Государственное образовательное учреждение высшего профессионального образования

“Московский государственный технический университет имени Н.Э.Баумана”

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Дисциплина: Анализ алгоритмов

Лабораторная работа № 6

Муравьиный алгоритм

Студент:

Ильясов Идрис Магомет-Салиевич

Группа: ИУ7-53Б

Преподаватели:

Волкова Лилия Леонидовна

Строганов Юрий Владимирович

Москва, 2019 г.

Содержание

Введение

Целью данной работы является изучение использования алгоритма полного перебора и муравьиного алгоритма для решения задачи коммивояжёра. Для достижения поставленной цели необходимо решить следующие задачи:

1. дать постановку решаемой задачи;
2. описать методы ее решения;
3. описать реализацию методов;
4. провести параметризация метода на основании муравьиного алгоритма для класса задач, подобрать такие параметры, при которых метод решает класс задач лучше всего;
5. интерпретировать результаты применения двух методов;
6. дать рекомендации в применимости метода.

1 Аналитическая часть

Задача коммивояжера формулируется как задача поиска минимального по стоимости замкнутого маршрута по всем вершинам без повторений на полном взвешенном графе с N вершинами. Содержательно вершины графа являются городами, которые должен посетить коммивояжер, а веса ребер отражают расстояния (длины) или стоимости проезда.

1.1 Методы решения задачи коммивояжера

1.1.1 Полный перебор

Алгоритм полного перебора (АПП) осуществляет поиск в пространстве решений посредством перебора всех вариантов. Результатом работы алгоритма является точное решение. Недостатком АПП является его временная сложность - пространство поиска растет экспоненциально, поэтому когда N не является значительно малым, используют эвристические и поисковые алгоритмы. К преимуществам данного алгоритма можно отнести возможность его распараллеливания и точное решение задачи (нахождение глобального минимума).

1.2 Эвристические методы и алгоритмы

В связи с отсутствием эффективных точных методов решения задачи коммивояжера, становится необходимым использование эвристических методов. Эвристическими называют методы, которые будучи основанными на некои эвристике (правиле), не всегда следующеи из строгих математических принципов, в подавляющем большинстве случаев дают решение, близкое к точному.

Частным случаем эвристического алгоритма для решения задачи коммивояжера является муравьиныи алгоритм. В его основе которого лежит моделирование поведения колонии муравьев. Колония представляет собой систему с очень простыми правилами автономного поведения особей. Однако, несмотря на примитивность поведения каждого отдельного муравья, поведение всей колонии оказывается достаточно разумным.

Основу поведения муравьиной колонии составляет самоорганизация, которая обеспечивает достижение общих целей колонии на основе низкоуровневого взаимодействия. Колония не имеет централизованного управления, а непрямои обмен информациеи (stigmergy) осуществляется посредством феромона.

Моделирование поведения муравьев связано с распределением феромона на тропе. 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 своих маршрутов. Моделирование такого подхода, использующего только положи- тельную 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 решении, с другой стороны, большое время испарения может привести к получению устойчивого локально оптимального решения.

Опишем локальные правила поведения муравьев при выборе пути:

1. муравьи имеют собственную «память», т.е. они запоминают посещенные города (маршрут);
2. муравьи обладают «зрением» - могут оценить длину ребра. Его привлекательность обратно пропорциональна расстоянию между городами;

3. муравьи обладают «обонянием» - они могут улавливать нахождение феромона и его концентрацию на ребре.

При старте матрица феромонов τ инициализируется равномерно некоторой константой τ_{start}. Если муравей находится в городе i и выбирает, куда ему пойти, то делает он это по вероятностному правилу:

$$P_{k,ij}(\tau) = \begin{cases} \frac{((\tau_{ij}(t))^{\alpha} * (\eta_{ij})^{\beta}}{\sum\limits_{q \in \text{цели}} ((\tau_{iq}(t))^{\alpha} * (\eta_{ij})^{\beta})}, \text{если город} j \in \text{списку целей, т.е.} \notin \text{списку посещенных городов} \\ 0, \text{если} j \in \text{списку посещенных городов} \end{cases}$$

где α, β - весовые коэффициенты, которые задают важность феромона и привлекательности ребра:
α - коэффициент стадности, β - коэффициент жадности
$\alpha + \beta = const$.
При $\alpha = 0$ алгоритм вырождается в жадный (т.е. будет выбран ближайший город).

