**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 курсу «Анализ алгоритмов»

**Тема** Методы решения задачи коммивояжёра

**Студент** Гаврилюк В. А.

**Группа** ИУ7-51Б

**Оценка (баллы)**

**Преподаватель** Волкова Л. Л.

Москва — 2025 г.

# СОДЕРЖАНИЕ

# ВВЕДЕНИЕ

**Цель лабораторной работы** — исследование методов решения задачи коммивояжёра: метода полного перебора и муравьиного алгоритма.

Для достижения поставленной цели необходимо выполнить следующие задачи:

- построить схемы для алгоритма на основе полного перебора и муравьиного алгоритма;
- выполнить оценку трудоёмкости данных алгоритмов;
- создать программное обеспечение (ПО), реализующее перечисленные выше алгоритмы;
- провести параметризацию для муравьиного алгоритма;
- провести сравнение затрат процессорного времени на работу алгоритмов.

# 1 Аналитический раздел

В данном разделе будет описана задача коммивояжёра, будут рассмотрены метод полного перебора и муравьиный алгоритм для решения задачи коммивояжёра, а также описаны изменения стоимости перехода по метке дуги на различной местности.

## 1.1 Задача коммивояжёра

Если ассоциировать города с вершинами графа, а пути сообщения и стоимости проезда с нагруженными рёбрами, то получается полный ориентированный асимметричный граф без собственных петель на $n$ вершинах. Граф может быть описан матрицей стоимости $C$, значение каждого элемента которой — $c_{ij}$ равно стоимости (измеряемой реально в единицах времени, денег или расстояния) прямого проезда из города $i$ в город $j$ [1]. Задача коммивояжёра называется симметричной, если $c_{ij} = c_{ji} \forall i \neq j, i, j = \overline{1, n}$, т. е. если стоимость проезда между каждыми двумя городами не зависит от направления, и несимметричной, если это не так.

Задача в терминах теории графов формулируется как задача нахождения покрывающего (полного) цикла наименьшей стоимости, который называется туром, на полном ориентированном графе, заданном несимметричной (в общем случае) матрицей стоимостей $C$.

## 1.2 Метод полного перебора

Метод полного перебора для решения задачи коммивояжёра представляет собой алгоритм, который перебирает все возможные пути между вершинами графа и выбирает путь с минимальной стоимостью. Точный переборный алгоритм решения данной задачи имеет факториальную сложность [1], но при этом гарантирует нахождение оптимального решения.

## 1.3 Муравьиный алгоритм

Идея муравьиного алгоритма заключается в моделировании поведения муравьев, связанного с их способностью находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь [1].

В данном алгоритме вводятся следующие локальные правила поведения муравьев при выборе пути [1]:

– муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, у каждого муравья сеть список уже посещённых городов — список запретов.

– муравьи обладают «зрением» — видимость есть эвристическое желание посетить город $j$, если муравей находится в городе $i$. Видимость вычисляется по следующей формуле:

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

где $D_{ij}$ — расстояние между городами $i$ и $j$;

– муравьи обладают «обонянием» — они могут улавливать след феромона, подтверждающий желание посетить город $j$ из города $i$, на основании опыта других муравьев.

Вероятность перехода $k$-го муравья из города $i$ в город $j$ определяется формулой [1]:

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

где $\tau_{ij}(t)$ — количество феромона на ребре $(i, j)$ в момент времени $t$, $J_{i,k}$ — список городов, которые необходимо посетить муравью $k$, находящемуся в городе $i$, $\alpha, \beta$ — параметры, задающие веса следа феромона.

Откладываемое количество феромона муравьев после прохождения ребра $(i, j)$ вычисляется по формуле (1.3), где $T_k(t)$ — маршрут, пройденный муравьём $k$ к моменту времени $t$, а $L_k(t)$ — длина этого маршрута. В начале алгоритма количество феромона на рёбрах принимается равным небольшому положительному числу [1].

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

Испарение феромона вычисляется по формуле:

$$\tau_{ij}(t+1) = (1-p) \cdot \tau_{ij}(t) + \Delta\tau_{ij}(t), \quad \Delta\tau_{ij}(t) = \sum_{k=1}^{m} \tau_{ij,k}(t), \tag{1.4}$$

где $m$ — количество муравьев в колонии, а $p \in [0, 1]$ — коэффициент испарения.

Если в колонии есть «элитные» муравьи, которые усиливают ребра наилучшего маршрута, найденного с начала работы алгоритма, рёбра маршрута получат дополнительное количество феромона, вычисляемое по формуле:

$$\Delta\tau_e = e \cdot Q/L^*, \tag{1.5}$$

где $L^*$ — длина маршрута $T^*$ — наилучшего из найденных маршрутов. Параметр $Q$ вычисляется по формуле:

$$Q = \frac{\sum_0^{n-1} D_{ij}}{n}, \tag{1.6}$$

где $n$ — количество вершин в графе.

## 1.4 Карта местности

Согласно варианту необходимо учесть изменение стоимости перемещения при различных условиях окружающей среды: по воде перемещение быстрее, а по пустыне медленнее (относительно некоторой стандартной, обыкновенной местности). Таким образом, были заданы следующие правила изменения стоимости перемещения:

- перемещение по воде (Water) — стоимость перехода уменьшается на 50 %;
- перемещение по земле (Land) — стоимость перехода остаётся неизменной;
- перемещение по пустыне (Desert) — стоимость перехода увеличивается на 30 %.

# 2 Конструкторский раздел

В данном разделе будут приведены требования к входным, выходным параметрам и представлены схемы для алгоритма решения задачи коммивояжёра методом полного перебора и муравьиного алгоритма.

## 2.1 Требования к входным и выходным параметрам

Требования к входным и выходным параметрам:

- в качестве входных параметров алгоритм принимает матрицу смежности;
- матрица смежности должна соответствовать полносвязному графу;
- выходными параметрами являются массив вершин и число — кратчайший из найденных маршрутов (маршрут незамкнутый) и его стоимость соответственно.

## 2.2 Метод полного перебора

На рисунке 2.1 представлена схема алгоритма решения задачи коммивояжёра методом полного перебора.

