



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»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

## ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Тема: Муравьиный алгоритм

Студент: Сироткина П.Ю.

Группа: ИУ7-56Б

Оценка: \_\_\_\_\_

Преподаватель: Волкова Л.Л.

Москва, 2021 г.

# Содержание

Введение . . . . .	3
1 Аналитический раздел . . . . .	4
1.1 Постановка задачи . . . . .	4
1.2 Алгоритм полного перебора для решения задачи коммивояжера . . . . .	5
1.3 Муравьиный алгоритм для решения задачи коммивояжера . . . . .	5
1.4 Вывод . . . . .	8
2 Конструкторский раздел . . . . .	10
2.1 Схема алгоритма полного перебора . . . . .	10
2.2 Съема муравьиного алгоритма . . . . .	11
2.3 Описание памяти, используемой программой . . . . .	12
2.4 Структура ПО . . . . .	12
2.5 Вывод . . . . .	12
3 Технологический раздел . . . . .	13
3.1 Средства реализации . . . . .	13
3.2 Листинг кода . . . . .	13
3.3 Тестирование ПО . . . . .	19
3.4 Вывод . . . . .	20
4 Экспериментальный раздел . . . . .	21
4.1 Пример работы . . . . .	21
4.2 Технические характеристики . . . . .	21
4.3 Постановка эксперимента . . . . .	22
4.4 Результаты эксперимента . . . . .	22
4.5 Время выполнения алгоритмов . . . . .	24
4.6 Вывод . . . . .	25
Заключение . . . . .	26
Список использованных источников . . . . .	27
А Параметрическая таблица исследований . . . . .	28

## Введение

Муравьиный алгоритм - алгоритм эвристической оптимизации путем подражания муравьиной колонии, использующийся для нахождения *приближенных* решений задач коммивояжера, а также решения аналогичных NP-трудных задач поиска маршрутов на графах.

Важно понимать, что большинство эвристических методов, к которым относится муравьиный алгоритм, не гарантируют точное решение, но являются достаточными для решения задачи за оптимальное время.

Общая постановка задачи коммивояжера:

Пусть есть  $N$  городов, соединенных между собой. Странствующий торговец (т.н. коммивояжер) объезжает эти города. Ему необходимо составить такой маршрут, чтобы он был самым выгодным, и при этом каждый из  $N$  городов должен быть посещен хотя бы по одному разу.

Выгодность указывается в рамках конкретной задачи: время в пути, стоимость поездки и т.д.

**Целью лабораторной работы** является изучение основ параметризации эвристических методов на основе муравьиного алгоритма для задачи коммивояжера.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- Изучить задачу коммивояжера, а также идеи ее решения с помощью муравьиного алгоритма и алгоритма полного перебора;
- Привести схемы изученных алгоритмов;
- Описать используемые типы данных;
- Описать структуру разрабатываемого ПО;
- Реализовать изученные алгоритмы;
- Протестировать разработанное ПО;
- Сравнить временные характеристики каждого из рассмотренных алгоритмов;
- Сделать выводы на основе проделанной работы.

# 1 Аналитический раздел

В данном разделе описаны основные идеи рассматриваемых алгоритмов. Поставлена задача, на примере которой будут исследоваться выбранные алгоритмы.

## 1.1 Постановка задачи

Пусть есть  $N$  городов, соединенных между собой. Странствующий торговец (т.н. коммивояжер) объезжает эти города. Ему необходимо составить такой маршрут, чтобы он был *наикратчайшим*, и при этом каждый из  $N$  городов должен быть посещен *ровно* по одному разу. В конце маршрута коммивояжер должен вернуться в город, с которого он начал маршрут. [1]

Для возможности применения математического аппарата для решения задачи её следует представить в виде математической модели.

Задачу коммивояжера можно представить в виде модели на графе. Вершины графа - города, ребра между вершинами - пути между этими городами. Каждому ребру в соответствие ставится его вес, который в данной задаче определяет расстояние между городами, определяемыми вершинами.

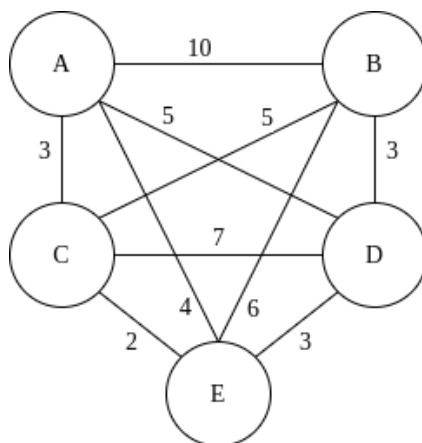


Рисунок 1.1 — Модель задачи коммивояжера на графе

Другими словами, задан взвешенный полностью связный граф. Для него необходимо найти гамильтонов цикл минимального веса.

## 1.2 Алгоритм полного перебора для решения задачи коммивояжера

Задачу коммивояжера можно решить, рассмотрев все возможные комбинации городов. Для каждого из маршрутов высчитывается его суммарная длина. Выбирается такой маршрут, для которого его суммарная длина минимальна.

Такой алгоритм гарантирует точное решение, однако, как известно, алгоритмическая сложность подобных алгоритмов составляет  $O(n!)$ , поэтому даже при небольшом количестве городов решение за оптимальное время становится невозможным.

## 1.3 Муравьиный алгоритм для решения задачи коммивояжера

Муравьиный алгоритм основан на особенностях поведения муравьев в природе.

При поиске путей к источникам пищи муравьи помечают пройденный путь специальным веществом — *феромоном*.

Феромон играет роль не прямой обратной связи в колонии муравьев — чем больше муравьев движется по помеченному пути, тем больше он становится привлекательным для других муравьев. По истечении определенного времени феромон испаряется.

В результате через некоторое время большая часть муравьев будет передвигаться от муравейника до найденного источника пищи по одному и тому же пути.

В вершинах описанного ранее графа располагаются муравьи, соответственно они перемещаются по ребрам графа.

Для каждого муравья вводится понятие *памяти* (или список запретов) - список пройденных им узлов за день. Выбирая узел для следующего шага, муравей 'помнит' об уже пройденных узлах и не рассматривает их в качестве возможных для перехода.

