Санкт-Петербургский государственный университет
Факультет прикладной математики и процессов управления

Иследование принципов работы муравьиного алгоритма

Лабараторная работа №6

Аннотация

В данной работе представлено описание принципов работы муравьиного алгоритма, подкреплённое програмной и графической реализацией.

Ключевые слова: Муравьиный алгоритм, Python, Tkinter

Автор работы: Шайдуров В.Д.
Группа: 21.Б15-пу
Научный руководитель: Дик А.Г.

Санкт-Петербург
2023 г.

Содержание

1 Вступление

Муравьиный алгоритм – это метод оптимизации, основанный на поведении муравьев в поисках кратчайшего пути от муравейника до источника пищи. Этот алгоритм используется для решения различных задач, таких как коммивояжер и задача о рюкзаке. В данной лабораторной работе мы будем изучать принципы работы муравьиного алгоритма и его применение в задаче коммивояжера. Мы также рассмотрим различные параметры алгоритма и их влияние на его результаты работы.

2 Цель работы

Целью данной лабораторной работы является изучение основных принципов работы муравьиного алгоритма, а также его применение для задачи поиска минимального Гамильтонова пути.

3 Задача

Оценить эффективность муравьиного алгоритма и реализовать его визуализацию.

4 Описание програмы

В данном разделе приводиться описание кода на уровне идеи. Для более подробного понимания, рекомендуется самостоятельно ознакомиться с кодом по ссылке (github). Каждое дейсвие в программе сопровожденно коментариями, если возникают вопросы по терменалогии или по общему устройству программы обращайтесь к этому разделу.

Програмная реализация написана на языке python 3.10, с использованием популярных и общеупотребимых пакетов.

4.1 Теоретические сведения

Необходимые термины:

Популяция муравьёв - это количество муравьев, которые будут использоваться в алгоритме для поиска оптимального пути..

Коэффициент испарения феромона - это скорость испарения феромона с течением времени. Чем меньше коэффициент испарения феромона, тем дольше феромон будет оставаться на пути.

Коэффициент значимости феромона - это значение, которое определяет, насколько важен феромон для выбора следующего шага муравья. Чем выше коэффициент значимости феромона, тем больше внимания муравьи будут уделять следам феромона.

Коэффициент значимости растояния - это значение, которое определяет, насколько важны расстояния между городами для выбора следующего шага муравья. Чем выше коэффициент значимости, тем больше внимания муравьи будут уделять наименьшему по длине достумному пути.

Интенсивность начального феромона - это количество феромона, которое будет изначально размещено на каждом пути. В моей програме оно не изменяемое пользователем и равно 1.

4.2 Общий ход программы

1. Первым вызывается класс *class Ant* в который передаётся, по усмотрению пользовался: **веса путей полного графа, размер популяции колонии, коэффеценты важности феромона и растояния, а также количесво поколений**. В начале происходит нормализация весов путей и определяется таблица феромонов, заполняемая еденицами в качетсве начального значения.

2. Далее вызывается метод *findbestway*, который повторяется "**количество поколений**"раз. Где в каждом поколении для муравьев создаётся пустая таблица, в которую в будущем будут добавленны их выбранные маршруты. **Используя таблицу нормализированного растояния и таблицу начальных феромонов, а также коэффеценты важности феромона и растояния**, мы создаём массив, который означает вероятность муравья выбрать тот или иной путь. (т.е. длина этого массива = кол-ву вершин в графе) - **массив вероятности по путям**.

3. Таким образом для каждого муравья в поколении создаётся более менее уникальный маршрут, используя при генерации **массив вероятности по путям**.

4. Когда пути для муравьёв были определены, подсчитывется их длина и запоминается наиболее короткий маршрут.

5. В конце поколения происходит мутация таблицы феромонов и шаги [2-5] повторяются.

