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"

Тема Муравьиный алгоритм

Студент Иммореева М.А.

Группа ИУ7-52Б

Оценка (баллы) _____

Преподаватель Строганов Д.В.

Москва — 2023 г.

Оглавление

Введение

Задача коммивояжера – одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанный города хотя бы по одному разу с последующим возвратом в исходный город, относится к числу транс вычислительных: уже при относительно небольшом числе городов (66 и более) она не может быть решена методом перебора вариантов никаким теоретически мыслимыми компьютерами за время, меньшее нескольких миллиардов лет.

Муравьиный алгоритм – алгоритм для нахождения приближенных решений задачи коммивояжера, а также решения аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьев, ищущих пути от колонии к источнику питания, и представляет собой метаэвристическую оптимизацию.

Целью данной лабораторной работы является исследование и реализация муравьиного алгоритма на примере задачи коммивояжера.

Для достижения поставленной цели необходимо выполнить следующие **задачи**:

1. изучение задачи коммивояжера;
2. исследование подходов к решению задачи коммивояжера;
3. построение схем разрабатываемых алгоритмов;
4. реализация алгоритмов для решения задачи коммивояжера;
5. проведение тестирования муравьиного алгоритма;
6. проведение сравнительного анализа времени выполнения реализованных алгоритмов решения задачи коммивояжера.

1 Аналитическая часть

В данном разделе будет рассмотрено теоретическое описание задачи коммивояжера и методы ее решения.

1.1 Задача коммивояжера

Задача коммивояжера – важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок. Коммивояжёру, чтобы распродать нужные и не очень нужные в хозяйстве товары, следует объехать n пунктов и в конце концов вернуться в исходный пункт. Требуется определить наиболее выгодный маршрут объезда. В качестве меры выгодности маршрута может служить суммарное время в пути, суммарная стоимость дороги, или, в простейшем случае, длина маршрута.

Ниже, на рис.1.1, визуализирована задача коммивояжера.

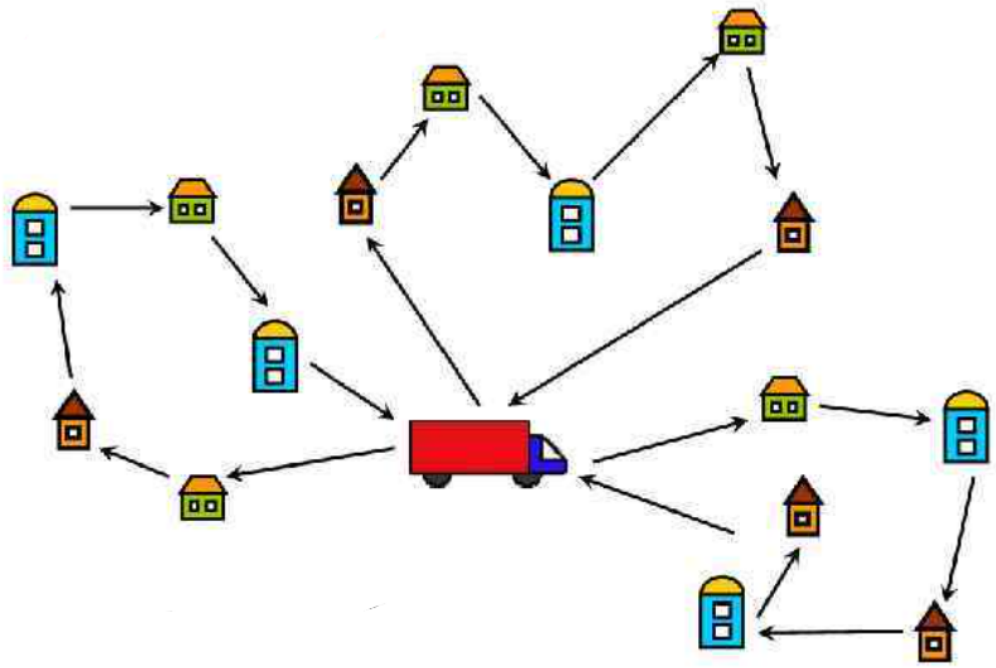

Рисунок 1.1 – Задача коммивояжера

В общем случае задача коммивояжера может быть сформулирована следующим образом: найти самый выгодный маршрут, начинающийся в

исходном городе и проходящий ровно один раз через каждый из указанных городов, с последующим возвратом в исходный город.

