Федеральное государственное бюджетное образовательное учреждение
высшего образования «Московский государственный технический университет
имени Н. Э. Баумана» (национальный исследовательский университет)

Дисциплина: «Анализ алгоритмов»
Отчет по лабораторной работе №6

Тема работы:
«Задача коммивояжера.
Муравьиный алгоритм»

Студент: Левушкин И. К.
Группа: ИУ7-52Б
Преподаватели: Волкова Л. Л.,
Строганов Ю. В.

Москва, 2019 г.

Содержание

Введение **3**

1 Аналитический раздел **4**

1.1 Описание алгоритмов 4

1.1.1 Муравьиный алгоритм 4

Выводы 5

2 Конструкторский раздел **5**

2.1 Разработка алгоритмов 5

Выводы 6

3 Технологический раздел **6**

3.1 Требования к программному обеспечению 6

3.2 Средства реализации 6

3.3 Листинг программы 6

3.4 Тестовые данные 10

Выводы 11

4 Исследовательский раздел **11**

4.1 Примеры работы 11

4.2 Постановка эксперимента 11

4.3 Сравнительный анализ на основе эксперимента 12

4.3.1 Сравнение времени работы 12

4.3.2 Параметризация в муравьином алгоритме 12

Выводы 17

Заключение **17**

Список литературы **17**

Введение

Задача коммивояжера занимает особое место в комбинаторной оптимизации и исследовании операций. Она формулируется как задача поиска минимального по стоимости замкнутого маршрута по всем вершинам без повторений на полном взвешенном графе. Содержательно вершины графа являются городами, которые должен посетить коммивояжер, а веса ребер отражают расстояния (длины) или стоимости проезда. Эта задача является NP-трудной, и точный переборный алгоритм её решения имеет факториальную сложность. [4]

Решение данной задачи важно в первую очередь для крупных транспортных компаний, которые стремятся оптимизировать перевозки и минимизировать расходы. [5]

Особый интерес представляет муравьиный алгоритм, способный эффективно находить приближенное решение задачи коммивояжера.

Цель лабораторной работы: изучение подходов к решению задачи коммивояжера на материале алгоритма полного перебора и муравьиного алгоритма.

Задачи работы:

1) изучить муравьиный алгоритм;
2) применить метод динамического программирования для реализации муравьиного алгоритма и полного перебора;
3) экспериментально подтвердить различия во временно́й эффективности алгоритмов при помощи разработанного программного обеспечения на материале замеров процессорного времени;
4) провести параметризацию муравьиного алгоритма;
5) описать и обосновать полученные результаты в отчете о лабораторной работе, выполненного как расчётно-пояснительная записка.

1 Аналитический раздел

В данном разделе будет рассмотрен муравьиный алгоритм.

1.1 Описание алгоритмов

1.1.1 Муравьиный алгоритм

Идея муравьиного алгоритма – моделирование поведения муравьёв, связанного с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь. При своём движении муравей метит путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьёв находить новый путь, если старый оказывается недоступным. [1]

У муравья есть 3 чувства:

1. Обоняние — муравей чует феромон и его концентрацию на ребре.
2. Зрение — муравей оценивает длину ребра.
3. Память — муравей запоминает посещенные города.

При старте матрица феромонов τ инициализируется равномерно некоторой константой.

Если муравей k находится в городе i и выбирает куда пойти, то делает это по вероятностному правилу:

$$P_{ij}(t) = \begin{cases} \dfrac{\tau_{ij}(t)^{\alpha}\eta_{ij}^{\beta}}{\sum_{q=1}^{m}\tau_{iq}(t)^{\alpha}\eta_{iq}^{\beta}}, & j, \\ 0, & , \end{cases} \tag{1}$$

где

α, β – весовые коэффициенты, которые задают важность феромона и привлекательность ребра, $\alpha + \beta = const$,
$\eta_{iq} = \frac{1}{d_{ij}}$ – привлекательность ребра (города),
d_{ij} – длина ребра.

