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-14 Качмар Андрій Дмитрович* _____
(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.

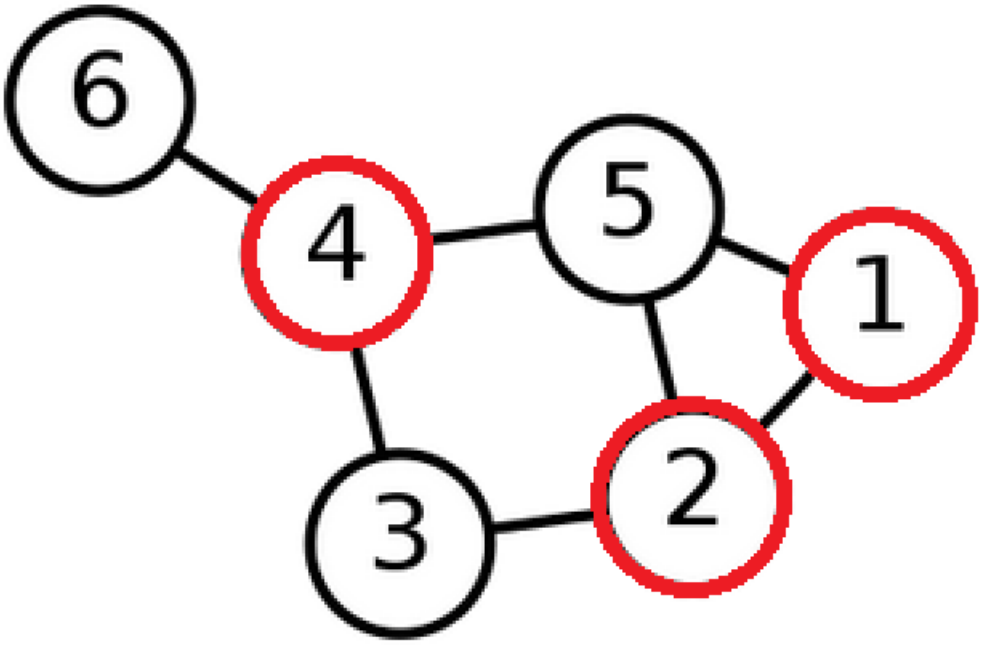
Зробити узагальнений висновок в якому обов’язково описати залежність якості розв’язку від вхідних параметрів.

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

1) Заповнення матриці відстаней (**300 вершин**)
2) Заповнення усіх вхідних параметрів алгоритму
    a. α, β , ρ , Lmin, M
    b. L_best
3) Заповнення матриці видимості ребер
4) Заповнення матриці рівня феромону на ребрах
5) Розміщення мурах в міста по заданому типу розміщення
6) Цикл за часом життя колонії
    a. Цикл по усіх мурахах колонії
        i. Побудова циклу обходу міст для мурахи
        ii. Цикл поки не буде знайдено обхід
            1. Отримання міста для переходу
                a. Для звичайних та елітних мурах Вираховування ймовірності переходу для звичайних та елітних мурах. Повернення найімовірніший перехід
                b. Для диких мурах повернення випадкове суміжне місто
        iii. Кінець циклу
        iv. Вираховування довжини шляху
    b. Кінець циклу по усіх мурахах колонії
7) Порівняння отриманих шляхів обходу для усіх мурах
8) Вибір найменшою шляху
9) Перевизначення найкращого знайденого шляху
10) Цикл по матриці феромону
    a. Вираховування рівня феромону для ребра
    b. Оновлення значення рівня феромону на ребрі
    c. Присвоєння додаткового феромону якщо мураха елітна
11) Кінець циклу по матриці феромону
12) Кінець циклу за часом життя колонії
13) Виведення знайденого найкращого шляху

