



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»

**Отчёт по лабораторной работе №6  
по дисциплине "Анализ алгоритмов"**

Тема Муравьиный алгоритм и метод полного перебора для решения задачи коммивояжёра

Студент Романов А.В.

Группа ИУ7-53Б

Преподаватели Волкова Л.Л., Строганов Ю.В.

Москва — 2020 г.

# Оглавление

|   |           |
|---|-----------|
| <b>Введение</b>                             | <b>2</b>  |
| <b>1 Аналитическая часть</b>                | <b>3</b>  |
| 1.1 Полный перебор . . . . .                | 3         |
| 1.2 Муравьиный алгоритм . . . . .           | 3         |
| <b>2 Конструкторская часть</b>              | <b>6</b>  |
| 2.1 Разработка алгоритмов . . . . .         | 6         |
| 2.2 Автоматическая параметризация . . . . . | 9         |
| <b>3 Технологическая часть</b>              | <b>10</b> |
| 3.1 Требование к ПО . . . . .               | 10        |
| 3.2 Средства реализации . . . . .           | 10        |
| 3.3 Реализация алгоритмов . . . . .         | 10        |
| 3.4 Тестовые данные . . . . .               | 17        |
| <b>4 Исследовательская часть</b>            | <b>18</b> |
| 4.1 Технические характеристики . . . . .    | 18        |
| 4.2 Время выполнения алгоритмов . . . . .   | 18        |
| 4.3 Автоматическая параметризация . . . . . | 19        |
| <b>Заключение</b>                           | <b>23</b> |
| <b>Литература</b>                           | <b>23</b> |

# Введение

Муравьиный алгоритм – один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьёв, ищущих пути от колонии к источнику питания, и представляет собой метаэвристическую оптимизацию.

## Цель лабораторной работы

Целью данной лабораторной работы является изучение муравьиного алгоритма и приобретение навыков параметризации методов на примере муравьиного алгоритма.

## Задачи лабораторной работы

В рамках выполнения работы необходимо решить следующие задачи:

- решить задачу коммивояжера при помощи алгоритма полного перебора и муравьиного алгоритма;
- замерить и сравнить время выполнения алгоритмов;
- протестировать муравьиный алгоритм на разных переменных;
- сделать выводы на основе проделанной работы.

# 1 | Аналитическая часть

В данном разделе представлены теоретические сведения о рассматриваемых алгоритмах.

## 1.1 Полный перебор

Пронумеруем все города от 1 до  $n$ . Базовому городу присвоим номер  $n$ . Каждый тур по городам однозначно соответствует перестановке целых чисел  $1, 2, \dots, n - 1$ .

Задачу коммивояжера можно решить образуя все перестановки первых  $n - 1$  целых положительных чисел. Для каждой перестановки строится соответствующий тур и вычисляется его стоимость. Обработывая таким образом все перестановки, запоминается тур, который к текущему моменту имеет наименьшую стоимость. Если находится тур с более низкой стоимостью, то дальнейшие сравнения производятся с ним.

Сложность алгоритма полного перебора составляет  $O(n!)$  [1].

## 1.2 Муравьиный алгоритм

Моделирование поведения муравьев 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, модели-

руя 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.

- муравьи имеют собственную «память». 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  $i$ ;
- муравьи обладают «зрением» — видимость есть эвристическое желание посетить город  $j$ , если муравей находится в городе  $i$ . Будем считать, что видимость обратно пропорциональна расстоянию между городами  $i$  и  $j$  —  $D_{ij}$

$$\eta_{ij} = \frac{1}{D_{ij}} \quad (1.1)$$

- муравьи обладают «обонянием» — они могут 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 мы можем сформулировать вероятностно-пропорциональное правило 1.2, определяющее вероятность перехода  $k$ -ого муравья из города  $i$  в город  $j$ :

$$\begin{cases} P_{i,j,k}(t) = \frac{[\tau_{ij}(t)]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in J_{i,k}} [\tau_{il}(t)]^\alpha [\eta_{il}]^\beta}, & j \in J_{i,k}; \\ P_{i,j,k}(t) = 0, & j \notin J_{i,k}, \end{cases} \quad (1.2)$$

где  $\alpha, \beta$  — параметры, задающие веса следа феромона, при  $\alpha = 0$  алгоритм вырождается до жадного алгоритма (будет выбран ближайший город). Заметим, что выбор города является вероятностным, правило 1.2 лишь определяет ширину зоны города  $j$ ; в общую зону всех городов  $J_{i,k}$ , бросается случайное число, которое и определяет выбор муравья. Правило 1.2 не изменяется в ходе алгоритма, но у двух разных муравьев значение вероятности перехода будут отличаться, т. к. они имеют разный список разрешенных городов.