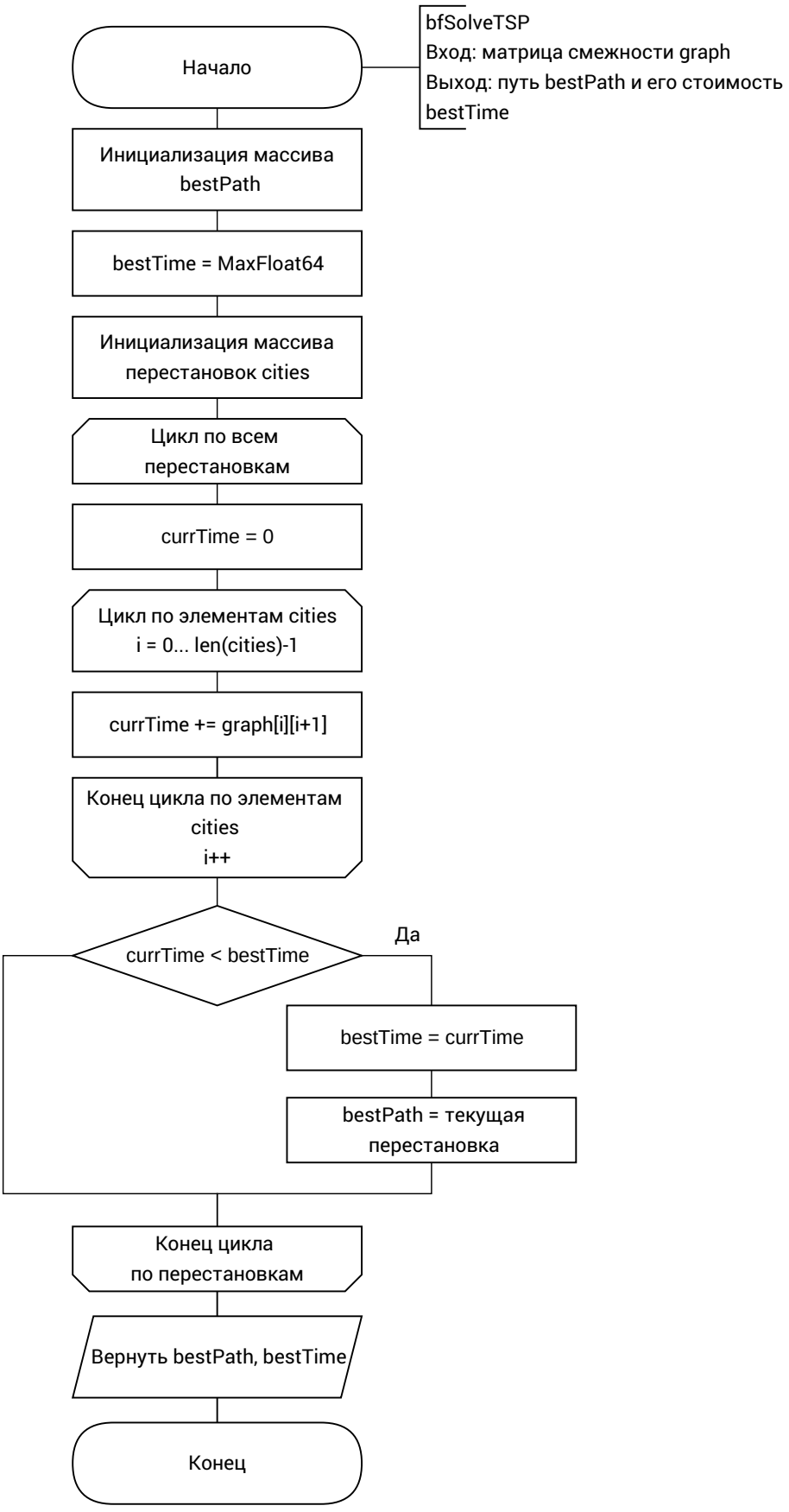


Рисунок 2.1 – Схема метода полного перебора

## 2.3 Муравьиный алгоритм

На рисунке 2.2 представлена схема муравьиного алгоритма.