Кроме того, надо учитывать изменение феромона по формуле 2:

$$\tau(t+1) = \tau_{ij}(t) \cdot (1-\rho) + \sum_{k=1}^{n} \Delta\tau_{k,ij}(t), \tag{2}$$

$$\Delta\tau_{k,ij}(t) = \begin{cases} \frac{Q}{L_k}, & ij, \\ 0, & , \end{cases} \quad (3)$$

где

L_k – длина маршрута k-ого муравья,

ρ – коэффициент испарения феромона,

Q – нормировочная константа порядка длины наилучшего маршрута.

Выводы

Рассмотрен муравьиный алгоритм, выделены ключевые моменты его работы.

2 Конструкторский раздел

В разделе приводится псевдокод муравьиного алгоритма.

2.1 Разработка алгоритмов

Псевдокод муравьиного алгоритма для решения задачи коммивояжера:

1. Ввод матрицы расстояний D
2. Инициализация рёбер – присвоение видимости η_{ij} и начальной концентрации феромона
3. Размещение муравьёв в случайно выбранные города без совпадений
4. Выбор начального кратчайшего маршрута и определение L^*
5. Цикл по времени жизни колонии t=1,tmax
6. Цикл по всем муравьям k=1,m
7. Построить маршрут $T_k(t)$ по правилу (1)
8. Рассчитать длину $L_k(t)$
9. Конец цикла по муравьям
10. Проверка всех $L_k(t)$ на лучшее решение по сравнению с $L*$
11. В случае если решение $L_k(t)$ лучше, обновить $L*$ и $T*$
12. Цикл по всем рёбрам графа
13. Обновить следы феромона на ребре согласно (2), (3)
14. Конец цикла по рёбрам
15. Конец цикла по времени
16. Вывести кратчайший маршрут $T*$ и его длину $L*$

Выводы

В разделе представлен пошаговый разбор муравьиного алгоритма для решения задачи коммивояжера.

3 Технологический раздел

Здесь описываются требования к программному обеспечению и средства реализации, приводятся листинги программы и тестовые данные.

3.1 Требования к программному обеспечению

Входные данные:

- количество городов (целое число, большее 2).

Выходные данные: длины кратчайшего пути, маршруты, найденные полным перебором или муравьиным алгоритмом.

3.2 Средства реализации

Программа написана на языке Python [8], который предоставляет программисту мощные инструменты для реализации различных алгоритмов и является достаточно надежным, эффективным и удобным для реализации сложных алгоритмов. Для написания использовался редактор исходного кода *PyCharm* [9].

Замер времени выполнения программы производится с помощью функции *process_time()* из библиотеки *time*, функционал которой позволяет подсчитывать процессорное время в тиках, а затем конвертировать полученный результат в секунды.

3.3 Листинг программы

Реализованная программа представлена в листингах 1-2.

Листинг 1: Реализация полного перебора для решения задачи коммивояжера

```
1 def calc_min_way(min_way, d_matrix, city_visit, i, j, way):
2   if (i != -1):
3     way += d_matrix[i][j]
4     city_visit[j] = True
5
6   count = 0
7   for k in range(len(d_matrix)):
8     if (not city_visit[k]):
```

```
9         min_way = min(min_way, calc_min_way(min_way, d_matrix, city_visit[:],
              j, k, way))
10      else:
11        count += 1
12
13    if (count == len(d_matrix)):
14      return way
15
16    return min_way
17
18  def get_min_way(min_way, d_matrix, city_visit):
19    for i in range(len(d_matrix)):
20      visit = city_visit[:]
21      visit[i] = True
22      min_way = min(min_way, calc_min_way(min_way, d_matrix, visit, -1, i, 0))
23    return min_way
24
25
26  def full_search(d_matrix):
27    min_way = 1000000
28    min_way = get_min_way(min_way, d_matrix, [False for i in range(len(
          d_matrix))])
29    return min_way
```

