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»

ОТЧЕТ

По лабораторной работе №6

По курсу: «Анализ алгоритмов»

Тема: «Муравьиный алгоритм и полный перебор»

Студент: Ле Ни Куанг

Группа: ИУ7и-56Б

Преподаватель: Волкова Л. Л.

Строганов Ю. В.

Москва

2021

Оглавление

Введение

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

Целью работы: провести сравнительный анализ метода полного перебора и эвристического метода на базе муравьиного алгоритма.

Задачи работы:

- реализовать метод полного перебора и метод на базе муравьиного алгоритма для решения задачи коммивояжера с возвращением последнего в город, с которого он начал обход;
- провести параметризацию второго метода для выбранного класса задач, т.е. определить такие комбинации параметров или их диапазонов, при которых метод дает наилучшие результаты на выбранном(ых) классе(ах) задач.

1 Аналитический раздел

В данном разделе будет приведено описание конвейерной обработки и параллельных вычислений.

1.1 Задача коммивояжера

Задача коммивояжера - одна из самых известных задач комбинаторной оптимизации, заключающаяся в поиске самого выгодного маршрута, проходящего через указанные города хотя бы по одному разу с последующим возвратом в исходный город. В условиях задачи указываются критерий выгодности маршрута (кратчайший, самый дешевый, совокупный критерий и тому подобное) и соответствующие матрицы расстояний, стоимости и тому подобного. Как правило, указывается, что маршрут должен проходить через каждый город только один раз - в таком случае выбор осуществляется среди гамильтоновых циклов.

1.2 Полный перебор

Алгоритм полного перебора для решения задачи коммивояжера является наиболее прямым решением, он пробует все перестановки (упорядоченные комбинации) и определяет, какой из них самый дешевый. Этот подход гарантирует точное решение задачи. Но сложность алгоритма $O(n!)$, факториал количества городов, поэтому это решение становится непрактичным даже для 20 городов.

1.3 Муравьиный алгоритм

Муравьиные алгоритмы представляют собой новый перспективный метод решения задач оптимизации, в основе которого лежит моделирование поведения колонии муравьев. Колония представляет собой систему с очень простыми правилами автономного поведения особей.

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, 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 дугой 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 сеть список уже посещенных городов — список запретов. Обозначим через $J_{i,k}$ список городов, которые необходимо пройти 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 пропорциональной расстоянию между соответствующими городами $\eta_{i,j} = 1/D_{i,j}$.

- Муравьи обладают «обонянием» – могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьев. Обозначим количество феромона на ребре (i, j) в момент времени t через $\tau_{i,j}(t)$.

Вероятность перехода из вершины i в вершину j определяется по следующей формуле 1.1

$$p_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1.1}$$

где $\tau_{i,j}$– количество феромонов на ребре i до j;
$\eta_{i,j}$– эвристическое расстояние от i до j;
α– параметр влияния феромона;
β– параметр влияния расстояния.

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 , $L_k(t)$ - длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути. Тогда откладываемое количество феромона может быть задано в виде:

$$\Delta\tau_{i,j}^{k} = \begin{cases} Q/L_k(t), & (i,j) \in T_k(t) \\ 0, & \text{иначе} \end{cases} \tag{1.2}$$

где Q - количество феромона, переносимого муравьем;
Тогда

$$\Delta\tau_{ij}(t) = \sum_{k=1}^{m} \Delta\tau_{ij,k}(t) \tag{1.3}$$

Сложность данного алгоритма определяется непосредственно из приведенного выше текста – $O(t_{max} \cdot m \cdot n^2)$, таким образом, сложность зависит от времени жизни колонии, количества городов и количества муравьев в колонии.

1.4 Вывод

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

2 Конструкторский раздел

В данном разделе представлены схемы рассматриваемых алгоритмов.

2.1 Разработка алгоритмов

На рисунках 2.1 и 2.2 приведены схемы алгоритмов решения задачи коммивояжера перебором и муравьиным алгоритмом соответственно.