4.3 Описание програмной структуры генетического алгоритма

В Листиге 1 привёдн испольняемый код для класса Ant.

```python
class Ant:
    def __init__(self, size_pop, numberGeneration, graphPaths, alpha, beta, rho):
        """ Класс реализующий муравьиный алгоритм """
        self.graphPaths = graphPaths  # веса путей заданных в графе
        self.vertexes = len(self.graphPaths) + 1  # количество вершин
        self.size_pop = size_pop  # популяция муравьёв
        self.numberGeneration = numberGeneration  # количество итераций
        self.alpha = alpha  # коэффициент важности феромонов при выборе пути
        self.beta = beta  # коэффициент значимости расстояния
        self.rho = rho  # скорость испарения феромонов

        # Нормализованное расстояние
        self.prob_matrix_distance = [list(map(lambda x: 1/x, sublist)) for sublist in
        ↪  self.graphPaths]
        # Матрица феромонов, обновляющаяся каждую итерацию
        self.Tau = [list(map(lambda x: 1, sublist)) for sublist in self.graphPaths]

        self.generation_best_path, self.generation_best_score = [], []  # лучшие в своём поколении
        self.best_path, self.best_score = None, None

    def find_best_way(self) -> None:
        for _ in range(self.numberGeneration):
            # Путь каждого муравья в определённом поколении
            TablePath = np.zeros((self.size_pop, self.vertexes)).astype(int)
            # вероятность перехода без нормализации
            a_alfa = [[x ** self.alpha for x in sublist] for sublist in self.Tau]
            b_beta = [[x ** self.beta for x in sublist] for sublist in self.prob_matrix_distance]
            prob_matrix = [[float(x) * float(y) for x, y in zip(row_a, row_b)] for row_a, row_b in
            ↪  zip(a_alfa, b_beta)]
            for ant in range(self.size_pop):  # для каждого муравья
                TablePath[ant, 0] = 0
                for v in range(self.vertexes - 1):  # каждая вершина, которую проходят муравьи
                    # точка, которая была пройдена и не может быть пройдена повторно
                    allow_list = list(set(range(self.vertexes)) - set(TablePath[ant, :v + 1]))
                    prob = []
                    for i in allow_list:
                        from_ = min(TablePath[ant, v], i)
                        to_ = max(TablePath[ant, v], i) - from_ - 1
                        prob.append(prob_matrix[from_][to_])
                    prob = [p / sum(prob) for p in prob]  # нормализация вероятности
                    next_point = np.random.choice(allow_list, size=1, p=prob)[0]
                    TablePath[ant, v + 1] = next_point

            # рассечёт расстояния
            scores = np.array([self.distance(list(path)) for path in TablePath])
            # фиксация лучшего решения
            index_best = scores.argmin()
            self.generation_best_path.append(TablePath[index_best, :])
            self.generation_best_score.append(scores[index_best])

            # мутация феромона, который будет добавлен к ребрам нового поколения
            self.Tau = [list(map(lambda x: (1 - self.rho) * x, sublist)) for sublist in self.Tau]
            for ant in range(self.size_pop):  # для каждого муравья
                for v in range(self.vertexes - 1):  # для каждой вершины
                    # муравьи перебираются из вершины from_ в вершину to_
                    from_ = min(TablePath[ant, v], TablePath[ant, v+1])
                    to_ = max(TablePath[ant, v], TablePath[ant, v+1]) - from_ - 1
                    self.Tau[from_][to_] += 1 / scores[ant]  # нанесение феромона
                # муравьи ползут от последней вершины обратно к первой
                from_ = min(TablePath[ant, self.vertexes - 1], TablePath[ant, 0])
                to_ = max(TablePath[ant, self.vertexes - 1], TablePath[ant, 0]) - from_ - 1
                self.Tau[from_][to_] += 1 / scores[ant]
```