Листинг 2: Реализация муравьиного алгоритма для решения задачи коммивояжера

```
1  from random import *
2
3  class Ant(object):
4    def __init__(self, i, towns_count):
5      self.city_visit = [False for i in range(towns_count)]
6      self.city_visit[i] = True
7      self.edge_visit = []
8      self.hole_way = 0
9      self.position = i
10
11   def get_edge_length(self, d_matrix, j):
12     return d_matrix[self.position][j]
13
14   def was(self, i):
15     return self.city_visit[i]
16
17   def move_ant(self, i, d_matrix):
18     self.hole_way += d_matrix[self.position][i]
19     self.edge_visit.append([self.position, i])
20     self.position = i
21     self.city_visit[i] = True
22
23
24
25 def calc_q(d_matrix):
26   sum = 0
27   count = 0
28   for i in d_matrix:
29     for j in i:
30       sum += j
```

```
31        count += 1
32
33    return sum / count * len(d_matrix)
34
35  def create_pheromon_matrix(d_matrix, q):
36    L = []
37    for i in range(len(d_matrix)):
38      L1 = []
39      for j in range(len(d_matrix[i])):
40        if (i == j):
41          L1.append(0)
42        else:
43          L1.append(q / d_matrix[i][j])
44      L.append(L1)
45    return L
46
47
48  def random_way_choice(cities_probability):
49    seed()
50    rand_number = random()
51    k = -1
52    current_prob = 0
53    for i in range(len(cities_probability)):
54      current_prob += cities_probability[i]
55      if (rand_number < current_prob):
56        k = i
57        break
58    if (k == -1):
59    for i in range(len(cities_probability)):
60      if (cities_probability[i] > 0.000001):
61        k = i
62    return k
63
64  def move_ant(town, ant, d_matrix):
65    ant.move_ant(town, d_matrix)
66
67
68  def calc_etta(d_matrix, i, j, ant):
69    if (ant.was(j)):
70      return 0
71    return 1 / d_matrix[i][j]
72
73  def calc_town_probability(ant, pheromon_matrix, d_matrix, alpha, j):
74    if (ant.was(j)):
75      return 0
76
77    betta = 1 - alpha
78
79    znam = 0
80    for i in range(len(d_matrix)):
81      znam += (pheromon_matrix[ant.position][i] ** alpha) * (calc_etta(
          d_matrix, ant.position, i, ant) ** betta)
82
83    p = ((pheromon_matrix[ant.position][j] ** alpha) * (calc_etta(d_matrix,
       ant.position, j, ant) ** betta) /
```

```
84    znam)
85
86    return p
87
88
89
90  def create_hole_way(ant, pheromon_matrix, d_matrix, alpha):
91  for i in range(len(d_matrix) - 1):
92    cities_probability = []
93    for j in range(len(d_matrix)):
94      cities_probability.append(calc_town_probability(ant, pheromon_matrix,
          d_matrix, alpha, j))
95    town = random_way_choice(cities_probability)
96    move_ant(town, ant, d_matrix)
97
98
99
100 def edge_belong_to_ant_way(i, j, ant):
101   for k in range(len(ant.edge_visit)):
102     if (ant.edge_visit[k][0] == i and ant.edge_visit[k][1] == j):
103       return True
104   return False
105
106 def refresh_pheromon_matrix(pheromon_matrix, p, q, ants, d_matrix):
107   one_minus_p = 1 - p
108   for i in range(len(pheromon_matrix)):
109     for j in range(len(pheromon_matrix)):
110       sum_pheromons = 0
111       for k in range(len(ants)):
112         if (edge_belong_to_ant_way(i, j, ants[k])):
113           sum_pheromons += q / ants[k].hole_way
114       pher = pheromon_matrix[i][j] * one_minus_p + sum_pheromons
115       if (i == j):
116         pher = 0
117       elif (pher < q / d_matrix[i][j]):
118         pher = q / d_matrix[i][j]
119       pheromon_matrix[i][j] = pher
120
121
122 def ants_algorythm(d_matrix):
123
124   alpha = float(input("alpha: "))
125   p = float(input("p: "))
126   t_max = int(input("t_max: "))
127
128   q = calc_q(d_matrix)
129   best_hole_way = 1000000
130   best_way = []
131
132   pheromon_matrix = create_pheromon_matrix(d_matrix, q)
133
134   for t in range(t_max):
135     ants = []
136     for i in range(len(d_matrix)):
137       ant = Ant(i, len(d_matrix))
```

```
138         ants.append(ant)
139
140       for i in range(len(ants)):
141         create_hole_way(ants[i], pheromon_matrix, d_matrix, alpha)
142         if (ants[i].hole_way < best_hole_way):
143           best_hole_way = ants[i].hole_way
144           best_way = ants[i].edge_visit
145
146       refresh_pheromon_matrix(pheromon_matrix, p, q, ants, d_matrix)
147     print(best_way)
148     return best_hole_way
```