Также вводится понятие *зрения* - муравей может оценить длины ребер до следующих городов.

Пусть имеется  $N$  городов и  $N$  муравьев. Вводится матрица смежности  $D$  и матрица феромонов  $T$ .

Колония помещается в одну из вершин графа, называемую стартовой вершиной. Все ребра графа помечаются одинаковым фоновым значением феромона  $T_{ij} = T_0$ . Муравьи выходят утром каждого из дней из муравейника и независимо друг от друга за день формируют по целому маршруту.

При выборе следующей вершины для посещения муравьи опираются на значения концентрации феромона на смежных ребрах, которые не находятся в списке запрета, и на значения длин этих ребер.

К закату муравьи возвращаются в муравейник и 'обсуждают', кто нашел самый короткий маршрут. Если новый маршрут лучше того, который был ранее заполнен колонией, то его запоминают. Ночью происходит обновление феромонов на ребрах.

Пусть  $k$ -ый муравей находится в  $i$ -ой вершине, а его запретный список  $V_k$  еще не является до конца заполненным. Тогда вероятность перемещения в  $j$ -ую вершину определяется формулой (1.1):

$$P_{ij}^k = \begin{cases} \frac{T_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{r \in V_{i,k}} T_{ir}^\alpha \eta_{ir}^\beta}, & \text{вершина } j \text{ не была посещена сегодня муравьем } k, \\ 0, & \text{иначе} \end{cases} \quad (1.1)$$

Здесь  $\alpha$  - коэффициент 'жадности',  $\beta$  - коэффициент 'стадности', а  $\eta_{ij} = 1/D_{ij}$  - 'привлекательность' ребра.  $\alpha + \beta = \text{Const}$ .

Если  $\alpha = 0$ , алгоритм вырождается в чисто стадный (куда стадо, туда и все).

Если  $\beta = 0$ , алгоритм вырождается в жадный, потому что опыт колонии не учитывается.

Когда все муравьи завершили обход, происходит обновление феромона по формуле (1.2):

$$T_{ij}(t+1) = (1-p) \cdot (T_{ij}(t) + \Delta T_{ij}), \Delta T_{ij} = \sum_{k=1}^N \Delta T_{ij}^k \quad (1.2)$$

где

$$\Delta T_{ij}^k = \begin{cases} Q/L_k, & \text{ребро посещено } k\text{-ым муравьем,} \\ 0, & \text{иначе} \end{cases} \quad (1.3)$$

$L_k$  — длина пути  $k$ -ого муравья,  $Q$  — некоторая константа соразмерная длине лучшего пути (запас феромона),  $\Delta T_{ij}^k$  — сколько феромона оставил  $k$ -ый муравей на  $ij$ -ом ребре. Таким образом, чем короче путь  $k$ -го муравья, тем больше феромона он оставит на ребрах, по которым он ходил.

Описанные действия представляют собой одну итерацию муравьиного алгоритма. Итерации повторяются до тех пор, пока не окажется выполненным какой-нибудь из критериев останова алгоритма — исчерпано число итераций, достигнута нужная точность, получен единственный путь (алгоритм сошелся к некоторому решению)[2].

Нужно также учитывать случай, когда феромон может обнулиться — тогда обнулится вся вероятность идти по этому ребру. Чтобы этого не допустить, заводится константа фонового значения феромона. Если значение феромона становится меньше фонового, то ему присваивается фоновое значение.

Чтобы оставить возможность пойти по не самому оптимальному с точки зрения вероятностей пути, муравей 'подбрасывает монетку', получая случайное число от 0 до 1. Если искать только максимум вероятностей — это в своем роде жадное решение.

## Параметризация муравьиного алгоритма

Для эвристических методов проводится параметризация. Параметризация заключается в определении на основе материалов экспериментов таких параметров или настроек работы метода, при которых для выбранного класса данных задача решается с наилучшим качеством согласно некоторым мерам оценки качества, например, точность, доля, полнота ответа.

Грубо говоря, метод складывается из некоторого набора алгоритмов и некоторой математической модели представления задачи в предметной области. Для эвристических методов вводятся некоторые дополнительные понятия, которые составляют часть модели, которая представляет предметную область.

Здесь вводятся понятия муравья (некоторая часть алгоритма, которая соответствует видению муравья части задачи) и феромона (средство обмена информацией).

Для этой задачи могут быть разные классы данных. Предположим, это один вид карт с однотипным разбросом длин ребер. Нужно подстроить метод под какой-то конкретный класс данных, т.к. придумать универсальный алгоритм - трудно.

У алгоритма есть 3 параметра:  $\alpha$  - коэффициент 'жадности',  $\rho$  - коэффициент испарения феромона и  $T_{max}$  - время жизни колонии.

Чтобы провести параметризацию, выбирается некоторое ограниченное количество значений исследуемых параметров и задача решается при этих коэффициентах. Полученное решение сравнивается с эталонным значением, определяемым подсчетом полным перебором. Необходимо отыскать набор параметров, который будет наиболее близок к эталонному решению. Это делается с помощью построения таблицы.

### 1.4 Вывод

В данном разделе были изучены задача коммивояжера, алгоритм полного перебора и муравьиный алгоритм для решения задачи коммивояжера.



Допущение в рамках лабораторной работы: рассматриваются только симметричные матрицы смежности и феромонов для простоты.

## **Требования к ПО**

К программе предъявляются следующие требования:

- Входные данные содержатся в файле `localfile.txt` в следующем формате: первой строкой идет количество исследуемых городов, дальше построчно описываются строки матрицы смежности;
- Все элементы матрицы смежности, кроме диагональных, должны быть положительными;
- Выходные данные: минимальная длина пути и сам путь. Результат выводится в консоль;
- Программа не проверяет корректность файла `localfile.txt`;
- Программа должна иметь возможность выводить таблицу исследования параметризации.

Тестирование ПО проводится путем сравнения заранее вычисленного гамильтонового цикла минимального веса с полученными значениями в результате выполнения алгоритмов.

## 2 Конструкторский раздел

В данном разделе представлены схемы муравьиного алгоритма и алгоритма полного перебора для задачи коммивояжера.

