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 Костев Д.И.

Группа ИУ7-51Б

Преподаватели Волкова Л.Л.

Москва — 2021 г.

Оглавление

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;
- реализовать алгоритмы полного перебора и муравьиный алгоритм для решения задачи коммивояжёра;
- замерить и сравнить время выполнения реализаций алгоритмов;
- провести параметризацию реализации муравьиного алгоритма;
- сделать выводы на основе проделанной работы.

1 Аналитическая часть

В данном разделе представлены теоретические сведения о рассматриваемых алгоритмах. Задача коммивояжёра – поиск минимального гамильтонова пути во взвешенном графе.

1.1 Полный перебор

Пронумеруем все города от 1 до n. Задачу коммивояжера можно решить, образуя все перестановки $n-1$ целых положительных чисел. Для каждой перестановки строится соответствующий маршрут и вычисляется его стоимость. По итогам обработки всех перестановок, запоминается маршрут, который к текущему моменту имеет наименьшую стоимость.

Сложность алгоритма полного перебора составляет $O(n!)$ [**?**], где n – число городов.

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, с большей вероятностью муравьи будут включать его в свой маршрут. Моделирование такого подхода, использующего только положительную обратную связь, приводит к преждевременной сходимости к локальному оптимуму. Избежать этого можно, моделируя отрицательную обратную связь в виде испарения феромона. При этом если феромон испаряется быстро, то это приводит к потере памяти колонии и забыванию хороших решений, с другой стороны, большое время испарения может привести к получению устойчивого локально оптимального решения. Теперь, с учетом особенностей за-

дачи коммивояжера, возможно описать локальные правила поведения муравьев при выборе пути.

- Муравьи имеют собственную «память». Поскольку каждый город может быть посещеи 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}:

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

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

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

где α, β — параметры, задающие веса следа феромона, при $\alpha = 0$ алгоритм вырождается в жадный алгоритм (будет выбран ближайший город). Заметим, что выбор города является вероятностным, правило 1.2 лишь определяет ширину зоны города j; в общую зону всех городов $J_{i,k}$;, генерируется случайное действительное число от 0 до 1, 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:

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

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

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

где m — количество муравьев в колонии. Обычно оно совпадает с n.

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.

Во время моделирования испарения феромона его количество на ребре может достигнуть нулевого значения, в результате чего вероятность продвижения по этому ребру всегда будет нулевой. Во избежание этой ситуации при обновлении количества феромона стоит сравнивать его с каким-либо положительным минимальным значением и не допускать выхода за его границу.

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

1.3 Вывод из аналитической части

В данном разделе были рассмотрены особенности алгоритмов решения задачи коммивояжёра.

2 Конструкторская часть

В данном разделе представлены схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2.1 - 2.3 приведены схемы алгоритмов решения задачи коммивояжера.