Чтобы повысить вариативность выбора, будем не выбирать $max(P_{k,ij})$, а подбрасывать монету. Выбираем случайное число от 0 до 1. Складываем элементы из P_k, пока сумма не превзойдет выбранное число. Индекс последнего прибавленного элемента и есть тот город, который надо выбрать.

За один день (шаг по t) муравей формирует маршрут. Ночью учитываем изменение феромона. Феромон испаряется с коэффициентом ρ.

$$\tau_{ij}(t+1) = \tau_{ij}(t) * (1 - \rho) + \Delta\tau_{ij};$$

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{k,ij};$$

$$\Delta\tau_{k,ij}; = \begin{cases} \frac{Q}{L_k}, \text{ребро} \in \text{маршруту } k\text{-го муравья} \\ 0, \text{ребро} \notin \text{маршруту } k\text{-го муравья}, \end{cases}$$

где L_k - длина маршрута k-го муравья,
Q - нормировочная константа порядка длины наилучшего маршрута (например, средняя длина ребра).

1.3 Выводы

В данном разделе были описаны алгоритмы полного перебора и муравьиный алгоритм.

2 Конструкторская часть

В данном разделе будут приведены функциональная модель и схемы алгоритмов.

2.1 Функциональная модель

На рисунке 1 приведена функциональная модель решаемой задачи.

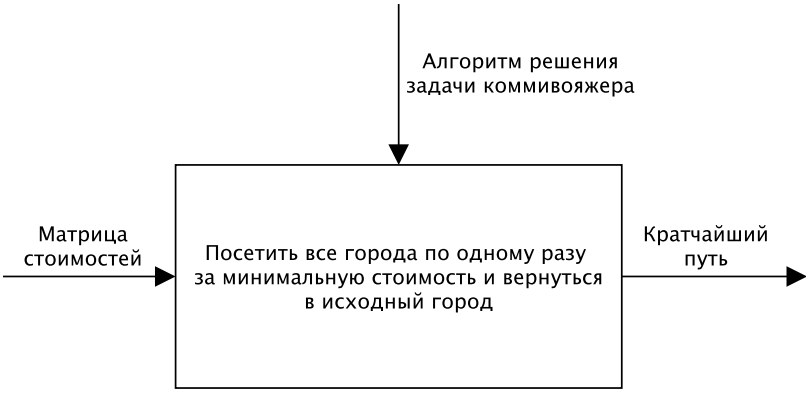

Рис. 1 - Функциональная модель алгоритма нахождения подстроки в строке.

2.2 Разработка алгоритмов

В данном разделе приведено описание схем алгоритмов решения задачи коммивояжера. На рис. 2 представлена схема полного перебора; на рис. 3, 4 - муравьиный алгоритм.

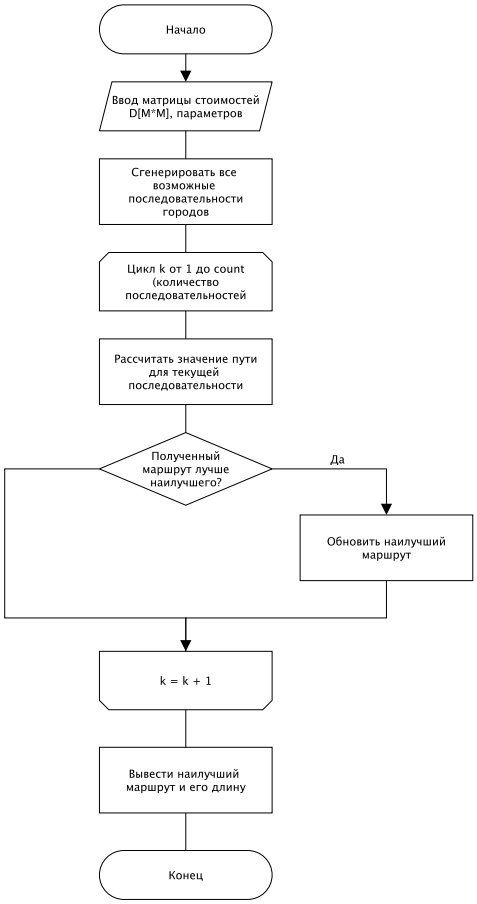

Рис. 2 - Схема алгоритма полного перебора.