### 2.1 Схема алгоритма полного перебора

На рисунке 2.1 представлена схема алгоритма полного перебора для задачи коммивояжера.

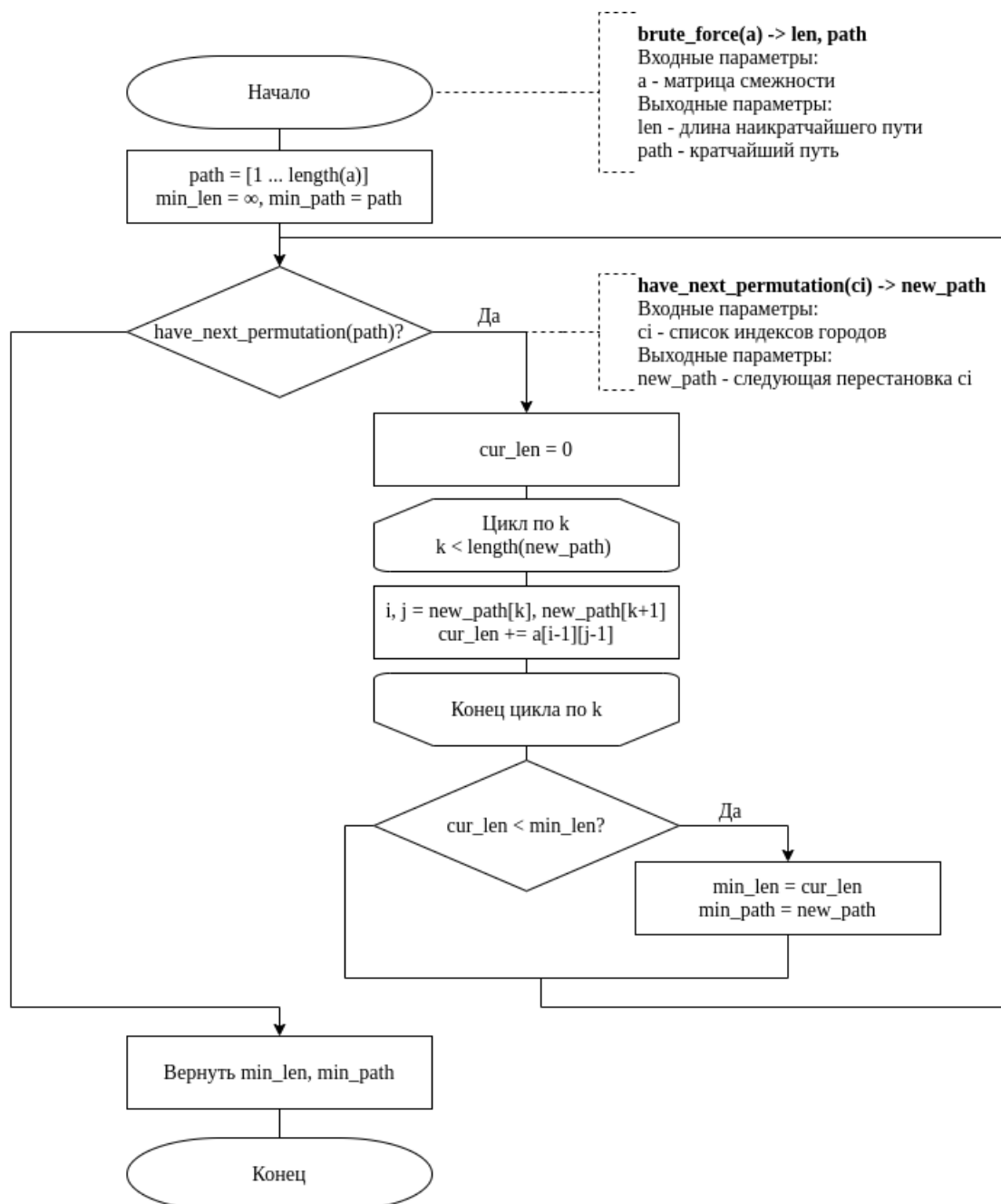


Рисунок 2.1 — Схема алгоритма полного перебора для задачи коммивояжера

## 2.2 Съема муравьиного алгоритма

На рисунке 2.2 представлена схема муравьиного алгоритма для задачи коммивояжера.

