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

Название: Параметризация задачи. Муравьиный алгоритм

Дисциплина: Анализ алгоритмов

Студент ИУ7-52Б (Группа) _____ (Подпись, дата) Н. В. Ляпина (И.О. Фамилия)

Преподователь _____ (Подпись, дата) Л.Л. Волкова (И.О. Фамилия)

Москва, 2022

Содержание

Введение

Существует довольно обширный класс задач оптимизации, для которых нет быстрых полиномиальных алгоритмов определения решений. Для таких задач имеет смысл использовать эвристические алгоритмы [1], которые не гарантируют оптимальных решений, однако, при правильной параметризации такие алгоритмы обеспечивают наличие решений наилучшего, насколько это возможно, качества.

На сегодняшний день большую популярность имеют методы, основывающиеся на природных механизмах. Такие алгоритмы позволяют решать множество сложных оптимизационных задач, таких как задачи полихромии графов, задачи о распределении и некоторые транспортные задачи. Алгоритм, имитирующий поведение муравьиной колонии, входит в класс алгоритмов «роевого интеллекта» [2].

В представленной работе будет предложен метод оптимизации задачи коммивояжера, использующий алгоритм муравьиной колонии.

Цель лабораторной работы:
проанализировать предложенную оптимизацию и параметризировать алгоритм для двух классов данных..

Задачи лабораторной работы:

1) реализовать алгоритм полного перебора и алгоритм муравьиной колонии;
2) разработать два класса эквивалентности для параметризации алгоритма муравьиной колонии;
3) провести параметризацию для разработанных классов эквивалентности;
4) сравнить алгоритмы, согласно следующим параметрам:
    а) качество полученного решения;
    б) временные затраты на получение решения.

Сравнительный анализ алгоритмов будет оформлен в виде таблиц, из которых можно будет сделать вывод об эффективности предложенной в работе оптимизации решения задачи коммивояжера.

1 Аналитический раздел

1.1 Задача коммивояжера

Задача коммивояжера (англ. The Travelling Salesman Problem) формулируется следующим образом [3]:

«Коммивояжер должен посетить N городов, побывав в каждом из них ровно по одному разу и завершив путешествие в том городе, с которого он начал. В какой последовательности ему нужно обходить города, чтобы общая длина его пути была наименьшей?»

Задача может быть смоделирована на ориентированном взвешенном графе согласно правилам, сформулированным ниже.

1) Город представлен вершиной графа;
2) Пути – это ребра графа;
3) Расстрояние – это вес ребра графа.

В целях упрощения задачи, стоит принять тот факт, что моделируемый граф является полностью связным, что обеспечивает наличие пути между двумя любыми произвольными городами. Таким образом, решение задачи – это нахождение Гамильтонова цикла минимального веса в полном взвешенном графе.

Совершенно очевидно, что задача может быть решена последовательным перебором всех возможных путей и выбором самого оптимального. Такой метод именуется как метод «грубой силы». Для N городов существует $\frac{(N-1)!}{2}$ различных путей. Факториал является чрезвычайно быстро растущей функцией — он растёт быстрее, чем любая показательная функция или любая степенная функция. Именно поэтому задача коммивояжёра интересна для тестирования различных алгоритмов.

1.2 Муравьиный алгоритм

Принципы поведения муравьиной колонии

Отдельного муравья нельзя назвать сообразительным существом. Все механизмы, обеспечивающие столь сложное поведение муравьиной колонии, обоснованы формированием роевого интеллекта – особи, составляющие колонию, исключают централизованное управление, основу их поведения составляет самоорганизация. Самоорганизация, в свою очередь, является результатом взаимодействия четырех компонентов [4]:

1) многократность;
2) положительная обратная связь;
3) отрицательная обратная связь;
4) случайность.

Муравьи для обмена информацией используют разнесенный во времени тип взаимодействия, при котором один субъект изменяет некоторый параметр окружающей среды, а осталь-

ные субъекты используют состояние этого параметра при нахождении в его окрестности [4]. Такой контакт называется стигмержи. Биологическим стигмержи для муравьиной колонии является феромон, оставляемый муравьем на земле как след своего перемещения.