Гамильтоновым циклом называется маршрут, включающий ровно по одному разу каждую вершину графа. Таким образом, решение задачи коммивояжера – это нахождение гамильтонова цикла минимального веса в полном взвешенном графе.

1.2 Алгоритм полного перебора

Пусть дано M — число городов, D — матрица смежности, каждый элемент которой — вес пути из одного города в другой. Задача может быть решена перебором всех возможных гамильтоновых циклов в заданном графе и выбором оптимального (минимального по весу). Но при таком подходе количество возможных маршрутов очень быстро возрастает с ростом M (сложность такого алгоритма составляет M! и время выполнения программы, реализующий такой подход, будет расти экспоненциально в зависимости от размеров входной матрицы).

1.3 Муравьиный алгоритм

В основе муравьиного алгоритма лежит моделирование поведение муравьиной колонии, которое связано с распределением феромона на тропе – ребре графа в задаче коммивояжера. При этом вероятность включения ребра в маршрут отдельного муравья пропорциональна количеству феромона на этом ребре, а количество откладываемого феромона пропорционально длине маршрута. Чем короче маршрут, тем больше феромона будет отложено на его ребрах, следовательно, большее количество муравьев будет включать его в синтез собственных маршрутов.

Моделирование такого подхода, использующего только положительную обратную связь, приводит к преждевременной сходимости — большинство муравьев двигается по локально оптимальному маршруту. Избежать, этого можно, моделируя отрицательную обратную связь в виде испарения феромона. При этом если феромон испаряется быстро, то это приводит

к потере памяти колонии и забыванию хороших решений, с другой стороны, большое время испарения может привести к получению устойчивого локально оптимального решения. Теперь, с учетом особенностей задачи коммивояжера, мы можем описать локальные правила поведения муравьев при выборе пути.

У муравья есть 3 чувства: память, зрение, обоняние.

«Память» — муравей запоминает свой маршрут. Поскольку каждый город может быть посещен только один раз, у каждого муравья есть список уже посещенных городов - список запретов. Обозначим через $J_{i,k}$ список городов, которые необходимо посетить муравью k, находящемуся в городе i;

«Зрение» — муравей может оценить длину ребра. Видимость есть эвристическое желание посетить город j, если муравей находится в городе i. Будем считать, что видимость — величина, обратная расстоянию между городами i и j — D_{ij}

$$\eta_{ij} = \frac{1}{D_{ij}} \tag{1.1}$$

«Обоняние» — муравей может улавливать след феромона, подтверждающий желание посетить город j из города i, на основании опыта других муравьев. Количество феромона на ребре (i, j) в момент времени t обозначим через $\tau_{ij}(t)$.

На этом основании можно сформулировать вероятностно-пропорциональное правило 1.2, определяющее вероятность перехода k-ого муравья из города i в город j:

$$P_{i,j} = \begin{cases} \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}, & \text{если ребро есть в списке городов;} \\ 0, & \text{иначе} \end{cases} \tag{1.2}$$

где $\tau_{i,j}$— количество феромонов на ребре ij; $\eta_{i,j}$—расстояние от города i до j; α и β— два регулируемых параметра, задающие веса следа феромона и видимости при выборе маршрута. Важным условием является то, что сумма α и β должна быть постоянной.

При $\alpha = 0$ алгоритм вырождается до алгоритма полного перебора (будет выбран ближайший город). Если $\beta = 0$, тогда работает лишь феромонное усиление, что влечет за собой быстрое вырождение маршрутов к

одному субоптимальному решению.

Пройдя ребро (i, j), муравей откладывает на нем некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t, а $L_k(t)$ — длина этого маршрута. Пусть также Q — параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано формулой 1.3:

$$\Delta\tau_{i,j}^{k}(t) = \begin{cases} Q/L_k(t), & \text{если k-ый мурваей прошел по ребру ij;} \\ 0, & \text{иначе} \end{cases} \tag{1.3}$$

где Q - количество феромона, переносимого муравьем.

Правила внешней среды определяют, в первую очередь, испарение феромона. Пусть $\rho \in [0, 1]$ есть коэффициент испарения, тогда правило испарения имеет вид

$$\tau_{ij}(t+1) = (1-\rho) * \tau_{ij}(t) + \Delta\tau_{ij}(t); \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t); \tag{1.4}$$

где m — количество муравьев в колонии.