Listing 1: class SimulatedAnn

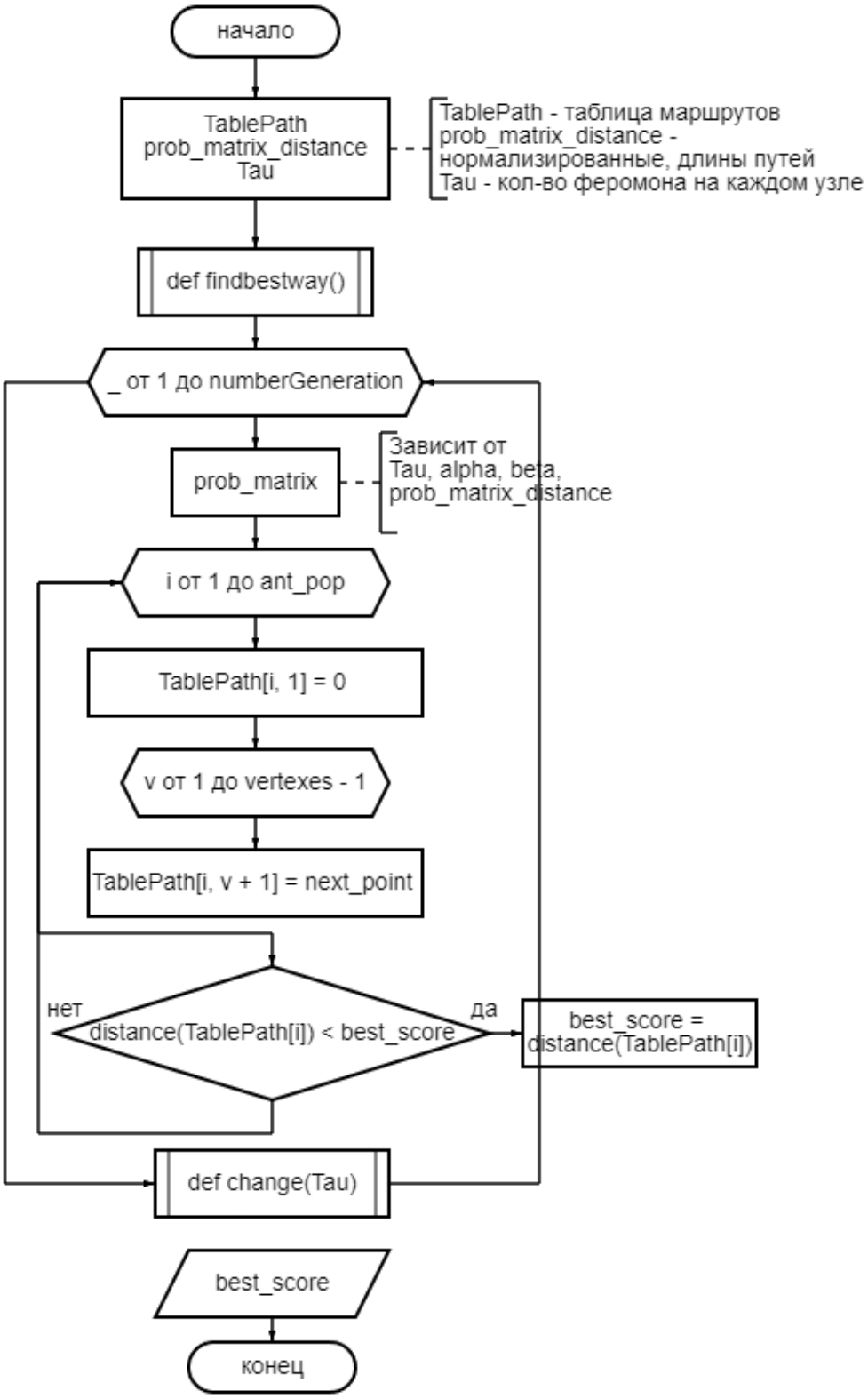

Рис. 1: Блоксхема программы.