Програмна реалізація алгоритму

3.1.1 Вихідний код

```
public PathSearchResult findSolution() {
    distanceMatrix = buildDistanceMatrix();
    pheromoneMatrix = buildPheromoneMatrix();
    ants = generateAnts(distanceMatrix, salesmanProblemDto);
    placeAnts(ants, salesmanProblemDto.getAntPlacementType());

    for (int i = 0; i < salesmanProblemDto.getColonyLife(); i++) {
        List<PathSearchResult> paths = buildPathsForAllAnts();
        PathSearchResult foundBestPath = paths.stream() Stream<PathSearchResult>
                .min(Comparator.comparing(PathSearchResult::getPathCost)) Op
                .orElseThrow(() -> new RuntimeException("Path not found"));
        if (foundBestPath.getPathCost() < this.bestPath.getPathCost()) {
            this.bestPath = foundBestPath;
        }
        updatePheromoneLevel(paths);
        clearAntMemory();
    }
    return bestPath;
}
```

```
public static int[][] buildDistanceMatrix() {
    int[][] distanceMatrix = new int[G_SIZE][G_SIZE];
    for (int i = 0; i < distanceMatrix.length; i++) {
        for (int j = 0; j < distanceMatrix[i].length; j++) {
            int distance = (i == j) ? Integer.MAX_VALUE : randomNumber(5, 150);
            distanceMatrix[i][j] = distance;
        }
    }
    return distanceMatrix;
}
public static double[][] buildVisionMatrix(int[][] distanceMatrix) {
    double[][] visionMatrix = new double[G_SIZE][G_SIZE];
    for (int i = 0; i < visionMatrix.length; i++) {
        for (int j = 0; j < visionMatrix.length; j++) {
            if (i == j) {
                visionMatrix[i][j] = 0;
            } else {
                visionMatrix[i][j] = round( value: (double) 1 / distanceMatrix[i][j], places: 3);
            }
        }
    }
    return visionMatrix;
}
```

```java
public static double[][] buildPheromoneMatrix() {
    double[][] pheromoneMatrix = new double[G_SIZE][G_SIZE];
    for (int i = 0; i < pheromoneMatrix.length; i++) {
        for (int j = 0; j < pheromoneMatrix.length; j++) {
            double pheromone = (i == j) ? 0 : (round(randomDouble(0.1, 0.2), places: 2));
            pheromoneMatrix[i][j] = pheromone;
        }
    }
    return pheromoneMatrix;
}
```

```java
private List<PathSearchResult> buildPathsForAllAnts() {
    List<PathSearchResult> paths = new ArrayList<>();
    for (Ant ant : ants) {
        PathSearchResult pathSearchResult = findAntPath(ant);
        pathSearchResult.countPathCost(distanceMatrix);
        paths.add(pathSearchResult);
    }
    return paths;
}
private PathSearchResult findAntPath(Ant ant) {
    int initialPosition = ant.getCurrentCityIndex();
    PathSearchResult pathSearchResult = new PathSearchResult(initialPosition, ant.getAntId());
    while (Boolean.FALSE.equals(ant.isFound())) {
        CityNode nextCityNode = (ant.getAntType() == AntType.WILD) ? nextCityMoveForWildAnt(ant) : nextCityMove(ant);
        if (nextCityNode.isNodeFound()) {
            ant.visitCity(nextCityNode.getIndex());
            pathSearchResult.addCityIndex(nextCityNode.getIndex(), isLast: false);
        } else {
            pathSearchResult.addCityIndex(initialPosition, isLast: true);
        }
    }
    return pathSearchResult;
}
```

```java
private CityNode nextCityMoveForWildAnt(Ant ant) {
    List<CityNode> availableCities = ant.availableCities();
    CityNode cityNode = new CityNode( index: null);
    if (!availableCities.isEmpty()) {
        cityNode = availableCities.get(randomNumber(0, availableCities.size()));
    }
    return cityNode;
}

private CityNode nextCityMove(Ant ant) {
    List<CityNode> availableCities = ant.availableCities();
    for (CityNode cityNode : availableCities) {
        double probability = countProbability(ant, ant.getCurrentCityIndex(), cityNode.getIndex());
        cityNode.setProbability(probability);
    }

    CityNode cityNode = new CityNode( index: null);
    if (!availableCities.isEmpty()) {
        cityNode = availableCities.stream().max(Comparator.comparing(CityNode::getProbability))
                .orElseThrow(() -> new RuntimeException("City not found"));
    }
    return cityNode;
}
```

```java
private double countProbability(Ant ant, int from, int to) {
    double pheromoneValue = Math.pow(pheromoneAtPath(from, to), salesmanProblemDto.getA());
    double visionValue = Math.pow(ant.visionAtPath(from, to), salesmanProblemDto.getB());
    double antsValue = sumAntCities(ant, from);
    double probability = 0;
    if (antsValue != 0) {
        probability = (pheromoneValue * visionValue) / (antsValue);
    }
    return probability;
}

private double sumAntCities(Ant ant, int from) {
    List<Integer> availableCitiesIndexes = ant.availableCities() List<CityNode>
            .stream() Stream<CityNode>
            .map(CityNode::getIndex) Stream<Integer>
            .collect(Collectors.toList());
    double sum = 0.0;
    for (Integer index : availableCitiesIndexes) {
        double pheromoneValue = Math.pow(pheromoneAtPath(from, index), salesmanProblemDto.getA());
        double visionValue = Math.pow(ant.visionAtPath(from, index), salesmanProblemDto.getB());
        sum += (pheromoneValue * visionValue);
    }
    return sum;
}
```