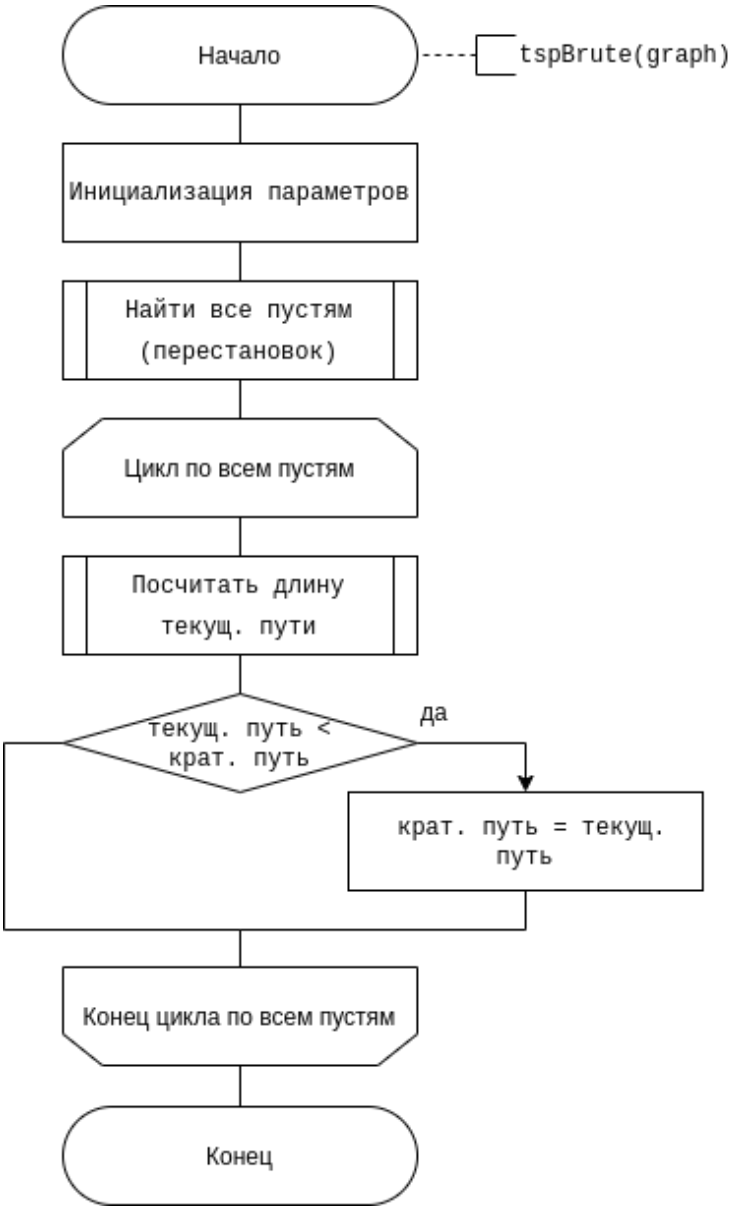

Рис. 2.1: Схема алгоритма полного перебора решения задачи коммивояжера

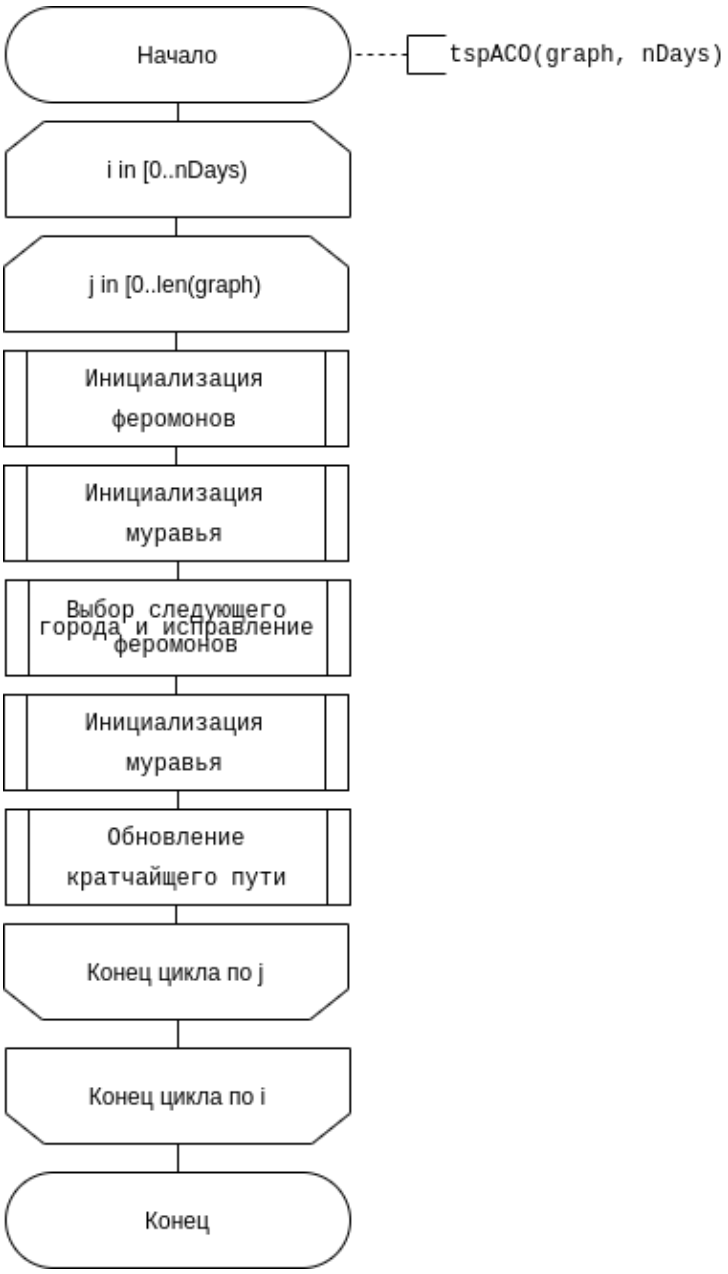

Рис. 2.2: Схема муравьиного алгоритма решения задачи коммивояжера

2.2 Вывод

В данном разделе были рассмотрены схемы алгоритмов для решения задачи коммивояжера.

3 Технологический раздел

В данном разделе будет приведены требования к программу и листинг кода.

3.1 Требования к программному обеспечению

Программа должна принимать в качестве входны данны граф, представленный в виде матрицы смежности.

Результатом программы являются решение задачи коммивояжера для текущего графа.

3.2 Средства реализации

Язык программирования: Go

Редактор: VS Code

Go - это новый мощный язык программирования, который я учил недавно, поэтому я хочу использовать его на практике.

3.3 Листинг кода

В листингах ниже представлен код программа.

Листинг 3.1: Файл main.go

```
package main

import (
    "math/rand"
    "time"
)

func main() {
    rand.Seed(int64(time.Now().Nanosecond()))
    // logOutput(10)
    nList := []int{2, 3, 4, 5, 6, 7, 8, 9, 10}
    Benchmark(nList)
}
```