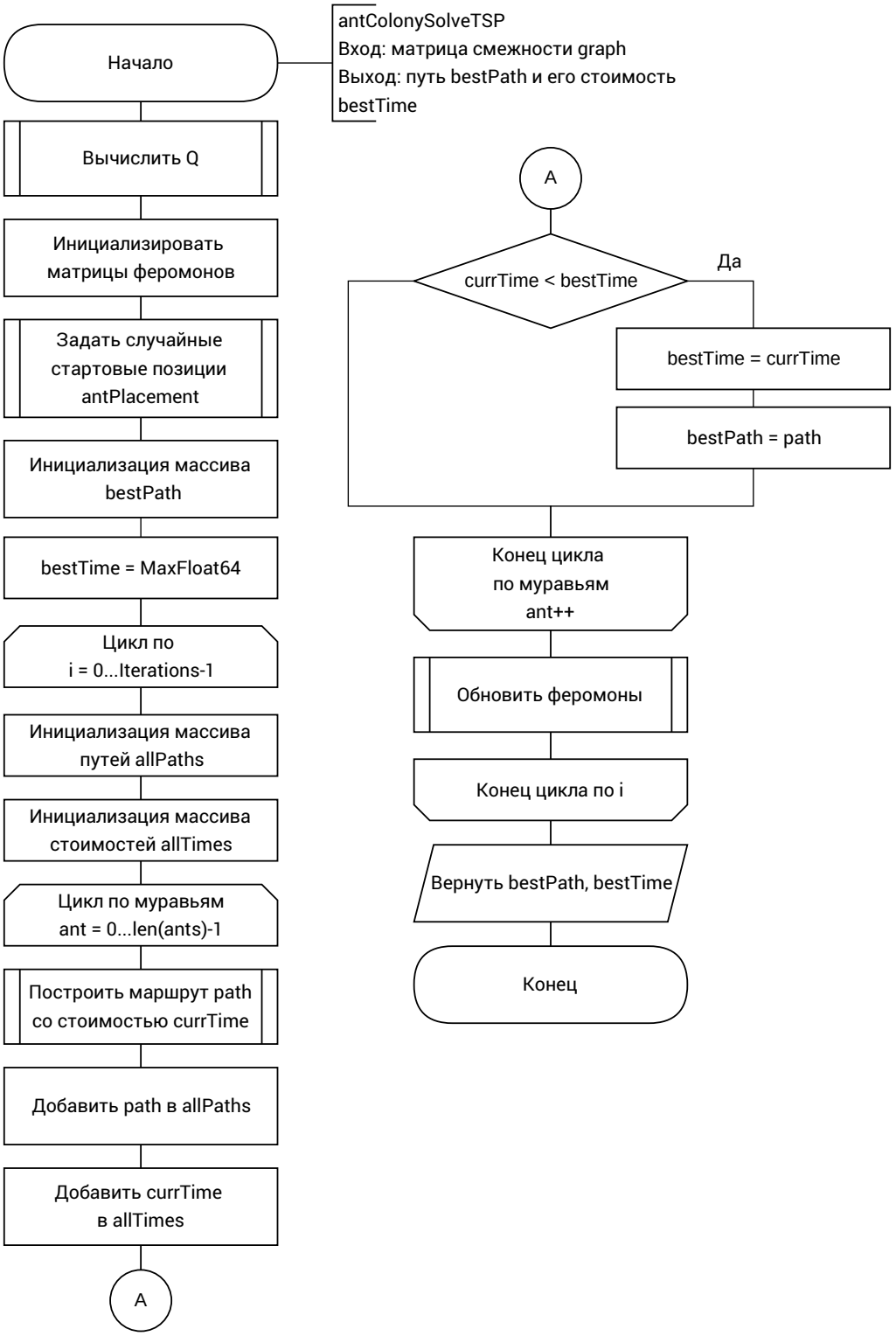


Рисунок 2.2 – Схема муравьиного алгоритма

На рисунках 2.3– 2.5 представлены схемы вспомогательных алгоритмов, использующихся в рамках муравьиного алгоритма.

