Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н. Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н. Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

ОТЧЕТ

по лабораторной работе №6

по курсу «Анализ Алгоритмов»

на тему: «Методы решения задачи коммивояжера»

Студент ИУ7-53Б _____ Лысцев Н. Д.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Волкова Л. Л.
(Подпись, дата) (И. О. Фамилия)

2024 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цель лабораторной работы — описание методов решения задачи коммивояжера полным перебором и на основе муравьиного алгоритма.

Для достижения поставленной цели необходимо решить следующие задачи:

1) изучить и описать задачу коммивояжера;
2) изучить и описать методы решения задачи коммивояжера — метод полного перебора и метод на основе муравьиного алгоритма;
3) разработать и реализовать программный продукт, позволяющий решить задачу коммивояжера исследуемыми методами;
4) сравнить по времени метод полного перебора и метод на основе муравьиного алгоритма.
5) обосновать полученные результаты в отчете о выполненной лабораторной работе.

Выданный индивидуальный вариант для выполнения лабораторной работы:

- неориентированый граф;
- без элитных муравьев;
- карта городов России от Калининграда до Владивостока;
- незамкнутый маршрут.

1 Аналитический раздел

В данном разделе будет рассмотрена задача коммивояжера и будут описаны алгоритмы её решения.

1.1 Задача коммивояжера

Цель задачи коммивояжера [1] заключается в нахождении самого выгодного маршрута (кратчайшего, самого быстрого, наиболее дешевого), проходящего через все заданные точки (пункты, города) по одному разу.

Условия задачи должны содержать критерий выгодности маршрута (должен ли он быть максимально коротким, быстрым, дешевым или все вместе), а также исходные данные в виде матрицы затрат (расстояния, стоимости, времени) при перемещении между рассматриваемыми пунктами.

1.2 Алгоритм полного перебора

Рассмотрим n городов и матрицу расстояний между ними. Найдем самый короткий маршрут посещения всех городов ровно по одному разу, без возвращения в первый город:

1) число вариантов для выбора первого города равно n;

2) число вариантов для выбора второго города равно $n - 1$;

3) с каждым следующим городом число вариантов уменьшается на 1;

4) число всех вариантов маршрутра равно $n!$;

5) минимальный по сумме значений матрицы расстояний вариант маршрута — искомый.

В связи со сложностью $n!$ полный перебор вариантов занимает существенное время, а при большом количестве городов становится технически невозможным.

1.3 Муравьиный алгоритм

Идея муравьиного алгоритма [2] — моделирование поведения муравьев, связанное с их способностью быстро находить кратчайший путь и адаптироваться к изменяющимся условиям, находя новый кратчайший путь.

Муравей обладает следующими чувствами:

- `зрение` — муравей k, находясь в городе i, может оценить длину (метку) D_{ij} дуги/ребра $i-j$ и привлекательность этого ребра/дуги, описываемую формулой 1.1:

$$\eta_{ij} = 1/D_{ij}, \tag{1.1}$$

- `обоняние` — муравей чует концентрацию феромона $\tau_{ij}(t)$ на ребре/дуге $i-j$ в текущий день t.

- `память` — муравей k запоминает список/кортеж посещенных за текущий день t городов — $J_k(t)$

Муравей выполняет следующую последовательность действий, пока не посетит все города:

1) находясь в городе i, муравей k выбирает следующий город j по вероятностному правилу (формула 1.2):

$$P_{ij,k}(t) = \begin{cases} 0, & j \in J_k(t) \\ \dfrac{\eta_{ij}^{\alpha} \cdot \tau_{ij}^{\beta}}{\sum\limits_{q \notin J_k(t)} \eta_{iq}^{\alpha} \cdot \tau_{iq}^{\beta}}, & \text{если город } j \text{ необходимо посетить} \end{cases} \tag{1.2}$$

где α — параметр влияния видимости пути, $\beta = 1 - \alpha$ — параметр влияния феромона, τ_{ij} — число феромона на дуге/ребре $i-j$, η_{ij} — эвристическое желание посетить город j, если муравей находится в городе i. Выбор города является вероятностным, данное правило определяет ширину зоны города j. В общую зону всех не посещенных муравьем городов бросается случайное число, которое и определяет выбор муравья;

2) муравей k проходит путь по дуге/ребре $i - j$ и оставляет на нем феромон.

Информация о числе феромона на пути используется другими муравьями для выбора пути. Те муравьи, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащен феромоном. Следующие муравьи будут предпочитать именно этот путь, продолжая обогащать его феромоном.

После прохождения маршрутов всеми муравьями (ночью, перед наступлением следующего дня) значение феромона на путях обновляется в соответствии со следующим правилом (1.3):

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \Delta\tau_{ij}, \tag{1.3}$$

где $\rho \in [0,1]$ — коэффициент испарения. Чтобы найденное локальное решение не было единственным, моделируется испарение феромона.