Листинг 3.2: Файл ant.go

```
package main

import (
```

```
    "math"
    "math/rand"
)

const (
    Alpha  = 1.8
    Beta   = 1.0
    Tau0   = 1.0
    Rho    = 0.5
    Q      = 20.0
    TauMin = 0.1
)

type Ant struct {
    env        *Env
    visited    [][]int
    isVisited  [][]bool
    pos        int
    route      []int
}

// graph + pheromone matrix + params
type Env struct {
    g             [][]int
    ph            [][]float64
    a, b, q, rho float64
}

func tspBrute(g [][]int) (int, []int) {
    n := len(g)
    min := math.MaxInt64
    rMin := make([]int, n)

    routes := [][]int{}
    r := []int{}
    getRoutes(0, g, r, &routes)

    for i := 0; i < len(routes); i++ {
        curRoute := routes[i]
        cur := calcLenRoute(g, curRoute)

        if cur < min {
            min = cur
            rMin = curRoute
        }
    }

    return min, rMin
```

```
}

func tspACO(e *Env, d int) (int, []int) {
    n := len(e.g)
    min := make([]int, n)
    rMin := make([][]int, n)

    for i := 0; i < d; i++ {
        for j := 0; j < n; j++ {
            a := e.createAnt(j)
            a.moveAnt()
            cur := calcLenRoute(e.g, a.route)

            if (cur < min[j]) || (min[j] == 0) {
                min[j] = cur
                rMin[j] = a.route
            }
        }
    }

    m := min[0]
    r := rMin[0]
    for i, v := range min {
        if v < m {
            m = v
            r = rMin[i]
        }
    }

    return m, r
}

func calcLenRoute(g [][]int, route []int) int {
    d := 0
    l := len(g) - 1

    for j := 0; j < l; j++ {
        d += g[route[j]][route[j+1]]
    }
    d += g[route[l]][route[0]]
    return d
}

func (e *Env) createAnt(pos int) *Ant {
    n := len(e.g)
    a := Ant{
        env:       e,
        visited:   make([][]int, n),
```

```
100         isVisited: make([][]bool, n),
101         pos:       pos,
102         route:     []int{pos},
103     }
104
105     for i := 0; i < n; i++ {
106         a.visited[i] = make([]int, n)
107         for j := 0; j < len(e.g[i]); j++ {
108             a.visited[i][j] = e.g[i][j]
109         }
110     }
111
112     for i := range a.isVisited {
113         a.isVisited[i] = make([]bool, n)
114     }
115
116     return &a
117 }
118
119 func createEnv(g [][]int) *Env {
120     e := Env{
121         g:   g,
122         ph:  make([][]float64, len(g), len(g)),
123         a:   Alpha,
124         b:   Beta,
125         q:   Q,
126         rho: Rho,
127     }
128
129     initPheromone(e.ph)
130
131     return &e
132 }
133
134 func initPheromone(ph [][]float64) {
135     for i := 0; i < len(ph); i++ {
136         ph[i] = make([]float64, len(ph))
137         for j := range ph[i] {
138             ph[i][j] = Tau0
139         }
140     }
141 }
142
143 func (a *Ant) moveAnt() {
144     l := len(a.env.g) - 1
145     for i := 0; i < l; i++ {
146         prob := a.getProb()
147         way := nextCity(prob)
```

```
148         a.follow(way)
149         a.route = append(a.route, way)
150         if way == -1 {
151             break
152         }
153         a.updatePheromone()
154     }
155 }
156
157 func (a *Ant) getProb() []float64 {
158     p := []float64{}
159     var sum float64
160
161     for i, l := range a.visited[a.pos] {
162         if l != 0 {
163             d := math.Pow((float64(1)/float64(l)), a.env.a) *
                    math.Pow(a.env.ph[a.pos][i], a.env.b)
164             p = append(p, d)
165             sum += d
166         } else {
167             p = append(p, 0)
168         }
169     }
170
171     for _, l := range p {
172         l /= sum
173     }
174
175     return p
176 }
177
178 func (a *Ant) updatePheromone() {
179     delta := 0.0
180
181     for i := 0; i < len(a.env.ph); i++ {
182         for j, ph := range a.env.ph[i] {
183             if a.env.g[i][j] != 0 {
184                 if a.isVisited[i][j] {
185                     delta = a.env.q / float64(a.env.g[i][j])
186                 } else {
187                     delta = 0
188                 }
189                 a.env.ph[i][j] = (1 - a.env.rho) * (ph + delta)
190             }
191
192             if a.env.ph[i][j] <= 0 {
193                 a.env.ph[i][j] = TauMin
194             }
```

```
195         }
196     }
197 }
198 
199 func (a *Ant) follow(path int) {
200     for i := range a.visited {
201         a.visited[i][a.pos] = 0
202     }
203     a.isVisited[a.pos][path] = true
204     a.pos = path
205 }
206 
207 func nextCity(p []float64) int {
208     var sum float64
209 
210     for _, j := range p {
211         sum += j
212     }
213 
214     r := rand.Float64() * sum
215     sum = 0.0
216 
217     for i, v := range p {
218         if r >= sum && r < sum+v {
219             return i
220         }
221         sum += v
222     }
223 
224     return len(p) - 1
225 }
226 
227 func getRoutes(pos int, w [][]int, path []int, rts *[][]int) {
228     path = append(path, pos)
229 
230     if len(path) < len(w) {
231         for i := 0; i < len(w); i++ {
232             if !isExist(path, i) {
233                 getRoutes(i, w, path, rts)
234             }
235         }
236     } else {
237         *rts = append(*rts, path)
238     }
239 }
240 
241 func isExist(a []int, v int) bool {
242     for _, val := range a {
```

```
243         if v == val {
244             return true
245         }
246     }
247
248     return false
249 }
```