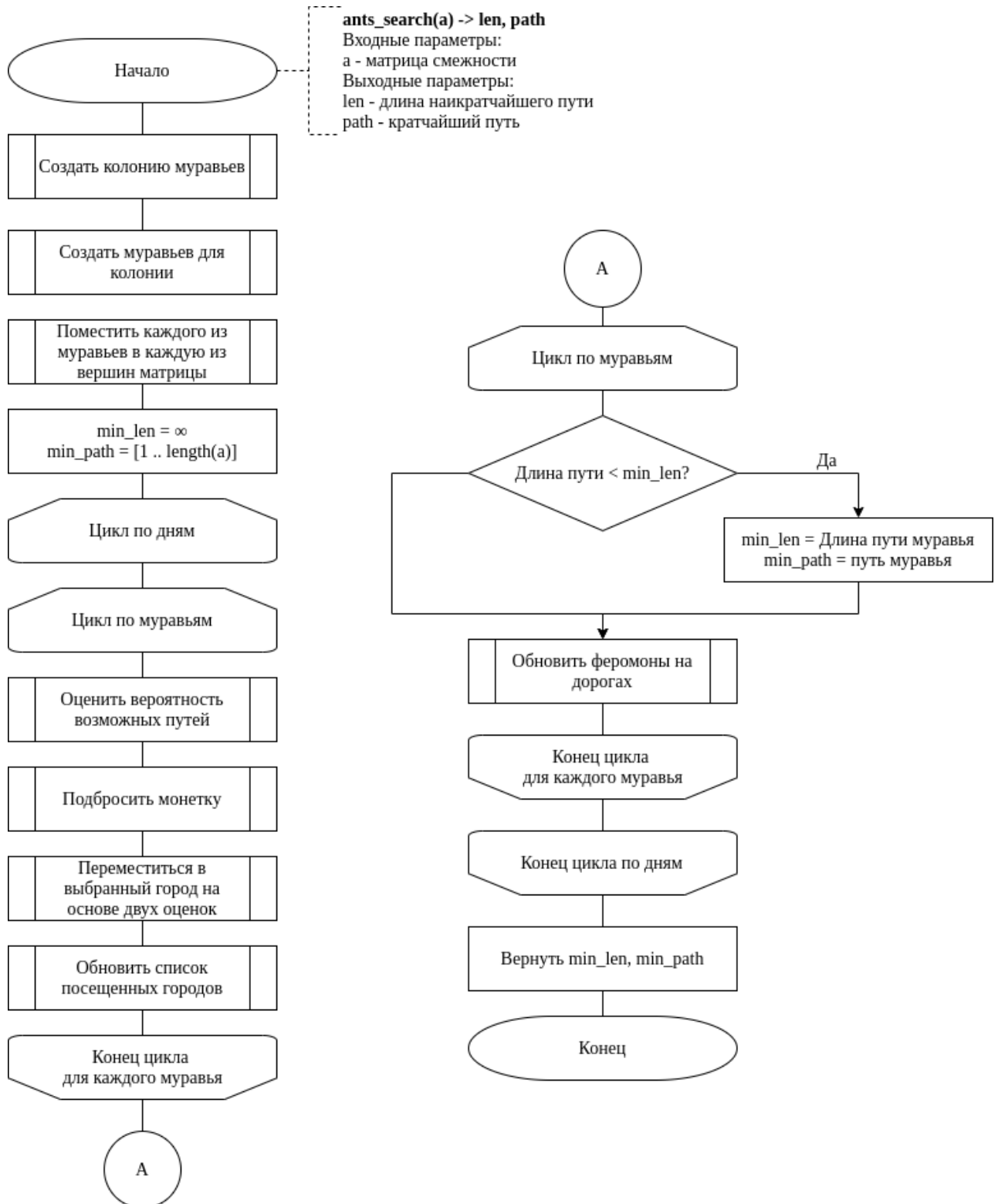


Рисунок 2.2 — Схема муравьиного алгоритма для задачи коммивояжера

## 2.3 Описание памяти, используемой программой

Затраты с точки зрения памяти на функционирование ПО складываются в большей степени за счет количества исследуемых городов и поддержания пользовательских типов данных.

## 2.4 Структура ПО

Программа будет включать в себя один смысловой модуль, называемый **algorithms**, который содержит в себе процедуры и функции, связанные с реализацией муравьиного алгоритма и алгоритма полного перебора. Независимо от модуля будет существовать файл `main.go`, реализующий мост между пользователем и модулем, описанным ранее.

## 2.5 Вывод

На основе теоретических данных, полученных из аналитического раздела, были разработаны схемы требуемых алгоритмов. Рассмотрены затраты по памяти и описана структура ПО.

## 3 Технологический раздел

В данном разделе приведены средства реализации и листинги кода.

### 3.1 Средства реализации

В качестве языка программирования для реализации лабораторной работы был выбран язык Go. Выбор этого языка обусловлен его быстрой работой и эффективностью, это лаконичный и гибкий язык.

### 3.2 Листинг кода

В листинге 3.1 представлена реализация алгоритма полного перебора для поставленной задачи.

Листинг 3.1 — Реализация алгоритма полного перебора

```
1 func brute_force(a [][]int, n int) (int, []int) {
2     path := make([]int, n)
3     for i := 0; i < n; i++ {
4         path[i] = i + 1
5     }
6
7     min_len := dist(a, path)
8     min_path := make([]int, n)
9
10    cur_path, state := path, true
11
12    for ; state == true; {
13        cur_len := dist(a, cur_path)
14        if cur_len < min_len {
15            min_len = cur_len
16            copy(min_path, cur_path)
17        }
18        cur_path, state = have_next_permutation(cur_path)
19    }
20    return min_len, min_path
21 }
22
23 func have_next_permutation(a []int) ([]int, bool) {
24     j := len(a) - 2
25     for j != -1 && a[j] >= a[j+1] {
26         j--
27     }
```

```

28     if j == -1 {
29         return a, false
30     }
31     k := len(a) - 1
32     for a[j] >= a[k] {
33         k--
34     }
35     swap(a, j, k)
36     l := j + 1
37     r := len(a) - 1
38     for l < r {
39         swap(a, l, r)
40         l++
41         r--
42     }
43     return a, true
44 }
45
46 func swap(a []int, i int, j int) {
47     s := a[i]
48     a[i] = a[j]
49     a[j] = s
50 }
51
52 func dist(a [][]int, path []int) (int) {
53     dist := 0
54     for k := 0; k < (len(path) - 1); k++ {
55         i := path[k] - 1
56         j := path[k + 1] - 1
57         dist += a[i][j]
58     }
59     dist += a[path[0] - 1][path[len(path) - 1] - 1]
60     return dist
61 }

```

В листинге 3.2 представлены используемые типы данных для реализации муравьиного алгоритма.

Листинг 3.2 — Реализация используемых типов данных

```

1 type Params_t struct {
2     alpha float64
3     beta float64
4     q float64
5     p float64
6     ph_coef float64
7     days int

```

```

8 }
9
10 type Colony_t struct {
11     matrix [][] int
12     pheromon [][] float64
13     params Params_t
14 }
15
16 type Ant_t struct {
17     colony *Colony_t
18     visited [][] int
19     path [] int
20     vertex int
21 }

```

В листинге 3.3 представлена реализация муравьиного алгоритма для поставленной задачи.