При этом

$$\Delta\tau_{ij} = \sum_{k=1}^{N} \Delta\tau_{ij,k}, \tag{1.4}$$

где N — число муравьев,

$$\Delta\tau_{ij,k} = \begin{cases} 0, & \text{если ребро/дуга } i-j \text{ не посещена муравьем } k \text{ в день } t \\ \frac{Q}{L_k}, & \text{иначе} \end{cases} \tag{1.5}$$

где L_k — длина маршрута (сумма меток дуг/ребер) k-го муравья в день t, Q — квота (некоторая константа) феромона 1-го муравья на день, Q выбирается соразмерной длине наименьшего маршрута (формула 1.6).

$$Q = \frac{\sum \text{недиагональных элементов матрицы смежности}}{\text{линейный размер матрицы смежности}} \tag{1.6}$$

Поскольку вероятность 1.2 перехода в заданную точку не должна быть равна нулю, необходимо обеспечить неравенство $\tau_{ij}(t)$ нулю посредством введения дополнительного минимально возможного значения феромона τ_{min} и в случае, если $\tau_{ij}(t+1)$ принимает значение, меньшее τ_{min}, откатывать значение феромона

до этой величины τ_{min}.

Вывод

В данном разделе была рассмотрена задача коммивояжера, а также способы её решения — алгоритм полного перебора и муравьиный алгоритм.

2 Конструкторский раздел

В данном разделе будут представлены схемы алгоритма полного перебора и муравьиного алгоритма.

2.1 Разработка алгоритмов

2.1.1 Алгоритм полного перебора

На рисунке 2.1 представлена схема алгоритма полного перебора для решения задачи коммивояжера.

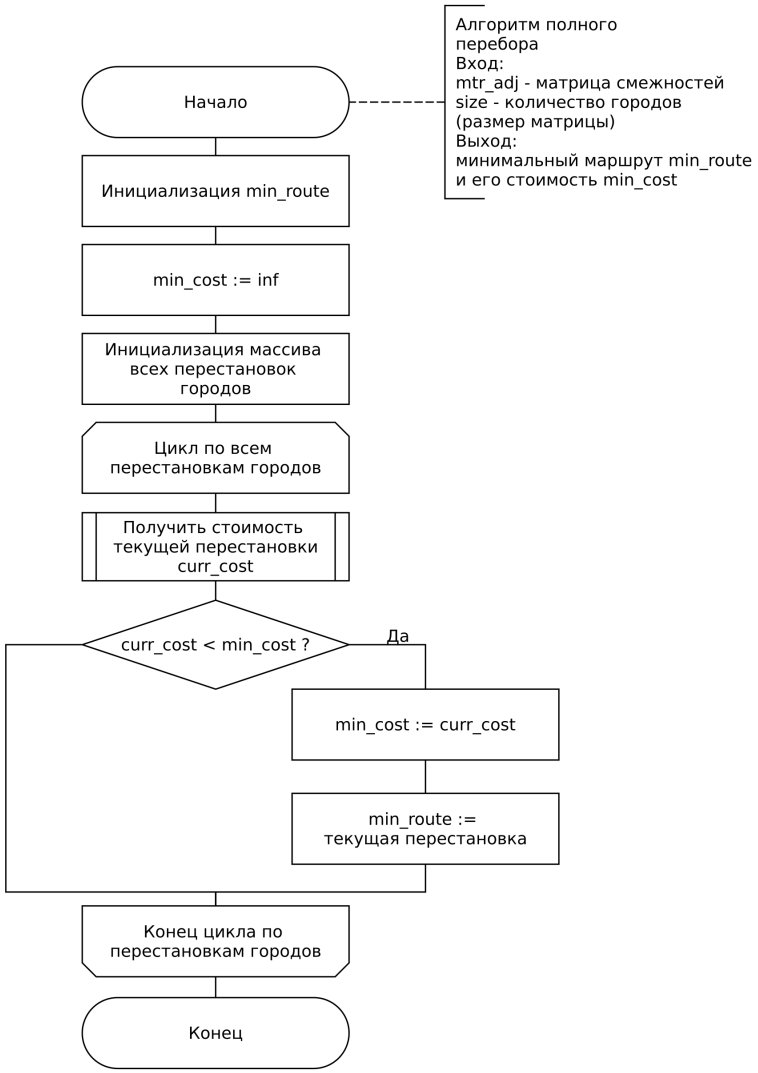

Рисунок 2.1 – Схема алгоритма полного перебора

2.1.2 Муравьиный алгоритм

На рисунке 2.2 представлена схема муравьиного алгоритма для решения задачи коммивояжера.