Листинг 3.3: Файл utils.go

```
1  package main
2
3  import (
4      "fmt"
5      "math/rand"
6      "strings"
7      "time"
8  )
9
10 const (
11     MaxEdgeLen = 9
12     NDays      = 50
13 )
14
15 func genGraph(n int) [][]int {
16     graph := make([][]int, n)
17     for i := 0; i < n; i++ {
18         graph[i] = make([]int, n)
19         for j := 0; j < n; j++ {
20             if i != j {
21                 graph[i][j] = rand.Intn(MaxEdgeLen) + 1
22             }
23         }
24     }
25     return graph
26 }
27
28 func printGraph(g [][]int) {
29     for i := range g {
30         fmt.Println(g[i])
31     }
32     println()
33 }
34
35 func measureTime(f func()) time.Duration {
36     start := time.Now()
37     f()
38     return time.Since(start)
39 }
```

```
40
41 func printResult(cost int, route []int) {
42
43     fmt.Printf("%4d|%v\n", cost, route)
44 }
45
46 func logOutput(n int) {
47     g := genGraph(n)
48     env := createEnv(g)
49     printGraph(g)
50
51     println("BRUTE")
52     cost, route := tspBrute(env.g)
53     fmt.Printf("%d\t%v\n", cost, route)
54     println("==========================\n")
55
56     fmt.Printf("ACO\tn ants * %d days\n", NDays)
57     println("\t{alpha, beta, rho, tau0, taumin, q}\n")
58
59     env.a = 1
60     env.b = 0
61     env.rho = 0.1
62     for i := 0; i < 6; i++ {
63         for j := 0; j < 3; j++ {
64             env.rho += 0.3
65             initPheromone(env.ph)
66             cost, route = tspACO(env, NDays)
67             fmt.Printf("%d\t%v\t", cost, route)
68             fmt.Printf("{%.1f, %.1f, %.1f, %.1f, %.1f, %.1f}\n",
69                 env.a, env.b, env.rho, Tau0, TauMin, Q)
70         }
71         env.a -= 0.2
72         env.b += 0.2
73     }
74 }
75
76 func Benchmark(nList []int) {
77     aco := []time.Duration{}
78     brute := []time.Duration{}
79     for _, n := range nList {
80         g := genGraph(n)
81         env := createEnv(g)
82
83         aco = append(aco, measureTime(func() {
84             tspACO(env, 100)
85         }))
86
87         brute = append(brute, measureTime(func() {
```

```
88              tspBrute(g)
89          }))
90      }
91
92      outBenchmark(nList, aco, brute)
93  }
94
95  func outBenchmark(nList []int, a []time.Duration, b []time.Duration) {
96      fmt.Printf("␣N|%13v|%13v\n", "ACO", "BRUTE")
97      fmt.Printf("%30v\n", strings.Repeat("-", 30))
98      for i, v := range nList {
99          fmt.Printf("%2v|%13v|%13v\n", v, a[i], b[i])
100     }
101     fmt.Printf("%30v\n\n", strings.Repeat("-", 30))
102
103     // to CSV format
104     println("n,aco,brute")
105     for i, v := range nList {
106         fmt.Printf("%d,%d,%d\n", v,
107             a[i].Round(time.Microsecond).Microseconds(),
108             b[i].Round(time.Microsecond).Microseconds())
109     }
110 }
```

3.4 Вывод

В этом разделе было рассмотрено требования к программу и кода программы.

4 Экспериментальный раздел

В данном разделе будет приведены пример работы программы и сравнение времени работы программы.