3.1.2 Приклади роботи

На рисунках 3.1 і 3.2 показані приклади роботи програми.

```
SalesmanProblemDto{A=1.0, B=3.0, L_MIN=2200, R=0.3, numberOfAnts=200, antPlacementType=MANY_WITH_REPEAT}
Solution
PathSearchResult{path=13-83-33-11-22-87-53-78-32-60-98-17-69-71-26-65-59-23-74-16-93-9-70-40-55-37-54-15-28-10-45-89-92-95-20-19-76-99-72-56-0-12-35-38-31-47-50-41-4-79-96-86-39-46-61-63-81-88-73
-27-77-97-66-3-90-82-2-58-94-75-91-30-44-25-6-8-84-18-36-34-29-64-24-57-67-42-80-5-21-43-7-51-68-62-1-48-52-14-85-49-13, pathCost=899}
```

Рисунок 3.1 – Робота при задних параметрах для 100 міст

```
SalesmanProblemDto{A=1.0, B=3.0, L_MIN=2200, R=0.3, numberOfAnts=200, antPlacementType=MANY_WITH_REPEAT}
Solution
PathSearchResult{path=79-27-284-253-266-48-197-228-56-126-203-211-127-171-147-22-201-245-113-247-65-235-108-262-46-44-204-39-251-17-227-124-222-281-268-158-21-267-78-43-114-221-168-20-30-117-135
-224-239-188-250-198-256-81-77-0-86-38-80-84-279-106-178-6-238-82-216-214-271-2-187-26-166-165-270-173-159-157-210-252-213-73-93-176-67-183-163-112-192-130-5-153-132-133-180-1-146-100-10-104-101
-119-233-254-209-282-175-217-99-291-276-179-275-285-66-261-36-76-71-290-184-297-277-195-62-55-32-292-61-40-33-3-215-174-218-125-128-54-143-69-177-150-182-264-98-223-60-91-109-29-141-244-193-31-283
-225-96-207-181-102-14-37-45-129-298-202-138-90-140-142-75-191-53-241-164-258-263-186-156-232-137-212-50-272-136-230-243-89-23-139-236-229-85-196-110-189-154-72-134-9-299-25-115-13-167-7-219-105
-286-63-111-208-255-295-242-107-148-246-155-259-265-120-83-206-64-144-278-131-4-169-293-11-231-18-15-234-161-123-289-274-12-294-269-103-92-16-41-200-248-199-149-19-95-94-185-74-287-87-237-59-162
-172-42-280-58-35-296-288-47-145-190-28-249-205-57-118-240-24-51-152-8-121-151-160-170-194-70-97-220-88-226-52-68-122-116-260-34-257-273-49-79, pathCost=1955}
```

Рисунок 3.2 – Робота при задних параметрах для 300 міст

3.2 Тестування алгоритму

Таблиця 3.1

Параметр	**Значення**
α	1
β	6
ρ	0,5
Lmin	2200
M (звичайні)	200
M (елітні)	20
M (дикі)	40
G_SIZE кількість міст	300
Розміщення	В усіх вершинах з повтореннями

Виконавши аналіз та тестування алгоритму було підібрано оптимальні вхідні значення. В ході підбору було визначено що найбільше на кінцевий результат впливають параметри α, β, ρ. Дані параметри використовуються для вираховування ймовірності включення ребра в цикл. При цьому ймовірність включення ребра в цикл окремого мурашки пропорційна кількості феромону на цьому ребрі, а кількість феромона, що відкладали пропорційна довжині циклу.