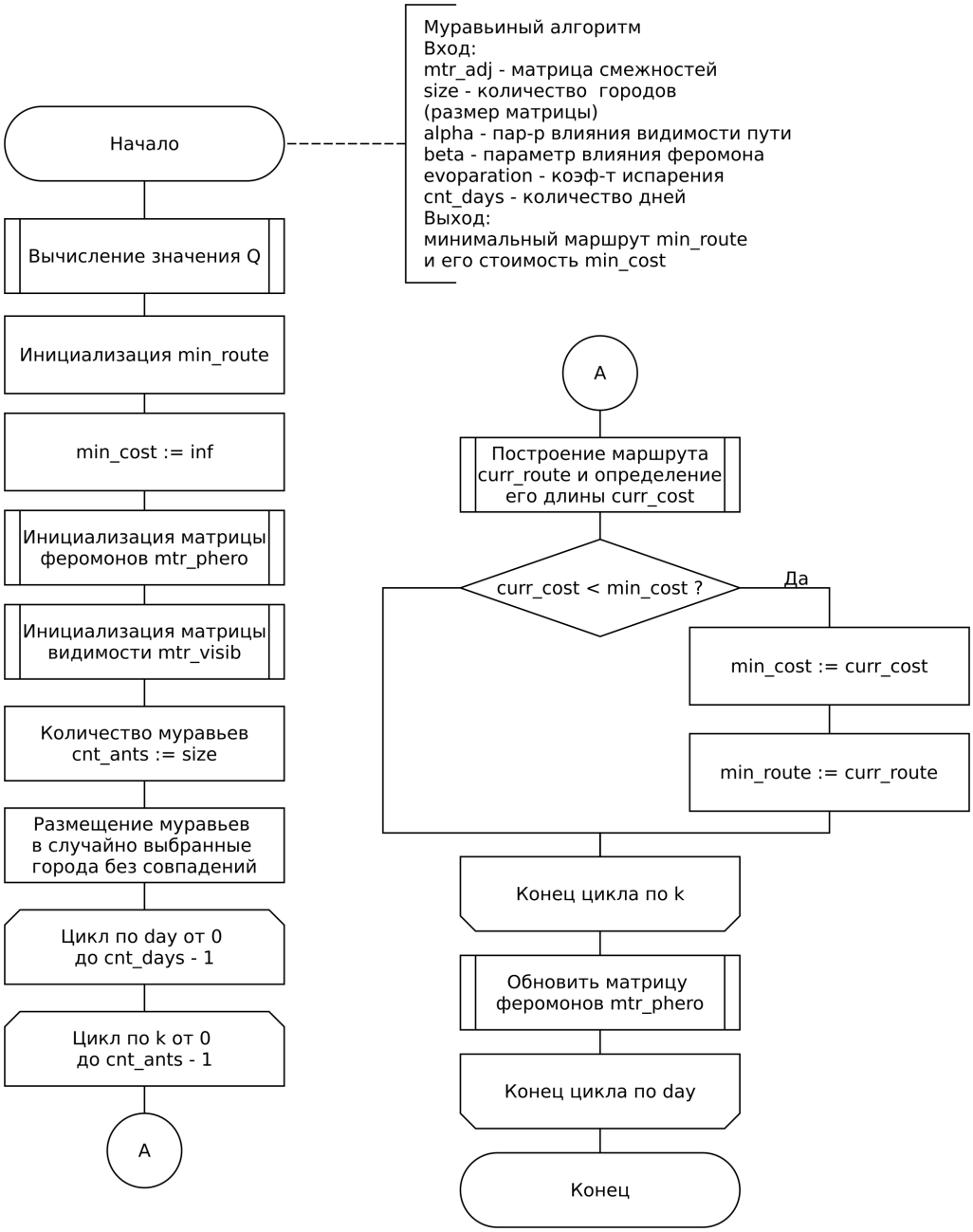

Рисунок 2.2 – Схема муравьиного алгоритма

2.2 Структура разрабатываемого ПО

Для реализации разрабатываемого программного обеспечения будет использоваться метод структурного программирования. Каждый из алгоритмов будет представлен отдельной функцией, при необходимости будут выделены подпрограммы для каждой из них. Также будут реализованы функции для ввода-вывода и функция, вызывающая все подпрограммы для связности и полноценности программы.

Вывод

В данном разделе были представлены схемы алгоритма полного перебора и муравьиного алгоритма, была описана структура разрабатываемого ПО.

3 Технологический раздел

В данном разделе будут перечислены требования к программному обеспечению, средства реализации, функциональные тесты и листинги кода.

3.1 Требования к программному обеспечению

К программе предъявляется ряд требований:

- программа должна получать на вход матрицу смежности, для которой можно будет выбрать один из алгоритмов поиска оптимальных путей (полным перебором или муравьиным алгоритмом);
- программа должна позволять пользователю определять коэффициенты и количество дней для муравьиного алгоритма;
- программа должна давать возможность получить минимальную сумму пути, а также сам путь, используя один из алгоритмов.

3.2 Средства реализации

В качестве языка программирования для этой лабораторной работы был выбран *Python* [3] по следующим причинам:

- в *Python* есть библиотека *itertools* [4], содержащая функцию *permutations* для вычисления всех перестановок переданной последовательности;
- в *Python* есть библиотека *numpy* [5], предоставляющая необходимый функционал для работы с многомерными массивами;
- в *Python* есть библиотека *time* [6], содержащая функцию *process_time* для замеров процессорного времени.

В качестве функции, которая будет осуществлять замеры процессорного времени, будет использована функция *process_time* из библиотека *time* [6].

3.3 Сведения о модулях программы

Программа состоит из пяти модулей:

– `main.py` — файл, содержащий интерфейс программы и точку входа;

– `brute_force_alg.py` — файл, содержащий код алгоритма полного перебора;

– `ant_alg.py` — файл, содержащий код муравьиного алгоритма;

– `utils.py` — файл, содержащий служебные алгоритмы;

– `measurements.py` — файл, содержащий функции для выполнения замеров времени.

3.4 Реализации алгоритмов

В листинге 3.1 представлена реализация алгоритма полного перебора.

Листинг 3.1 – Реализация алгоритма полного перебора

```
def get_curr_cost(curr_route: list[int], mtr_adj: np.ndarray) ->
   int:
    curr_cost = 0

    for i in range(len(curr_route) - 1):
        curr_cost += mtr_adj[curr_route[i]][curr_route[i + 1]]

    return curr_cost


def brute_force_alg(mtr_adj: np.ndarray, size: int) ->
   tuple[list[int], int]:
    min_route = list()
    min_cost = float("inf")

    for perm in it.permutations(list(range(len(mtr_adj)))):
        curr_route = list(perm)

        curr_cost: int = get_curr_cost(curr_route, mtr_adj)

        if curr_cost < min_cost:
            min_route = curr_route
            min_cost = curr_cost

    return min_route, min_cost
```

В листингах 3.2 и 3.3 представлена реализация муравьиного алгоритма.