### Листинг 3.3 — Реализация муравьиного алгоритма

```

1 func AntsSearch(matrix [][] int) (int, [] int) {
2     params := new(Params_t)
3     params.alpha = alpha
4     params.beta = beta
5     params.q = q
6     params.p = p
7     params.ph_coef = ph_coef
8     params.days = days
9     params.q = float64(calc_q(matrix))
10    colony := create_colony(matrix, params)
11    return live(colony)
12 }
13
14 func calc_q(matrix [][] int) (float64) {
15     sum := float64(0)
16     for i := 0; i < len(matrix); i++ {
17         for j := 0; j < len(matrix); j++ {
18             sum += float64(matrix[i][j])
19         }
20     }
21     sum /= float64(len(matrix))
22     return sum
23 }
24
25 func create_colony(matrix [][] int, params *Params_t) *Colony_t {
26     n := len(matrix)
27     colony := new(Colony_t)

```

```

28     colony.matrix = matrix
29     colony.params = *params
30     colony.pheromon = make([[]] float64 , n)
31     for i := 0; i < n; i++ {
32         colony.pheromon[i] = make([] float64 , n)
33         for j := 0; j < n; j++ {
34             colony.pheromon[i][j] = colony.params.ph_coef
35         }
36     }
37     return colony
38 }
39
40 func live(colony *Colony_t) (int , []int) {
41     n := len(colony.matrix)
42     min_dist := 0
43     min_path := make([]int , n)
44     for i := 0; i < colony.params.days; i++ {
45         for j := 0; j < n; j++ {
46             ant := colony.create_ant(j)
47             ant.find_path()
48             ant.update_pheromon()
49             cur_dist := ant.distance()
50             if (cur_dist < min_dist) || (min_dist == 0) {
51                 min_dist = cur_dist
52                 copy(min_path, ant.path)
53             }
54         }
55     }
56     return min_dist , min_path
57 }
58
59 func (colony *Colony_t) create_ant(vertex int) *Ant_t {
60     n := len(colony.matrix)
61     ant := new(Ant_t)
62     ant.colony = colony
63     ant.visited = make([[]] int , n)
64     for i := 0; i < n; i++ {
65         ant.visited[i] = make([]int , n)
66         for j := 0; j < n; j++ {
67             ant.visited[i][j] = colony.matrix[i][j]
68         }
69     }
70     ant.vertex = vertex
71     ant.path = make([]int , 0)
72     ant.path = append(ant.path , ant.vertex)
73     return ant

```



```

74 }
75
76 func (ant *Ant_t) find_path() {
77     for {
78         p := ant.get_probabilities()
79         vertex := choose_vertex(p)
80         if vertex != -1 {
81             ant.move(vertex)
82         } else {
83             break
84         }
85     }
86 }
87
88 func (ant *Ant_t) get_probabilities() []float64 {
89     n := len(ant.colony.matrix)
90     probabilities := make([]float64, n)
91     psum := float64(0)
92     var (
93         param1 float64
94         param2 float64
95     )
96     for i := 0; i < n; i++ {
97         if ant.visited[ant.vertex][i] != 0 {
98             param1 = ant.colony.pheromon[ant.vertex][i]
99             param2 = (float64(1) /
100                 float64(ant.colony.matrix[ant.vertex][i]))
101             d := math.Pow(param1, ant.colony.params.beta) *
102                 math.Pow(param2, ant.colony.params.alpha)
103             probabilities[i] = d
104             psum += d
105         } else {
106             probabilities[i] = 0
107         }
108     }
109     for i := 0; i < n; i++ {
110         probabilities[i] /= psum;
111     }
112     return probabilities
113 }
114
115 func choose_vertex(p []float64) int {
116     n := len(p)
117     sum := float64(0)
118     for i := 0; i < n; i++ {
119         sum += p[i]

```

```

118     }
119     random_probability :=
120         rand.New(rand.NewSource(time.Now().UnixNano())).Float64() * sum
121     sum = 0
122     for i := 0; i < n; i++ {
123         if random_probability < sum + p[i] && random_probability > sum {
124             return i
125         }
126         sum += p[i]
127     }
128     return -1
129 }
130 func (ant *Ant_t) move(new_vertex int) {
131     n := len(ant.colony.matrix)
132     for i := 0; i < n; i++ {
133         ant.visited[i][ant.vertex] = 0
134     }
135     ant.path = append(ant.path, new_vertex)
136     ant.vertex = new_vertex
137 }
138
139 func (ant *Ant_t) update_pheromon() {
140     delta := float64(0)
141     for i := 0; i < len(ant.colony.pheromon); i++ {
142         for j := 0; j < len(ant.colony.pheromon); j++ {
143             if ant.colony.matrix[i][j] != 0 {
144                 f := false
145                 for k := 0; k < len(ant.path); k++ {
146                     src := ant.path[k]
147                     dst := ant.path[(k + 1) % len(ant.path)]
148                     if (src == i && dst == j) || (dst == i && src == j) {f
149                         = true}
150                 }
151                 if f {
152                     delta += ant.colony.params.q /
153                         float64(ant.colony.matrix[i][j])
154                 }
155             }
156             ant.colony.pheromon[i][j] = (1 - p) *
157                 (float64(ant.colony.pheromon[i][j]) + delta)
158         }
159     }

```

```

160     }
161 }
162
163 func (ant *Ant_t) distance() int {
164     distance := 0
165     n := len(ant.path)
166     for i := 0; i < n; i++ {
167         src := ant.path[i]
168         dst := ant.path[(i + 1) % len(ant.path)]
169         distance += ant.colony.matrix[src][dst]
170     }
171     return distance
172 }

```

### 3.3 Тестирование ПО

В таблице 3.1 приведены тестовые данные, на которых было протестировано разработанное ПО. Все тасты были успешно пройдены.