3.4 Тестовые данные

Тестирование полного перебора производится на следующей матрице размерностью 10 (i – номер строки, j – номер столбца):

Таблица 1: Матрица

ij	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
1	0	3	10	2	5	2	8	8	8	7
2	3	0	4	2	8	1	7	7	10	1
3	10	4	0	8	5	4	10	2	6	1
4	2	2	8	0	1	1	9	1	7	4
5	5	8	5	1	0	7	1	9	4	8
6	2	1	4	1	7	0	8	9	4	6
7	8	7	10	9	1	8	0	4	4	8
8	8	7	2	1	9	9	4	0	5	1
9	8	10	6	7	4	4	4	5	0	7
10	7	1	1	4	8	6	8	1	7	0

Полный перебор:

1. Путь: $(0 \to 5 \to 1 \to 9 \to 2 \to 7 \to 3 \to 4 \to 6 \to 8)$
2. Минимальный путь: 14

Муравьиный алгоритм:

1. $\alpha = 0.3$; $\rho = 0.3$; t_max = 100
   (a) Путь: $(7 \to 2 \to 9 \to 1 \to 5 \to 0 \to 3 \to 4 \to 6 \to 8)$
   (b) Минимальный путь: 15
2. $\alpha = 0.4$; $\rho = 0.5$; t_max = 100
   (a) Путь: $(0 \to 3 \to 5 \to 1 \to 9 \to 2 \to 7 \to 6 \to 4 \to 8)$
   (b) Минимальный путь: 17

3. $\alpha = 0.2$; $\rho = 0.6$; t_max = 90
   (a) Путь: $(8 \to 5 \to 0 \to 1 \to 9 \to 2 \to 7 \to 3 \to 4 \to 6)$
   (b) Минимальный путь: 16
4. $\alpha = 0.6$; $\rho = 0.45$; t_max = 120
   (a) Путь: $(0 \to 5 \to 1 \to 2 \to 9 \to 7 \to 3 \to 4 \to 6 \to 8)$
   (b) Минимальный путь: 16
5. $\alpha = 0.4$; $\rho = 0.4$; t_max = 200
   (a) Путь: $(8 \to 6 \to 4 \to 3 \to 7 \to 9 \to 2 \to 1 \to 5 \to 0)$
   (b) Минимальный путь: 16

Выводы

В данном разделе были рассмотрены требования к программному обеспечению, обоснован выбор средств реализации, приведены листинги программы и тестовые данные. Все заявленные тесты успешно пройдены муравьиным алгоритмом и полным перебором.

4 Исследовательский раздел

В разделе представлены примеры выполнения программы, результаты сравнения алгоритмов решения задачи коммивояжера, а также исследование эффективности поиска муравьиным алгоритмом при различных параметрах.

4.1 Примеры работы

4.2 Постановка эксперимента

1. Сравнить время работы полного перебора и муравьиного алгоритма на 5 матрицах размерностью 10, для получения более точного результата произвести вычисления над каждой матрицей 10 раз. Параметры: $\alpha = 0.5, \rho = 0.5, tmax = 300$.
2. Выяснить при каких параметрах $\alpha \in [0; 1]$, $\rho \in (0; 1]$, $tmax \in [10; 200]$, где $\alpha, \rho \in \mathbb{R}$, $t \in \mathbb{N}$ муравьиный алгоритм будет работать лучше. При этом значения α и ρ меняются с шагом 0.1, t_max — с шагом 10.

4.3 Сравнительный анализ на основе эксперимента

4.3.1 Сравнение времени работы

Замеры произведены на 4-ядерном процессоре *Intel Core i7* с тактовой частотой 2,4 ГГц, оперативная память — 8 ГБ.