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 в виде:

$$P_{i,j,k}(t) = \begin{cases} \frac{Q}{L_k(t)}, & (i, j) \in T_k(t); \\ 0, & (i, j) \notin T_k(t). \end{cases} \quad (1.3)$$

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

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

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 [2].

## Вывод

В данном разделе были рассмотрены особенности алгоритмов решения задачи коммивояжера.

## 2 | Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов.

### 2.1 Разработка алгоритмов

На рисунках 2.1 - 2.2 приведены схемы алгоритмов решения задачи коммивояжера.

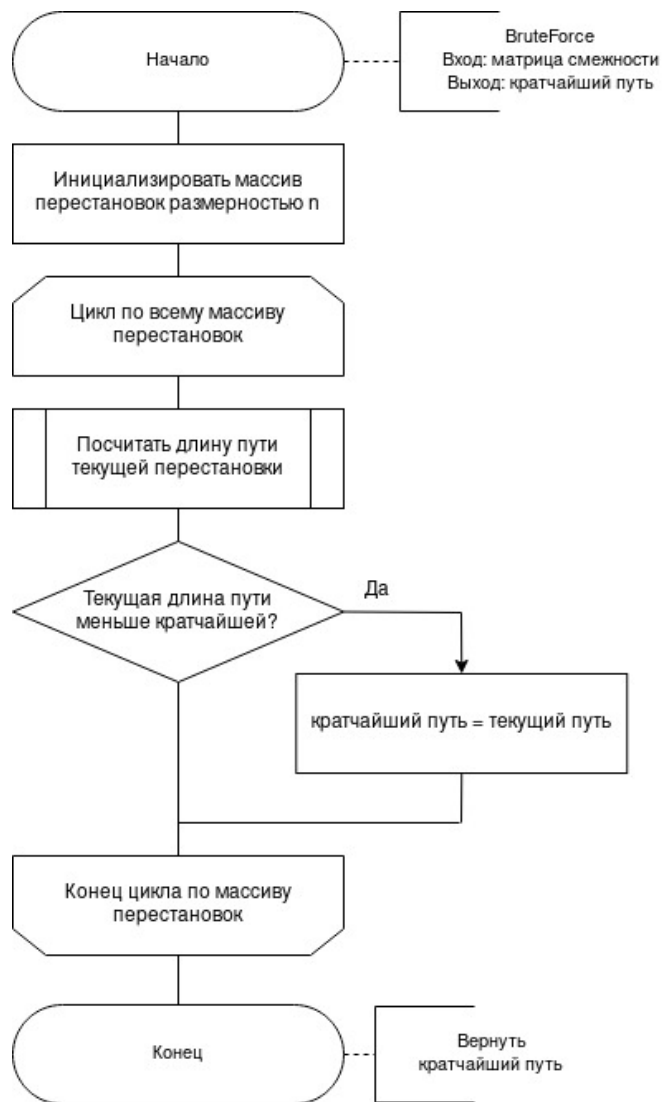


Рис. 2.1: Схема алгоритма полного перебора.