В начале алгоритма количество феромона на ребрах принимается равным небольшому положительному числу. Общее количество муравьев остается постоянным и равным количеству городов, каждый муравей начинает маршрут из своего города.

Сложность алгоритма: $O(t_{max} * max(m, n^2))$, где t_{max} — время жизни колонии, m — количество муравьев в колонии, n — размер графа [**?**].

Вывод

В данном разделе была рассмотрена предметная область работы и основополагающие материалы, которые в дальнейшем потребуются при реализации алгоритмов решения задачи коммивояжера.

Существуют две группы методов для решения задачи коммивояжера — точные и эвристические. К точным относится алгоритм полного перебора, к эвристическим — муравьиный алгоритм. Применение муравьиного алгоритма обосновано в тех случаях, когда необходимо быстро найти решение или когда для решения задачи достаточно получения первого приближения. В случае необходимости максимально точного рещения используется алгоритм полного перебора.

2 Конструкторская часть

В данном разделе приводится схема рассматриваемого алгоритма, определяются требования к программе.

2.1 Требования к программе

На основе изученной информации можно описать входные и выходные данные. На вход программе должна подаваться матрица смежности либо координаты x y городов, количество городов. На выходе программа должна выдавать найденный кратчайший путь.

Так как перевод из муравьиных дней в человеческие невозможен, выбран подход случайного леса. Итог программы рассчитывается в муравьиных днях, несоизмеримых с человеческими.

2.2 Разработка алгоритмов

На рисунке 2.1 приведена схема муравьиного алгоритма. На рисунке 2.2 приведена схема алгоритма полного перебора.