Муравьиный алгоритм в решении задачи коммивояжера

Алгоритм, основанный на имитации природных механизмов самоорганизации муравьиной колонии, может быть применен в решении задачи коммивояжера. Для описания данного подхода к решению поставленной задачи, следует рассмотреть, каким образом реализуются компоненты социального взаимодействия в рамках выбранной математической модели.

1) **Многократность**. Один муравей рассматривается как самостоятельный коммивояжер, решающий задачу. Одновременно несколько муравьев занимаются поиском одного маршрута.

2) **Положительная обратная связь**. Муравей оставлет после перемещения след из феромона и перемещается по следу. Чем больше феромона оставлено на пути – ребре графа, тем больше муравьев будут перемещаться по этому пути.

3) **Отрицательная обратная связь**. Использование лишь положительной обратной связи может привести к тому, что алгоритм сведется в классический «жадный» алгоритм. Чтобы не допустить подобного случая, феромон имеет свойство испаряться.

4) **Случайность**. Вероятностное правило, которое гласит, что вероятность включения ребра графа в маршрут муравья пропорциональна количеству феромона на нем, гарантирует реализацию данного компонента взаимодействия.

Параметризация метода

Параметризация – это определение таких параметров, надстроек и возможных режимов работы метода, при котором задача будет решена с наилучшим качеством, согласно мерам оценки этого качества. При параметризации метода, основанного на алгоритме муравьиной колонии, вводится понятие муравья. Муравей обладает памятью, представленной списком посещенных за день городов, зрением и виртуальным следом феромона на ребре.

Зрение – это локальная статистическая величина, выражающая эвристическое желание посетить город j из города i [4]. Виртуальный след, в свою очередь, представлет подтвержденное опытом колонии желание посетить город j из города i. Вероятностное правило, определяющее вероятность перехода муравья k из города i в город j на итерации t определено согласно 1.1:

$$P_{ij,k}(t) = \begin{cases} \dfrac{[\tau_{i,j}(t)]^{\alpha} \cdot [\eta_{i,j}(t)]^{\beta}}{\sum\limits_{l \in J_{i,k}} [\tau_{i,l}(t)]^{\alpha} \cdot [\eta_{i,l}(t)]^{\beta}} & \text{если } j \in J_{i,k}, \\ 0 & \text{иначе.} \end{cases} \tag{1.1}$$

В формуле 1.1 приняты следующие обозначения. Список городов, которые еще не посещены муравьем k, который находится в городе i обозначен как $J_{i,k}$, η – видимость, величина, обратная расстоянию между городами i и j. Виртуальный след феромона на ребре графа (i,j) на итерации t обозначен как $\tau_{i,j}(t)$. α – регулируемый параметр, задающий вес следа феромона, β задает видимость при выборе маршрута.

Однако, правило 1.1 определяет лишь ширину зоны города j [5]: общая зона определяется по принципу "колеса рулетки": каждый город на ней имеет свой сектор с площадью, пропорциональной вероятности 1.1. Для выбора вбрасывается случайное число, и по нему определяется сектор, соответствующий городу.

После завершения маршрута муравей k должен оставить на ребре (i,j) количество феромона согласно следующему правилу 1.2:

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

Здесь $T_k(t)$ – маршрут, который был пройден муравьем k за итерацию t, $L_k(t)$ – длина этого маршрута, а Q – регулируемый параметр, имеющий значение порядка длины оптимального пути [?].

Испарение феромона обеспечивается правилом 1.3:

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

где m – количество муравьев в колонии.

В начале алгоритма количество феромона принимается равным небольшому положительному числу. Количество муравьев постоянно и равно количеству вершин графа.