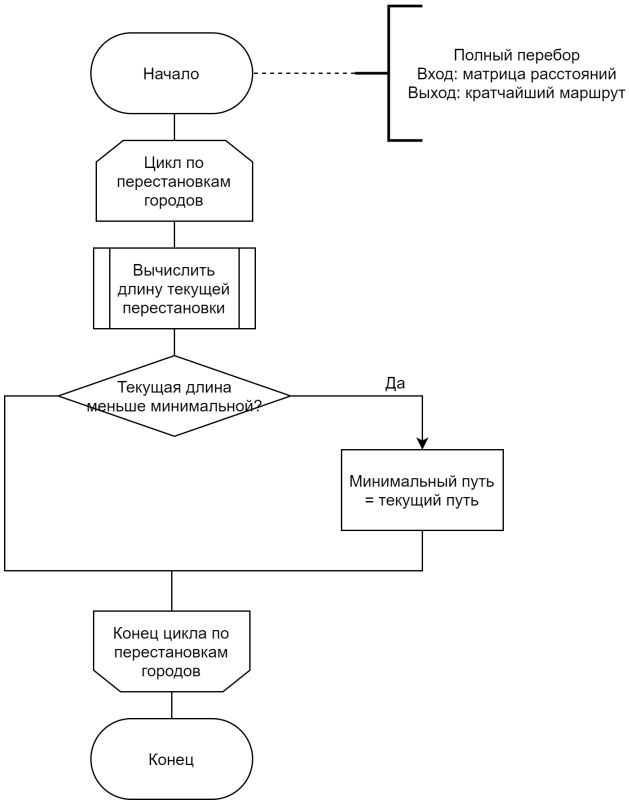

Рис. 2.1: Схема алгоритма полного перебора

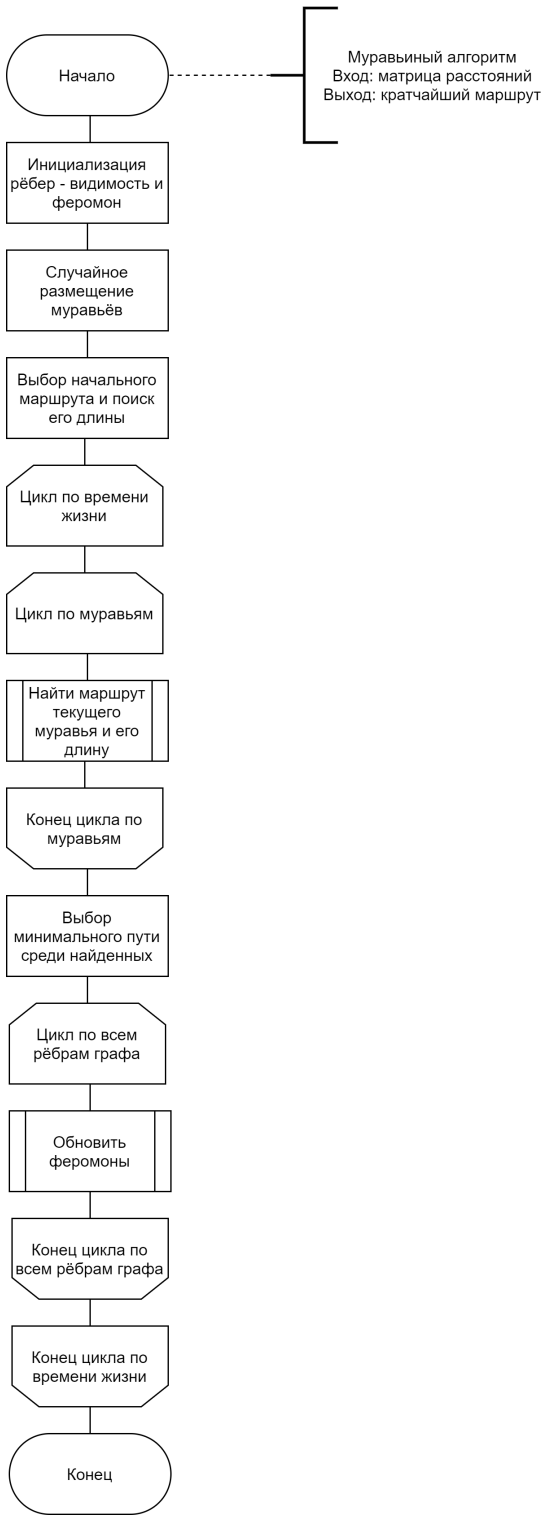

Рис. 2.2: Схема муравьиного алгоритма

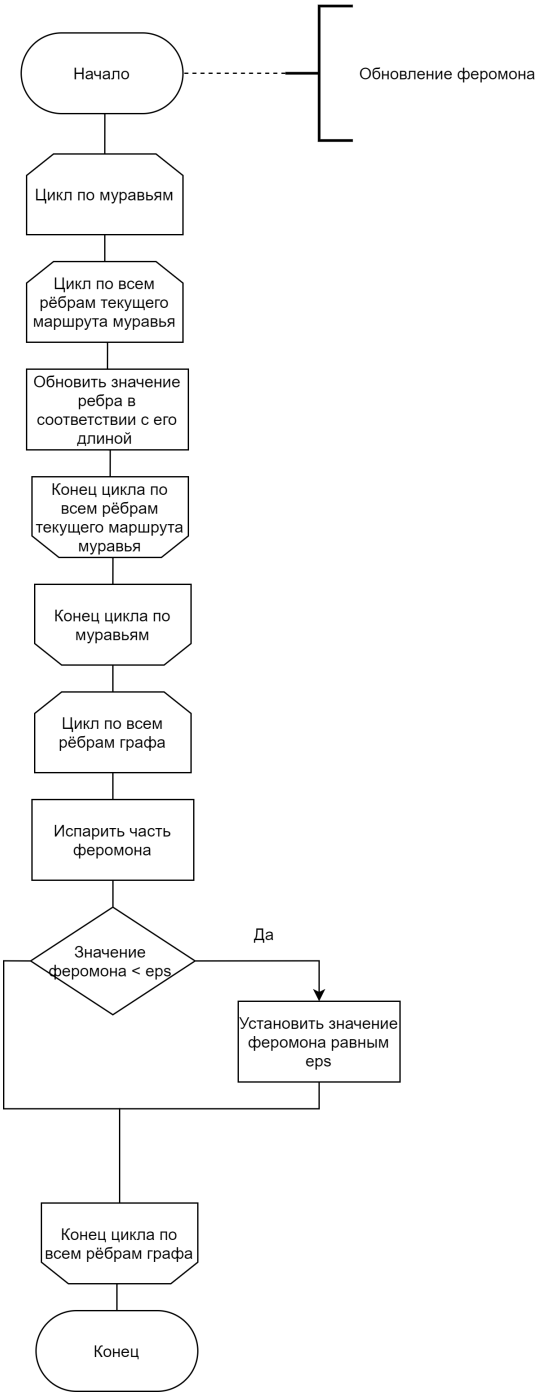

Рис. 2.3: Схема алгоритма обновления феромона

2.2 Вывод из конструкторской части

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 Требования к ПО

К программе предъявляется ряд требований:

- на вход подается матрица расстояний графа;
- на выходе – кратчайший путь.

3.2 Средства реализации

В качестве языка программирования был выбран C.

3.3 Реализация алгоритмов

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

Листинг 3.1: Алгоритм полного перебора

```
1
2  int nperm(size_t *seq, size_t n)
3  {
4    size_t i = n;
5    do {
6      if (i < 2)
7        return EXIT_FAILURE;
8      i--;
9    } while (seq[i - 1] > seq[i]);
10
11   size_t j;
12   for (j = n; j > i && seq[i - 1] > seq[--j];) {}
13
14   swp(seq + i - 1, seq + j);
15
16   for (j = n; i < --j; i++)
17     swp(seq + i, seq + j);
18
19   return EXIT_SUCCESS;
20 }