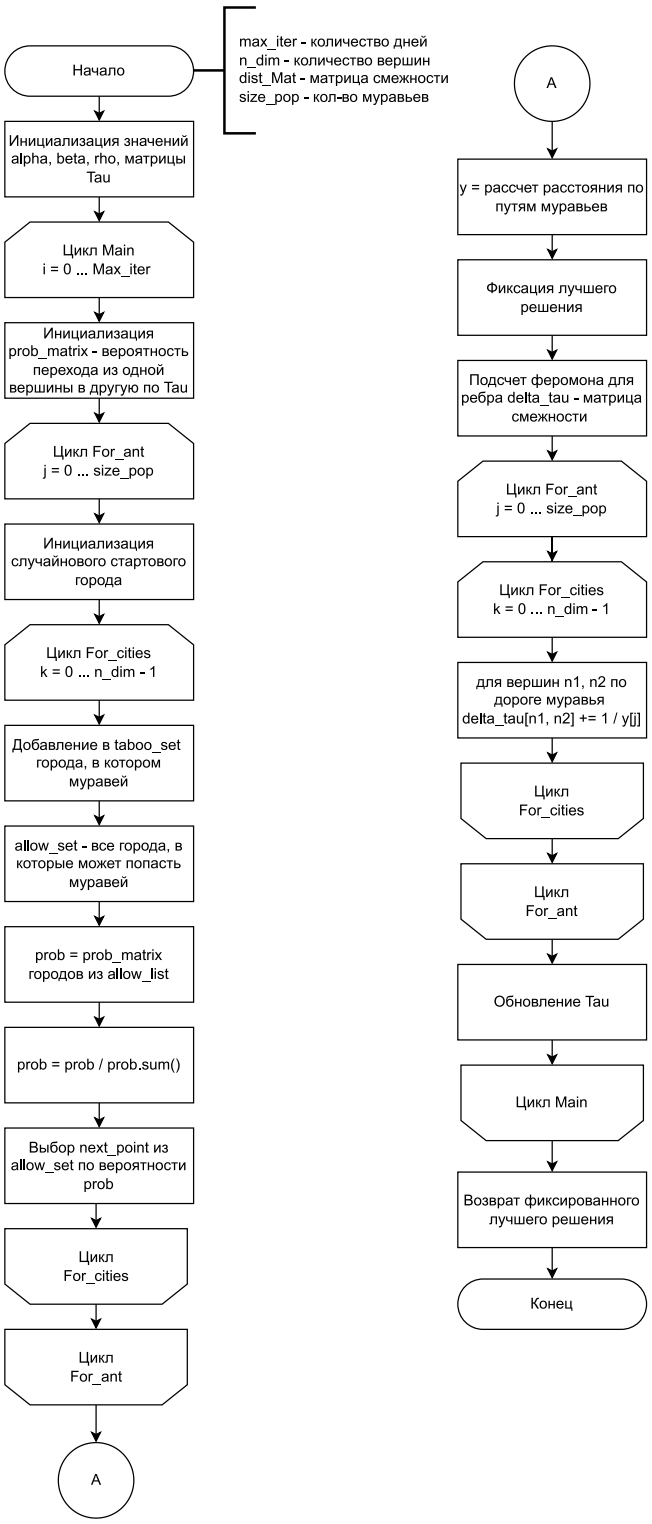

Рисунок 2.1 – Схема муравьиного алгоритма

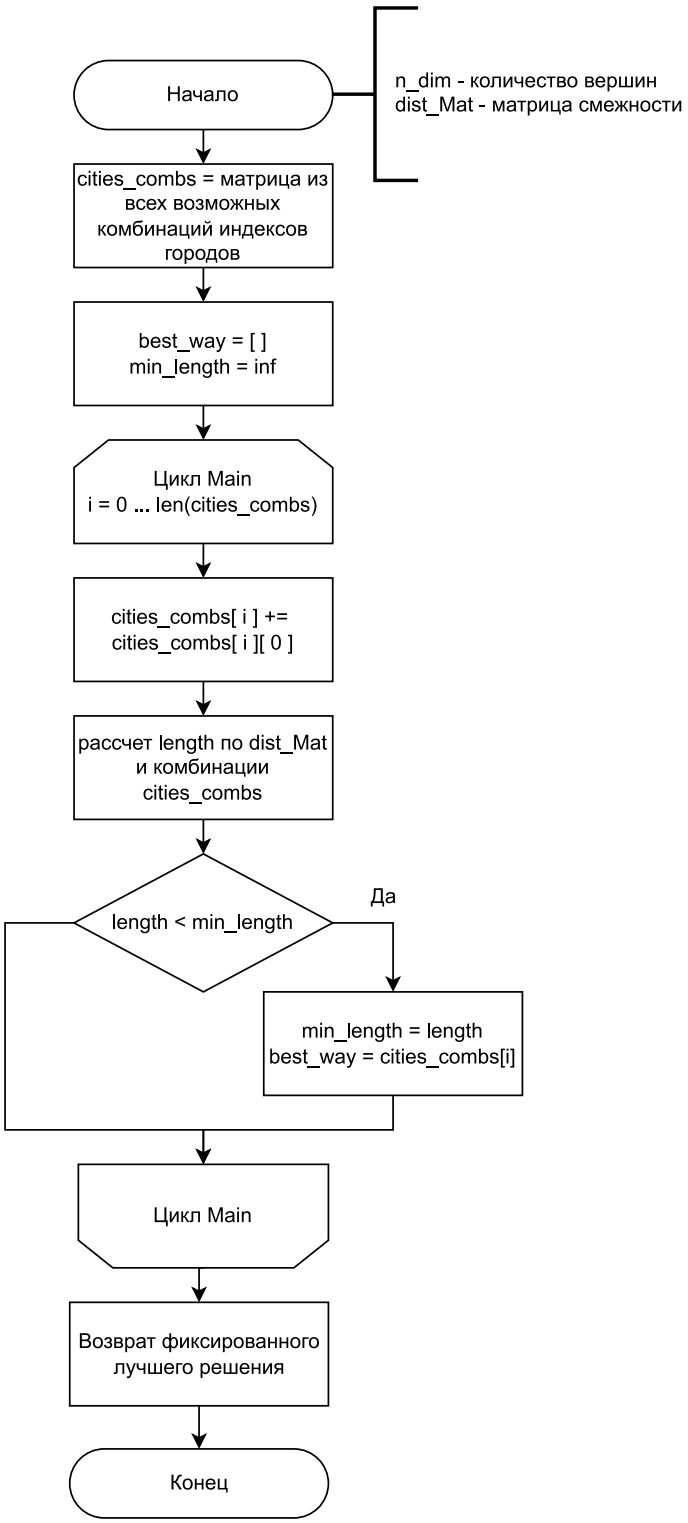

Рисунок 2.2 – Схема алгоритма полного перебора

Вывод

В данном разделе была рассмотрена схема муравьиного алгоритма, алгоритма полного перебора, были приведены требования к программе.

3 Технологическая часть

В данном разделе будут приведены средства реализации и листинги кода.

3.1 Средства реализации

В качестве языка программирования для реализации данной лабораторной работы был выбран язык программирования Python. Выбор обусловлен наличием библиотек для измерения времени, наличием инструментов для работы с массивами.