Листинг 3.2 – Реализация муравьиного алгоритма (начало)

```
def ant_alg(
        mtr_adj: np.ndarray,
        size: int,
        alpha: float,
        beta: float,
        evoparation: float,
        cnt_days: int
):
    Q: float = calc_Q(mtr_adj, size)
```

Листинг 3.3 – Реализация муравьиного алгоритма (конец)

```
    min_route = list()
    min_cost = float("inf")

    mtr_phero: np.ndarray = init_mtr_phero(size)
    mtr_visib: np.ndarray = init_mtr_visib(mtr_adj, size)

    cnt_ants = size

    for day in range(cnt_days):
        ants: list[dict] = [{'tabu': [j], 'cost': 0} for j in
           range(cnt_ants)]
        for k in range(cnt_ants):
            ants[k]: dict = get_ant_route(ants[k], mtr_adj,
               mtr_phero, mtr_visib, size, alpha, beta)

            if ants[k]['cost'] < min_cost:
                min_cost = ants[k]['cost']
                min_route = ants[k]['tabu']

        mtr_phero = update_phero(mtr_phero, size, ants, Q,
           evoparation)

    return min_route, min_cost
```

В листинге 3.4 представлена реализация функции вычисления константы Q.

Листинг 3.4 – Реализация функции для вычисления константы Q

```
def calc_Q(mtr_adj: np.ndarray, size: int) -> float:
    _sum = 0
    count = 0
    for i in range(size):
        for j in range(size):
            if i != j:
                _sum += mtr_adj[i][j]
                count += 1

    return _sum / count
```

В листинге 3.5 представлена реализация функции создания и инициализации матрицы феромонов и матрицы видимости.

Листинг 3.5 – Реализация функции создания и инициализации матрицы феромонов и матрицы видимости

```
def init_mtr_phero(size: int) -> np.ndarray:

    return np.ones((size, size))

def init_mtr_visib(mtr_adj: np.ndarray, size: int) -> np.ndarray:
    mtr_visib = [[(1.0 / mtr_adj[i][j] if i != j else 0) for j in
       range(size)] for i in range(size)]
    mtr_visib = np.array(mtr_visib)

    return mtr_visib
```

В листингах 3.6 и 3.7 представлена реализация функций для подсчета необходимых вероятностей.

Листинг 3.6 – Реализация функций подсчета необходимых вероятностей (начало)

```
def get_prob_numerator(
        city_from: int,
        city_to: int,
        mtr_phero: np.ndarray,
        mtr_visib: np.ndarray,
        alpha: float,
        beta: float
) -> float:
    P = (mtr_phero[city_from][city_to] ** alpha) *
       (mtr_visib[city_from][city_to] ** beta)

    return P

def get_probabilities(
        size: int,
        tabu_k: list,
        mtr_phero: np.ndarray,
        mtr_visib: np.ndarray,
        alpha: float,
```

Листинг 3.7 – Реализация функций подсчета необходимых вероятностей (конец)

```
        beta: float
) -> np.ndarray:
    probabilities = np.zeros(size)

    for city in range(size):
        if city not in tabu_k:
            curr_city = tabu_k[-1]

            probabilities[city] = get_prob_numerator(curr_city,
               city, mtr_phero, mtr_visib, alpha, beta)

    denominator = sum(probabilities)

    probabilities /= denominator

    return probabilities
```

В листингах 3.8 и 3.9 представлена реализация функций для определения маршрута муравья.

Листинг 3.8 – Реализация функции для определения следующего города

```
def get_next_city(
        tabu_k: list[int],
        mtr_phero: np.ndarray,
        mtr_visib: np.ndarray,
        size: int,
        alpha: float,
        beta: float
) -> int:
    probabilities: np.ndarray = get_probabilities(size, tabu_k,
       mtr_phero, mtr_visib, alpha, beta)
    choice: float = random()
    probSum = probabilities[0]
    next_city = 0
    while choice > probSum and next_city < size - 1:
        next_city += 1
        probSum += probabilities[next_city]
    return next_city
```

Листинг 3.9 – Реализация функции для определения маршрута муравья

```
def get_ant_route(
        ant_k: dict,
        mtr_adj: np.ndarray,
        mtr_phero: np.ndarray,
        mtr_visib: np.ndarray,
        size: int,
        alpha: float,
        beta: float
) -> dict:
    while len(ant_k['tabu']) != size:
        nextCity = get_next_city(ant_k['tabu'], mtr_phero,
           mtr_visib, size, alpha, beta)

        ant_k['cost'] += mtr_adj[ant_k['tabu'][-1]][nextCity]
        ant_k['tabu'].append(nextCity)

    return ant_k
```

В листингах 3.10 и 3.11 представлена реализация функции для обновления матрицы феромонов.

Листинг 3.10 – Реализация функции для обновления матрицы феромонов (начало)

```
def update_phero(
        mtr_phero: np.ndarray,
        size: int,
        ants: list[dict],
        Q: float,
        evoparation: float
) -> np.ndarray:
    add_mtr_phero = np.zeros((size, size))

    for ant_k in ants:
        delta = Q / ant_k['cost']

        for i in range(size - 1):
            add_mtr_phero[ant_k['tabu'][i]][ant_k['tabu'][i + 1]]
               += delta
```

Листинг 3.11 – Реализация функции для обновления матрицы феромонов (конец)

```
    mtr_phero += ((1 - evoparation) * mtr_phero + add_mtr_phero)

    for i in range(size):
        for j in range(size):
            mtr_phero[i][j] = 0.1 if mtr_phero[i][j] <
               MIN_PHEROMONE else mtr_phero[i][j]

    return mtr_phero
```

3.5 Функциональные тесты

В таблице 3.1 приведены тесты для функций программы. Все функциональные тесты пройдены успешно.