21
22 int cnt_dist(mtx_t *mtx, size_t *route)
23 {
24   int dst = 0, td;
25
26   for (size_t i = 0; i < mtx->n - 1; i++)
27   {
28     td = mtx->data[route[i]][route[i + 1]];
29     if (td < 0)
30       return -1;
31
32     dst += td;
```

```
33    }
34
35    td = mtx->data[0][mtx->n - 1];
36    if (td < 0)
37      return -1;
38
39    dst += td;
40
41    return dst;
42  }
43
44  int commis_voyageur_bf(mtx_t *mtx, size_t **route)
45  {
46    *route = malloc(mtx->n * sizeof(size_t));
47    size_t *tr = malloc(mtx->n * sizeof(size_t));
48
49    for (size_t i = 0; i < mtx->n; i++)
50      tr[i] = i;
51
52    bool found = false;
53    int dst, m;
54
55    do {
56      dst = cnt_dist(mtx, tr);
57
58      if (dst >= 0 && (!found || dst < m))
59      {
60        found = true;
61        m = dst;
62        memcpy(*route, tr, mtx->n * sizeof(mtx->n));
63      }
64    } while (nperm(tr, mtx->n) == EXIT_SUCCESS);
65
```

```
66    free(tr);
67
68    if (!found)
69    {
70      free(*route);
71      return -1;
72    }
73
74    return m;
75  }
```