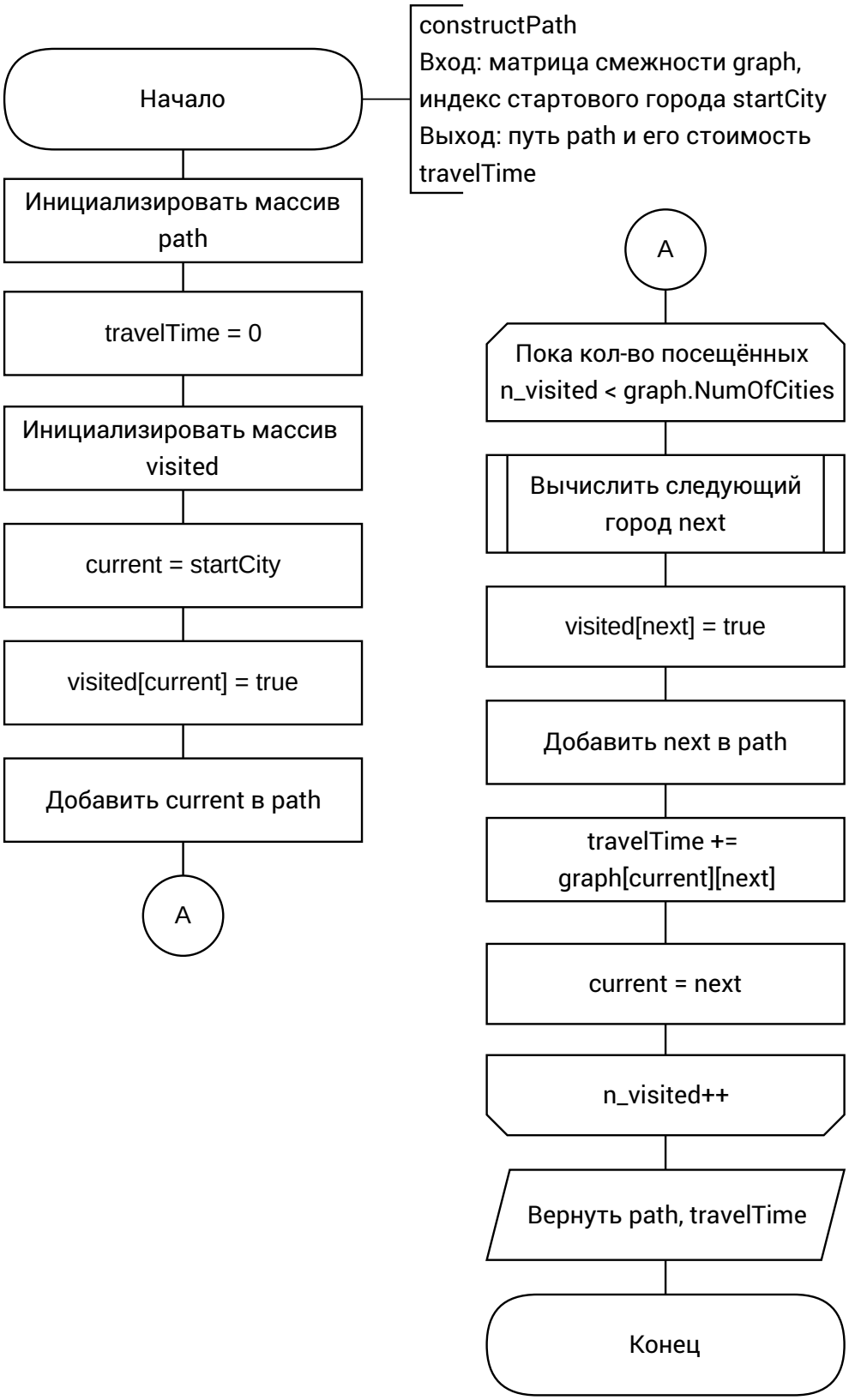


Рисунок 2.3 – Схема алгоритма создания маршрута

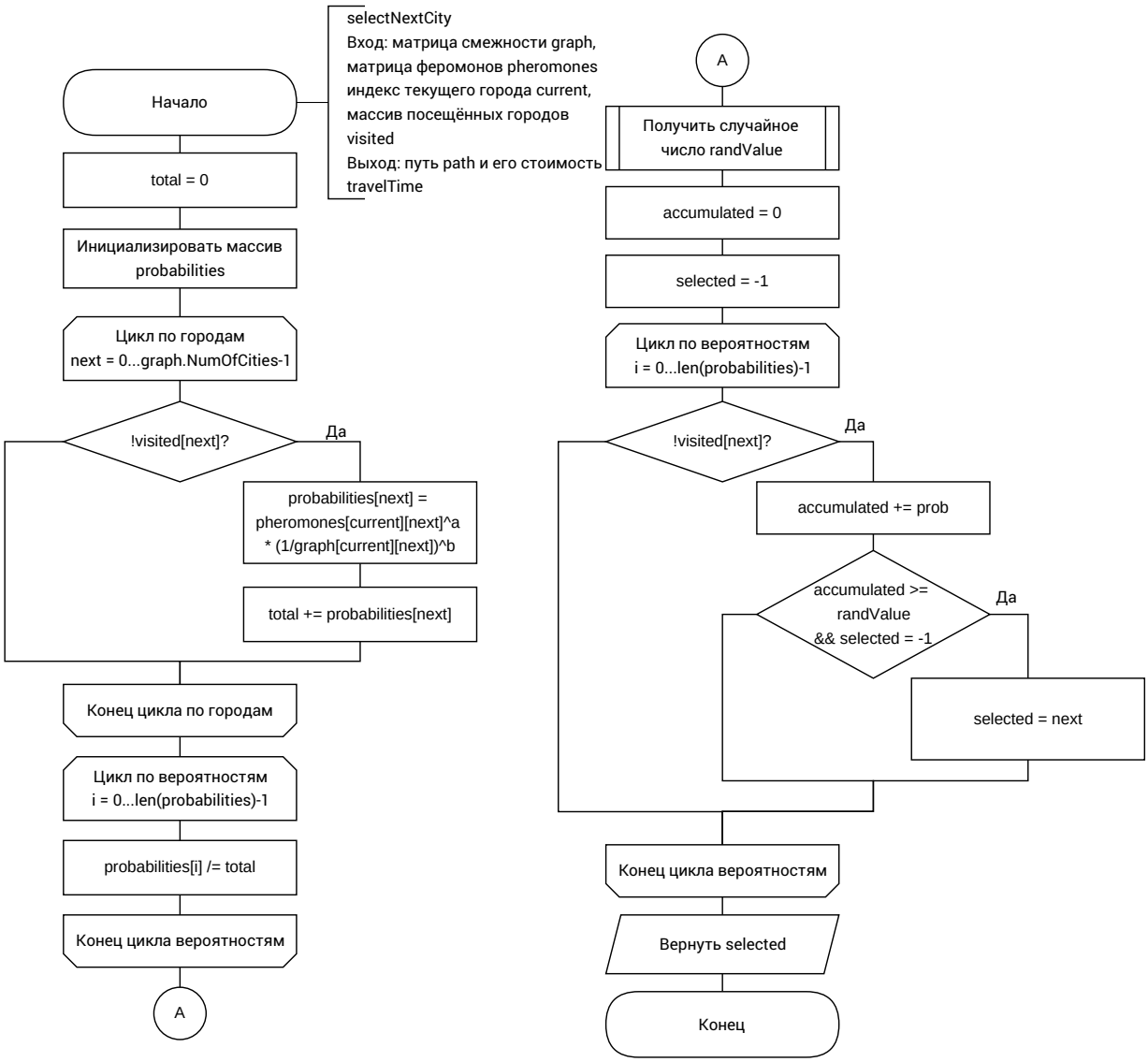


Рисунок 2.4 – Схема алгоритма выбора следующего города

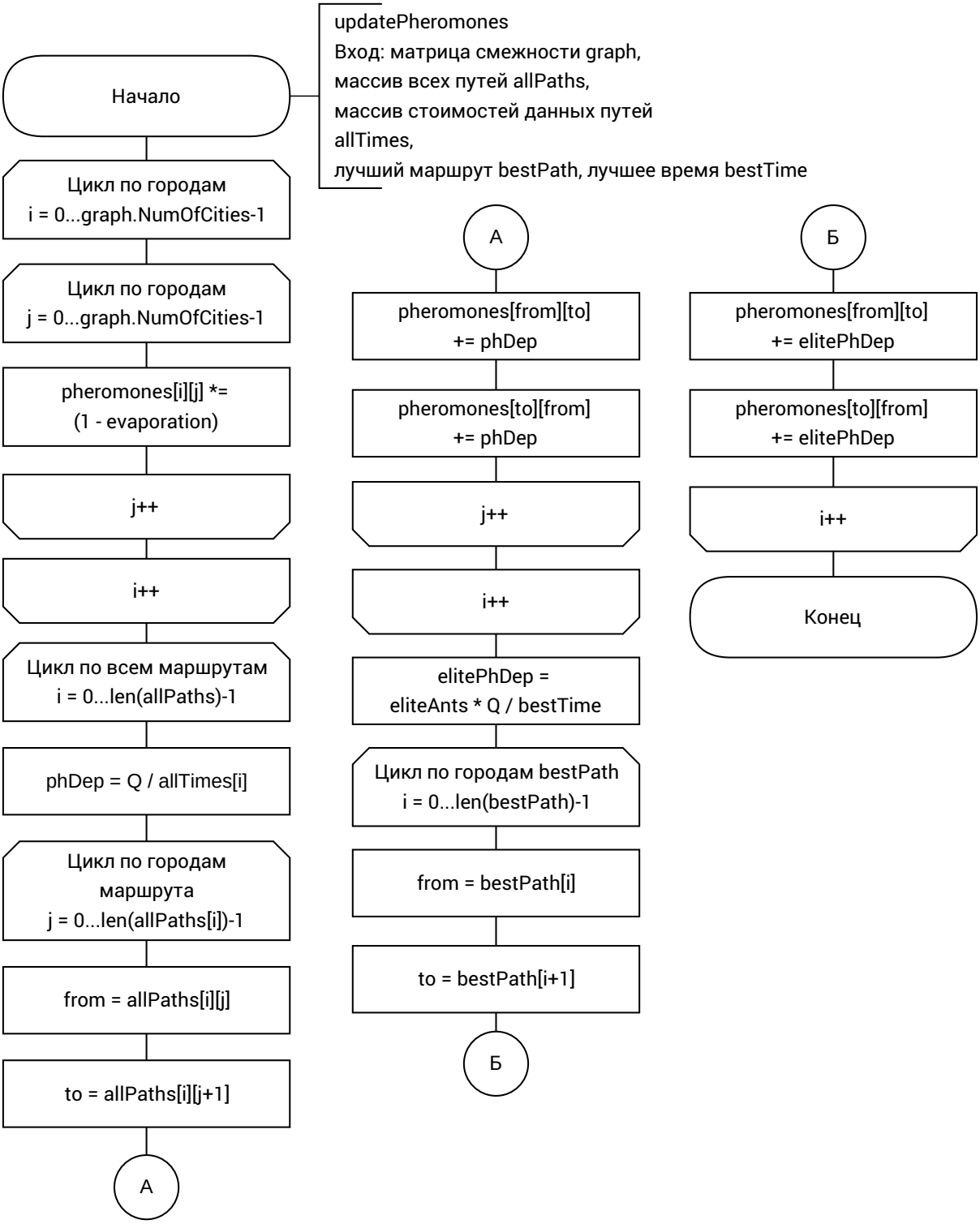


Рисунок 2.5 – Схема алгоритма обновления феромонов

## 2.4 Оценка трудоёмкости

### 2.4.1 Модель вычислений

Операции, имеющие единичную стоимость:

$$+, -, =, +=, -=, ==, !=, <, >, <=, >=, [idx], ++, --, \\ \&\&, >>, <<, ||, \&, | \tag{2.1}$$

Операции, имеющие двойную стоимость:

$$*, /, \%, *=, /=, \%=, len() \tag{2.2}$$

Стоимость операции возведения в степень будет оцениваться в 3 единицы.