Таблица 3.1 – Функциональные тесты

Матрица смежности	Ожидаемый результат	Результат программы
$\begin{pmatrix} 0 & 4 & 2 & 1 & 7 \\ 4 & 0 & 3 & 7 & 2 \\ 2 & 3 & 0 & 10 & 3 \\ 1 & 7 & 10 & 0 & 9 \\ 7 & 2 & 3 & 9 & 0 \end{pmatrix}$	15, [0, 2, 4, 1, 3, 0]	15, [0, 2, 4, 1, 3, 0]
$\begin{pmatrix} 0 & 1 & 2 \\ 1 & 0 & 1 \\ 2 & 1 & 0 \end{pmatrix}$	4, [0, 1, 2, 0]	4, [0, 1, 2, 0]
$\begin{pmatrix} 0 & 15 & 19 & 20 \\ 15 & 0 & 12 & 13 \\ 19 & 12 & 0 & 17 \\ 20 & 13 & 17 & 0 \end{pmatrix}$	64, [0, 1, 2, 3, 0]	64, [0, 1, 2, 3, 0]

Вывод

В данном разделе были перечислены требования к программному обеспечению, средства реализации, функциональные тесты и листинги кода.

4 Исследовательский раздел

В данном разделе будет проведен сравнительный анализ алгоритмов при различных ситуациях на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором проводились исследования:

- операционная система: Ubuntu 22.04.3 LTS x86_64 [7];
- оперативная память: 16 Гб;
- процессор: 11th Gen Intel® Core™ i7-1185G7 @ 3.00 ГГц × 8, 4 физических ядра, 8 логических ядер.

4.2 Проведение первого исследования

Цель исследования

Целью исследования является проведение сравнительного анализа времени работы алгоритма полного перебора и муравьиного алгоритма от линейного размера матрицы смежности.

Наборы варьируемых и фиксированных параметров

Замеры времени проводились для линейных размеров матрицы, равных 2, 3, 4, 5, 6, 7, 8, 9, 10.

В качестве фиксированного параметра был выбран разброс значений для матрицы смежности от 5 до 10. Для муравьиного алгоритма были фиксированы следующие параметры:

- α — параметр влияния видимости пути (значение 0.5);
- β — параметр влияния феромона (значение 0.5);

– ρ — коэффициент испарения (значение 0.5);

– время жизни муравьиной колонии (значение 250).

Замеры времени для каждого линейного размера матрицы проводились 10 раз.

Результаты первого исследования

Таблица 4.1 – Замер времени для для линейных размеров матрицы от 2 до 10

Линейный размер матрицы, единицы	**Время, с**	
	Алгоритм полного перебора	**Муравьиный алгоритм**
2	0.000019	0.000323
3	0.000010	0.000901
4	0.000034	0.001879
5	0.000156	0.003271
6	0.001017	0.005237
7	0.007903	0.007688
8	0.070830	0.010854
9	0.708097	0.014859
10	7.946709	0.019937