Таблица 3.1 — Тестирование алгоритмов

Матрица смежности	Ожидание	Муравьиный алгоритм	Полный перебор
$\begin{bmatrix} 0 & 20 & 30 \\ 20 & 0 & 10 \\ 30 & 10 & 0 \end{bmatrix}$	60	60	60
$\begin{bmatrix} 0 & 4 & 2 & 7 & 3 \\ 4 & 0 & 6 & 10 & 9 \\ 2 & 6 & 0 & 8 & 5 \\ 7 & 10 & 8 & 0 & 1 \\ 3 & 9 & 5 & 1 & 0 \end{bmatrix}$	22	22	22
$\begin{bmatrix} 0 & 10 & 5 & 9 & 11 \\ 10 & 0 & 11 & 8 & 12 \\ 5 & 11 & 0 & 3 & 9 \\ 9 & 8 & 3 & 0 & 12 \\ 11 & 12 & 9 & 12 & 0 \end{bmatrix}$	39	39	39

### **3.4 Вывод**

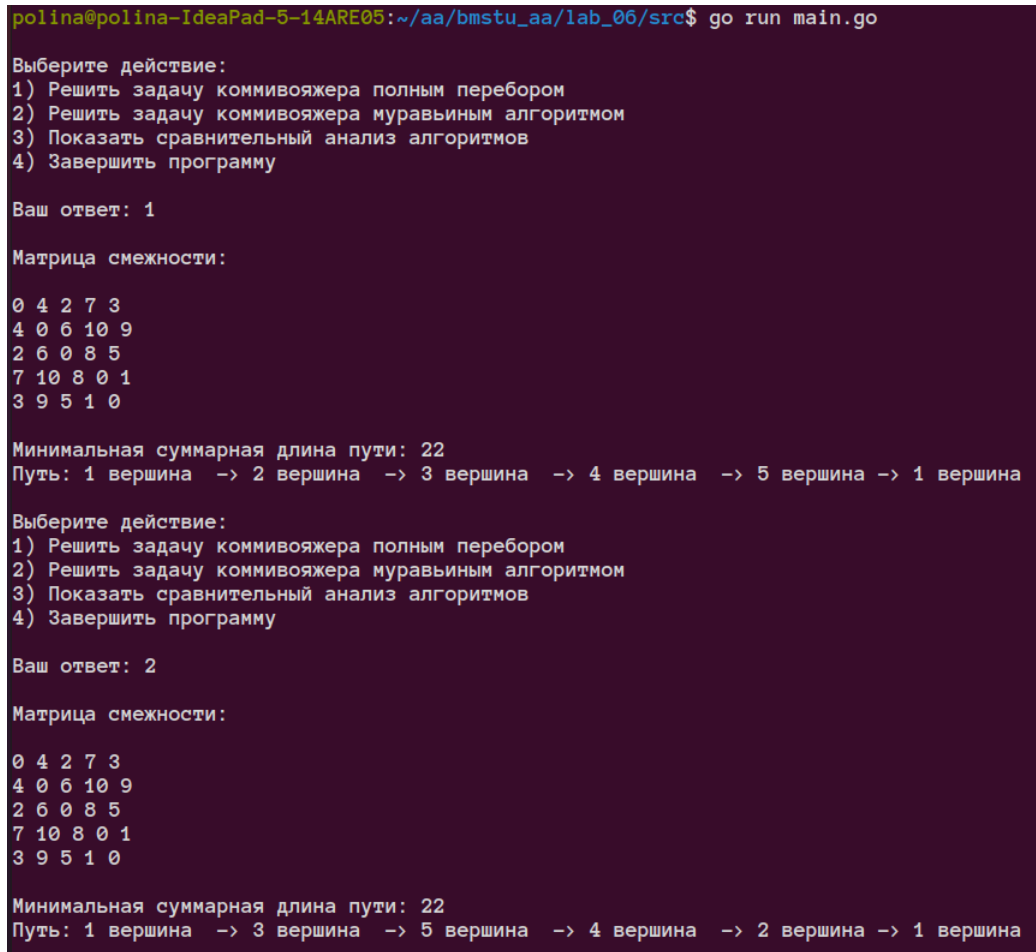
В данном разделе было представлено реализованное ПО для решения поставленной задачи, а также проведено тестирование.

## 4 Экспериментальный раздел

В данном разделе представлен пользовательский интерфейс, а также проведена оценка эффективности алгоритмов.

### 4.1 Пример работы

На рисунке 4.1 приведен пример работы программы.



```
polina@polina-IdeaPad-5-14ARE05:~/aa/bmstu_aa/lab_06/src$ go run main.go
Выберите действие:
1) Решить задачу коммивояжера полным перебором
2) Решить задачу коммивояжера муравьиным алгоритмом
3) Показать сравнительный анализ алгоритмов
4) Завершить программу

Ваш ответ: 1

Матрица смежности:

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

Минимальная суммарная длина пути: 22
Путь: 1 вершина -> 2 вершина -> 3 вершина -> 4 вершина -> 5 вершина -> 1 вершина

Выберите действие:
1) Решить задачу коммивояжера полным перебором
2) Решить задачу коммивояжера муравьиным алгоритмом
3) Показать сравнительный анализ алгоритмов
4) Завершить программу

Ваш ответ: 2

Матрица смежности:

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

Минимальная суммарная длина пути: 22
Путь: 1 вершина -> 3 вершина -> 5 вершина -> 4 вершина -> 2 вершина -> 1 вершина
```

Рисунок 4.1 — Пример работы программы

### 4.2 Технические характеристики

Технические характеристики машины, на которой выполнялось тестирование:

- Операционная система: Ubuntu[3] Linux[4] 20.04 64-bit;
- Оперативная память: 16 Gb;
- Процессор: AMD(R) Ryzen(TM)[5] 5 4500U CPU @ 2.3 CHz.

### 4.3 Постановка эксперимента

Проводятся эксперименты следующего типа: рассматривается 3 вида единообразных карт, и, варьируя коэффициенты параметризации муравьиного алгоритма, составляется таблица, содержащая следующие колонки: номер карты, коэффициент 'жадности', коэффициент испарения феромона, кол-во дней жизни колонии, полученный результат эвристического метода, эталонный результат.

Каждый из типов карт характеризуется тем, что дорога между каждым городом не больше 20 единиц.

В виду того, что эта таблица получится большой, ее полная версия приведена во вложении. В этом разделе представлена наиболее информативная выжимка этой таблицы.

Поле погрешность в данном случае несет следующий смысл: максимальная из погрешностей из погрешностей для 3-х однотипных карт для определенного одинакового набора параметров.

### 4.4 Результаты эксперимента

В таблицах 4.1-4.2 представлены результаты эксперимента.