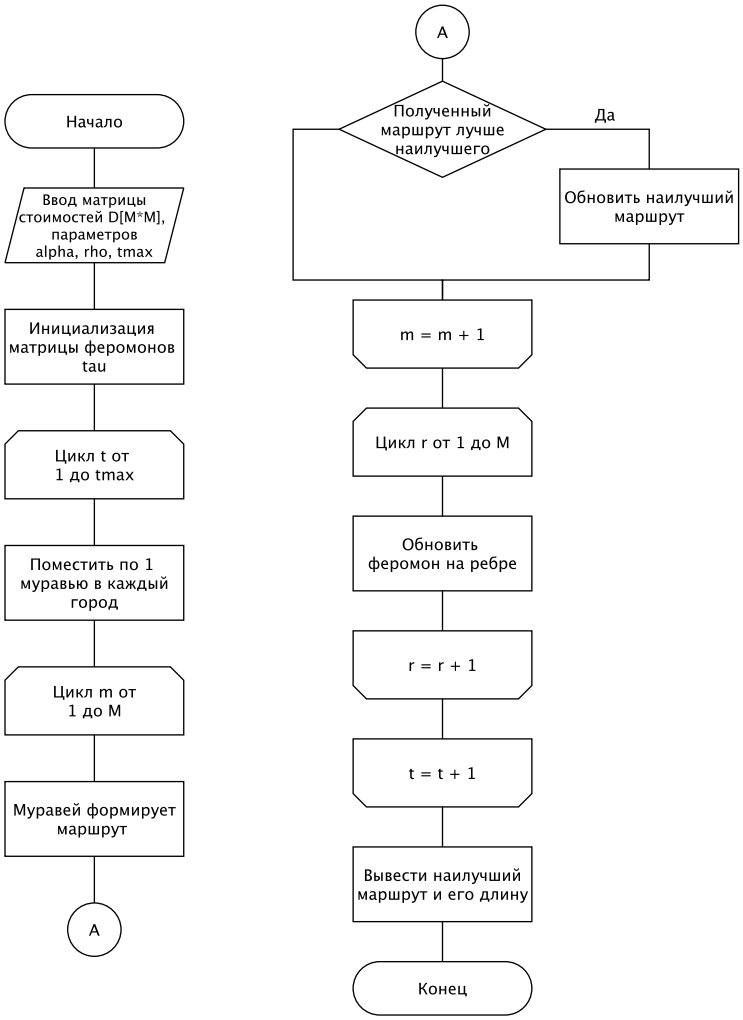

Рис. 3- Схема муравьиного алгоритма.

2.3 Выводы

В данном разделе были рассмотрены схемы алгоритмов.

3 Технологическая часть

В данном разделе представлены требования к разрабатываемому программному обеспечению, средства, использованные в процессе разработки для реализации поставленных задач, а также листинг кода программы.

3.1 Требования к программному обеспечению

Программа должна вычислять кратчайший путь между всеми городами. Результатом работы про- граммы является наилучший маршрут и его длина.

3.2 Средства реализации

Для реализации поставленной задачи был использован язык программирования Python.

3.3 Листинг кода

В листингах 1-2 представлены реализации исследуемых алгоритмов решения задачи коммивояжера.

Листинг 1: Полный перебор

```
1  def generate(t, n, a, array_sequences):
2      if t == (n - 1):
3          b = copy.deepcopy(a)
4          array_sequences.append(b)
5      else:
6          for j in range(t, n, 1):
7              a[j], a[t] = a[t], a[j]
8              generate(t+1, n, a, array_sequences)
9              a[j], a[t] = a[t], a[j]
10
11
12 def full_search(matrix, count_cities):
13     source_seq = list(range(count_cities))
14     array_sequences = []
15
16     generate(0, count_cities, source_seq, array_sequences)
17     min_distance = inf
18
19     for i in range(len(array_sequences)):
20         array_sequences[i].append(array_sequences[i][0])
21         cur_distance = 0
22
23         for j in range(count_cities):
24             first = array_sequences[i][j]
25             second = array_sequences[i][j+1]
```

```
            cur_distance += matrix[first][second]

        if cur_distance < min_distance:
            min_distance = cur_distance
            best_route = array_sequences[i]

    return min_distance, best_route
```

.

