МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Лабораторная работа №6
по дисциплине
«Генетические алгоритмы»
Вариант 18

Студент,
группы 5130201/20101 _____ Тищенко А. А.

Преподаватель _____ Большаков А. А.

«_____»_____ 2025г.

Санкт-Петербург, 2025

Содержание

1 Постановка задачи **3**

2 Теоретические сведения **4**

2.1 Общие сведения о муравьиных алгоритмах 4

2.2 Простой муравьиный алгоритм (SACO) 4

2.3 Обновление феромона 4

2.4 Испарение феромона 5

2.5 Критерии остановки алгоритма 5

2.6 Описание общего алгоритма 5

3 Особенности реализации **7**

3.1 Структуры данных конфигурации и результата 7

3.2 Класс AntColonyOptimizer и инициализация 7

3.3 Построение тура муравьём 8

3.4 Основной цикл алгоритма 8

3.5 Точка входа 9

3.6 Визуализация 9

4 Результаты работы **10**

4.1 Сравнение с результатами лабораторной работы №3 11

5 Ответ на контрольный вопрос **12**

Заключение **13**

Список литературы **14**

1 Постановка задачи

В данной работе были поставлены следующие задачи:

- Реализовать с использованием муравьиных алгоритмов решение задачи коммивояжера по индивидуальному заданию согласно номеру варианта.
- Представить графически найденное решение
- Сравнить найденное решение с представленным в условии задачи оптимальным решением и результатами, полученными в лабораторной работе №3.

Индивидуальное задание вариант 18:

Дано: Эвклидовы координаты городов 38 городов в Джибути (см. Приложение А). Оптимальный тур представлен на Рис. 1, его длина равна 6659.

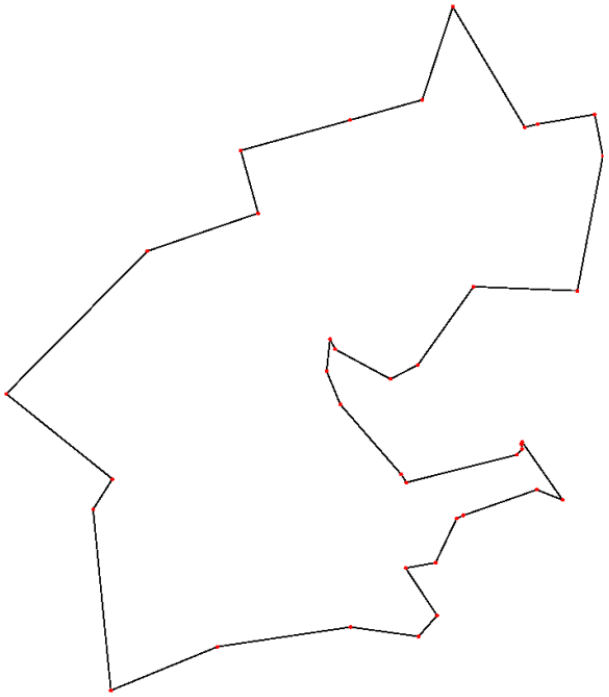

Рис. 1. Оптимальный тур для заданного набора данных

2 Теоретические сведения

2.1 Общие сведения о муравьиных алгоритмах

Муравьиные алгоритмы (MA) относятся к метаэвристическим методам оптимизации и предназначены преимущественно для решения задач комбинаторной оптимизации, в частности задачи поиска оптимальных путей на графах. Основная идея таких алгоритмов основана на моделировании коллективного поведения реальных муравьёв, использующих феромонные следы для обмена информацией.

Каждый агент, называемый *искусственным муравьём*, поэтапно строит решение задачи, перемещаясь по графу и выбирая следующую вершину на основе вероятностного правила, учитывающего концентрацию феромона на дугах графа. Феромон отражает привлекательность соответствующих маршрутов: чем выше его концентрация на дуге, тем вероятнее выбор этой дуги муравьём.

2.2 Простой муравьиный алгоритм (SACO)

Для иллюстрации рассмотрим простой муравьиный алгоритм SACO (Simple Ant Colony Optimization). Пусть задан граф

$$G = (V, E),$$

где V — множество вершин, E — множество рёбер. Каждой дуге (i, j) сопоставлена величина феромона τ_{ij}.

В начальный момент концентрация феромона обычно принимается нулевой, однако для предотвращения зацикливания каждому ребру присваивается малое случайное начальное значение $\tau_{ij}^{(0)}$.