Листинг 3.2: Муравьиный алгоритм

```
1
2   int commis_voyageur_ant(mtx_t *mtx, size_t **route, antp_t *param)
3   {
4     mtxf_t *dist = create_mtxf(mtx->n, mtx->m);
5     mtxf_t *fero = create_mtxf(mtx->n, mtx->m);
6
7     ant_t *colony = malloc(mtx->n * sizeof(ant_t));
8     float *p = malloc((mtx->n - 1) * sizeof(float));
9     *route = malloc(mtx->n * sizeof(size_t));
10
11    int L = 0;
12
13    for (size_t i = 0; i < mtx->n; i++)
14    {
15      for (size_t j = 0; j < mtx->m; j++)
16      {
17        dist->data[i][j] = 1.0 / (float)mtx->data[i][j];
18
19        if (mtx->data[i][j] > 0)
20        {
21          fero->data[i][j] = 1.0;
```

```
22          L += mtx->data[i][j];
23        }
24        else
25          fero->data[i][j] = 0.0;
26      }
27
28      colony[i].unvis = malloc((mtx->n - 1) * sizeof(size_t));
29      colony[i].route = malloc(mtx->n * sizeof(size_t));
30    }
31
32    L /= 2;
33
34    //start alg
35
36    bool found = false;
37
38    for (size_t i = 0; i < param->ANT_TIME; i++)
39    {
40      for (size_t ant = 0; ant < mtx->n; ant++)
41      {
42        colony[ant].route[0] = colony[ant].cur = ant;
43        size_t c = 1;
44        for (size_t j = 0; j < mtx->n; j++, c++)
45        {
46          if (ant == j)
47            c--;
48          else
49            colony[ant].unvis[c - 1] = j;
50        }
51
52        colony[ant].unv_len = mtx->n - 1;
53        size_t route_cnt = 1;
54
```

```
55        while (colony[ant].unv_len)
56        {
57          float s = 0.0;
58          for (size_t q = 0; q < colony[ant].unv_len; q++)
59          {
60            size_t ai = colony[ant].cur, aj = colony[ant].unvis[q];
61            p[q] = powf(fero->data[ai][aj], param->ALPHA) + powf(dist->data[ai
                  ][aj], param->BETA);
62            s += p[q];
63          }
64          float r = (float)rand() / (float)RAND_MAX;
65          p[0] /= s;
66          for (size_t q = 1; q < colony[ant].unv_len; q++)
67          {
68            p[q] /= s;
69            p[q] += p[q - 1];
70          }
71
72          size_t q = 0;
73          bool select = false;
74          while (q < colony[ant].unv_len && !select)
75          {
76            if (r < p[q])
77              select = true;
78            q++;
79          }
80
81          colony[ant].route[route_cnt] = colony[ant].cur = colony[ant].unvis[q
                - 1];
82          route_cnt++;
83
84
85          size_t k;
```

```
86          for (k = 0; k < colony[ant].unv_len; k++)
87            if (k == q - 1)
88              break;
89
90          for (size_t e = k; e < colony[ant].unv_len - 1; e++)
91            colony[ant].unvis[e] = colony[ant].unvis[e + 1];
92
93          colony[ant].unv_len--;
94        }
95
96        int dst = cnt_dist(mtx, colony[ant].route);
97        if (dst >= 0 && (!found || dst < L))
98        {
99          found = true;
100         L = dst;
101         memcpy(*route, colony[ant].route, mtx->n * sizeof(size_t));
102       }
103
104       colony[ant].dst = dst;
105     }
106
107     for (size_t i = 0; i < mtx->n; i++)
108       for (size_t j = 0; j < mtx->m; j++)
109         if (fero->data[i][j] > 0)
110         {
111           fero->data[i][j] *= 1.0 - param->RO;
112           if (fero->data[i][j] < 0.1)
113             fero->data[i][j] = 0.1;
114         }
115
116     int from, to;
117     for (size_t ant = 0; ant < mtx->n; ant++)
118     {
```

```
119         for (size_t r = 0; r < mtx->n - 1; r++)
120         {
121           from = colony[ant].route[r];
122           to = colony[ant].route[r + 1];
123 
124           fero->data[from][to] += param->Q / colony[ant].dst;
125         }
126 
127         fero->data[mtx->n - 1][0] += param->Q / colony[ant].dst;
128       }
129     }
130 
131     destroy_mtxf(dist);
132     destroy_mtxf(fero);
133     for (size_t i = 0; i < mtx->n; i++)
134     {
135       free(colony[i].unvis);
136       free(colony[i].route);
137     }
138     free(colony);
139     free(p);
140 
141     if (!found)
142       return -1;
143 
144     return L;
145 }
```

3.4 Тестирование

При оценивании успешности выполнения реализации алгоритмов стоит отметить, что муравьиный алгоритм не гарантирует нахождения глобального оптимума, поэтому длины полученного минимального маршрута у муравьиного алгоритма и полного перебора могут отличаться. В приведённых тестах значения длины пути у обоих алгоритмов совпали.