Листинг 2: Муравьиный алгоритм

```
def calculate_lk(D, visited_cities):
    lk = 0

    for l in range(1, len(visited_cities)):
        i = visited_cities[l-1]
        j = visited_cities[l]
        lk += D[i][j]

    return lk

def ant_alg(D, count_cities):
    avg = 0
    count = 0
    for i in range(count_cities):
        for j in range(count_cities):
            if 0 < D[i][j] < inf:
                avg += D[i][j]
                count += 1
    avg /= count

    alpha, beta, Q = 2, 3, avg
    tau0 = randint(2, 10)
    tau_limit, rho = 0.001, 0.75
    count_ants = count_cities

    eta = [0]*count_cities
    tau = [0]*count_cities

    for i in range(count_cities):
        eta[i] = [0]*count_cities
```

```
31          tau[i] = [0]*count_cities
32
33      for i in range(count_cities):
34          for j in range(count_cities):
35              if i != j:
36                  eta[i][j] = 1/D[i][j]
37                  tau[i][j] = tau0
38              else:
39                  tau[i][j] = 0
40
41      t = 0
42      tmax = 10
43      L_best = inf
44
45      while t < tmax:
46          l = list(range(count_cities))
47          shuffle(l)
48
49          visited_cities = [0]*count_ants
50          for i in range(count_ants):
51              visited_cities[i] = []
52              visited_cities[i].append(l[i])
53
54
55          for k in range(count_ants):
56              while len(visited_cities[k]) != count_cities:
57                  Pk = [0]*count_cities
58                  for i in range(count_cities):
59                      if i in visited_cities[k]:
60                          Pk[i] = 0
61                      else:
62                          cur_i = visited_cities[k][-1]
63                          Pk[i] = (tau[cur_i][i]**alpha) * (eta[cur_i][i]**beta)
64
65                  summ = sum(Pk)
66                  for i in range (count_cities):
67                      Pk[i] /= summ
68
69                  coin = random()
70                  summ, i = 0, 0
```

```
71                  while (i < len(Pk)) and summ < coin:
72                      summ += Pk[i]
73                      i += 1
74                  visited_cities[k].append(i-1)
75
76              visited_cities[k].append(visited_cities[k][0])
77              Lk = calculate_lk(D, visited_cities[k])
78
79              if Lk < L_best:
80                  L_best = Lk
81                  best_route = visited_cities[k]
82
83          for i in range(count_cities):
84              for j in range(count_cities):
85                  summ = 0
86                  for k in range(count_ants):
87                      Lk = calculate_lk(D, visited_cities[k])
88                      fl = False
89                      for m in range(1, len(visited_cities[k])):
90                          if ((visited_cities[k][m]== i and
91                              visited_cities[k][m-1] == j)
92                              or (visited_cities[k][m]==j and
93                                  visited_cities[k][m-1] == i)):
94                              fl = True
95
96                      if fl:
97                          summ += Q/Lk
98
99                  delta_tau = summ
100                 tau[i][j] = tau[i][j]*(1 - rho) + delta_tau
101                 if tau[i][j] < tau_limit:
102                     tau[i][j] = tau_limit
103
104         t += 1
105     return L_best, best_route
```

3.4 Описание тестирования

Для тестирования программы были заготовлены следующие тесты.

Таблица 1: Заготовки тестов

Входная матрица	Минимальное расстояние
0 5 8 2 5 0 10 1 8 10 0 4 2 1 4 0	18
0 1 5 7 1 1 0 1 4 3 5 1 0 1 6 7 4 1 0 1 1 3 6 1 0	5
0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0	6

3.5 Выводы

На основе схем алгоритмов, представленных в конструкторском разделе, в соответствии с указанными требованиями к реализации с использованием средств языка Python было разработано программное обеспечение, содержащее реализации выбранных алгоритмов.

4 Экспериментальная часть

В данном разделе будут приведены примеры работы разработанного программного обеспечения.

4.1 Примеры работы

На приведенных ниже рисунках продемонстрирована работа выбранных алгоритмов.