Каждый муравей $k = 1, \ldots, n_k$ помещается в стартовую вершину и начинает построение пути. Если муравей находится в вершине i, он выбирает следующую вершину $j \in N_i^k$ на основе вероятностного правила

$$p_{ij}^k(t) = \frac{\tau_{ij}^{\alpha}(t)}{\sum\limits_{l \in N_i^k} \tau_{il}^{\alpha}(t)},$$

где α — параметр, определяющий степень влияния феромона.

При отсутствии допустимых переходов допускается возврат в предыдущую вершину, что приводит к появлению петель, которые впоследствии удаляются.

После завершения построения полного пути $x_k(t)$ выполняется его оценка. Длина пути обозначается как $L_k(t)$ и равна числу пройденных дуг.

2.3 Обновление феромона

Каждый муравей откладывает феромон на рёбрах своего пути согласно правилу

$$\Delta\tau_{ij}^k(t) = \begin{cases} \frac{1}{L_k(t)}, & \text{если дуга } (i,j) \in x_k(t), \\ 0, & \text{иначе.} \end{cases}$$

Общее обновление феромона на дуге (i, j):

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n_k} \Delta\tau_{ij}^{k}(t).$$

Чем короче путь, тем больше феромона откладывается на его рёбрах, что повышает вероятность выбора коротких маршрутов в последующих итерациях.

2.4 Испарение феромона

Чтобы предотвратить преждевременную сходимость алгоритма к локальным минимумам, применяется механизм *искусственного испарения феромона*. На каждом шаге выполняется:

$$\tau_{ij}(t) = (1 - \rho)\,\tau_{ij}(t),$$

где $\rho \in [0, 1]$ — коэффициент испарения. Большие значения ρ усиливают случайность поиска, малые — повышают устойчивость к изменениям.

2.5 Критерии остановки алгоритма

Муравьиные алгоритмы могут завершаться при выполнении одного из условий:

- достигнуто максимальное число итераций;
- найдено решение приемлемого качества $f(x_k(t)) \leq \varepsilon$;
- все муравьи начинают строить одинаковые маршруты, что говорит о стабилизации процесса.

2.6 Описание общего алгоритма

Алгоритм SACO можно представить в следующем виде:

1. Инициализация феромона малыми случайными значениями $\tau_{ij}^{(0)}$.
2. Размещение всех муравьёв в начальной вершине.
3. Для каждой итерации:
   (a) Каждый муравей строит путь согласно вероятностному правилу выбора вершины.
   (b) Выполняется удаление петель.
   (c) Вычисляется длина пути $L_k(t)$.
4. Выполняется испарение феромона.
5. Каждый муравей откладывает феромон на рёбрах своего пути.

6. Итерация продолжается до выполнения критерия остановки.

Муравьиные алгоритмы позволяют эффективно находить приближённые решения задач комбинаторной оптимизации, таких как задача коммивояжёра, что и является целью данной лабораторной работы.

3 Особенности реализации

Код решения собран в модуле `lab6/aco.py`. Реализация использует объектно-ориентированный подход с явной типизацией через современные аннотации типов Python (PEP 604). Ниже приведены ключевые элементы реализации с сигнатурами функций и пояснениями.

3.1 Структуры данных конфигурации и результата

Конфигурация алгоритма оформлена через `@dataclass` и включает все параметры, влияющие на поведение ACO:

```
@dataclass
class ACOConfig:
    cities: Sequence[City]      # список координат городов
    n_ants: int                 # число муравьев
    n_iterations: int           # число итераций
    alpha: float = 1.0          # влияние феромона
    beta: float = 5.0           # влияние эвристики расстояние(1/)
    rho: float = 0.5            # коэффициент испарения
    q: float = 1.0              # константа для отложения феромона
    seed: int | None = None     # зерно ГСЧ воспроизводимость()
```

Результат работы алгоритма представлен структурой:

```
@dataclass
class ACOResult:
    best_tour: Tour             # индексы городов в порядке обхода
    best_length: float          # длина лучшего маршрута
    history: List[float]        # история длин по итерациям
```

3.2 Класс AntColonyOptimizer и инициализация

Основная логика инкапсулирована в классе `AntColonyOptimizer`, который принимает конфигурацию при создании:

```
class AntColonyOptimizer:
    def __init__(self, config: ACOConfig)
```

В конструкторе выполняются следующие действия:

- инициализация генератора случайных чисел через `random.seed(config.seed)` для обеспечения воспроизводимости экспериментов;
- вычисление матрицы расстояний между всеми городами с помощью `build_distance_matrix`
- создание матрицы феромона размером $n \times n$, где все недиагональные элементы инициализируются единицами, а диагональные — нулями (для предотвращения самопереходов).