Экспериментально получена таблица сравнения времени (табл. 2, время в секундах (с)):

Таблица 2: Сравнение времени выполнения алгоритмов решения задачи коммивояжера

Матрица	Перебор, с	Муравьи, с
1	24.16445	1.737721
2	23,37577	1,538537
3	22,36178	1,673532
4	25,36118	1,80861
5	24,35346	1,742806

Видно, что как и ожидалось, муравьиный алгоритм быстрее решает поставленную задачу — на экспериментальных данных в среднем затрачено в 18 раз меньше времени, чем при полном переборе.

4.3.2 Параметризация в муравьином алгоритме

Для проведения параметризации в процессе случайной генерации получены 5 матриц размерностью 10, элементы матрицы находятся в диапазоне от 0 до 100 и кратны 10.

1. Путь: $(1 \to 8 \to 3 \to 10 \to 2 \to 5 \to 6 \to 4 \to 7 \to 9)$

   Минимальная длина: 130

$$\begin{bmatrix} 0 & 50 & 70 & 90 & 10 & 30 & 70 & 10 & 60 & 90 \\ 50 & 0 & 60 & 60 & 10 & 80 & 40 & 10 & 80 & 80 \\ 50 & 80 & 0 & 10 & 70 & 70 & 50 & 10 & 70 & 10 \\ 10 & 80 & 70 & 0 & 30 & 30 & 40 & 80 & 40 & 40 \\ 40 & 20 & 50 & 20 & 0 & 10 & 10 & 70 & 20 & 20 \\ 10 & 60 & 70 & 10 & 60 & 0 & 40 & 70 & 20 & 50 \\ 80 & 80 & 60 & 90 & 20 & 70 & 0 & 40 & 10 & 60 \\ 50 & 30 & 10 & 60 & 50 & 10 & 50 & 0 & 10 & 90 \\ 10 & 70 & 90 & 30 & 10 & 50 & 40 & 10 & 0 & 40 \\ 90 & 10 & 50 & 10 & 10 & 20 & 80 & 90 & 20 & 0 \end{bmatrix}$$

2. Путь: $(1 \to 7 \to 3 \to 5 \to 9 \to 10 \to 8 \to 4 \to 6 \to 2)$

   Минимальная длина: 170

$$\begin{bmatrix} 0 & 60 & 30 & 90 & 90 & 70 & 30 & 80 & 20 & 40 \\ 10 & 0 & 10 & 50 & 70 & 70 & 20 & 20 & 60 & 40 \\ 80 & 20 & 0 & 60 & 10 & 40 & 60 & 80 & 20 & 80 \\ 70 & 80 & 50 & 0 & 40 & 20 & 50 & 30 & 30 & 90 \\ 90 & 20 & 10 & 80 & 0 & 50 & 80 & 70 & 20 & 30 \\ 30 & 30 & 70 & 70 & 70 & 0 & 90 & 10 & 90 & 80 \\ 30 & 30 & 10 & 60 & 30 & 50 & 0 & 30 & 70 & 10 \\ 70 & 60 & 30 & 10 & 10 & 50 & 50 & 0 & 90 & 80 \\ 60 & 50 & 30 & 60 & 60 & 60 & 60 & 20 & 0 & 10 \\ 50 & 70 & 80 & 40 & 60 & 70 & 70 & 20 & 30 & 0 \end{bmatrix}$$