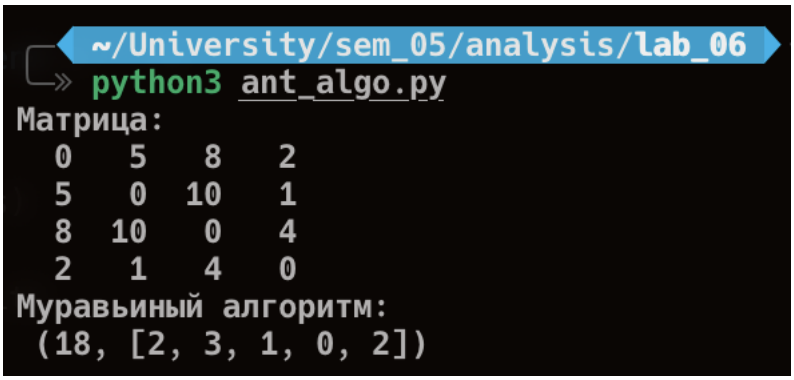

Рис. 4 - Пример работы муравьиного алгоритма

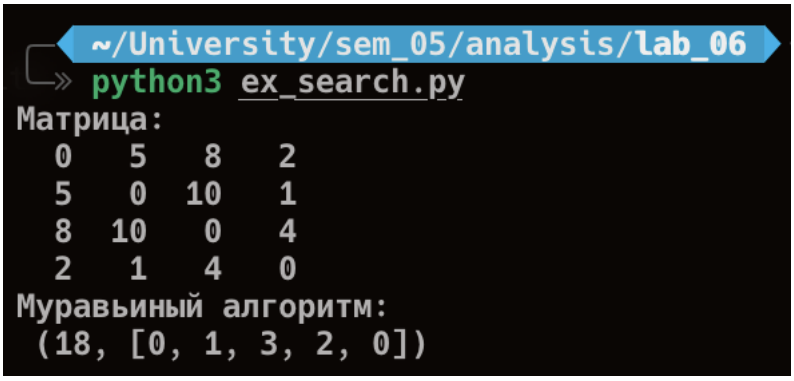

Рис. 5 - Пример работы алгоритма полного перебора

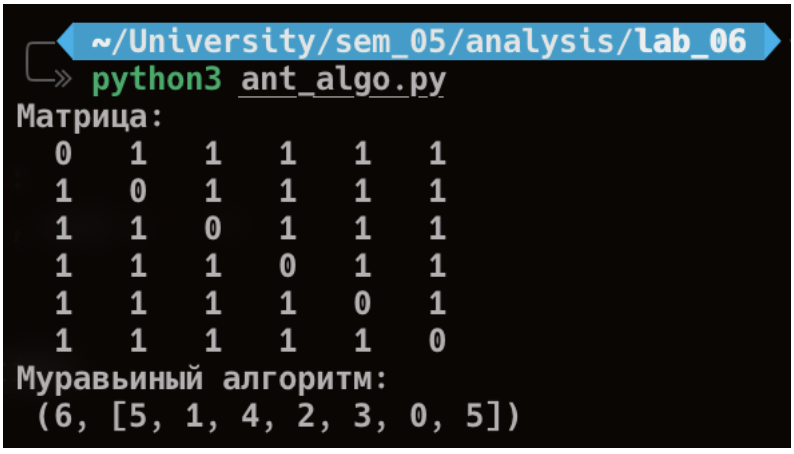

Рис. 6 - Пример работы муравьиного алгоритма

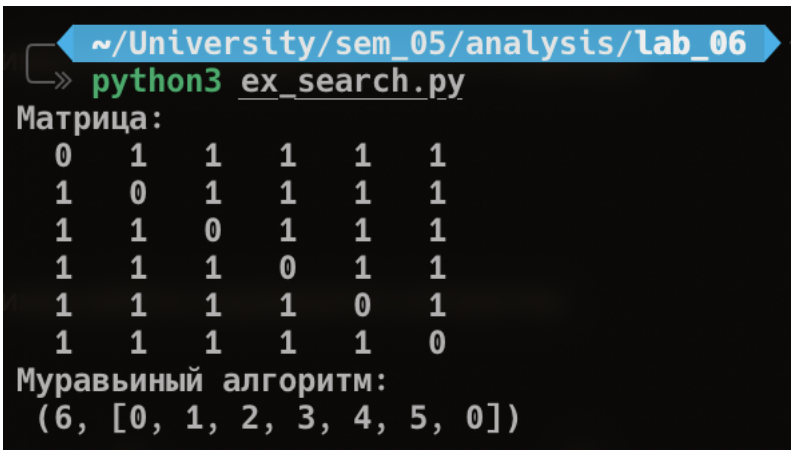

Рис. 7 - Пример работы алгоритма полного перебора

4.2 Результаты тестирования

Для тестирования алгоритмов были использованы данные из Таблицы 1. В Таблице 2 приведены полученные при тестировании результаты. Они совпадают с ожидаемыми, тестирование пройдено успешно.