4.1 Примеры работы

На рисунке 4.2 приведен пример работы программы.

```
→ go run *.go
 N|          ACO|        BRUTE
------------------------------
 2|    137.244µs|        652ns
 3|    387.956µs|        873ns
 4|    794.909µs|      3.783µs
 5|   1.697386ms|      7.778µs
 6|   2.877174ms|     31.754µs
 7|   4.510631ms|    160.892µs
 8|   7.177821ms|   1.016292ms
 9|  10.362828ms|  14.296797ms
10|   14.94792ms| 133.040387ms
------------------------------

n,aco,brute
2,137,1
3,388,1
4,795,4
5,1697,8
6,2877,32
7,4511,161
8,7178,1016
9,10363,14297
10,14948,133040
```

Рис. 4.1: Примеры работы программы

```
→ go run *.go
[0 8 1 5 7 8 1 6 7 1]
[1 0 7 6 6 2 2 1 7 6]
[5 2 0 6 1 7 6 8 1 9]
[6 3 4 0 7 8 3 2 5 8]
[6 4 9 5 0 8 1 4 6 6]
[5 9 2 3 7 0 9 7 8 3]
[3 4 3 8 8 9 0 8 6 5]
[2 3 3 9 6 6 5 0 6 8]
[4 9 7 2 2 5 8 3 0 5]
[3 5 8 3 1 6 7 6 6 0]

BRUTE
18      [0 9 4 6 1 5 2 8 3 7]
=========================

ACO     n ants * 50 days
        {alpha, beta, rho, tau0, taumin, q}

18      [9 4 6 1 5 2 8 3 7 0]   {1.0, 0.0, 0.4, 1.0, 0.1, 20.0}
25      [0 6 2 8 4 3 7 1 5 9]   {1.0, 0.0, 0.7, 1.0, 0.1, 20.0}
18      [1 5 2 8 3 7 0 9 4 6]   {1.0, 0.0, 1.0, 1.0, 0.1, 20.0}
23      [0 9 4 6 5 2 8 3 7 1]   {0.8, 0.2, 1.3, 1.0, 0.1, 20.0}
18      [3 7 0 9 4 6 1 5 2 8]   {0.8, 0.2, 1.6, 1.0, 0.1, 20.0}
25      [1 5 2 8 9 4 6 0 3 7]   {0.8, 0.2, 1.9, 1.0, 0.1, 20.0}
22      [5 9 4 6 0 2 8 3 1 7]   {0.6, 0.4, 2.2, 1.0, 0.1, 20.0}
25      [5 9 4 6 2 8 3 7 0 1]   {0.6, 0.4, 2.5, 1.0, 0.1, 20.0}
23      [7 0 9 4 6 2 8 3 1 5]   {0.6, 0.4, 2.8, 1.0, 0.1, 20.0}
23      [2 8 3 7 6 0 9 4 1 5]   {0.4, 0.6, 3.1, 1.0, 0.1, 20.0}
28      [0 6 2 8 4 9 3 7 1 5]   {0.4, 0.6, 3.4, 1.0, 0.1, 20.0}
26      [8 5 9 4 6 0 3 1 7 2]   {0.4, 0.6, 3.7, 1.0, 0.1, 20.0}
23      [5 3 8 7 0 9 4 6 2 1]   {0.2, 0.8, 4.0, 1.0, 0.1, 20.0}
31      [5 9 1 0 2 8 3 7 4 6]   {0.2, 0.8, 4.3, 1.0, 0.1, 20.0}
29      [5 2 8 7 1 6 0 9 4 3]   {0.2, 0.8, 4.6, 1.0, 0.1, 20.0}
32      [0 9 7 8 4 6 2 1 5 3]   {0.0, 1.0, 4.9, 1.0, 0.1, 20.0}
25      [3 1 7 0 5 9 4 6 2 8]   {0.0, 1.0, 5.2, 1.0, 0.1, 20.0}
31      [7 2 8 9 4 6 0 3 1 5]   {0.0, 1.0, 5.5, 1.0, 0.1, 20.0}
```

Рис. 4.2: Параметризация для матрицы размером 10 × 10

4.2 Сравнение времени работы

Операционная система - Ubuntu 20.04.1 LTS

Процессор - Intel® CoreTM i5-7300HQ CPU @ 2.50GHz × 4 (ЦП 4 ядра 4 потока)

В таблице 4.1 приведены замеры времени работы реализации алгоритмов решения задачи коммивояжера.