В таблице 3.1 приведены тестовые данные.

Таблица 3.1: Тестирование реализаций алгоритмов

Матрица смежности	Ожидаемый результат	Полученный результат
$\begin{bmatrix} 0 & 3 & 1 & 6 & 8 \\ 3 & 0 & 4 & 1 & 0 \\ 1 & 4 & 0 & 5 & 0 \\ 6 & 1 & 5 & 6 & 1 \\ 8 & 0 & 0 & 1 & 0 \end{bmatrix}$	10	10
$\begin{bmatrix} 0 & 10 & 15 & 20 \\ 10 & 0 & 35 & 25 \\ 15 & 35 & 0 & 30 \\ 20 & 25 & 30 & 0 \end{bmatrix}$	70	70

Все тесты пройдены успешно.

3.5 Вывод из технологической части

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

4 | Исследовательская часть

В данном разделе приведен анализ характеристик разработанного ПО.

4.1 Время выполнения алгоритмов

Время выполнения алгоритма замерялось с помощью стандартной функции clock(). Для каждого размера матрицы замер проводился 10 раз, затем усреднялся. Полученные результаты приведены в таблице 4.1. Время указано в мс.

Таблица 4.1: Сравнение времени исполнения алгоритмов решения задачи коммивояжера.

Размер графа	Полный перебор	Муравьиный алгоритм
3	0	10
4	0	20
5	0	40
6	0	30
7	0	30
8	20	50
9	200	70
10	2099	90
11	24272	110

4.2 Автоматическая параметризация

В ходе параметризации были проанализированы следующие значения параметров алгоритма: α изменяется от 0 до 6, β вычисляется как разность $6 - \alpha$; время жизни колонии от 10 до 190, коэффициент испарения от 0 до 1. Исследуется разность длины найденного пути, найденного муравьиным алгоритмом, и длины оптимального пути.

В таблицах 4.2 - 4.5 приведены выборка результатов параметризации для трёх матриц расстояний размером 10×10. Полным перебором был посчитан оптимальный путь – он составил 226 для первой матрицы, 249 и 166 для второй и тертьей. В столбцах разность 1 – 3 записаны разности между результатом алгоритма и оптимальным значение пути для каждой матрицы.

Таблица 4.2: Выборка из результатов параметризации для матриц размером 10×10.

α	Время	ρ	Разность 1	Разность 2	Разность 3
3.2	130	0.3	3	25	18
3.2	130	0.6	0	25	15
3.2	130	0.9	15	0	0
3.2	190	0.1	0	4	18
3.2	190	0.3	7	19	26
3.2	190	0.6	26	27	18
3.2	190	0.9	15	12	23
3.6	10	0.1	64	63	62
3.6	10	0.3	30	61	44
3.6	10	0.6	32	32	19
3.6	10	0.9	5	10	2
3.6	70	0.1	3	4	16
3.6	70	0.3	7	0	2
3.6	70	0.6	10	0	13
3.6	70	0.9	7	0	2
3.6	130	0.1	3	6	10
3.6	130	0.3	7	9	3
3.6	130	0.6	0	4	8
3.6	130	0.9	3	0	21
3.6	190	0.1	0	0	0
3.6	190	0.3	3	0	13
3.6	190	0.6	0	0	16
3.6	190	0.9	0	6	3
4.0	10	0.1	31	64	60
4.0	10	0.3	33	19	36
4.0	10	0.6	7	21	18
4.0	10	0.9	7	27	8
4.0	70	0.1	5	0	13

Таблица 4.3: Выборка из результатов параметризации для матриц размером 10×10.