Также для улучшения временных характеристик муравьиного алгоритма вводят так называемых элитных муравьев [4]. Элитный муравей усиливает ребра наилучшего маршрута, найденного с начала работы алгоритма. Количество феромона, откладываемого на ребрах наилучшего текущего маршрута T^+, принимается равным $\frac{Q}{L^+}$, где L^+ – длина маршрута T^+. Этот феромон побуждает муравьев к исследованию решений, содержащих несколько ребер наилучшего на данный момент маршрута. Если в муравейнике есть e элитных муравьев, то ребра маршрута T^+ будут получать общее усиление

$$\Delta\tau_e = e\cdot\frac{Q}{L^+}. \tag{1.4}$$

Вывод

Программное обеспечение, решающее поставленную задачу, может работать следующим образом. На вход алгоритму подается матрица смежности и описанные в разделе регулируемые параметры. Программа возвращает длину кратчайшего пути.

Моделируемый граф является полностью связным, что обеспечивает наличие пути между двумя произвольными городами.

Разрабатываемое программное обеспечение должно реализовывать два алгоритма – алгоритм муравьиной колонии и алгоритм «грубой силы». Таким образом, можно будет сравнить временные характеристики описанных в разделе методов решения поставленной задачи.

2 Констукторский раздел

В данном разделе будут рассмотрены схемы алгоритмов, требования к функциональности ПО, и определены способы тестирования.

2.1 Описание структур данных

Муравьиная колония представляет собой структуру данных, которая содержит информацию об абстракциях, представленных ниже.

1) Граф, с которым будет работать алгоритм. Сам граф представлен матрицей смежности.
2) Матрицу феромонов, содержащую количество феромона в каждом узле.
3) Количество дней, которое «проживет» колония.
4) Конфигурационную структуру, содержащую регулируемые параметры, описанные в разделе 1.

Муравей – структура, содержащая информацию о типе муравья, колонии, начальную и текущую позицию и матрицы пройденного пути и запретных городов. Матрицу запретных городов было решено заполнить логическими значениями. Матрица пройденного пути содержит расстояния между двумя соответствующими узлами.

2.2 Описание памяти, используемой алгоритмом

2.2.1 Алгоритм полного перебора

Оценка памяти рекурсивного алгоритма сводится к подсчету количества рекурсивных вызовов и объема работы, выполняемого для каждого вызова. В данном случае расчет производится по формуле 2.1:

$$M_{brute} = ((n \cdot lvar_{prem} + ret) \cdot depth + m \cdot lvar) \cdot \\ |graph| + size_{int} \cdot |graph| \tag{2.1}$$

где $depth$ – глубина стека вызовов, равная количеству перестановок при нахождении полного перебора, $|graph|$ – размер матрицы смежности, $lvar_{prem}$ – размер аллоцированных переменных в рекурсивной части функции и n, m – их количество. Аналогично, $lvar$ – размер аллоцированных в итеративной части функции. Выражение $size_{int} \cdot |graph|$ определяет размер временного массива для хранения текущего пути.

2.2.2 Муравьиный алгоритм

Память, используемая муравьиным алгоритмом, складывается из содержимого структуры колонии и содержимого структуры каждого муравья. Таким образом, формула принимает

вид 2.2:

$$
\begin{aligned}
&M_{ant-alg} = |graph| \cdot M_{ant} + M_{colony}, \text{ где:} \\
&M_{ant} = (size_{int} \cdot |graph|)^2 + (size_{bool} \cdot |graph|)^2 + 2 \cdot size_{int} + \\
&+ptr\left(M_{col}\right), \\
&M_{col} = (size_{int} \cdot |graph|)^2 + (size_{float} \cdot |graph|)^2 + size_{int} + \\
&+k \cdot size_{float}
\end{aligned}
\tag{2.2}
$$

Здесь M_{ant} – размер памяти используемый каждым муравьем в колонии, M_{col} – размер памяти, используемый колонией, $ptr\left(M_{col}\right)$ – размер указателя на структуру колонии. Остальные обозначения аналогичны 2.1.

2.3 Схемы алгоритмов

Ниже будут представлены схемы алгоритмов:

1) муравьиного алгоритма (рисунки 2.1, 2.2, 2.3);
2) алгоритма полного перебора (рисунок 2.4).