3.2 Сведения о модулях программы

Программа состоит из следующих модулей: main.py - главный файл программы, в котором располагается вся программа, timer.py - файл программы с замерами времени.

3.3 Реализация алгоритмов

В листингах 3.1, 3.2 приведены реализации алгоритмов.

Листинг 3.1 – Муравьиный алгоритм.

```
def run(self, max_iter=None):
    self.max_iter = max_iter or self.max_iter
    for i in range(self.max_iter):
        prob_matrix = (self.Tau ** self.alpha) *
           (self.prob_matrix_distance) ** self.beta
        for j in range(self.size_pop):
            self.Table[j, 0] = 0
            for k in range(self.n_dim - 1):
                taboo_set = set(self.Table[j, :k + 1])
                allow_list = list(set(range(self.n_dim)) -
                   taboo_set)
                prob = prob_matrix[self.Table[j, k], allow_list]
                prob = prob / prob.sum() # нормализация вероятност
                   и
                next_point = np.random.choice(allow_list, size=1,
                   p=prob)[0]
                self.Table[j, k + 1] = next_point

        y = np.array([self.func(i) for i in self.Table])

        index_best = y.argmin()
        x_best, y_best = self.Table[index_best, :].copy(),
           y[index_best].copy()
        self.generation_best_X.append(x_best)
        self.generation_best_Y.append(y_best)

        delta_tau = np.zeros((self.n_dim, self.n_dim))
        for j in range(self.size_pop):  # для каждого муравья
            for k in range(self.n_dim - 1):  # для каждой вершины
                n1, n2 = self.Table[j, k], self.Table[j, k + 1]
                delta_tau[n1, n2] += 1 / y[j]  # нанесение феромон
                   а
            n1, n2 = self.Table[j, self.n_dim - 1], self.Table[j,
               0]
            delta_tau[n1, n2] += 1 / y[j]  # нанесение феромона

        self.Tau = (1 - self.rho) * self.Tau + delta_tau

    best_generation = np.array(self.generation_best_Y).argmin()
    self.best_x = self.generation_best_X[best_generation]
```

```
34        self.best_y = self.generation_best_Y[best_generation]
35
36        return self.best_x, self.best_y
```

Листинг 3.2 – Алгоритм полного перебора.

```
def full_combinations(matrix, size):
    cities = np.arange(size)
    cities_combs = []

    for combination in it.permutations(cities):
        cities_combs.append(list(combination))

    best_way = []
    min_length = float("inf")

    for i in range(len(cities_combs)):
        cities_combs[i].append(cities_combs[i][0])

        length = calc_length(matrix, size, cities_combs[i])

        if length < min_length:
            min_length = length
            best_way = cities_combs[i]

    return min_length, best_way
```

3.4 Функциональные тесты

В таблице 3.1 приведены функциональные тесты для алгоритмов.

Таблица 3.1 – Тестирование функций

Количество городов	Координаты	Ожидаемый результат
2	$\begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 3.0 \end{bmatrix}$	6
6	$\begin{bmatrix} 0.0 & 0.0 \\ 1.0 & 3.0 \\ 1.0 & 5.0 \\ 0.0 & 7.0 \\ -1.0 & 5.0 \\ -1.0 & 3.0 \end{bmatrix}$	14.7
4	$\begin{bmatrix} 0.0 & 0.0 \\ 0.0 & 3.0 \\ 2.0 & 0.0 \\ 2.0 & 3.0 \end{bmatrix}$	10
10	$\begin{bmatrix} 7.0 & 1.0 \\ 0.0 & 7.0 \\ 0.0 & 6.0 \\ 6.0 & 0.0 \\ 7.0 & 0.0 \\ 8.0 & 2.0 \\ 6.0 & 9.0 \\ 6.0 & 2.0 \\ 7.0 & 9.0 \\ 3.0 & 7.0 \end{bmatrix}$	28.3

Алгоритмы прошли проверку.

Вывод

Были разработан и протестирован муравьиный алгоритм, алгоритм полного перебора.

4 Исследовательская часть

В данном разделе будет приведена постановка замера времени и сравнительный анализ алгоритмов на основе полученных данных.

4.1 Технические характеристики

Технические характеристики устройства, на котором выполнялся замер времени:

Операционная система: 64-разрядная операционная система, процессор x64.