Трудоёмкость условного оператора:

$$f_{if} = f_{\text{условия}} + \begin{cases} \min(f_1, f_2), & \text{лучший случай} \\ \max(f_1, f_2), & \text{худший случай,} \end{cases} \tag{2.3}$$

где $f_1$ — трудоёмкость тела условного выражения, $f_2$ — трудоёмкость тела блока else.

Трудоёмкость цикла:

$$f_{loop} = f_{\text{инициализация}} + f_{\text{сравнения}} + N \times (f_{\text{тело}} + \\ + f_{\text{инкремент}} + f_{\text{сравнения}}) \tag{2.4}$$

Трудоёмкость вызова функции равна нулю.

### 2.4.2 Трудоёмкость метода полного перебора

Стоимость инициализации переменных:

$$f_{init} = n + 1 + 1 + 1 + n(1 + 1 + 2) = 3 + n + 4n = 3 + 5n \tag{2.5}$$

Стоимость основного цикла по всем перестановкам рассчитывается по формуле (2.6).

$$f_{loop} = 2 + n!(2 + f_{body}) \tag{2.6}$$

Трудоёмкость $f_{body}$ можно рассчитать по формуле (2.7).

$$f_{body} = 1 + 1 + 1 + (n-1)(1 + 1 + 4) + f_{check} \tag{2.7}$$

Трудоёмкость $f_{check}$ рассчитывается по формуле (2.8).

$$1 + \begin{cases} 0, & \text{лучший случай (bestTime <= currTime)}, \\ 2 & \text{худший случай (bestTime > currTime)} \end{cases} \tag{2.8}$$

Итоговая трудоёмкость в лучшем случае рассчитывается по формуле (2.9).

$$\begin{gathered} f = f_{init} + f_{loop} = 3 + 5n + 2 + n!(2 + f_{body}) = \\ 5 + 5n + n!(5 + (n-1) \cdot 6) + 1) = \\ 6 + 5n + n!(6n - 1) \approx O(n! \cdot n) \end{gathered} \tag{2.9}$$

Итоговая трудоёмкость в худшем случае рассчитывается по формуле (2.10).

$$\begin{gathered} f = f_{init} + f_{loop} = 3 + 5n + 2 + n!(2 + f_{body}) = \\ 5 + 5n + n!(5 + (n-1) \cdot 6) + 3) = \\ 6 + 5n + n!(6n + 2) \approx O(n! \cdot n) \end{gathered} \tag{2.10}$$

### 2.4.3 Трудоёмкость муравьиного алгоритма

Стоимость инициализации переменных:

$$f_{init} = 3 + n(4 + 3n) + 2 + 3n = 5 + 7n + 3n^2 \tag{2.11}$$

Пусть $k$ — количество итераций алгоритма. Тогда стоимость основного цикла в муравьином алгоритме вычисляется по формуле (2.12).

$$\begin{gathered} f_{loop} = 1 + 1 + k(6 + n(1 + 1 + f_{path} + 1 + 1 + f_{check}) \\ + f_{upd\_pheromones}) \end{gathered} \tag{2.12}$$

В формуле (2.12) $f_{path}$ — стоимость построения маршрута муравья, $f_{check}$

— стоимость проверки и обновления лучшего найденного пути и $f_{upd\_pheromones}$ — стоимость обновления феромонов с учётом испарения и добавления феромонов обычными и элитными муравьями. Данные стоимости вычисляются с помощью формул (2.14)— (2.16). Стоимость выбора следующего города при построении маршрута вычисляется по формуле (2.13).

$$f_{select\_next} = 4 + n(20 + 2) + 2 + 4n + 5 + n(1 + 1 + 5) = \\ 11 + 22n + 4n + 7n = 11 + 33n \tag{2.13}$$

$$f_{path} = 7 + 1 + 1 + n(1 + 1 + f_{select\_next} + 2 + 1 + 3 + 1 + 1) = \\ 9 + n(10 + f_{select\_next}) = 9 + n(10 + 11 + 33n) = 9 + 21n + 33n^2 \tag{2.14}$$

Формула для обновления лучшего результата *ant_check* совпадает с формулой (2.8) в алгоритме полного перебора:

$$f_{ant\_check} = 1 + \begin{cases} 0, & \text{лучший случай (bestTime <= currTime),} \\ 2 & \text{худший случай (bestTime > currTime).} \end{cases} \tag{2.15}$$

$$f_{upd\_pheromones} = 11 + 4n + 7n^2 + n(7 + 13n - 13) + 7 + 14(n - 1) = \\ 11 + 4n + 7n^2 - 6 + 13n + 14n - 11 = -2 + 31n + 7n^2 \tag{2.16}$$

Итоговая трудоёмкость муравьиного алгоритма в худшем случае вычисляется по формуле (2.17).

$$f = f_{init} + f_{loop} = 7 + 6k + 7n + 3n^2 \\ +16nk + 21n^2k + 33n^3k \approx 33n^3k = O(n^3k) \tag{2.17}$$

В лучшем случае трудоёмкость алгоритма составляет:

$$f = f_{init} + f_{loop} = 7 + 6k + 7n + 3n^2 \\ +13nk + 21n^2k + 33n^3k \approx 33n^3k = O(n^3k) \tag{2.18}$$

## Вывод

В данном разделе были приведены схемы алгоритмов решения задачи коммивояжёра: на основе полного перебора и муравьиного алгоритма. В рамках модели вычислений, определённой в подразделе 2.4.1, алгоритм на основе полного перебора обладает факториальной сложностью, а муравьиный алгоритм — $O(n^3k)$.