Размер	Муравьиный	Перебор
2	137	1
3	388	1
4	795	4
5	1697	8
6	2877	32
7	4511	161
8	7178	1016
9	10363	14297
10	14948	133040

Таблица 4.1: Время работы (μc)

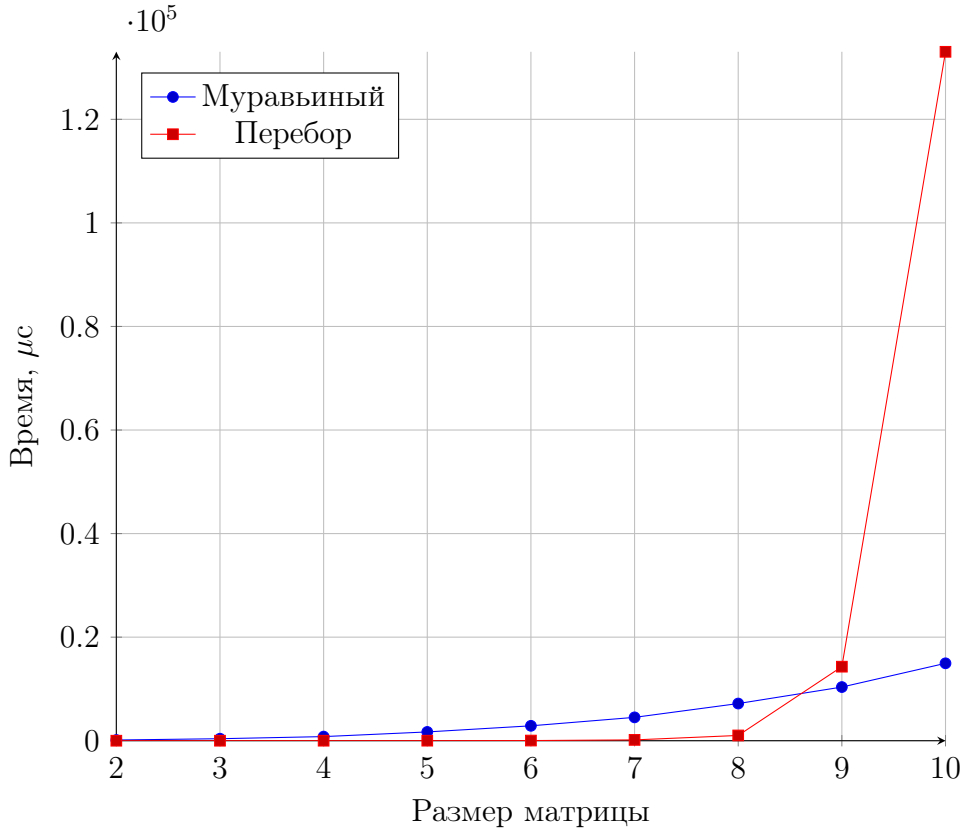

Рис. 4.3: Зависимость времени работы реализации алгоритмов решения задачи коммивояжера от размерности

4.3 Вывод

График показывает, что муравьиный алгоритм решения задачи коммивояжера выигрывает у алгоритма полного перебора начиная с графа, количество вершин в котором больше 8. Решение методом грубой силы становится непрактичным даже для 20 городов. С плохими параметрами конфигурации муравьиный алгоритм может «застрять» на локальных экстремумах

Заключение

В ходе лабораторной работы было изучены метод полного перебора и метод на базе муравьиного алгоритма для решения задачи коммивояжера и сделаны следующие выводы:

- муравьиный алгоритм решения задачи коммивояжера выигрывает у алгоритма полного перебора начиная с графа, количество вершин в котором больше 8;
- решение методом грубой силы становится непрактичным даже для 20 городов;
- с плохими параметрами конфигурации муравьиный алгоритм может «застрять» на локальных экстремумах.

Литература

[1] М.В. Ульянов. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. ФИЗМАТЛИТ, 2007. с. 376.

[2] Effective Go
`https://golang.org/doc/effective_go.html` [Электронный ресурс] (дата обращения: 25.01.21)