Память: 16 Гб.

Процессор: Intel(R) Core(TM) i7–4700HQ CPU @ 2.40 ГГц.

Во время замера времени ноутбук был нагружен только встроенными приложениями окружения, а также непосредственно системой тестирования.

4.2 Время выполнения алгоритмов

Алгоритмы замерялись при помощи функции $time.perf\_counter()$ из библиотеки *time* языка *Python*. Данная функция возвращает значение в долях секунды счетчика производительности, то есть часов с наибольшим доступным разрешением для измерения короткой длительности.

Замеры времени для каждого выполнения проводились 10 раз. В качестве результата взято среднее время работы алгоритма.

По результатам эксперимента видно, что муравьиный алгоритм начинает значительно выигрывать по времени у алгоритма полного перебора при количестве вершин графа начиная с 9.

Таким образом можно сделать вывод, что использовать муравьиный алгоритм для решения задачи коммивояжера выгодно с точки зрения времени выполнения, в сравнении с алгоритмом полного перебора, в случае если в анализируемом графе вершин больше либо равно 9.

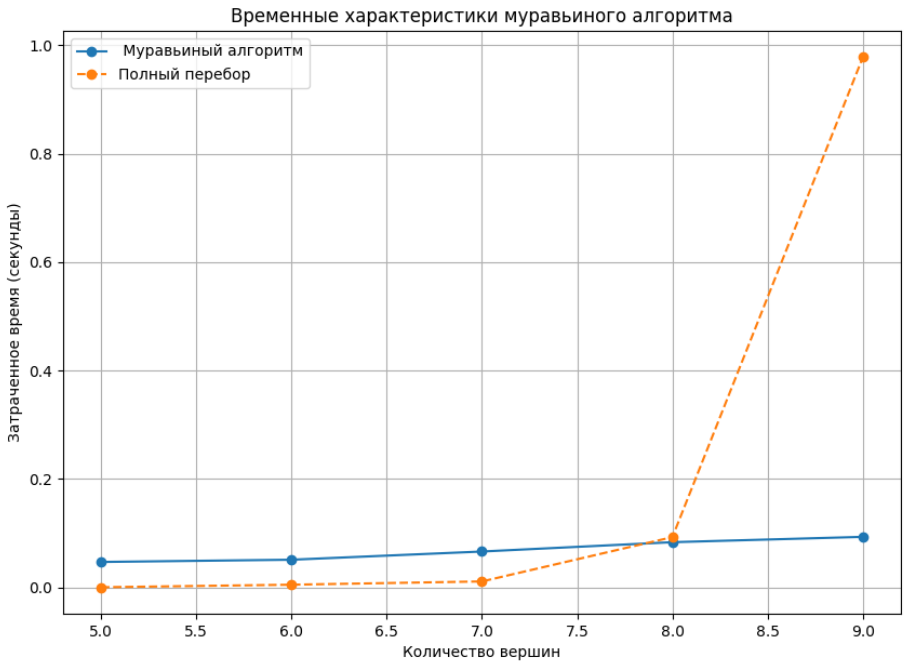

Рисунок 4.1 – Замеры времени для муравьиного алгоритма в сравнении с алгоритмом, использующим полный перебор

4.3 Параметризация муравьиного алгоритма

В муравьином алгоритме вычисления производятся на основе настраиваемых параметров. Проведем параметризацию муравьиного алгоритма для матрицы 10×10:

$$\begin{pmatrix} 0 & 3 & 6 & 5 & 3 & 1 & 1 & 7 & 8 & 3 \\ 2 & 0 & 5 & 8 & 1 & 1 & 3 & 2 & 9 & 7 \\ 5 & 3 & 0 & 4 & 9 & 8 & 3 & 3 & 2 & 6 \\ 2 & 3 & 4 & 0 & 9 & 2 & 3 & 7 & 8 & 8 \\ 6 & 2 & 5 & 9 & 0 & 6 & 7 & 2 & 2 & 3 \\ 6 & 1 & 4 & 2 & 6 & 0 & 6 & 3 & 10 & 9 \\ 7 & 5 & 3 & 3 & 7 & 6 & 0 & 6 & 3 & 2 \\ 1 & 6 & 2 & 7 & 2 & 3 & 6 & 0 & 4 & 1 \\ 1 & 7 & 9 & 8 & 2 & 10 & 3 & 4 & 0 & 9 \\ 3 & 8 & 2 & 8 & 3 & 9 & 2 & 1 & 9 & 0 \end{pmatrix}$$

В каждом эксперименте фиксировались значения alpha, beta, p и количество дней. При этом, соблюдалось условие, что alpha+beta=const. В течение экспериментов значения alpha менялись от 0 до 1 с шагом 0.25, значения p менялись от 0 до 1 с шагом 0.25, количество дней - от 50 до 400 с шагом 50. Количество повторов каждого эксперимента равнялось 100, результатом проведения эксперимента считалась усредненная разница между длиной маршрута, рассчитанного алгоритмом полного перебора и муравьиным алгоритмом с текущими параметрами.