# 3 Технологический раздел

В данном разделе будет представлена реализация муравьиного алгоритма и метода полного перебора для решения задачи коммивояжёра.

## 3.1 Средства реализации

Для реализации был выбран язык программирования Golang [2]. Выбор обусловлен наличием функций для измерения времени работы *time.Now* и *time.Since* в библиотеке *time*.

## 3.2 Реализация метода полного перебора

В листинге 3.1 представлена реализация алгоритма решения задачи коммивояжёра методом полного перебора.

Листинг 3.1 – Реализация метода полного перебора

```
func (b *BruteForceSolver) SolveTSP(graph *graphmap.Graph)
   ([]int, float64) {
    if graph.NumOfCities < 1 {
        return []int{}, 0
    }
    if graph.NumOfCities == 1 {
        return []int{0}, 0
    }
    b.graph = graph
    cities := make([]int, graph.NumOfCities)
    for i := range cities {
        cities[i] = i
    }
    bestPath := []int{}
    bestTime := math.MaxFloat64
    permute(cities, func(path []int) {
        travelTime := b.calculateTravelTime(path)
        if travelTime < bestTime {
            bestTime = travelTime
            bestPath = make([]int, len(path))
            copy(bestPath, path)
        }
    })
    return bestPath, bestTime
}
```

```
func (b *BruteForceSolver) calculateTravelTime(route []int)
   float64 {
    var distance float64
    for i := 0; i < len(route)-1; i++ {
        distance += b.graph.GetTimeOnTerrain(route[i],
           route[i+1])
    }
    return distance
}

func permute(arr []int, callback func([]int)) {
    var generate func([]int, int)
    generate = func(a []int, n int) {
        if n == 1 {
            callback(a)
            return
        }
        for i := 0; i < n; i++ {
            generate(a, n-1)
            if n%2 == 1 {
                a[0], a[n-1] = a[n-1], a[0]
            } else {
                a[i], a[n-1] = a[n-1], a[i]
            }
        }
    }

    generate(arr, len(arr))
}
```

## 3.3 Реализация муравьиного алгоритма

В листингах 3.2 — 3.5 продемонстрирована реализация муравьиного алгоритма.

Листинг 3.2 – Основная функция муравьиного алгоритма

```
func (c *Colony) SolveTSP(graph *graphmap.Graph) ([]int,
   float64) {
    if graph.NumOfCities < 1 {
        return []int{}, 0
    }
    if graph.NumOfCities == 1 {
        return []int{0}, 0
    }
    c.graph = graph
    c.setQ()
    c.initPheromones()
    antPlacement := c.randomAntPlacement()

    bestPath := []int{}
    bestTime := math.MaxFloat64
    for range c.params.Iterations {
        allPaths := make([][]int, 0, c.graph.NumOfCities)
        allTimes := make([]float64, 0, c.graph.NumOfCities)

        for antStartCity := range antPlacement {
            path, travelTime := c.constructPath(antStartCity)
            allPaths = append(allPaths, path)
            allTimes = append(allTimes, travelTime)
            if travelTime < bestTime {
                bestPath = path
                bestTime = travelTime
            }
        }
        c.updatePheromones(allPaths, allTimes, bestPath,
           bestTime)
    }

    return bestPath, bestTime
}
```

Листинг 3.3 – Функция построения пути

```
func (c *Colony) constructPath(startCity int) ([]int, float64) {
    path := []int{}
    var travelTime float64
    visited := make(map[int]struct{}, c.graph.NumOfCities)
    current := startCity
    visited[current] = struct{}{}
    path = append(path, current)

    for len(visited) < c.graph.NumOfCities {
        next := c.selectNextCity(current, visited)
        if next == -1 {
            break
        }
        visited[next] = struct{}{}
        path = append(path, next)
        travelTime += c.graph.GetTimeOnTerrain(current, next)
        current = next
    }

    return path, travelTime
}
```

Листинг 3.4 – Функция выбора следующего города

```
func (c *Colony) selectNextCity(current int, visited
   map[int]struct{}) int {
    var total float64
    probabilities := make([]float64, c.graph.NumOfCities)
    for next := range c.graph.NumOfCities {
        if _, ok := visited[next]; ok {
            continue
        }
    probabilities[next] = math.Pow(c.pheromones[current][next],
       c.params.Alpha)
    * math.Pow(1.0/float64(c.graph.TravelTime[current][next]),
       c.params.Beta)
        total += probabilities[next]
    }
    for i := range probabilities {
        probabilities[i] /= total
    }
    randValue := random.Float64()
```

```
    accumulated := 0.0
    for next, prob := range probabilities {
        if _, ok := visited[next]; ok {
            continue
        }
        accumulated += prob
        if accumulated >= randValue {
            return next
        }
    }
    return -1
}
```

Листинг 3.5 – Функция обновления феромонов с учётом испарения и элитных муравьев

```
func (c *Colony) updatePheromones(allPaths [][]int, allTimes
   []float64, bestPath []int, bestTime float64) {
    for i := range c.pheromones {
        for j := range c.pheromones[i] {
            c.pheromones[i][j] *= (1 - c.params.Evaporation)
        }
    }
    for i := range allPaths {
        pheromoneDeposit := c.params.q / float64(allTimes[i])
        for j := 0; j < len(allPaths[i])-1; j++ {
            from, to := allPaths[i][j], allPaths[i][j+1]
            c.pheromones[from][to] += pheromoneDeposit
            c.pheromones[to][from] += pheromoneDeposit
        }
    }
    elitePheromoneDeposit := float64(c.params.EliteAnts) *
       c.params.q / float64(bestTime)
    for i := 0; i < len(bestPath)-1; i++ {
        from, to := bestPath[i], bestPath[i+1]
        c.pheromones[from][to] += elitePheromoneDeposit
        c.pheromones[to][from] += elitePheromoneDeposit
    }
}
```

# 4 Исследовательский раздел

## 4.1 Технические характеристики

Исследование проводилось на ЭВМ со следующими характеристиками:

– операционная система Ubuntu 22.04.5 LTS;

– объем оперативной памяти 16 ГБ;

– процессор Intel Core i7-8700K CPU 3.70ГГц × 12 [3].

## 4.2 Параметризация муравьиного алгоритма

Выполнена параметризация муравьиного алгоритма по трём параметрам: $\alpha$, $\beta$ и коэффициенту испарения феромонов $p$. В качестве критерия качества получаемого решения были использованы максимальное, медианное и среднее арифметическое значения отклонения затрат полученного маршрута от эталонного значения. Эталонные значения были получены в результате применения алгоритма на основе полного перебора.