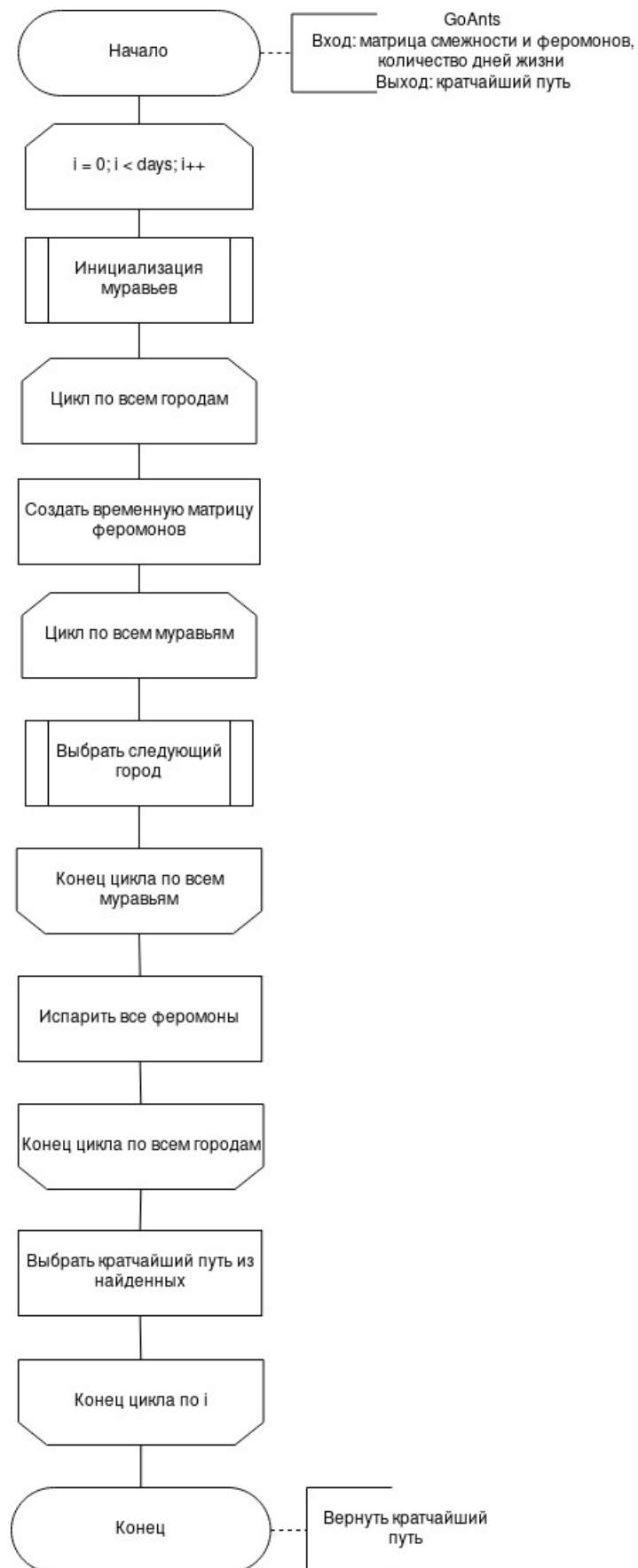


Рис. 2.2: Схема муравьиного алгоритма.

## 2.2 Автоматическая параметризация

Автоматическая параметризация выполняет проверку дней  $= [1..v]$  (где  $v$  — размер графа),  $\alpha = [0..1]$ ,  $\rho = [0..1]$  независимо друг от друга.

Алгоритм запускается три раза и минимальное значение сравнивается с эталонным. Затем на экран выводятся параметры и сравниваются с эталонными значениями.

## Вывод

На основе теоретических данных, полученных из аналитического раздела, были построены схемы алгоритмов для решения задачи коммивояжёра.

## 3 | Технологическая часть

В данном разделе приведены средства реализации и листинги кода.

### 3.1 Требование к ПО

К программе предъявляется ряд требований:

- на вход подается матрица смежности, со значениями не более чем максимальное целое число деленное пополам;
- на выходе – кратчайший путь.

### 3.2 Средства реализации

Для реализации ПО я выбрал язык программирования Golang [3]. Данный выбор обусловлен моим желанием расширить свои знания в области применения данного языка программирования.

### 3.3 Реализация алгоритмов

В листингах 3.1 - 3.2 представлены листинги алгоритмов решения задачи коммивояжёра.

Листинг 3.1: Алгоритм полного перебора

```
1 func BruteForce(graph [][]int) []int {  
2     path := make([]int, 0)  
3     shortest := make([]int, len(graph))  
4 }
```

```

5  for r := 0; r < len(graph); r++ {
6      routes := make([][]int, 0)
7      calculateRoutes(r, graph, path, &routes)
8
9      minSum := int(MaxInt)
10
11     for i := 0; i < len(routes); i++ {
12         curr := 0
13         for j := 0; j < len(routes[i])-1; j++ {
14             curr += graph[routes[i][j]][routes[i][j+1]]
15         }
16
17         if curr < minSum {
18             minSum = curr
19         }
20     }
21
22     shortest[r] = minSum
23 }
24
25 return shortest
26 }
27
28 func calculateRoutes(position int, graph [][]int, path []int, routes *[][]
    int) {
29     path = append(path, position)
30
31     if len(path) < len(graph) {
32         for i := 0; i < len(graph); i++ {
33             if !pathContains(path, i) {
34                 calculateRoutes(i, graph, path, routes)
35             }
36         }