1	ro	Alpha	tmax	Difference
2	0.250000	0.750000	400	0.000000
3	0.000000	0.750000	350	0.020000
4	0.000000	0.750000	400	0.020000
5	0.250000	0.750000	350	0.040000
6	0.250000	0.750000	300	0.060000
7	0.000000	0.750000	300	0.100000
8	0.250000	0.750000	250	0.100000
9	0.250000	0.750000	200	0.100000
10	0.000000	0.500000	400	0.140000
11	0.250000	1.000000	400	0.180000
12	0.000000	1.000000	350	0.180000
13	0.250000	1.000000	350	0.180000
14	0.250000	0.500000	400	0.200000
15	0.000000	0.750000	250	0.220000
16	0.250000	1.000000	300	0.220000

Рисунок 4.2 – Результат эксперимента с изменением параметров

В результате усреднения массового эксперимента было полученр, что наилучшим значением настроенного параметра alpha является значение, равное 0.75, а коэффициента испарения p — 0.25. При количестве дней равном 310 достигается наименьшее различие с минимальным путем.

Вывод

Из проведенного анализа можно сделать следующие выводы: использование муравьиного алгоритма более выгодно с точки зрения временных

затрат, тогда как алгоритм полного перебора эффективен только при малом количестве вершин графа (2 – 8).

На основе проведенной в данном разделе параметризации были выявлены оптимальные параметры для муравьиного алгоритма: a = 0.75, p = 0.75, tmax (количество дней) = 350. Однако стоит учитывать, что чем больше значение tmax, тем больше вероятность того, что будет найден идеальный маршрут, но при этом будет возрастать время выполнения программы. Также был проведен сравнительный анализ времени выполнения двух алгоритмов для решения задачи коммивояжера: муравьиный алгоритм значительно выигрывает по времени выполнения у алгоритма полного перебора при количестве вершин в рассматриваемом графе больше 9-ти. При небольших размерах матриц и в случае, если необходимо получить наименьшее расстояние, рационально использовать алгоритм полного перебора.

Заключение

В ходе выполнения лабораторной работы были решены следующие задачи:

1. исследованы задачи коммивояжера;
2. исследованы подходы к решению задачи коммивояжера;
3. построены схемы разрабатываемых алгоритмов;
4. реализованы алгоритмы для решения задачи коммивояжера;
5. проведены тестирования муравьиного алгоритма;
6. проведен сравнительный анализ времени выполнения реализованных алгоритмов решения задачи коммивояжера.

Из проведенного анализа можно сделать следующие выводы: использование муравьиного алгоритма более выгодно с точки зрения временных затрат, тогда как алгоритм полного перебора эффективен только при малом количестве вершин графа (2 – 8).

На основе проведенной параметризации были выявлены оптимальные параметры для муравьиного алгоритма: a = 0.75, p = 0.75, tmax (количество дней) = 350. Однако стоит учитывать, что чем больше значение tmax, тем больше вероятность того, что будет найден идеальный маршрут, но при этом будет возрастать время выполнения программы. Также был проведен сравнительный анализ времени выполнения двух алгоритмов для решения задачи коммивояжера: муравьиный алгоритм значительно выигрывает по времени выполнения у алгоритма полного перебора при количестве вершин в рассматриваемом графе больше 9-ти. При небольших размерах матриц и в случае, если необходимо получить наименьшее расстояние, рационально использовать алгоритм полного перебора.

Список использованных источников

[1] Волосова А. В. Параллельные методы и алгоритмы. – Москва: МАДИ, 2020.

[2] Штовба С. Д. Муравьиные алгоритмы, Exponenta Pro. Математика в приложениях. 2004. № 4

[3] МакКоннелл Дж. Основы современных алгоритмов. - М.: Техносфера, 2004. - 368 с.