Параметризация проводилась на трёх полносвязных графах с 10 вершинами с одинаковыми значениями математического ожидания и разбросом значений меток дуг графа. Данные графы представлены формулами (4.1) — (4.3).

$$F_1 = \begin{pmatrix} 0 & 9 & 59 & 85 & 97 & 42 & 48 & 71 & 87 & 13 \\ 9 & 0 & 17 & 11 & 45 & 9 & 66 & 11 & 15 & 11 \\ 59 & 17 & 0 & 12 & 49 & 79 & 57 & 76 & 26 & 94 \\ 85 & 11 & 12 & 0 & 31 & 87 & 2 & 41 & 25 & 57 \\ 97 & 45 & 49 & 31 & 0 & 57 & 43 & 58 & 61 & 25 \\ 42 & 9 & 79 & 87 & 57 & 0 & 8 & 59 & 98 & 97 \\ 48 & 66 & 57 & 2 & 43 & 8 & 0 & 70 & 21 & 70 \\ 71 & 11 & 76 & 41 & 58 & 59 & 14 & 0 & 76 & 98 \\ 87 & 15 & 26 & 25 & 61 & 98 & 51 & 76 & 0 & 34 \\ 13 & 11 & 94 & 57 & 25 & 97 & 10 & 98 & 34 & 0 \end{pmatrix} \tag{4.1}$$

$$F_2 = \begin{pmatrix} 0 & 9 & 59 & 85 & 97 & 42 & 48 & 71 & 87 & 13 \\ 9 & 0 & 17 & 11 & 45 & 9 & 66 & 11 & 15 & 11 \\ 59 & 17 & 0 & 12 & 49 & 79 & 57 & 76 & 26 & 94 \\ 85 & 11 & 12 & 0 & 31 & 87 & 2 & 41 & 25 & 57 \\ 97 & 45 & 49 & 31 & 0 & 57 & 43 & 58 & 61 & 25 \\ 42 & 9 & 79 & 87 & 57 & 0 & 8 & 59 & 98 & 97 \\ 48 & 66 & 57 & 2 & 43 & 8 & 0 & 70 & 21 & 70 \\ 71 & 11 & 76 & 41 & 58 & 59 & 70 & 0 & 76 & 98 \\ 87 & 15 & 26 & 25 & 61 & 98 & 34 & 76 & 0 & 34 \\ 13 & 11 & 94 & 57 & 25 & 97 & 54 & 98 & 34 & 0 \end{pmatrix} \tag{4.2}$$

$$F_3 = \begin{pmatrix} 0 & 28 & 99 & 83 & 21 & 10 & 49 & 60 & 31 & 77 \\ 28 & 0 & 68 & 14 & 79 & 81 & 35 & 43 & 88 & 51 \\ 99 & 68 & 0 & 39 & 79 & 95 & 85 & 41 & 90 & 74 \\ 83 & 14 & 39 & 0 & 94 & 99 & 98 & 49 & 35 & 63 \\ 21 & 79 & 79 & 94 & 0 & 79 & 31 & 81 & 98 & 16 \\ 10 & 81 & 95 & 99 & 79 & 0 & 26 & 73 & 12 & 40 \\ 49 & 35 & 85 & 98 & 31 & 26 & 0 & 94 & 72 & 19 \\ 60 & 43 & 41 & 49 & 81 & 73 & 94 & 0 & 58 & 48 \\ 31 & 88 & 90 & 35 & 98 & 12 & 72 & 58 & 0 & 61 \\ 77 & 51 & 74 & 63 & 16 & 40 & 19 & 48 & 61 & 0 \end{pmatrix} \tag{4.3}$$

Для каждого графа для каждого варианта сочетания параметров $\alpha$ $\beta$ и $\rho$ было выполнено 10 запусков. Значения параметров выбирались из списка значений:

$$0.1, 0.3, 0.5, 0.7, 0.9. \tag{4.4}$$

На основе результатов параметризации можно выделить следующие комбинации параметров, при которых значения отклонений являются минимальными:

$$\begin{gathered} \alpha = 0.9, \beta = 0.9, \rho = 0.1; \\ \alpha = 0.5, \beta = 0.9, \rho = 0.7. \end{gathered} \tag{4.5}$$

Полная таблица, полученная в результате параметризации, находится в

приложении А.

## 4.3 Проведение исследования

Было проведено исследование зависимости времени работы алгоритмов решения задачи коммивояжёра от количества городов (вершин во входном графе). Для каждого размера были сгенерировано 10 случайных матриц смежности и было проведено 10 замеров времени. Измерение времени проводилось с помощью функций *time.Now* и *time.Since* из библиотеки *time*. В таблице 4.1 приведены результаты замера процессорного времени работы алгоритмов.

Таблица 4.1 – Зависимость времени работы алгоритмов от количества вершин в графе

| **Кол-во вершин** | **Полный перебор, мкс** | **Муравьиный алгоритм, мкс** |
|---|---|---|
| 3 | 7.097 | 1,890.857 |
| 4 | 5.921 | 1,950.345 |
| 5 | 23.305 | 2,940.739 |
| 6 | 185.098 | 4,507.259 |
| 7 | 1,361.954 | 7,030.611 |
| 8 | 12,881.386 | 10,118.585 |
| 9 | 126,270.046 | 15,486.321 |
| 10 | 1,407,765.411 | 20,927.566 |
| 11 | 17,253,809.982 | 27,470.175 |
| 12 | 225,634,368,708.000 | 36,125.691 |

На рисунках 4.1 и 4.2 продемонстрированы графики, построенные на основе табличных данных.