3.3 Построение тура муравьём

Каждый муравей строит полный гамильтонов цикл, начиная со случайно выбранного стартового города. Ключевой метод выбора следующего города:

```
1 def _choose_next_city(self, current: int,
2                       unvisited: set[int]) -> int
```

Метод реализует вероятностный выбор на основе формулы:

$$p_{ij} = \frac{[\tau_{ij}]^{\alpha} \cdot [\eta_{ij}]^{\beta}}{\sum_{k \in \text{unvisited}} [\tau_{ik}]^{\alpha} \cdot [\eta_{ik}]^{\beta}}$$

где τ_{ij} — уровень феромона на ребре (i, j), а $\eta_{ij} = 1/d_{ij}$ — эвристическая привлекательность (обратная величина расстояния). К расстоянию добавляется малая константа 10^{-12} для численной стабильности при делении. Финальный выбор осуществляется через `random.choices` с вычисленными вероятностями.

Построение полного тура выполняет метод:

```
1 def _build_tour(self, start: int) -> Tour
```

Начиная со стартового города, муравей последовательно выбирает следующие непосещённые города до тех пор, пока множество `unvisited` не станет пустым.

Вычисление длины построенного тура:

```
1 def _tour_length(self, tour: Sequence[int]) -> float
```

Метод суммирует расстояния между последовательными городами в туре, включая замыкающее ребро от последнего города к первому, используя предвычисленную матрицу расстояний.

3.4 Основной цикл алгоритма

Главный метод запуска оптимизации:

```
1 def run(self) -> ACOResult
```

На каждой из `n_iterations` итераций выполняются следующие шаги:

1. **Построение туров**: каждый из `n_ants` муравьёв создаёт свой маршрут, начиная со случайного города. Вычисляется длина каждого маршрута, и глобально лучший тур обновляется при обнаружении более короткого.

2. **Испарение феромона**: все элементы матрицы феромона умножаются на $(1 - \rho)$, моделируя естественное испарение. Это предотвращает неограниченный рост концентрации феромона и позволяет алгоритму «забывать» плохие решения.

3. **Отложение феромона**: для каждого муравья вычисляется вклад $\Delta\tau = q/L$, где L — длина его маршрута. Этот вклад добавляется симметрично на оба направления каждого ребра в туре. Таким образом, короткие маршруты откладывают больше феромона.

4. **Запись истории**: лучшая на данный момент длина добавляется в список `history` для последующего анализа сходимости.

По завершении всех итераций метод возвращает `ACOResult` с лучшим найденным туром, его длиной и историей оптимизации.

3.5 Точка входа

Для удобства использования предоставлена функция верхнего уровня:

```
1 def run_aco(config: ACOConfig) -> ACOResult
```

Она создаёт экземпляр оптимизатора и запускает алгоритм, возвращая результат.

3.6 Визуализация

Модуль включает две функции для визуализации результатов средствами `matplotlib`:

Функция построения графика маршрута:

```
1 def plot_tour(cities: Sequence[City], tour: Sequence[int],
2               save_path: str) -> None
```

Отображает города в виде точек и соединяет их ломаной линией в порядке обхода, включая возврат к начальной точке. Используется соотношение сторон `aspect="equal"` для сохранения геометрии, сетка для лучшей читаемости координат. Результат сохраняется в PNG с разрешением 220 DPI.

Функция построения графика сходимости:

```
1 def plot_history(best_lengths: Sequence[float],
2                  save_path: str) -> None
```

Строит линейный график изменения длины лучшего найденного тура по итерациям. Позволяет визуально оценить скорость сходимости и стабильность алгоритма.

4 Результаты работы

Алгоритм был запущен со следующими параметрами: 50 муравьёв, 50 итераций, $\alpha = 1{,}2$, $\beta = 5$, $\rho = 0{,}5$, $q = 1$. Лучший найденный тур имеет длину 6662,35, что на 0,05% отличается от оптимального значения 6659.