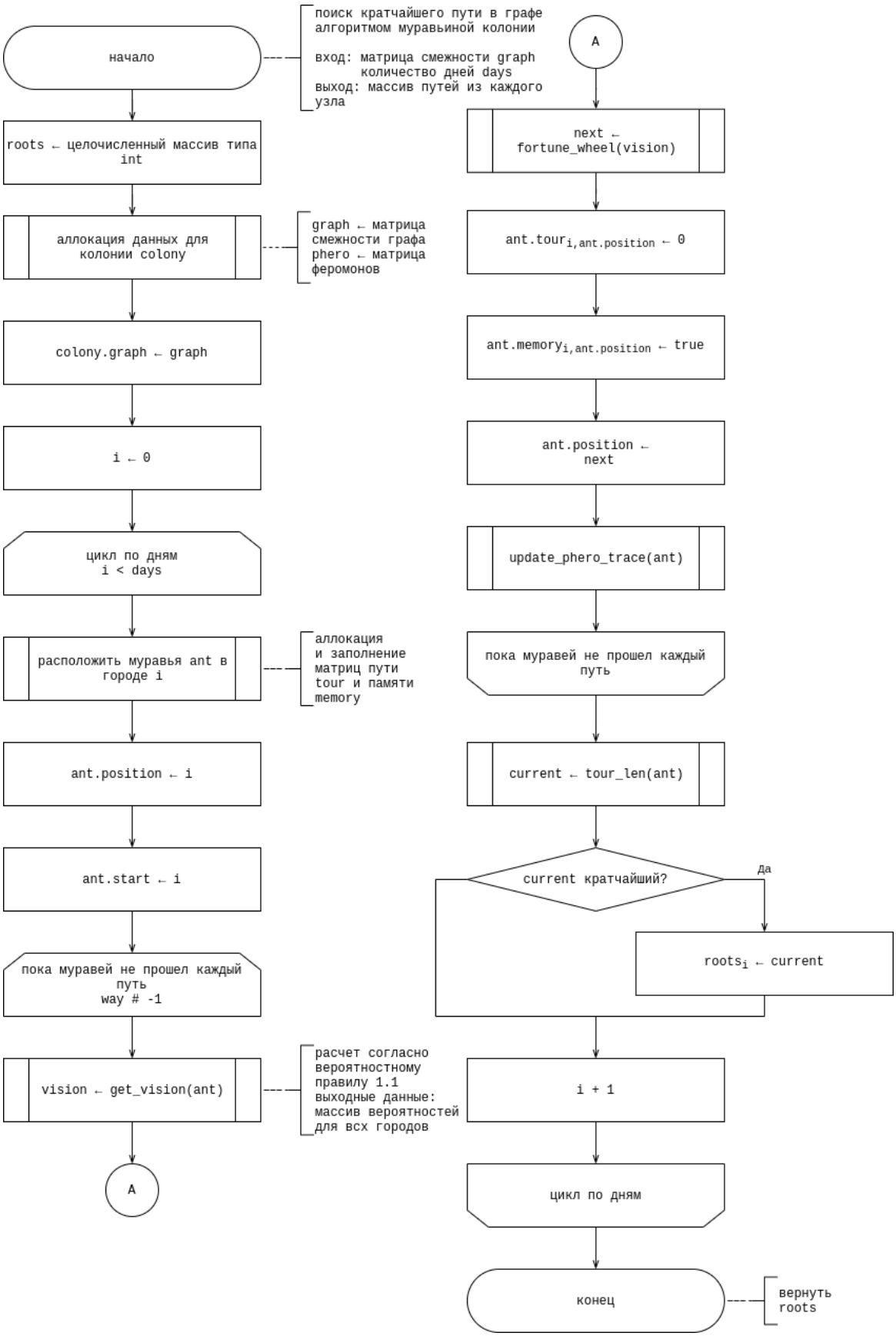

Рисунок 2.1 — Схема работы муравьиного алгоритма