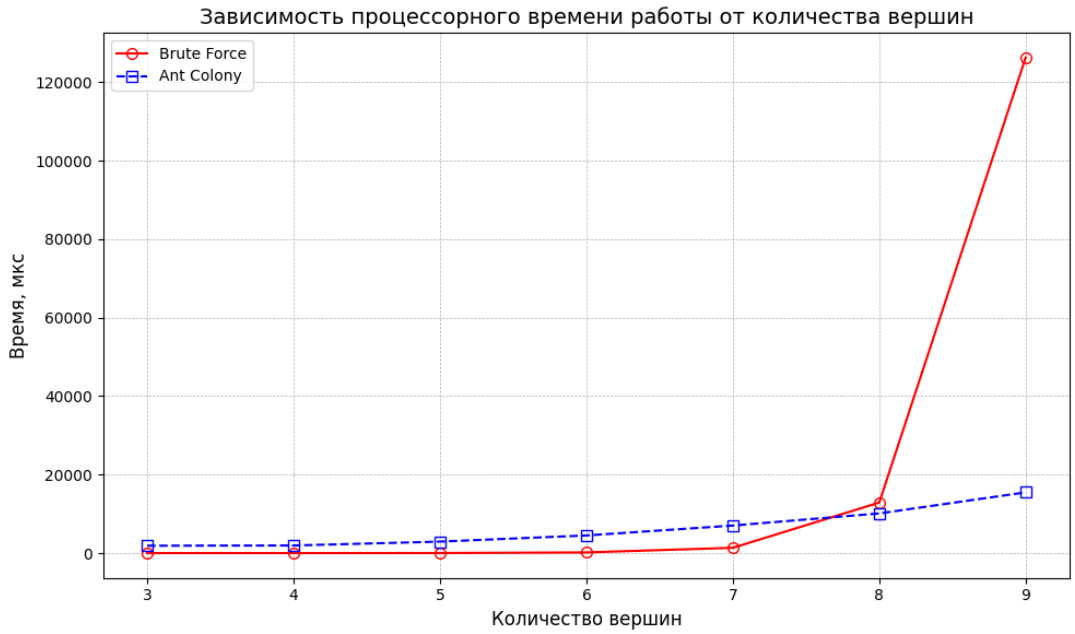


Рисунок 4.1 – Зависимость процессорного времени работы алгоритмов от количества вершин (линейная шкала)

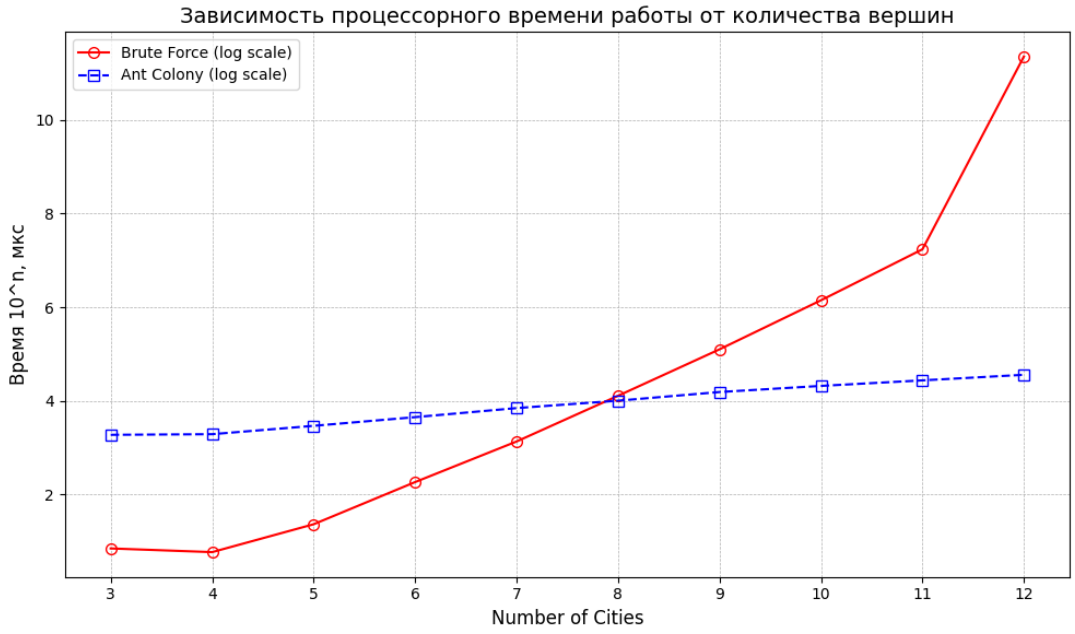


Рисунок 4.2 – Зависимость процессорного времени работы алгоритмов от количества вершин (логарифмическая шкала)

## Вывод

По результатам, полученным при параметризации муравьиного алгоритма, было установлено, что точность алгоритма зависит от выбранных параметров и класса данных, на котором выполняется параметризация. Т.е.

для каждой конкретной задачи и типа данных необходимо подбирать и вычислять индивидуальные параметры для муравьиного алгоритма.

Рекомендованными комбинациями параметров для выбранного класса данных являются следующие комбинации:

$$\begin{aligned} &\alpha = 0.9, \beta = 0.9, \rho = 0.1; \\ &\alpha = 0.5, \beta = 0.9, \rho = 0.7. \end{aligned} \tag{4.6}$$

В рамках исследования процессорного времени работы алгоритмов было выявлено, что при количестве вершин, меньшем 8, реализация алгоритма на основе полного перебора является более эффективной по времени.

# ЗАКЛЮЧЕНИЕ

На основе результатов параметризации было установлено, что для каждой конкретной задачи и типа данных необходимо подбирать и вычислять индивидуальные параметры для муравьиного алгоритма.

Рекомендованными комбинациями параметров для выбранного класса данных являются следующие комбинации: $\alpha = 0.9, \beta = 0.9, \rho = 0.1$ и $\alpha = 0.5, \beta = 0.9, \rho = 0.7$.

Исследование процессорного времени работы подтвердило теоретические расчёты трудоёмкости алгоритмов решения задачи коммивояжёра. Также в рамках исследования было выявлено, что при количестве вершин, меньшем 8, реализация алгоритма на основе полного перебора является более эффективной по времени.

В рамках лабораторной работы:

- были построены схемы для алгоритма на основе полного перебора и муравьиного алгоритма;
- была выполнена оценка трудоёмкости данных алгоритмов;
- создано программное обеспечение (ПО), реализующее перечисленные выше алгоритмы;
- произведена параметризацию для муравьиного алгоритма;
- произведено сравнение затрат процессорного времени на работу алгоритмов.

Все задачи выполнены. Цель лабораторной работы достигнута.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. *Ульянов А.* Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. — Москва : Наука, 2010.
2. The Go Programming Language [Электронный ресурс]. — Режим доступа: `https://go.dev/` (дата обращения: 5.12.2024).
3. Intel Core i7-8700K Processor [Электронный ресурс]. — Режим доступа: `https://ark.intel.com/content/www/us/en/ark/products/126775/intel-core-i78700k-processor-12m-cache-up-to-4-70-ghz.html` (дата обращения: 5.12.2024).