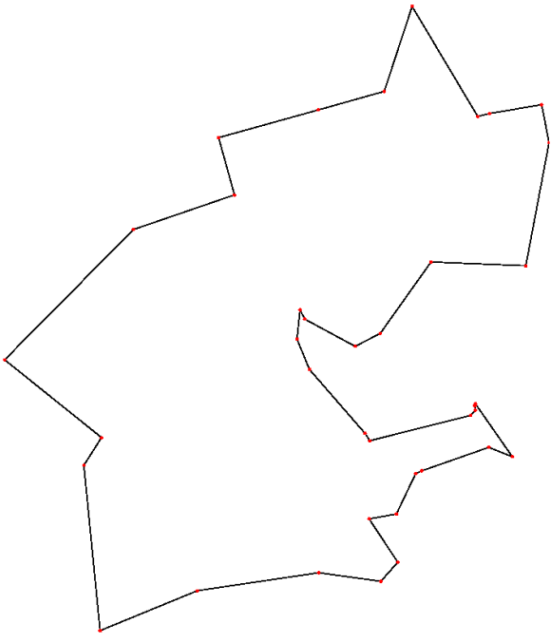

Рис. 2. Оптимальный маршрут длиной 6659

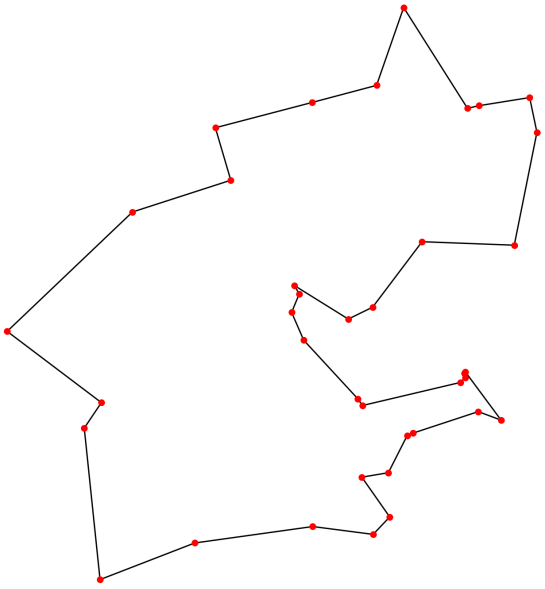

Рис. 3. Лучший маршрут, найденный муравьиным алгоритмом (6662,35)

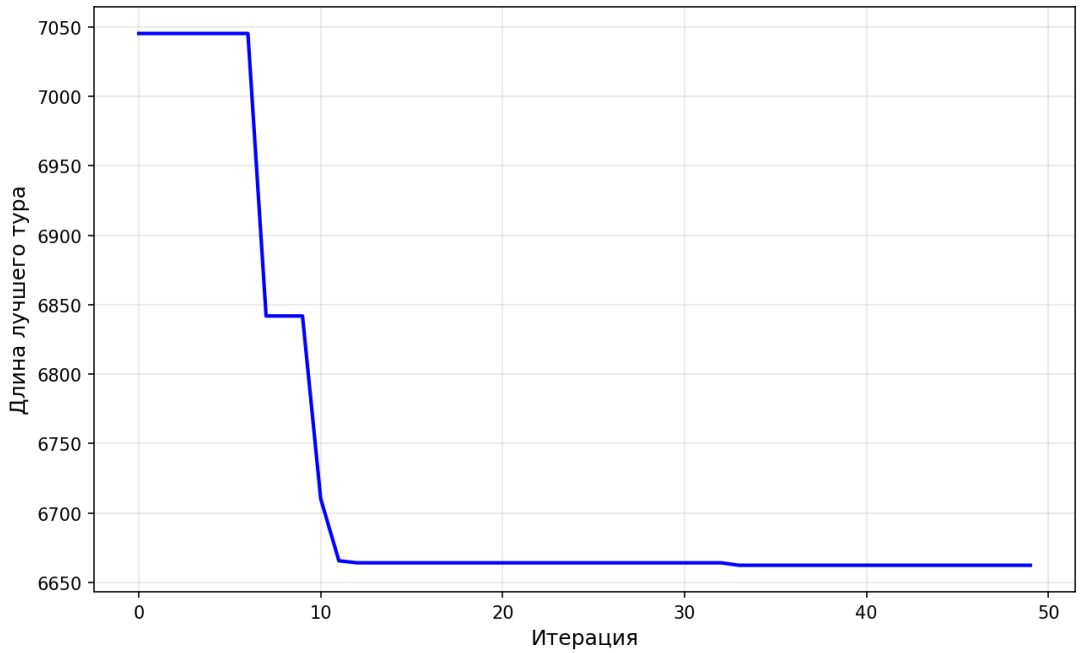

Рис. 4. Сходимость длины лучшего тура по итерациям

4.1 Сравнение с результатами лабораторной работы №3

Для лабораторной работы №3 с генетическим алгоритмом лучший результат составил **6667,03** при популяции $N = 500$, вероятностях $P_c = 0{,}9$ и $P_m = 0{,}5$. Муравьиный алгоритм показал более точное решение: длина тура **6662,35** против оптимального 6659. Разница с оптимумом составила 3,35 единицы (0,05%), тогда как в лабораторной работе №3 отклонение было 8,03 (0,12%).