5 Визуализация работы программы

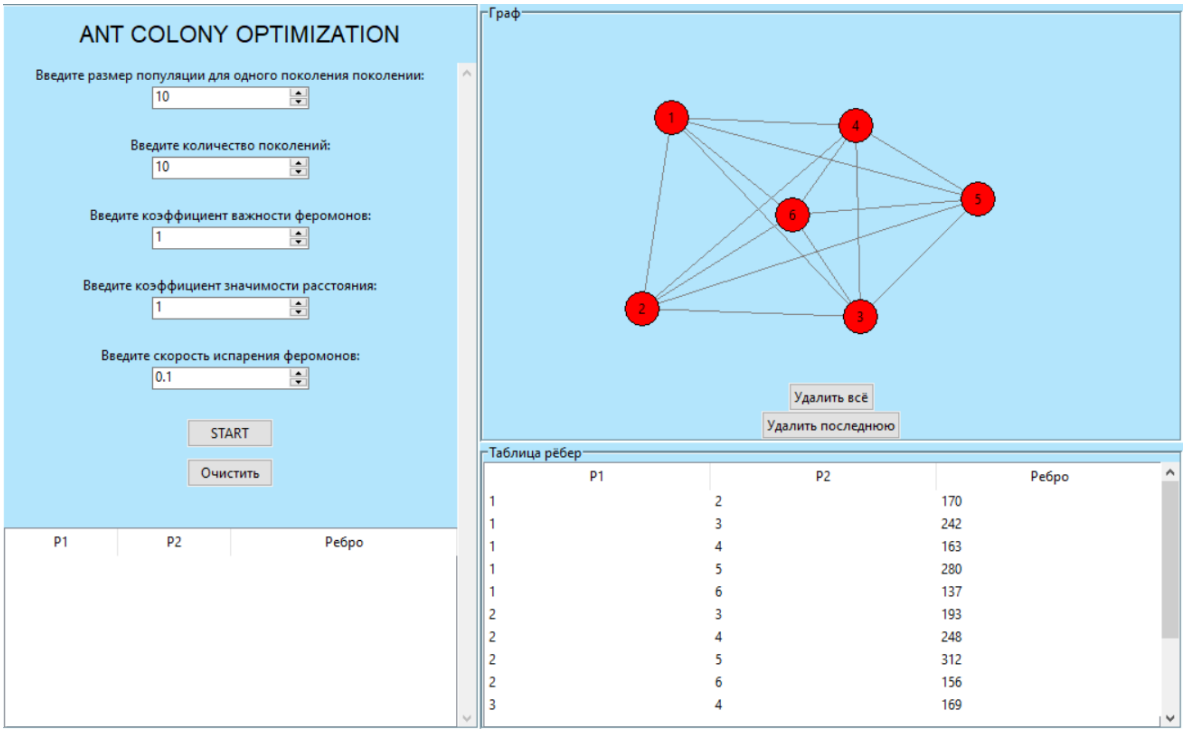

Рис. 2: Окно ввода.