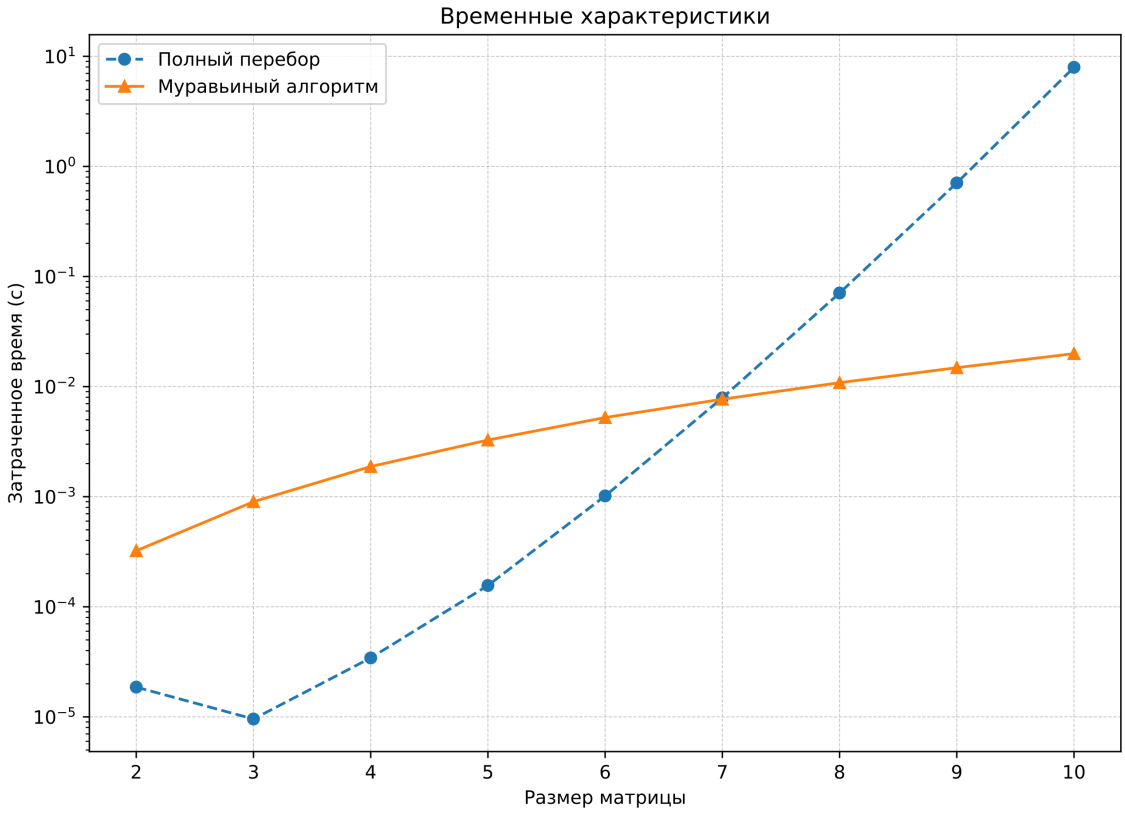

Рисунок 4.1 – Результаты замеров времени для для линейных размеров матрицы от 2 до 10

На линейных размерах матрицы смежности от 2 до 6 алгоритм полного перебора работает значительно быстрее муравьиного алгоритма (на матрице смежности размером 2×2 в 17 раз, а на матрице смежности размером 6×6 в 5.14 раза), но на линейных размерах матрицы смежности от 7 до 10 муравьиный алгоритм выигрывает у алгоритма полного перебора по времени выполнения (на матрице смежности размером 7×7 в 1.02 раз, а на матрице смежности размером 10×10 в уже в 398.59 раза). Это связано с тем, что алгоритм полного перебора имеет вычислительную сложность $O(n!)$, в то время как сложность муравьиного алгоритма $O(t \cdot m \cdot n^2)$, где t — количество итераций, m — количество муравьев, n — число вершин [8].

4.3 Проведение второго исследования

Автоматическая параметризация была проведена на единственном классе данных, состоящем из трех полносвязных графов с 10 вершинами в каждом с одинаковым разбросом значений меток ребер/дуг графа.

Итоговая таблица значений параметризации будет состоять из следующих колонок:

— α — параметр влияния видимости пути;

— ρ — коэффициент испарения;

— *days* — количество дней жизни колонии муравьев;

— *Result* — эталонный результат, полученный методом полного перебора для проведения данного эксперимента;

— *Mistake* — разность полученного основанным на муравьином алгоритме методом значения и эталонного значения на данных значениях параметров, показатель качества решения.

Цель исследования

Цель исследования — определить комбинацию параметров, которые позволяют решить задачу наилучшим образом для выбранного класса данных. Качество решения зависит от количества дней и погрешности измерений.

4.3.1 Граф №1

Согласно с вариантом индивидуального задания представим полносвязный неориентированный граф, вершинами которого являются города от Калининграда до Владивостока:

1) Калининград;

2) Санкт-Петербург;

3) Москва;

4) Нижний Новгород;

5) Екатеринбург;

6) Омск;

7) Новосибирск;

8) Красноярск;

9) Иркутск;

10) Владивосток.

Меткой ребра этого графа будет выступать расстояние между городами в километрах.

Матрица расстояний этого графа (формула 4.1):

$$K_1 = \begin{pmatrix} 0 & 826 & 1089 & 1483 & 2483 & 3303 & 3858 & 4362 & 5209 & 7359 \\ 826 & 0 & 634 & 896 & 1782 & 2584 & 3105 & 3574 & 4416 & 6538 \\ 1089 & 634 & 0 & 401 & 1415 & 2236 & 2812 & 3354 & 4203 & 6418 \\ 1483 & 896 & 401 & 0 & 1016 & 1835 & 2412 & 2962 & 3810 & 6031 \\ 2483 & 1782 & 1415 & 1016 & 0 & 820 & 1398 & 1968 & 2811 & 5060 \\ 3303 & 2584 & 2236 & 1835 & 820 & 0 & 609 & 1229 & 2043 & 4324 \\ 3858 & 3105 & 2812 & 2412 & 1398 & 609 & 0 & 629 & 1434 & 3716 \\ 4362 & 3574 & 3354 & 2962 & 1968 & 1229 & 629 & 0 & 849 & 3103 \\ 5029 & 4416 & 4203 & 3810 & 2811 & 2043 & 1434 & 849 & 0 & 2285 \\ 7359 & 6538 & 6418 & 6031 & 5060 & 4324 & 3716 & 3103 & 2285 & 0 \end{pmatrix} \quad (4.1)$$

Для данного графа приведена таблица 4.2 с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.2 – Выборка из параметров для графа №1

α	ρ	Days	Result	Mistake
0.1	0.1	50	8069	0
0.1	0.1	300	8069	0
0.1	0.2	50	8069	0
0.1	0.2	100	8069	0
0.1	0.2	500	8069	0
0.1	0.3	3	8069	0
0.1	0.3	500	8069	0
0.1	0.4	500	8069	0
0.1	0.5	500	8069	0
0.1	0.6	300	8069	0
0.1	0.6	500	8069	0
0.1	0.7	10	8069	0
0.1	0.7	300	8069	0
0.1	0.7	500	8069	0
0.1	0.8	50	8069	0
0.1	0.8	500	8069	0
0.2	0.1	300	8069	0
0.2	0.1	500	8069	0
0.2	0.2	300	8069	0
0.2	0.4	300	8069	0
0.2	0.4	500	8069	0
0.2	0.5	50	8069	0
0.2	0.5	500	8069	0
0.2	0.6	500	8069	0
0.2	0.7	100	8069	0
0.2	0.8	100	8069	0
0.2	0.8	300	8069	0
0.2	0.8	500	8069	0
0.3	0.1	300	8069	0
0.3	0.5	500	8069	0
0.3	0.6	500	8069	0
0.4	0.3	500	8069	0
0.4	0.4	300	8069	0
0.5	0.8	500	8069	0