```

```

37 } else {
38     *routes = append(*routes, path)
39 }
40 }
41
42 func pathContains(path []int, value int) bool {
43     for i := 0; i < len(path); i++ {
44         if path[i] == value {
45             return true
46         }
47     }
48
49     return false
50 }

```

Листинг 3.2: Муравьиный алгоритм

```

1 func GoAnts(colony *Colony, days int) []int {
2     shortest := make([]int, len(colony.graph))
3
4     for i := 0; i < days; i++ {
5         for j := 0; j < len(colony.graph); j++ {
6             ant := colony.createAnt(j)
7             ant.startMove()
8             current := ant.distance()
9
10            if (current < shortest[j]) || (0 == shortest[j]) {
11                shortest[j] = current
12            }
13        }
14    }
15
16    return shortest
17 }

```

```

18
19 func CreateColony(graph [][]int) *Colony {
20     colony := new(Colony)
21     colony.graph = graph
22     colony.pheromon = make([][]float64, len(colony.graph))
23
24     for i := 0; i < len(colony.graph); i++ {
25         colony.pheromon[i] = make([]float64, len(colony.graph[i]))
26         for j := 0; j < len(colony.pheromon[i]); j++ {
27             colony.pheromon[i][j] = pheromonCoef
28         }
29     }
30
31     return colony
32 }
33
34
35 func (colony *Colony) createAnt(position int) *Ant {
36     ant := new(Ant)
37     ant.colony = colony
38     ant.visited = make([][]int, len(colony.graph))
39
40     for i := 0; i < len(colony.graph); i++ {
41         ant.visited[i] = make([]int, len(colony.graph[i]))
42
43         for j := 0; j < len(colony.graph[i]); j++ {
44             ant.visited[i][j] = colony.graph[i][j]
45         }
46     }
47
48     ant.pos = position
49     ant.path = make([][]bool, len(colony.graph))
50

```

```

51  for i := 0; i < len(colony.graph); i++ {
52      ant.path[i] = make([] bool, len(colony.graph))
53  }
54
55  return ant
56 }
57
58 func (ant *Ant) makeMove(j int) {
59     for i := range ant.visited {
60         ant.visited[i][ant.pos] = 0
61     }
62
63     ant.path[ant.pos][j] = true
64     ant.pos = j
65 }
66
67 func (ant *Ant) startMove() {
68     way := MaxInt
69
70     for cond := true; cond; cond = (way != MinInt) {
71         way = chooseWay(ant.getProbability())
72
73         if MinInt != way {
74             ant.makeMove(way)
75             ant.updatePheromon()
76         }
77     }
78 }
79
80 func (ant *Ant) getProbability() []float64 {
81     probability := make([] float64, 0)
82     sum := float64(0)
83

```

```

84  for i, j := range ant.visited[ant.pos] {
85      if 0 != j {
86          d := math.Pow(ant.colony.pheromon[ant.pos][i], betta) * math.Pow((
            float64(1)/float64(j)), alpha)
87          probability = append(probability, d)
88          sum += d
89      } else {
90          probability = append(probability, 0)
91      }
92  }
93
94  for _, val := range probability {
95      val = val / sum
96  }
97
98  return probability
99 }
100
101 func chooseWay(path []float64) int {
102     sum := float64(0)
103
104     for _, j := range path {
105         sum += j
106     }
107
108     random := rand.New(rand.NewSource(time.Now().UnixNano())).Float64() * sum
109     sum = 0
110
111     for i, j := range path {
112         if random < sum+j && random > sum {
113             return i
114         }
115

```



```

116     sum += j
117 }
118
119 return MinInt
120 }
121
122 func (ant *Ant) updatePheromon() {
123     delta := float64(0)
124
125     for r := 0; r < len(ant.colony.pheromon); r++ {
126         for i, j := range ant.colony.pheromon[r] {
127
128             if 0 != ant.colony.graph[r][i] {
129                 delta = 0
130                 if ant.path[r][i] {
131                     delta = q / float64(ant.colony.graph[r][i])
132                 }
133
134                 ant.colony.pheromon[r][i] = (1 - p) * (float64(j) + delta)
135             }
136
137             if ant.colony.pheromon[r][i] <= 0 {
138                 ant.colony.pheromon[r][i] = 0.1
139             }
140         }
141     }
142 }
143
144 func (ant *Ant) distance() int {
145     distance := 0
146
147     for i, j := range ant.path {
148         for k, sign := range j {

