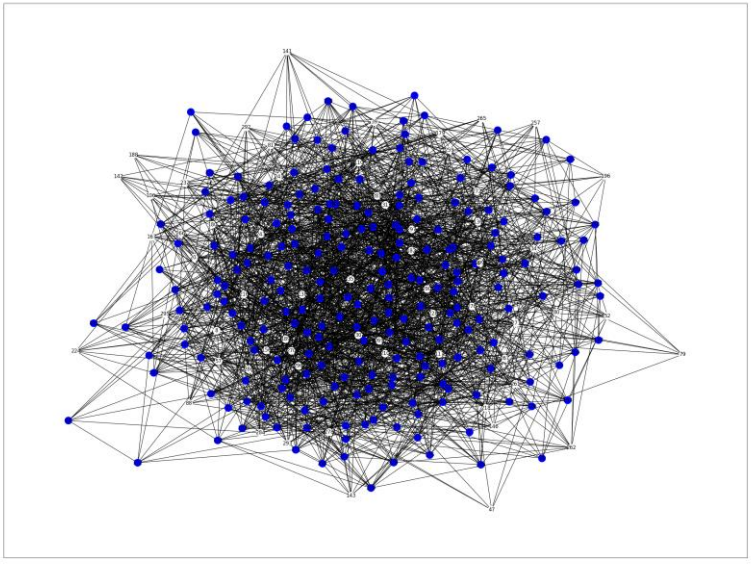


Рисунок 3.2 –



3.3 Тестування алгоритму

Метод схрещування	Метод мутації	Середнє покращення завдяки алгоритму
Кожен ген з одного з батьків	Кожен ген з ймовірністю	18
одноточкове	Кожен ген з ймовірністю	17.6
двоточкове	Кожен ген з ймовірністю	16.4
Кожен ген з одного з батьків	Випадкова кількість генів не більше (10)	13.4
одноточкове	Випадкова кількість генів не більше (10)	16.4
двоточкове	Випадкова кількість генів не більше (10)	14.8

Рисунок 3.3 –

ВИСНОВОК

В рамках даної лабораторної роботи був спроектований генетичний алгоритм для розвязку NP-складної задачі покриття графа, було використано схрещування у вигляді випадкового вибору відповідного гену одного з батьків ,одноточковий та двоточковий, для мутацій було використана статична ймовірність кожного гену змінити своє значення, вибір декількох генів для зміни, це забезпечує більшу варіативність серед нащадків, як локальне покращення був застосований алгоритм видалення незадіяних в покритті вершин з використанням випадкового порядку проходження по вершинам, що забезпечує уникнення локальних рішень.

Оскільки ймовірність мутації конкретного гену є стала, вона відіграє велику роль, якщо вона буде занадто мала нам не уникнути не оптимальних локальних рішень, але й занадто великою вона не має бути, оскільки так зменшується вплив генів батьків, наприклад якщо ця ймовірність буде 0,5 то в середньому половина генів будуть спадковані від батьків а інша половина будуть обернені до тих що мали б бути спадковані.

Розмір популяції хоча і не так сильно впливає на результат ніж ймовірність мутації, тим не менш від нього залежить швидкість покращення вихідного результату, в той же час занадто велика популяція зменшує вірогідність вибору кращих батьків у популяції з великим розривом якості.

КРИТЕРІЇ ОЦІНЮВАННЯ

При здачі лабораторної роботи до 24.12.2023 включно максимальний бал дорівнює – 5. Після 24.12.2023 максимальний бал дорівнює – 4,5.

Критерії оцінювання у відсотках від максимального балу:

- покроковий алгоритм – 10%;
- програмна реалізація алгоритму – 45%;
- робота з гіт – 20%;
- тестування алгоритму– 20%;
- висновок – 5%.

+1 додатковий бал можна отримати за виконання та захист роботи до 17.12.2023