Таблица 2: Результаты тестирования

Входная матрица	Муравьиный алгоритм	Полный перебор
0 5 8 2 5 0 10 1 8 10 0 4 2 1 4 0	18	18
0 1 5 7 1 1 0 1 4 3 5 1 0 1 6 7 4 1 0 1 1 3 6 1 0	5	5
0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0 1 1 1 1 1 1 0	6	6

4.3 Постановка эксперимента

На рисунке 8 и в Таблице 3 представлены результаты замеров для варьирующихся размеров квадратной матрицы от 3 до 10 с шагом 1 для алгоритма муравья и полного перебора. Симметричные матрицы были сгенерированы случайным образом. Один эксперимент ставился 100 раз, вычислялось среднее время работы.

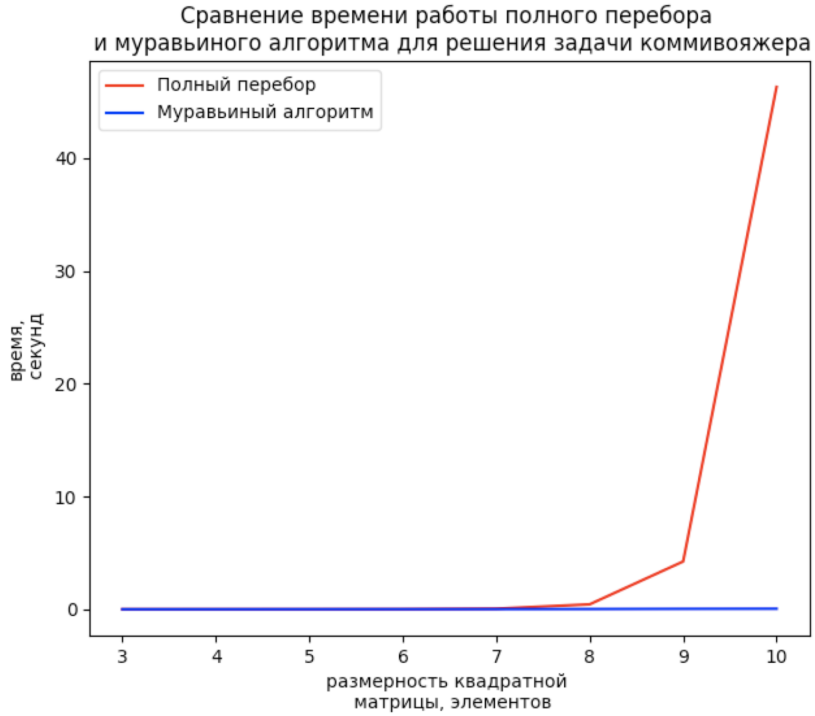

Рис. 8 - Сравнение времени работы полного перебора и муравьиного алгоритма

Таблица 3: Сравнение времени работы полного перебора и муравьиного алгоритма

Входная матрица	Муравьиный алгоритм	Полный перебор
3	7.009512325585938e-05	0.0012590862478235316
4	0.00023698806762695312	0.002931833267211914
5	0.0018529891967773438	0.0062220096588134766
6	0.008517265319824219	0.008891820907592773
7	0.05505704879760742	0.012941837310791016
8	0.4212532043457031	0.020035743713378906
9	4.131577968597412	0.03218197822570801
10	47.601372718811035	0.0479891300201416

Как уже было отмечено ранее, время работы полного перебора экспоненциально зависит от размерности матрицы, в то время как трудоемкость муравьиного алгоритма можно описать полиномиальной функцией.

Для того чтобы провести параметризацию метода на основании муравьиного алгоритма, было проведено сравнение результатов работы алгоритма для различных параметров α, ρ и $t_{max}(\alpha = 0, 2, 4, 6, 8, 10, \rho = 0, 0.25, 0.5, 0.75, 1, t_{max} = 10, 50, 100, 200, 300, 400)$. Эксперименты проводились на двух матрицах: первая содержит 29 строк и представляет собой расстояние между 29 городами Баварии (BAYS-29.tsp), вторая содержит 10 строк с единичным расстоянием между всеми пунктами. В Таблице 4 приведены результаты проведенных экспериментов.