α	Время	ρ	Разность 1	Разность 2	Разность 3
4.0	70	0.3	3	0	10
4.0	70	0.6	3	4	18
4.0	70	0.9	0	0	17
4.0	130	0.1	0	0	8
4.0	130	0.3	0	0	25
4.0	130	0.6	0	0	3
4.0	130	0.9	0	0	1
4.0	190	0.1	0	0	26
4.0	190	0.3	0	0	8
4.0	190	0.6	0	0	2
4.0	190	0.9	0	0	13
4.4	10	0.1	57	46	82
4.4	10	0.3	34	4	13
4.4	10	0.6	23	0	22
4.4	10	0.9	10	32	10
4.4	70	0.1	0	14	18
4.4	70	0.3	7	0	21
4.4	70	0.6	0	0	16
4.4	70	0.9	0	0	10
4.4	130	0.1	0	9	18
4.4	130	0.3	0	0	16
4.4	130	0.6	0	0	15
4.4	130	0.9	0	4	19
4.4	190	0.1	3	0	17
4.4	190	0.3	0	0	18
4.4	190	0.6	0	0	3
4.4	190	0.9	0	0	14
4.8	10	0.1	27	53	69

Таблица 4.4: Выборка из результатов параметризации для матриц размером 10×10.

α	Время	ρ	Разность 1	Разность 2	Разность 3
4.8	10	0.3	21	35	25
4.8	10	0.6	30	31	8
4.8	10	0.9	0	0	23
4.8	70	0.1	3	0	22
4.8	70	0.3	0	0	20
4.8	70	0.6	3	6	2
4.8	70	0.9	3	10	3
4.8	130	0.1	7	4	15
4.8	130	0.3	0	4	13
4.8	130	0.6	0	6	19
4.8	130	0.9	0	9	3
4.8	190	0.1	4	0	18
4.8	190	0.3	0	4	10
4.8	190	0.6	0	0	5
4.8	190	0.9	0	0	15
5.2	10	0.1	44	19	60
5.2	10	0.3	35	27	3
5.2	10	0.6	49	49	39
5.2	10	0.9	5	35	38
5.2	70	0.1	15	19	13
5.2	70	0.3	10	14	16
5.2	70	0.6	12	10	21
5.2	70	0.9	12	28	0
5.2	130	0.1	0	9	16
5.2	130	0.3	0	27	6
5.2	130	0.6	7	6	15
5.2	130	0.9	10	20	2
5.2	190	0.1	0	9	17

Таблица 4.5: Выборка из результатов параметризации для матриц размером 10×10.

α	Время	ρ	Разность 1	Разность 2	Разность 3
5.2	190	0.3	0	4	15
5.2	190	0.6	7	23	2
5.2	190	0.9	4	13	0
5.6	10	0.1	52	69	62
5.6	10	0.3	58	63	86
5.6	10	0.6	66	60	46
5.6	10	0.9	88	57	22
5.6	70	0.1	19	47	21
5.6	70	0.3	27	56	31
5.6	70	0.6	31	36	25
5.6	70	0.9	23	42	49

4.3 Вывод из исследовательской части

При небольших размерах графа (от 3 до 8) алгоритм полного перебора выигрывает по времени у муравьиного. 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 220 раз быстрее.

При параметризации в качестве тестовых данных использовались матрицы расстояний с одинаковым разбросом длин рёбер. Наиболее стабильные результаты автоматической параметризации получаются при наборе $\alpha = 3.6..4.8$, $\beta = 6 - \alpha$, $\rho = 0.6$. При времени жизни колонии, превышающем число городов, результат является более точным.

Тогда можно сделать вывод, что при числе городов, меньшем девяти стоит использовать алгоритм полного перебора для получения точного результата. Для иного числа городов стоит использовать муравьиный алгоритм с выявленными параметрами. Однако нужно понимать, что результат работы муравьиного алгоритма не всегда является оптимальным маршрутом.

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:

- решена задача коммивояжера при помощи алгоритма полного перебора и муравьиного алгоритма;
- реализованы алгоритмы полного перебора и муравьиный алгоритм для решения задачи коммивояжёра;
- замерено и сравнено время выполнения реализаций алгоритмов;
- проведена параметризация реализации муравьиного алгоритма;
- сделаны выводы на основе проделанной работы.