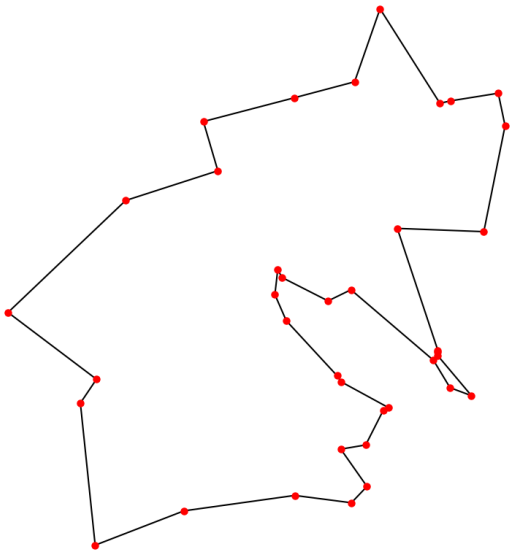

Рис. 5. Лучший маршрут из лабораторной работы №3 (ГА): длина 6667,03

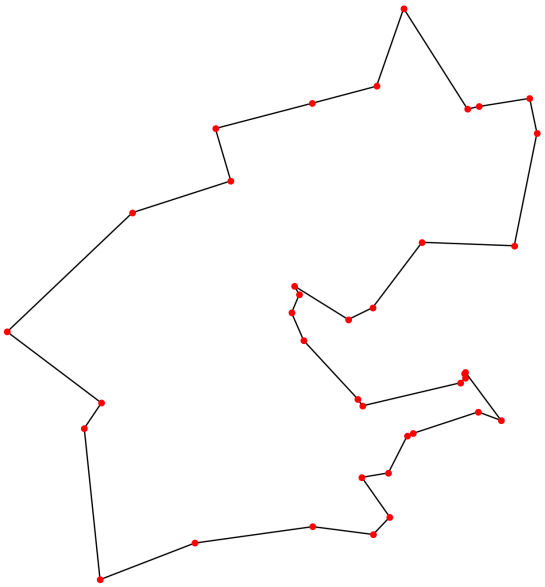

Рис. 6. Лучший маршрут лабораторной работы №6 (МА): длина 6662,35

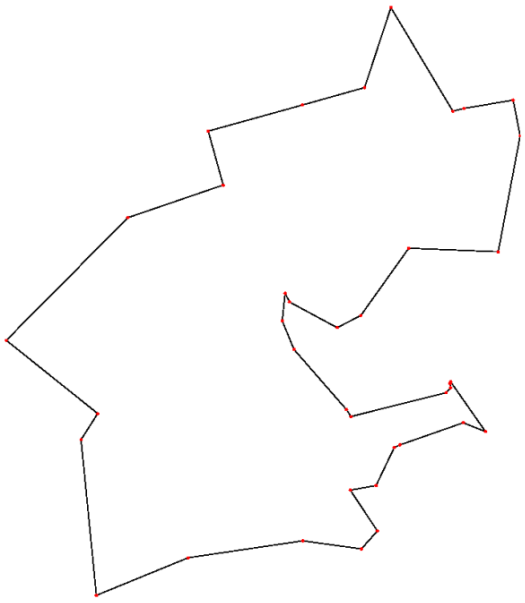

Рис. 7. Оптимальный маршрут длиной 6659

5 Ответ на контрольный вопрос

Вопрос: Какие критерии окончания могут быть использованы в простом МА?

Ответ: В простом муравьином алгоритме могут использоваться следующие критерии завершения работы:

- окончание при превышении заданного числа итераций;
- окончание по достижению приемлемого решения;
- окончание в случае, когда все муравьи начинают следовать одним и тем же путём.

Заключение

В ходе шестой лабораторной работы выполнена реализация простого муравьиного алгоритма для задачи коммивояжёра:

1. Разработан модуль `aco.py` с конфигурацией алгоритма, построением туров, обновлением феромона и визуализацией результатов с помощью `matplotlib`.
2. Проведён численный эксперимент на данных из варианта 18 (38 городов Джибути); подобраны параметры $\alpha = 1{,}2$, $\beta = 5$, $\rho = 0{,}5$, 50 муравьёв, 400 итераций.
3. Получено приближённое решение длиной 6662,35, что всего на 0,05% хуже известного оптимума 6659 и лучше результата, достигнутого генетическим алгоритмом из лабораторной работы №3.

Список литературы

[1] Методические указания по выполнению лабораторных работ к курсу «Генетические алгоритмы», 119 стр.