4.3.2 Граф №2

Граф №2 представляет собой матрицу смежности размером 10 элементов (разброс значений от 500 до 9999), которая представлена в формуле 4.2.

$$K_2 \begin{pmatrix} 0 & 2054 & 4772 & 2285 & 8108 & 7511 & 1603 & 4980 & 4946 & 6961 \\ 2054 & 0 & 9820 & 7343 & 1913 & 3186 & 9741 & 9824 & 8436 & 4025 \\ 4772 & 9820 & 0 & 2700 & 7491 & 1448 & 1528 & 5163 & 9493 & 5676 \\ 2285 & 7343 & 2700 & 0 & 7228 & 5495 & 8920 & 6178 & 6876 & 8871 \\ 8108 & 1913 & 7491 & 7228 & 0 & 897 & 3805 & 8851 & 8109 & 7628 \\ 7511 & 3186 & 1448 & 5495 & 897 & 0 & 9043 & 7877 & 7899 & 1560 \\ 1603 & 9741 & 1528 & 8920 & 3805 & 9043 & 0 & 3334 & 8791 & 5418 \\ 4980 & 9824 & 5163 & 6178 & 8851 & 7877 & 3334 & 0 & 4370 & 633 \\ 4946 & 8436 & 9493 & 6876 & 8109 & 7899 & 8791 & 4370 & 0 & 3545 \\ 6961 & 4025 & 5676 & 8871 & 7628 & 1560 & 5418 & 633 & 3545 & 0 \end{pmatrix} \quad (4.2)$$

Для данного графа приведена таблица 4.3 с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.3 – Выборка из параметров для графа №2 (начало)

α	ρ	Days	Result	Mistake
0.1	0.1	100	17258	0
0.1	0.1	300	17258	0
0.1	0.1	500	17258	0
0.1	0.2	50	17258	0
0.1	0.2	500	17258	0
0.1	0.3	50	17258	0
0.1	0.3	300	17258	0
0.1	0.3	500	17258	0
0.1	0.4	100	17258	0
0.1	0.4	500	17258	0
0.1	0.5	300	17258	0
0.1	0.5	500	17258	0
0.1	0.6	300	17258	0
0.1	0.7	100	17258	0
0.1	0.7	300	17258	0
0.1	0.8	300	17258	0
0.1	0.8	500	17258	0
0.2	0.1	300	17258	0
0.2	0.1	500	17258	0
0.2	0.4	500	17258	0
0.2	0.5	300	17258	0
0.2	0.5	500	17258	0
0.2	0.6	100	17258	0
0.2	0.6	300	17258	0
0.2	0.6	500	17258	0
0.2	0.7	500	17258	0
0.2	0.8	500	17258	0
0.3	0.2	3	17258	0
0.3	0.2	300	17258	0
0.3	0.2	500	17258	0
0.3	0.6	300	17258	0
0.3	0.6	500	17258	0
0.3	0.7	50	17258	0
0.3	0.7	100	17258	0
0.3	0.7	500	17258	0
0.3	0.8	500	17258	0

Таблица 4.4 – Выборка из параметров для графа №2 (конец)

α	ρ	Days	Result	Mistake
0.4	0.3	500	17258	0
0.4	0.6	100	17258	0
0.4	0.6	300	17258	0
0.4	0.6	500	17258	0
0.4	0.8	500	17258	0
0.5	0.7	500	17258	0
0.5	0.8	500	17258	0
0.6	0.5	50	17258	0

4.3.3 Граф №3

Граф №3 представляет собой матрицу смежности размером 10 элементов (разброс значений от 500 до 9999), которая представлена в формуле 4.3.

$$K_3 \begin{pmatrix} 0 & 7350 & 4050 & 1676 & 6394 & 4900 & 5605 & 8001 & 8137 & 1936 \\ 7350 & 0 & 705 & 5103 & 1038 & 4199 & 1786 & 3492 & 6111 & 5218 \\ 4050 & 705 & 0 & 6688 & 9899 & 4109 & 3122 & 8410 & 1078 & 4511 \\ 1676 & 5103 & 6688 & 0 & 8275 & 9030 & 1376 & 7076 & 8435 & 8500 \\ 6394 & 1038 & 9899 & 8275 & 0 & 7240 & 2432 & 4687 & 1021 & 9136 \\ 4900 & 4199 & 4109 & 9030 & 7240 & 0 & 6837 & 2817 & 1987 & 4830 \\ 5605 & 1786 & 3122 & 1376 & 2432 & 6837 & 0 & 7360 & 4111 & 7236 \\ 8001 & 3492 & 8410 & 7076 & 4687 & 2817 & 7360 & 0 & 9470 & 8066 \\ 8137 & 6111 & 1078 & 8435 & 1021 & 1987 & 4111 & 9470 & 0 & 1313 \\ 1936 & 5218 & 4511 & 8500 & 9136 & 4830 & 7236 & 8066 & 1313 & 0 \end{pmatrix} \quad (4.3)$$

Для данного графа приведена таблица 4.5 с выборкой параметров, которые наилучшим образом решают поставленную задачу.