Таблица 4.1 — Результаты эксперимента

$\alpha$	$\rho$	Кол-во дней	Погрешность
1	0.2	2	7
1	0.2	3	9
1	0.2	4	5
1	0.4	2	6
1	0.4	3	9
1	0.4	4	6
1	0.6	2	7
1	0.6	3	6
1	0.6	4	8
1	0.8	2	9

Таблица 4.2 — Результаты эксперимента (продолжение)

$\alpha$	$\rho$	Кол-во дней	Погрешность
1	0.8	3	8
1	0.8	4	6
2	0.2	2	4
2	0.2	3	0
2	0.2	4	7
2	0.4	2	4
2	0.4	3	5
2	0.4	4	4
2	0.6	2	8
2	0.6	3	4
2	0.6	4	0
2	0.8	2	4
2	0.8	3	5
2	0.8	4	0
3	0.2	2	3
3	0.2	3	5
3	0.2	4	3
3	0.4	2	4
3	0.4	3	3
3	0.4	4	3
3	0.6	2	4
3	0.6	3	3
3	0.6	4	1
3	0.8	2	5
3	0.8	3	2
3	0.8	4	0

## 4.5 Время выполнения алгоритмов

В таблице 4.3 представлены результаты замеров времени выполнения алгоритмов.

Таблица 4.3 — Время выполнения алгоритмов, микросекунды

Размерность матрицы	Муравьиный алгоритм	Полный перебор
1	600	600
3	4 662	4 461
5	13 539	13 541
7	27 916	28 001
10	64 847	135 884
12	95 035	1 029 8702

На рисунке 4.2 представлена зависимость времени выполнения муравьиного алгоритма и алгоритма полного перебора от размерности матрицы смежности.

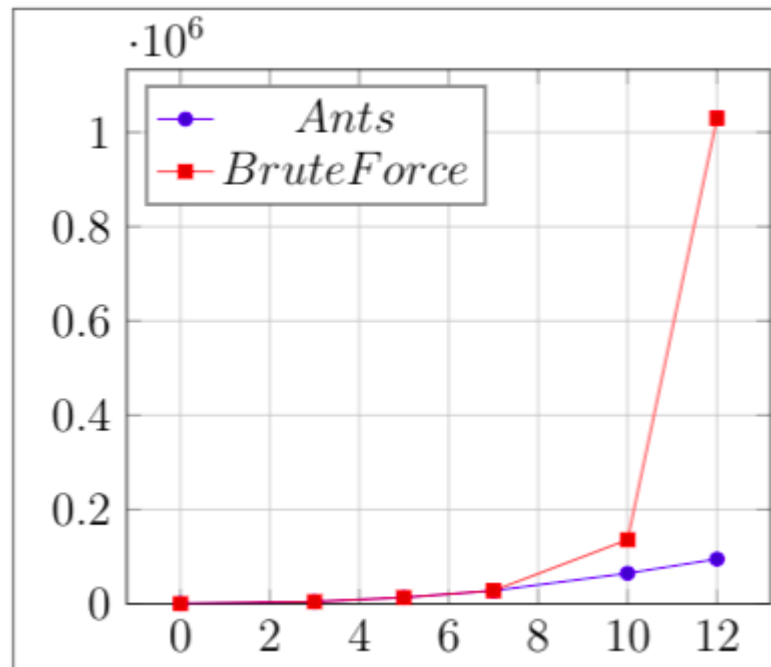


Рисунок 4.2 — Зависимость времени выполнения от размерности матрицы



## 4.6 Вывод

Как видно из графика, при небольших числах алгоритмы работают одинаково. Однако, уже на размерности  $7 \times 7$  алгоритм полного перебора начинает проигрывать, что вполне ожидаемо, т.к. алгоритмическая сложность полного перебора  $O(n!)$ .

С помощью реализации муравьиного алгоритма удалось добиться как минимум 10-кратного выигрыша на небольших размерностях. Чем больше будет размерность матрицы смежности тем больше будет выигрыш.

## Заключение

В данной лабораторной работе был изучен и реализован эвристический метод для решения задачи коммивояжера - муравьиный алгоритм. Его идея основывается на биологическом принципе поведения муравьев.

Большинство эвристических методов, к которым относится муравьиный алгоритм, не гарантируют точное решение, но являются достаточными для решения задачи за оптимальное время.

Важным этапом реализации эвристических методов является его параметризация, т.е. настройка на соответствующий тип данных, т.к. эвристические методы решают задачу не на универсальном наборе данных, а только на заранее заданном единообразном типе матриц. Например, карты смежности между городами и между странами - два разных набора данных.

Параметризация заключается в варьировании параметров, решении задачи с этими параметрами, а затем происходит сравнение полученного значения с эталонным, полученным точным, но долгим методом. В данной лабораторной работе параметризация настраивалась на трех картах, таким образом суммарная погрешность относительно эталонного решения определялась как максимум из погрешностей для каждой из карт - выбирая оптимальные настройки, нужно добиться того, чтобы худшая погрешность среди трех карт была минимальной.

С помощью реализации муравьиного алгоритма удалось добиться как минимум 10-кратного выигрыша на небольших размерностях. Чем больше будет размерность матрицы смежности тем больше будет выигрыш.

## **Список использованных источников**

1. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. — М.В. Ульянов, 2007.
2. Муравьиные алгоритмы. — Ершов Н.М., 2011.
3. Ubuntu. — [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Ubuntu>., 16.09.2021.
4. Linux. — [Электронный ресурс]. Режим доступа: <https://ru.wikipedia.org/wiki/Linux>., 16.09.2021.
5. Процессор AMD Ryzen(ТМ) 5. — [Электронный ресурс]. Режим доступа: <https://shop.lenovo.ru/product/81YM007FRU/>., 21.09.2021.

## Приложение А Параметрическая таблица исследований

В таблицах А.1 - А.5 представлены результаты параметризации.  
Здесь МА - муравьиный алгоритм, ПП - полный перебор.