3. Путь: $(1 \to 7 \to 3 \to 4 \to 6 \to 5 \to 9 \to 2 \to 8 \to 10)$

   Минимальная длина: 180

$$\begin{bmatrix} 0 & 70 & 30 & 20 & 90 & 40 & 10 & 40 & 10 & 70 \\ 70 & 0 & 30 & 10 & 80 & 80 & 20 & 20 & 40 & 30 \\ 50 & 70 & 0 & 10 & 10 & 90 & 10 & 70 & 10 & 40 \\ 50 & 70 & 60 & 0 & 10 & 20 & 70 & 50 & 60 & 20 \\ 30 & 80 & 10 & 90 & 0 & 10 & 10 & 20 & 10 & 30 \\ 50 & 80 & 30 & 70 & 10 & 0 & 50 & 90 & 50 & 30 \\ 10 & 50 & 10 & 30 & 10 & 20 & 0 & 70 & 80 & 30 \\ 20 & 70 & 10 & 30 & 30 & 30 & 90 & 0 & 10 & 20 \\ 80 & 50 & 60 & 80 & 90 & 60 & 70 & 70 & 0 & 80 \\ 20 & 90 & 20 & 20 & 20 & 90 & 50 & 80 & 90 & 0 \end{bmatrix}$$

4. Путь: $(1 \to 3 \to 2 \to 8 \to 5 \to 6 \to 10 \to 4 \to 9 \to 7)$

   Минимальная длина: 110

$$\begin{bmatrix} 0 & 10 & 10 & 10 & 80 & 70 & 40 & 20 & 10 & 10 \\ 20 & 0 & 60 & 70 & 70 & 80 & 40 & 10 & 90 & 10 \\ 10 & 10 & 0 & 50 & 70 & 90 & 50 & 40 & 20 & 40 \\ 60 & 10 & 10 & 0 & 60 & 90 & 50 & 60 & 10 & 20 \\ 60 & 60 & 60 & 20 & 0 & 10 & 10 & 10 & 30 & 70 \\ 60 & 60 & 80 & 10 & 90 & 0 & 70 & 80 & 90 & 10 \\ 10 & 70 & 30 & 60 & 30 & 20 & 0 & 50 & 10 & 30 \\ 40 & 10 & 80 & 80 & 10 & 50 & 60 & 0 & 70 & 30 \\ 60 & 20 & 70 & 30 & 20 & 80 & 20 & 60 & 0 & 20 \\ 20 & 20 & 90 & 10 & 10 & 50 & 20 & 50 & 70 & 0 \end{bmatrix}$$

5. Путь: $(1 \to 3 \to 4 \to 10 \to 7 \to 8 \to 2 \to 6 \to 9 \to 5)$

   Минимальная длина: 160

$$\begin{bmatrix}
0 & 90 & 10 & 40 & 10 & 90 & 90 & 70 & 60 & 60 \\
70 & 0 & 90 & 50 & 40 & 10 & 40 & 90 & 50 & 50 \\
10 & 30 & 0 & 50 & 20 & 10 & 90 & 60 & 50 & 20 \\
70 & 60 & 90 & 0 & 60 & 30 & 60 & 10 & 20 & 20 \\
10 & 10 & 90 & 60 & 0 & 30 & 70 & 40 & 60 & 50 \\
20 & 40 & 80 & 60 & 60 & 0 & 30 & 80 & 10 & 70 \\
50 & 40 & 10 & 50 & 40 & 60 & 0 & 10 & 40 & 30 \\
60 & 20 & 50 & 80 & 70 & 70 & 50 & 0 & 10 & 40 \\
30 & 60 & 30 & 80 & 10 & 10 & 40 & 30 & 0 & 80 \\
60 & 30 & 10 & 90 & 20 & 30 & 10 & 20 & 10 & 0
\end{bmatrix}$$

В таблице 3 приведен пример соответствия параметров с полученными результатами для матрицы 1.

Для остальных матриц получены соответствующие таблицы. Тем не менее, гораздо удобнее анализировать результаты в графическом виде.