Чим коротша довжина циклу тим більше буде відкладено феромона на ребрах тому для коефіцієнта випаровування ρ було підібране значення 0,5. Адже якщо феромон випаровується швидко то це призводить до втрати пам’яті колонії і оптимальні рішення втрачаються. В свою чергу швидке випаровування призводить до отримання постійного локального рішення.

Значення Lmin на початку буде менше за розв’язок але після декількох ітерацій при буде більше за розв’язок. Тоді на ребрі буде відкладатися більше феромону що буде свідчити про знаходження оптимального рішення.

Кількість мурах розраховується залежно від кількості вершин які потрібно пройти. Кількість елітних та диких мурах не повинна перевищувати кількість звичайни тому, що у іншому випадку рішення буде не оптимальне.

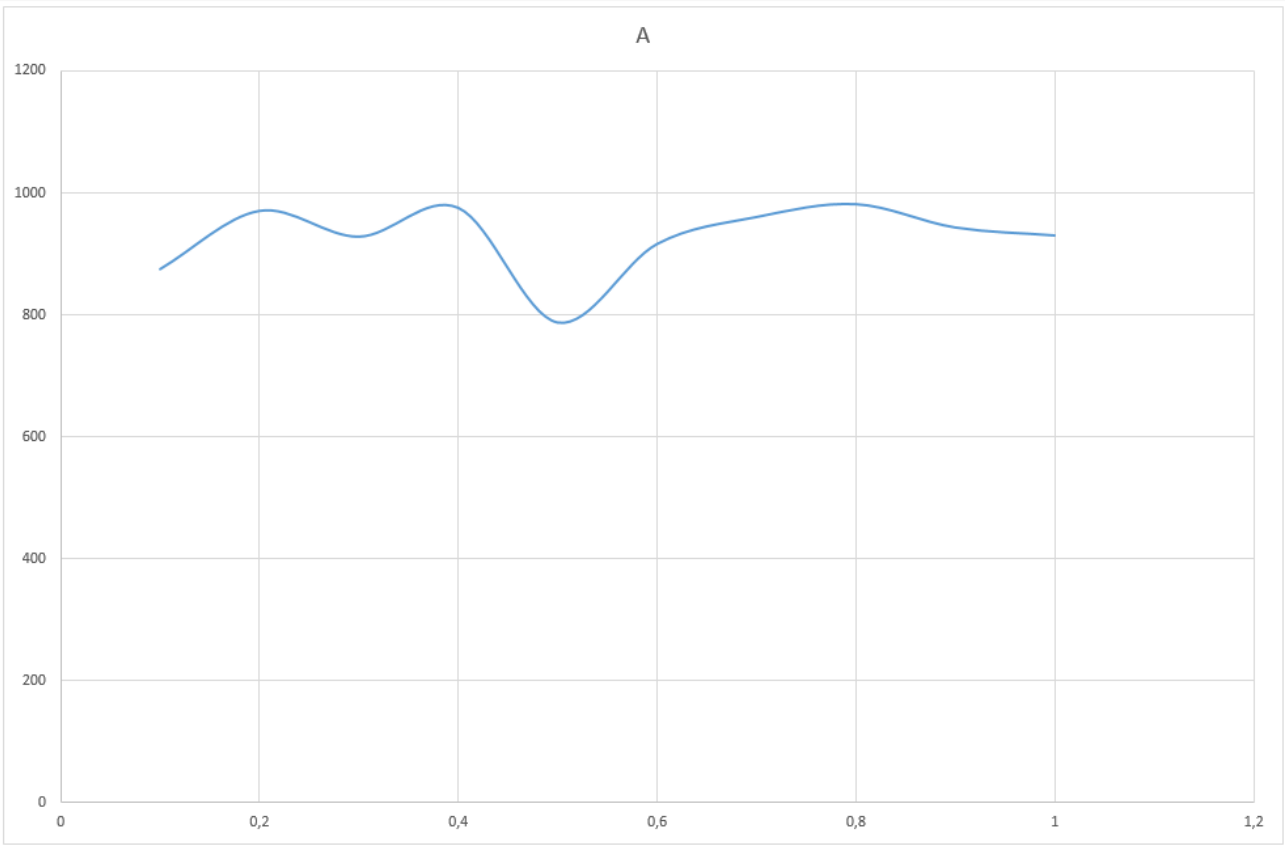


Рисунок 3.3 – Графік залежності значення розв’язку від параметра α

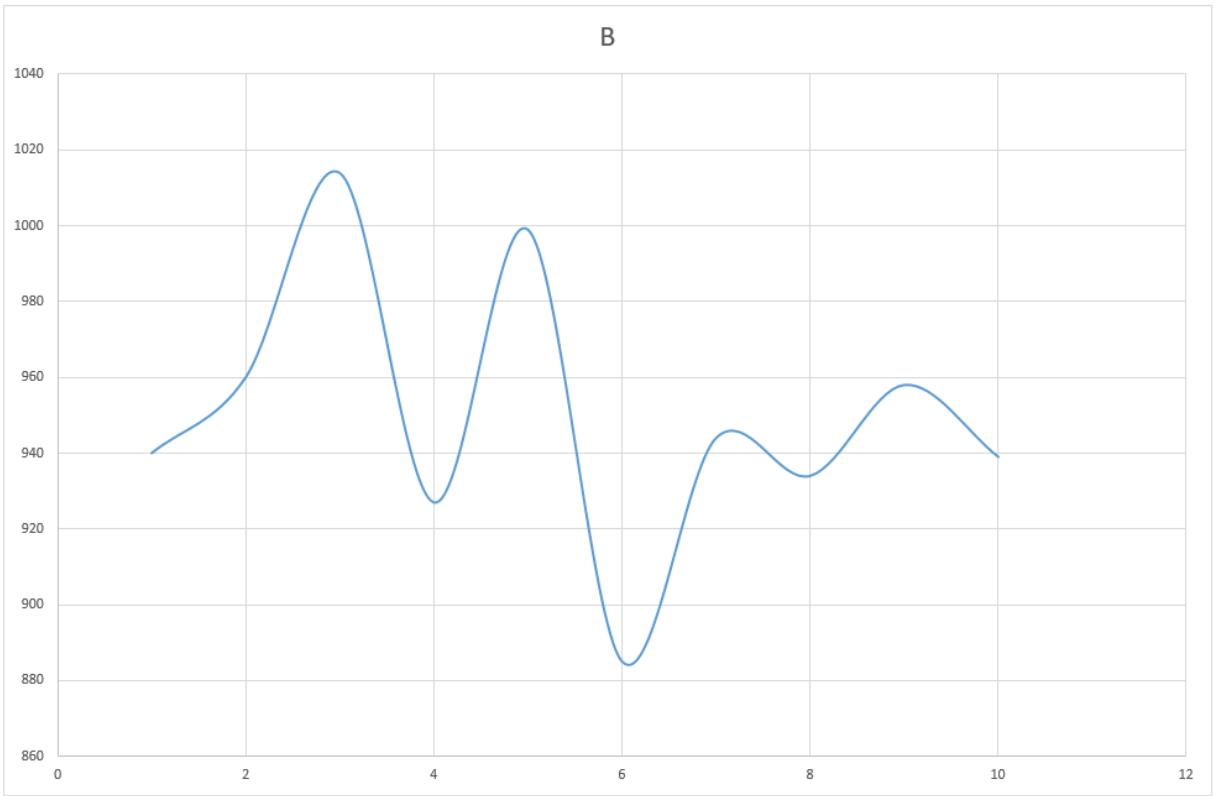


Рисунок 3.4 – Графік залежності значення розв’язку від параметра β

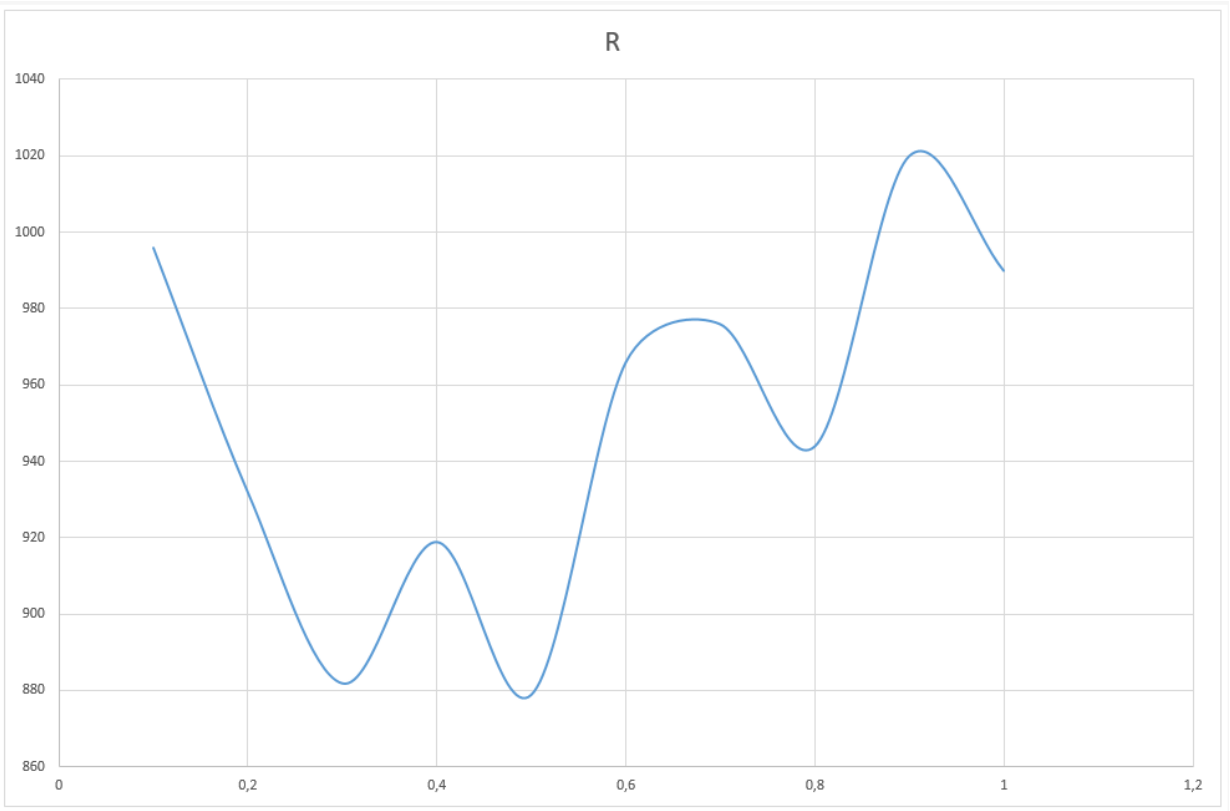


Рисунок 3.4 – Графік залежності значення розв’язку від параметра ρ

ВИСНОВОК

В рамках даної лабораторної роботи було вивчено основні підходи розробки метаевристичних алгоритмів для типових прикладних задач. А саме було написано програму яка вирішує типову задачу комівояжера з використанням Мурашиного алгоритму. В ході виконання даної лабораторної роботи було підібрано параметри алгоритму а саме (α, β, ρ, Lmin, кількість мурах M і їх типи (елітні, звичайні, дикі) , маршрути з однієї чи різних вершин) Отже виконавши дану лабораторну роботу було опрацювано методологію підбору прийнятних параметрів алгоритму та засвоєно розробку метаевристичних алгоритмів.

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%.