Таблица А.1 — Параметрическая таблица исследований

№ карты	$\alpha$	$\rho$	Кол-во дней	МА	ПП	Погрешность
1	0	0.2	2	43	25	18
2	0	0.2	2	38	23	15
3	0	0.2	2	23	14	9
1	0	0.2	3	39	25	14
2	0	0.2	3	40	23	17
3	0	0.2	3	18	14	4
1	0	0.2	4	39	25	14
2	0	0.2	4	36	23	13
3	0	0.2	4	29	14	15
1	0	0.4	2	35	25	10
2	0	0.4	2	36	23	13
3	0	0.4	2	29	14	15
1	0	0.4	3	36	25	11
2	0	0.4	3	36	23	13
3	0	0.4	3	23	14	9
1	0	0.4	4	38	25	13
2	0	0.4	4	34	23	11
3	0	0.4	4	31	14	17
1	0	0.6	2	38	25	13
2	0	0.6	2	40	23	17
3	0	0.6	2	29	14	15
1	0	0.6	3	35	25	10
2	0	0.6	3	35	23	12
3	0	0.6	3	30	14	16

Таблица А.2 — Параметрическая таблица исследований (продолжение)

№ карты	$\alpha$	$\rho$	Кол-во дней	МА	ПП	Погрешность
1	0	0.6	4	42	25	17
2	0	0.6	4	36	23	13
3	0	0.6	4	31	14	17
1	0	0.8	2	39	25	14
2	0	0.8	2	31	23	8
3	0	0.8	2	24	14	10
1	0	0.8	3	39	25	14
2	0	0.8	3	35	23	12
3	0	0.8	3	31	14	17
1	0	0.8	4	39	25	14
2	0	0.8	4	41	23	18
3	0	0.8	4	30	14	16
1	1	0.2	2	30	25	5
2	1	0.2	2	28	23	5
3	1	0.2	2	23	14	9
1	1	0.2	3	33	25	8
2	1	0.2	3	34	23	11
3	1	0.2	3	14	14	0
1	1	0.2	4	31	25	6
2	1	0.2	4	27	23	4
3	1	0.2	4	20	14	6
1	1	0.4	2	34	25	9
2	1	0.4	2	28	23	5
3	1	0.4	2	17	14	3
1	1	0.4	3	25	25	0
2	1	0.4	3	30	23	7
3	1	0.4	3	23	14	9
1	1	0.4	4	29	25	4
2	1	0.4	4	28	23	5
3	1	0.4	4	17	14	3

Таблица А.3 — Параметрическая таблица исследований (продолжение)

№ карты	$\alpha$	$\rho$	Кол-во дней	МА	ПП	Погрешность
1	1	0.6	2	34	25	9
2	1	0.6	2	27	23	4
3	1	0.6	2	20	14	6
1	1	0.6	3	29	25	4
2	1	0.6	3	28	23	5
3	1	0.6	3	19	14	5
1	1	0.6	4	31	25	6
2	1	0.6	4	24	23	1
3	1	0.6	4	19	14	5
1	1	0.8	2	31	25	6
2	1	0.8	2	31	23	8
3	1	0.8	2	20	14	6
1	1	0.8	3	31	25	6
2	1	0.8	3	28	23	5
3	1	0.8	3	22	14	8
1	1	0.8	4	30	25	5
2	1	0.8	4	28	23	5
3	1	0.8	4	22	14	8
1	2	0.2	2	31	25	6
2	2	0.2	2	27	23	4
3	2	0.2	2	20	14	6
1	2	0.2	3	31	25	6
2	2	0.2	3	27	23	4
3	2	0.2	3	14	14	0
1	2	0.2	4	28	25	3
2	2	0.2	4	25	23	2
3	2	0.2	4	14	14	0
1	2	0.4	2	28	25	3
2	2	0.4	2	25	23	2
3	2	0.4	2	14	14	0

Таблица А.4 — Параметрическая таблица исследований (продолжение)

№ карты	$\alpha$	$\rho$	Кол-во дней	МА	ПП	Погрешность
1	2	0.4	3	29	25	4
2	2	0.4	3	24	23	1
3	2	0.4	3	14	14	0
1	2	0.4	4	28	25	3
2	2	0.4	4	28	23	5
3	2	0.4	4	14	14	0
1	2	0.6	2	30	25	5
2	2	0.6	2	28	23	5
3	2	0.6	2	16	14	2
1	2	0.6	3	28	25	3
2	2	0.6	3	25	23	2
3	2	0.6	3	14	14	0
1	2	0.6	4	29	25	4
2	2	0.6	4	26	23	3
3	2	0.6	4	14	14	0
1	2	0.8	2	32	25	7
2	2	0.8	2	26	23	3
3	2	0.8	2	14	14	0
1	2	0.8	3	32	25	7
2	2	0.8	3	24	23	1
3	2	0.8	3	14	14	0
1	2	0.8	4	25	25	0
2	2	0.8	4	26	23	3
3	2	0.8	4	14	14	0
1	3	0.2	2	25	25	0
2	3	0.2	2	27	23	4
3	3	0.2	2	14	14	0
1	3	0.2	3	28	25	3
2	3	0.2	3	24	23	1
3	3	0.2	3	14	14	0

Таблица А.5 — Параметрическая таблица исследований (продолжение)

№ карты	$\alpha$	$\rho$	Кол-во дней	МА	ПП	Погрешность
1	3	0.2	4	28	25	3
2	3	0.2	4	26	23	3
3	3	0.2	4	14	14	0
1	3	0.4	2	31	25	6
2	3	0.4	2	26	23	3
3	3	0.4	2	14	14	0
1	3	0.4	3	25	25	0
2	3	0.4	3	23	23	0
3	3	0.4	3	14	14	0
1	3	0.4	4	30	25	5
2	3	0.4	4	26	23	3
3	3	0.4	4	14	14	0
1	3	0.6	2	33	25	8
2	3	0.6	2	23	23	0
3	3	0.6	2	14	14	0
1	3	0.6	3	25	25	0
2	3	0.6	3	26	23	3
3	3	0.6	3	14	14	0
1	3	0.6	4	28	25	3
2	3	0.6	4	24	23	1
3	3	0.6	4	14	14	0
1	3	0.8	2	29	25	4
2	3	0.8	2	23	23	0
3	3	0.8	2	14	14	0
1	3	0.8	3	25	25	0
2	3	0.8	3	23	23	0
3	3	0.8	3	14	14	0
1	3	0.8	4	28	25	3
2	3	0.8	4	27	23	4
3	3	0.8	4	14	14	0