Таблица 4.5 – Выборка из параметров для графа №3

α	ρ	Days	Result	Mistake
0.1	0.1	300	15045	0
0.1	0.1	500	15045	0
0.1	0.2	300	15045	0
0.1	0.2	500	15045	0
0.1	0.3	300	15045	0
0.1	0.3	500	15045	0
0.1	0.4	500	15045	0
0.1	0.5	100	15045	0
0.1	0.5	500	15045	0
0.1	0.6	300	15045	0
0.1	0.6	500	15045	0
0.1	0.7	300	15045	0
0.1	0.8	50	15045	0
0.2	0.1	300	15045	0
0.2	0.1	500	15045	0
0.2	0.2	50	15045	0
0.2	0.2	500	15045	0
0.2	0.3	300	15045	0
0.2	0.5	500	15045	0
0.2	0.6	300	15045	0
0.2	0.6	500	15045	0
0.2	0.8	500	15045	0
0.3	0.2	300	15045	0
0.3	0.2	500	15045	0
0.3	0.4	500	15045	0
0.3	0.5	500	15045	0
0.3	0.7	300	15045	0
0.3	0.8	50	15045	0
0.3	0.8	500	15045	0
0.5	0.2	300	15045	0
0.5	0.4	500	15045	0
0.5	0.5	100	15045	0
0.5	0.6	300	15045	0
0.5	0.6	500	15045	0
0.6	0.6	300	15045	0
0.6	0.7	500	15045	0
0.7	0.7	500	15045	0

Вывод

В результате исследования было получено, что использование муравьиного алгоритма наиболее эффективно при больших размерах матриц. Так, на матрице смежности размером 2×2 в 17 раз, а на матрице смежности размером 6×6 в 5.14 раза), но на линейных размерах матрицы смежности от 7 до 10 муравьиный алгоритм выигрывает у алгоритма полного перебора по времени выполнения (на матрице смежности размером 7×7 в 1.02 раз, а на матрице смежности размером 10×10 в уже в 398.59 раза). Следовательно, при размерах матриц больше 9 следует использовать муравьиный алгоритм, но стоит учитывать, что он не гарантирует получения глобального оптимума при решении задачи.

Для класса данных было получено, что наилучшим образом алгоритм работает на значениях параметров, которые представлены далее:

— $\alpha = 0.1, \rho = 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8$;

— $\alpha = 0.2, \rho = 0.1, 0.4, 0.5, 0.6, 0.8$;

— $\alpha = 0.3, \rho = 0.6$;

Для этого класса данных рекомендуется использовать данные параметры.

ЗАКЛЮЧЕНИЕ

В ходе выполнения лабораторной работы были решены следующие задачи:

1) изучены и описаны задачу коммивояжера;

2) изучены и описаны методы решения задачи коммивояжера — метод полного перебора и метод на основе муравьиного алгоритма;

3) разработан и реализован программный продукт, позволяющий решить задачу коммивояжера исследуемыми методами;

4) проведено сравнение по времени метод полного перебора и метод на основе муравьиного алгоритма.

5) обоснованы полученные результаты в отчете о выполненной лабораторной работе.

Цель данной лабораторной работы, а описание методов решения задачи коммивояжера полным перебором и на основе муравьиного алгоритма, также была достигнута.

Исходя из полученных результатов можно сказать, что использование муравьиного алгоритма наиболее эффективно при больших размерах матриц. Так, на матрице смежности размером 2×2 в 17 раз, а на матрице смежности размером 6×6 в 5.14 раза), но на линейных размерах матрицы смежности от 7 до 10 муравьиный алгоритм выигрывает у алгоритма полного перебора по времени выполнения (на матрице смежности размером 7×7 в 1.02 раз, а на матрице смежности размером 10×10 в уже в 398.59 раза). Следовательно, при размерах матриц больше 9 следует использовать муравьиный алгоритм, но стоит учитывать, что он не гарантирует получения глобального оптимума при решении задачи.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Задача коммивояжера — метод ветвей и границ [Электронный ресурс]. — Режим доступа: https://galyautdinov.ru/post/zadacha-kommivoyazhera (дата обращения: 02.02.2024).
2. М.В. Ульянов Ресурсно-эффективные компютерные алгоритмы. Разработка и анализ. — 2007.
3. Python 3.12.1 documentation [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/index.html (дата обращения: 03.02.2024).
4. itertools - Functions creating iterators for efficient looping [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/library/itertools.html (дата обращения: 03.02.2024).
5. NumPy documentation [Электронный ресурс]. — Режим доступа: https://numpy.org/doc/stable/ (дата обращения: 03.02.2024).
6. time - Time access and conversions [Электронный ресурс]. — Режим доступа: https://docs.python.org/3/library/time.html (дата обращения: 03.02.2024).
7. Ubuntu 22.04.3 LTS (Jammy Jellyfish) [Электронный ресурс]. — Режим доступа: https://releases.ubuntu.com/22.04/ (дата обращения: 28.09.2022).
8. Реализация и сравнительный анализ алгоритмов различных типов для решения задачи коммивояжера [Электронный ресурс]. — Режим доступа: http://elibrary.sgu.ru/VKR/2016/02-03-03_006.pdf (дата обращения: 03.02.2024).