```

```

149     if sign {
150         distance += ant.colony.graph[i][k]
151     }
152 }
153 }
154
155 return distance
156 }

```

### 3.4 Тестовые данные

В таблице 3.1 приведены тестовые данные. Все тесты были пройдены успешно.

| Матрица смежности   | Ожидаемый результат | Полученный результат |
|---|---------------------|----------------------|
| $\begin{bmatrix} 0 & 3 & 1 & 6 & 8 \\ 3 & 0 & 4 & 1 & 0 \\ 1 & 4 & 0 & 5 & 0 \\ 6 & 1 & 5 & 6 & 1 \\ 8 & 0 & 0 & 1 & 0 \end{bmatrix}$ | 15                  | 15                   |
| $\begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{bmatrix}$                          | 80                  | 80                   |

Таблица 3.1: Тестирование алгоритмов.

## Вывод

В данном разделе были разработаны и протестированны алгоритмы решения задачи коммивояжёра.

## 4 | Исследовательская часть

В данном разделе приведен анализ характеристик разработанного ПО.

### 4.1 Технические характеристики

Ниже приведены технические характеристики устройства, на котором было проведено тестирование ПО:

- Операционная система: Debian [4] Linux [5] 11 «bullseye» 64-bit.
- Оперативная память: 12 GB.
- Процессор: Intel(R) Core(TM) i5-3550 CPU @ 3.30GHz [6].

### 4.2 Время выполнения алгоритмов

Время выполнения алгоритма замерялось с помощью встроенной функции `time.Now()` из библиотеки `Time` [7]. Полученные результаты приведены в таблице 4.1.

| Размер графа | Полный перебор | Муравьиный алгоритм |
|--------------|----------------|---------------------|
| 3            | 1 490          | 81 200              |
| 5            | 2 500          | 143 100             |
| 7            | 43 220         | 698 700             |
| 9            | 2 240 905      | 1 290 500           |
| 11           | 31 470 700     | 2 100 340           |

Таблица 4.1: Сравнение времени исполнения алгоритмов решения задачи коммивояжера.

## 4.3 Автоматическая параметризация

В таблице 4.2 приведена выборка результатов параметризации для матрицы смежности размером 10x10. Количество дней принято равным 100. Полным перебором был посчитан оптимальный путь – он составил 130.

Таблица 4.2: Выборка из параметризации для матрицы размером  $10 \times 10$ .

| $\alpha$ | $\beta$ | $\rho$ | Длина | Разница |
|----------|---------|--------|-------|---------|
| 0        | 1       | 0.0    | 130   | 0       |
| 0        | 1       | 0.3    | 130   | 0       |
| 0        | 1       | 0.5    | 131   | 1       |
| 0        | 1       | 1.0    | 130   | 0       |
| 0.1      | 0.9     | 0.0    | 130   | 0       |
| 0.1      | 0.9     | 0.3    | 130   | 0       |
| 0.1      | 0.9     | 0.6    | 131   | 1       |
| 0.1      | 0.9     | 1.0    | 130   | 0       |
| 0.2      | 0.8     | 0.0    | 130   | 0       |
| 0.2      | 0.8     | 0.3    | 131   | 1       |
| 0.2      | 0.8     | 0.6    | 131   | 1       |
| 0.2      | 0.8     | 1.0    | 130   | 0       |
| 0.3      | 0.7     | 0.0    | 131   | 1       |
| 0.3      | 0.7     | 0.4    | 130   | 0       |
| 0.3      | 0.7     | 0.9    | 131   | 1       |
| 0.3      | 0.7     | 1.0    | 130   | 0       |
| 0.4      | 0.6     | 0.0    | 130   | 0       |
| 0.4      | 0.6     | 0.4    | 131   | 1       |
| 0.4      | 0.6     | 0.5    | 130   | 0       |
| 0.4      | 0.6     | 1.0    | 130   | 0       |
| 0.5      | 0.5     | 0.0    | 130   | 0       |
| 0.5      | 0.5     | 0.3    | 131   | 1       |
| 0.5      | 0.5     | 0.7    | 131   | 1       |
| 0.5      | 0.5     | 1.0    | 130   | 0       |
| 0.6      | 0.4     | 0.2    | 136   | 6       |
| 0.6      | 0.4     | 0.6    | 133   | 3       |
| 0.6      | 0.4     | 0.7    | 130   | 0       |
| 0.6      | 0.4     | 0.7    | 130   | 0       |