Таблица 4: Сравнение времени работы полного перебора и муравьиного алгоритма

α	ρ	t_{max}	Матрица 1	Матрица 2
0	0	10	2669	10
0	0	50	2584	10
0	0	100	2410	10
0	0	200	2413	10
0	0	300	2291	10
0	0	400	2384	10
0	0.25	10	2585	10
0	0.25	50	2459	10
0	0.25	100	2419	10
0	0.25	200	2255	10
0	0.25	300	2317	10
0	0.25	400	2193	10
0	0.5	10	2432	10
0	0.5	50	2422	10
0	0.5	100	2440	10
0	0.5	200	2237	10
0	0.5	300	2294	10
0	0.5	400	2260	10
0	0.75	10	2553	10
0	0.75	50	2498	10
0	0.75	100	2402	10
0	0.75	200	2327	10
0	0.75	300	2187	10
0	0.75	400	2307	10
0	1	10	2554	10
0	1	50	2452	10
0	1	100	2320	10
0	1	200	2410	10
0	1	300	2286	10
0	1	400	2240	10
2	0	10	2262	10
2	0	50	2080	10
2	0	100	2081	10
2	0	200	2081	10
2	0	300	2070	10
2	0	400	2057	10

α	ρ	t_{max}	Матрица 1	Матрица 2
2	0.25	10	2248	10
2	0.25	50	2136	10
2	0.25	100	2142	10
2	0.25	200	2088	10
2	0.25	300	2087	10
2	0.25	400	2116	10
2	0.5	10	2085	10
2	0.5	50	2054	10
2	0.5	100	2080	10
2	0.5	200	2106	10
2	0.5	300	2052	10
2	0.5	400	2162	10
2	0.75	10	2105	10
2	0.75	50	2204	10
2	0.75	100	2159	10
2	0.75	200	2119	10
2	0.75	300	2118	10
2	0.75	400	2118	10
2	1	10	2112	10
2	1	50	2123	10
2	1	100	2101	10
2	1	200	2140	10
2	1	300	2145	10
2	1	400	2050	10
4	0	10	2128	10
4	0	50	2145	10
4	0	100	2086	10
4	0	200	2126	10
4	0	300	2098	10
4	0	400	2114	10
4	0.25	10	2140	10
4	0.25	50	2167	10
4	0.25	100	2100	10
4	0.25	200	2132	10
4	0.25	300	2143	10
4	0.25	400	2253	10

α	ρ	t_{max}	Матрица 1	Матрица 2
4	0.5	10	2089	10
4	0.5	50	2156	10
4	0.5	100	2163	10
4	0.5	200	2175	10
4	0.5	300	2143	10
4	0.5	400	2135	10
4	0.75	10	2074	10
4	0.75	50	2176	10
4	0.75	100	2085	10
4	0.75	200	2176	10
4	0.75	300	2135	10
4	0.75	400	2138	10
4	1	10	2121	10
4	1	50	2152	10
4	1	100	2123	10
4	1	200	2283	10
4	1	300	2082	10
4	1	400	2143	10
6	0	10	2223	10
6	0	50	2081	10
6	0	100	2113	10
6	0	200	2098	10
6	0	300	2085	10
6	0	400	2142	10
6	0.25	10	2208	10
6	0.25	50	2156	10
6	0.25	100	2114	10
6	0.25	200	2126	10
6	0.25	300	2138	10
6	0.25	400	2140	10
6	0.5	10	2253	10
6	0.5	50	2098	10
6	0.5	100	2123	10
6	0.5	200	2173	10
6	0.5	300	2143	10
6	0.5	400	2115	10

α	ρ	t_{max}	Матрица 1	Матрица 2
6	0.75	10	2188	10
6	0.75	50	2163	10
6	0.75	100	2138	10
6	0.75	200	2219	10
6	0.75	300	2153	10
6	0.75	400	2204	10
6	1	10	2140	10
6	1	50	2202	10
6	1	100	2376	10
6	1	200	2087	10
6	1	300	2275	10
6	1	400	2147	10
8	0	10	2088	10
8	0	50	2134	10
8	0	100	2253	10
8	0	200	2124	10
8	0	300	2100	10
8	0	400	2080	10
8	0.25	10	2120	10
8	0.25	50	2116	10
8	0.25	100	2062	10
8	0.25	200	2163	10
8	0.25	300	2111	10
8	0.25	400	2106	10
8	0.5	10	2100	10
8	0.5	50	2151	10
8	0.5	100	2118	10
8	0.5	200	2106	10
8	0.5	300	2129	10
8	0.5	400	2138	10
8	0.75	10	2188	10
8	0.75	50	2232	10
8	0.75	100	2141	10
8	0.75	200	2292	10
8	0.75	300	2211	10
8	0.75	400	2087	10
8	1	10	2126	10
8	1	50	2297	10
8	1	100	2124	10