Таблица 3: Параметризация на матрице 1

№	α	ρ	t	Ответ	Муравьи
0	0.0	0.1	10	130	190
1	0.0	0.1	20	130	190
2	0.0	0.1	30	130	190
3	0.0	0.1	40	130	190
4	0.0	0.1	50	130	190
5	0.0	0.1	60	130	190
...	...	...	...	...	...
360	0.1	0.9	10	130	170
361	0.1	0.9	20	130	170
362	0.1	0.9	30	130	160
363	0.1	0.9	40	130	160
364	0.1	0.9	50	130	160
365	0.1	0.9	60	130	170
...	...	...	...	...	...
806	0.4	0.1	70	130	160
807	0.4	0.1	80	130	150
808	0.4	0.1	90	130	150
809	0.4	0.1	100	130	170
810	0.4	0.1	110	130	150
...	...	...	...	...	...
1517	0.7	0.6	180	130	190
1518	0.7	0.6	190	130	190
1519	0.7	0.6	200	130	190
1520	0.7	0.7	10	130	190
1521	0.7	0.7	20	130	190
1522	0.7	0.7	30	130	190
...	...	...	...	...	...
2195	1.0	1.0	160	130	270
2196	1.0	1.0	170	130	300
2197	1.0	1.0	180	130	270
2198	1.0	1.0	190	130	280
2199	1.0	1.0	200	130	280

Значения α, ρ, t, на которых муравьиный алгоритм показал приемлемый результат для всех 5 выбранных матриц, представлены в таблице 4.

Таблица 4: Результаты параметризации

α	ρ	t
0.1	0.1	60
0.1	0.2	50
0.1	0.2	180
0.1	0.3	170
0.1	0.9	50
0.1	0.9	150
0.1	0.9	190
0.2	0.2	80
0.2	0.2	200
0.2	0.3	60
0.2	0.3	140
0.2	0.4	190
0.2	0.5	90
0.2	0.5	100
0.2	0.6	90
0.2	0.6	140
0.2	0.6	200
0.2	0.7	10
0.2	0.7	190
0.2	0.8	60
0.2	0.8	180
0.2	0.9	190
0.2	0.9	200
0.3	0.1	140
0.3	0.3	10
0.3	0.9	160
0.3	1.0	110
0.4	0.1	60
0.4	0.1	160
0.4	0.3	110
0.4	0.5	40
0.4	0.6	160
0.4	0.8	60
0.4	1.0	70

Выводы

Можно заметить, что хорошие результаты получены при $\alpha \in [0,1; 0,4]$ в сочетании с $\rho \in (0; 1]$ и $t \in \{50; 150; 200\}$. Однако, стоит учесть, что для других классов задач параметры могут иметь совсем не такие значения. Для каждого отдельного случая необходимо проводить параметризацию.

Заключение

В работе экспериментально подтверждена эффективность муравьиного алгоритма в сравнении с точным перебором всех маршрутов (на матрицах размерностью 10 быстрее в 2 раза). Также проведена параметризация вероятностного алгоритма муравья. Результаты свидетельствуют, что при $\alpha \in [0,1; 0,4]$ в сочетании с $\rho \in (0; 1]$ и $t \in \{50; 150; 200\}$ получаются приемлемые результаты на случайных матрицах размерностью 10. Тем не менее, стоит учесть, что для других классов задач параметры могут иметь совсем не такие значения. Для каждого отдельного случая необходимо проводить новую параметризацию.

Список литературы

[1] Штовба С.Д. Муравьиные алгоритмы // Exponenta Pro. Математика в приложениях, 2003, №4, с.70-75

[2] Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход.- М.:Техносфера, 2009.

[3] Д. Кнут. Искусство программирования, М., Мир, 1978

[4] Задача коммивояжера [Электронный ресурс]. – Режим доступа: http://www.math.nsc.ru/LBRT/k5/OR-MMF/TSPr.pdf, свободный – (02.12.2019)

[5] Практическое применение алгоритма решения задачи коммивояжера [Электронный ресурс]. – Режим доступа: https://cyberleninka.ru/article/n/prakticheskoe-primenenie-algoritma-resheniya-zadachi-kommivoyazhera/, свободный – (01.12.2019)

[6] Алгоритмы муравья [Электронный ресурс]. – Режим доступа: http://www.berkut.mk.ua/download/pdfsmyk/algorMurav.pdf, свободный – (24.11.2019)

[7] Оптимизация методом колонии муравьев [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/163887/, свободный – (28.11.2019)

[8] Python 3.8.2rc1 documentation [Электронный ресурс]. - Режим доступа: https://docs.python.org/3/, свободный - (28.11.2019)

[9] PyCharm documentation [Электронный ресурс]. - Режим доступа: https://www.jetbrains.com/pycharm/documentation/ - (28.11.2019)