| $\alpha$ | $\beta$ | $\rho$ | Длина | Разница |
|----------|---------|--------|-------|---------|
| 0.7      | 0.3     | 0.0    | 130   | 0       |
| 0.7      | 0.3     | 0.3    | 134   | 4       |
| 0.7      | 0.3     | 0.6    | 132   | 2       |
| 0.7      | 0.3     | 0.8    | 139   | 9       |
| 0.8      | 0.2     | 0.0    | 140   | 10      |
| 0.8      | 0.2     | 0.5    | 134   | 4       |
| 0.8      | 0.2     | 0.7    | 131   | 1       |
| 0.8      | 0.2     | 1.0    | 130   | 0       |
| 0.9      | 0.1     | 0.0    | 134   | 4       |
| 0.9      | 0.1     | 0.3    | 132   | 2       |
| 0.9      | 0.1     | 0.5    | 134   | 4       |
| 0.9      | 0.1     | 1.0    | 130   | 0       |
| 1.0      | 0.0     | 0.0    | 145   | 25      |
| 1.0      | 0.0     | 0.4    | 133   | 3       |
| 1.0      | 0.0     | 0.7    | 142   | 22      |
| 1.0      | 0.0     | 1.0    | 138   | 8       |

## Вывод

При небольших размерах графа (от 3 до 7) алгоритм полного перебора выигрывает по времени у муравьиного. Например, при размере графа 5, полный перебор работает быстрее примерно в 57 раз. Однако, при увеличении размера графа (от 9 и выше), ситуация меняется в обратную сторону: муравьиный алгоритм начинает значительно выигрывать по времени у алгоритма полного перебора. На размерах графа 11, муравьиный алгоритм работает в 15 раз быстрее.

Наиболее стабильные результаты автоматической параметризации получаются при наборе  $\alpha = 0.1..0.5$ ,  $\beta = 0.1..0.5$ ,  $\rho = \text{любое}$ . При таких параметрах полученный результат не отличается более чем на 1 от эталонного, и, в около 75% (на промежутке  $\rho = 0.0..1.0$ ) случаев полученный результат совпадает с эталонным. Наиболее нестабильные результаты получен-

ны при  $\alpha = 1.0$ ,  $\beta = 0.0$ ,  $\rho = \text{любое}$ .

# Заключение

В рамках данной лабораторной работы лабораторной работы была достигнута её цель: изучен муравьиный алгоритм и приобретены навыки параметризации методов на примере муравьиного алгоритма. Также выполнены следующие задачи:

- реализованны два алгоритма решения задачи коммивояжера;
- замерено время выполнения алгоритмов;
- муравьиный алгоритм протестирован на разных переменных;
- сделаны выводы на основе проделанной работы;

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. Так, например, при размере графа 11, муравьиный алгоритм работает быстрее чем алгоритм полного перебора в 15 раз. Стоит отметить, что муравьиный алгоритм не гарантирует что найденный путь будем оптимальным, так как он является эвристическим алгоритмом, в отличии от алгоритма полного перебора.



# Литература

- [1] Гудман С. Хидетниеми С. Введение в разработку и анализ алгоритмов. Мир, 1981. с. 368.
- [2] М.В. Ульянов. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. ФИЗМАТЛИТ, 2007. с. 308.
- [3] Go is an open source programming language that makes it easy to build simple, reliable, and efficient software. Режим доступа: <https://golang.org/>. Дата обращения: 10.12.2020.
- [4] Debian – универсальная операционная система [Электронный ресурс]. Режим доступа: <https://www.debian.org/>. Дата обращения: 20.09.2020.
- [5] Linux – Getting Started [Электронный ресурс]. Режим доступа: <https://linux.org>. Дата обращения: 20.09.2020.
- [6] Процессор Intel® Core™ i5-3550 [Электронный ресурс]. Режим доступа: <https://ark.intel.com/content/www/ru/ru/ark/products/65516/intel-core-i5-3550-processor-6m-cache-up-to-3-70-ghz.html>. Дата обращения: 20.09.2020.
- [7] Golang – package time. Режим доступа: <https://golang.org/pkg/time/>. Дата обращения: 10.12.2020.