α	ρ	t_{max}	Матрица 1	Матрица 2
8	1	200	2172	10
8	1	300	2146	10
8	1	400	2243	10
10	0	10	2121	10
10	0	50	2156	10
10	0	100	2116	10
10	0	200	2114	10
10	0	300	2107	10
10	0	400	2081	10
10	0.25	10	2185	10
10	0.25	50	2139	10
10	0.25	100	2107	10
10	0.25	200	2105	10
10	0.25	300	2113	10
10	0.25	400	2156	10
10	0.5	10	2123	10
10	0.5	50	2087	10
10	0.5	100	2140	10
10	0.5	200	2095	10
10	0.5	300	2109	10
10	0.5	400	2140	10
10	0.75	10	2253	10
10	0.75	50	2188	10
10	0.75	100	2265	10
10	0.75	200	2243	10
10	0.75	300	2294	10
10	0.75	400	2220	10
10	1	10	2327	10
10	1	50	2261	10
10	1	100	2113	10
10	1	200	2220	10
10	1	300	2210	10
10	1	400	2269	10

Изменение параметров никак не влияет на найденный кратчайший маршрут для второй матрицы. Однако для первой можно выделить следующие диапазоны параметров, при которых были получены наилучшие решения: $\alpha = 2, 4, \rho = 0$, с увеличением значения параметра t_{max} в большей части экспериментов уменьшалось расстояние найденного лучшего маршрута.

5 Заключение

В ходе данной работы был изучен муравьиный алгоритм для решения задачи коммивояжера, про- веден сравнительный анализ указанных алгоритмов на варьирующейся размерности массива по затрачиваемым ресурсам времени и получено экспериментальное подтверждение различии во временной эффективности при помощи разработанного программного обеспечения на материале замеров процессорного времени выполнения реализации на варьирующейся размерности матриц. Проведена параметризация метода на основании муравьиного алгоритма для класса задач, подобраны такие параметры, при которых метод решает класс задач лучше всего; было замечено и доказано экспериментально, что качество полу- чаемых решений во многом зависит от настроечных параметров и от параметров правил откладывания и испарения феромона. На основе проведенного исследования можно дать следующие рекомендации в применимости метода.

Поскольку в основе муравьиного алгоритма лежит моделирование передвижения муравьёв по некоторым путям, то его можно использовать для поиска рациональных решений в задачах, допускающих графовую интерпретацию. Ряд экспериментов показывает, что эффективность муравьиных алгоритмов растёт с ростом размерности решаемых задач оптимизации. Хорошие результаты получаются для нестационарных систем с изменяемыми во времени параметрами, например, для расчётов телекоммуникационных и компьютерных сетей, а также для разработки оптимальной структуры съёмочных сетей GPS в рамках создания высокоточных геодезических и съёмочных технологий. В настоящее время муравьиные алгоритмы находят применение для таких сложных оптимизационных задач, как задача коммивояжёра, транспортная задача, задача календарного планирования, задача раскраски графа, квадратичная задача о назначениях, задача оптимизации сетевых трафиков и ряд других. Высокую эффективность муравьиные алгоритмы демонстрируют при оптимизации распределенных нестационарных систем. Ярким примером может служить нахождение муравьиными алгоритмами оптимальных трафиков в телекоммуникационных сетях.

Список литературы

[1] Штовба С.Д. Муравьиные алгоритмы // Exponenta Pro Математика в приложениях. 2003. No4. С.70–75

[2] Ульянов М. В. Ресурсо-эффективные компьютерные алгоритмы. Разработка и анализ – Москва: Наука Физматлит, 2007. – 376 стр

[3] M.Dorigo,“Ottimizzazione,apprendimentoautomatico,edalgoritmibasatisumetaforanaturale(Optimization, Learning, and Natural Algorithms)”, диссертация на соискание ученой степени “Doctorate in Systems and Information Electronic Engineering”, Politecnico di Milano, 1992 г.