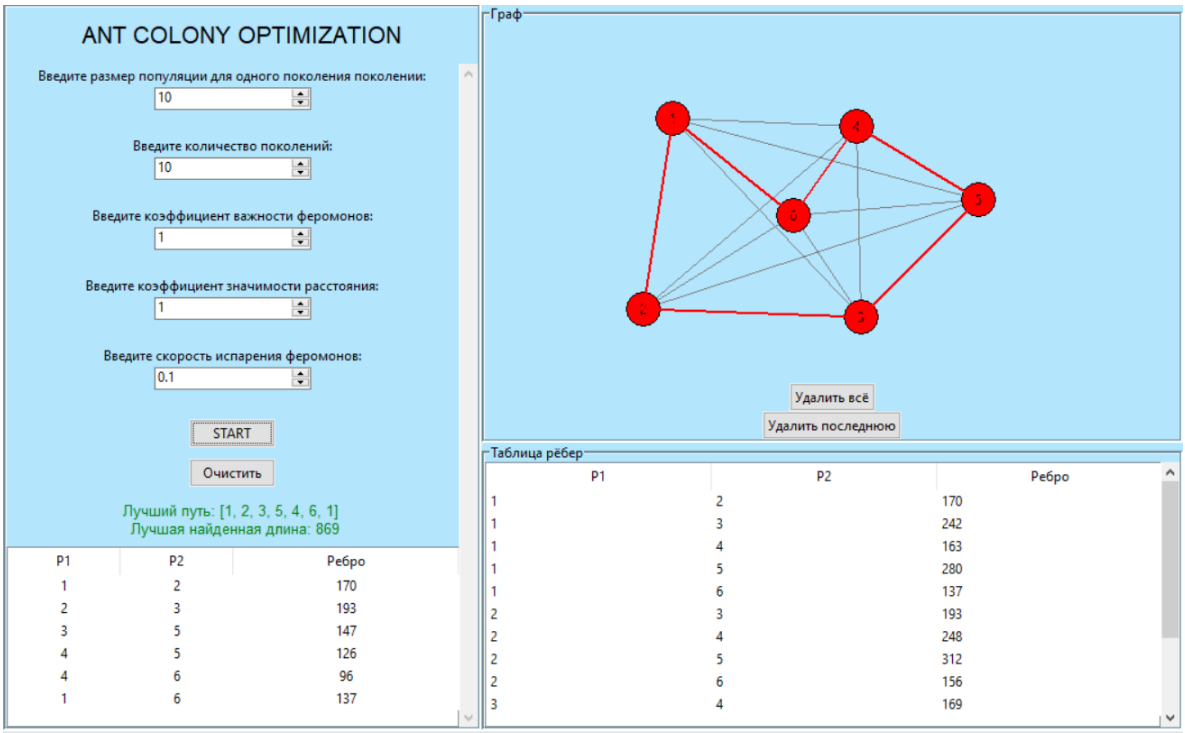

Рис. 3: Окно вывода.

Устройсво приложения:

В левой части окна можно вводить параметры влияющие на систему, такие как: **размер колонии (популяции), количество поколений, коэффициенты важности феровонов и значимости растояния, а также скорость испарения феромонов**.

Вверху справа графическое интерактивное окно, в котором можно добавлять точки одним щелчком мыши. Пути графа рисуются автоматически и добавляются в таблицу рёбер ниже. Весом ребра мо умолчанию является его длина в пикселях, которую можно легко изменить двойным нажатием мыши по нужной ячейке. Также в последней версии приложения была добавленна фкнкция перемещения точек графа.

Нажимая на кнопку старт, производяться рассчёты, результат которых выводиться под кнопкой зелёным цветом, а таблица заполняется маршрутом в указанном алгоритмом порядке.

6 Оценка эффективности алгоритма

Оценка эффективности муравьиного алгоритма в решении задачи коммивояжера показала, что он способен почти всегда найти лучшее решения при правильном выборе начальных параметров. Так на личном опыте для графа состоящего из 20 вершин, муравьиный алгоритм справился так же хорошо как и метод отжига , однако при одинаковом количестве поколеений муравьиный алгоритм обладает большим количством различных итераций из-за чего время его выполнения будет дольше, хоть и принято считать что его погрешность для графов большей размерности является меньшей.

Муравьиный алгоритм является эффективным алгоритмом для задачи коммивояжера, особенно когда другие методы оптимизации не могут обеспечить достаточную точность результата. Однако он как и свои предшественники входит в группу эврестических алгоритмов из-за чего он по прежнему не может гаронтировать, что найденное решаения является наилучшим.

Заключение

В данной работе был расмотрен муравьиный алгоритм на поиск минимального по длине Гамельтона цикла. Были выявленны его преимущества в сравнение с методом отжига. Была представленна его прогрманая визуализация и теоретическое растолкование.