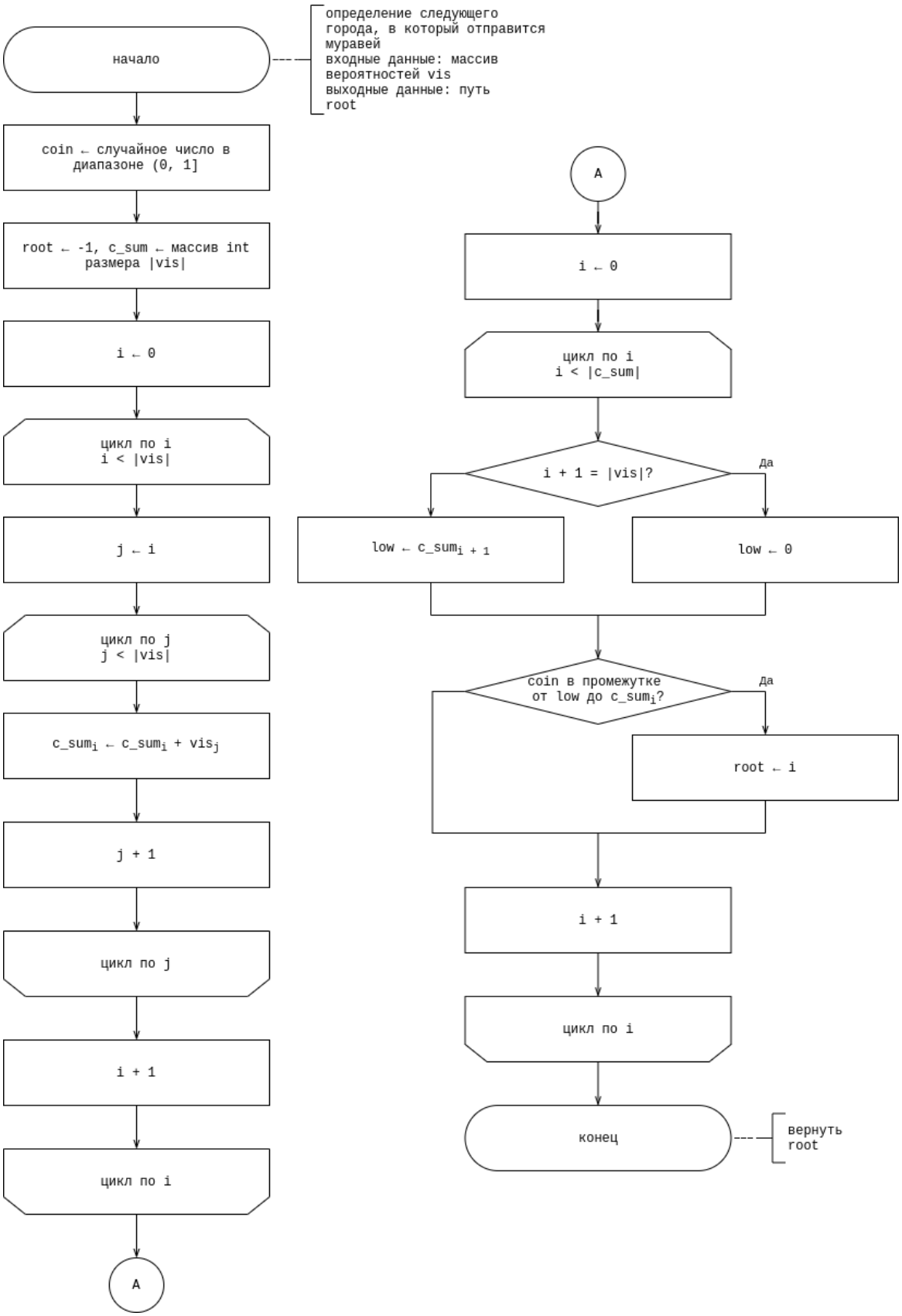

Рисунок 2.2 — Схема работы алгоритма случайного выбора города

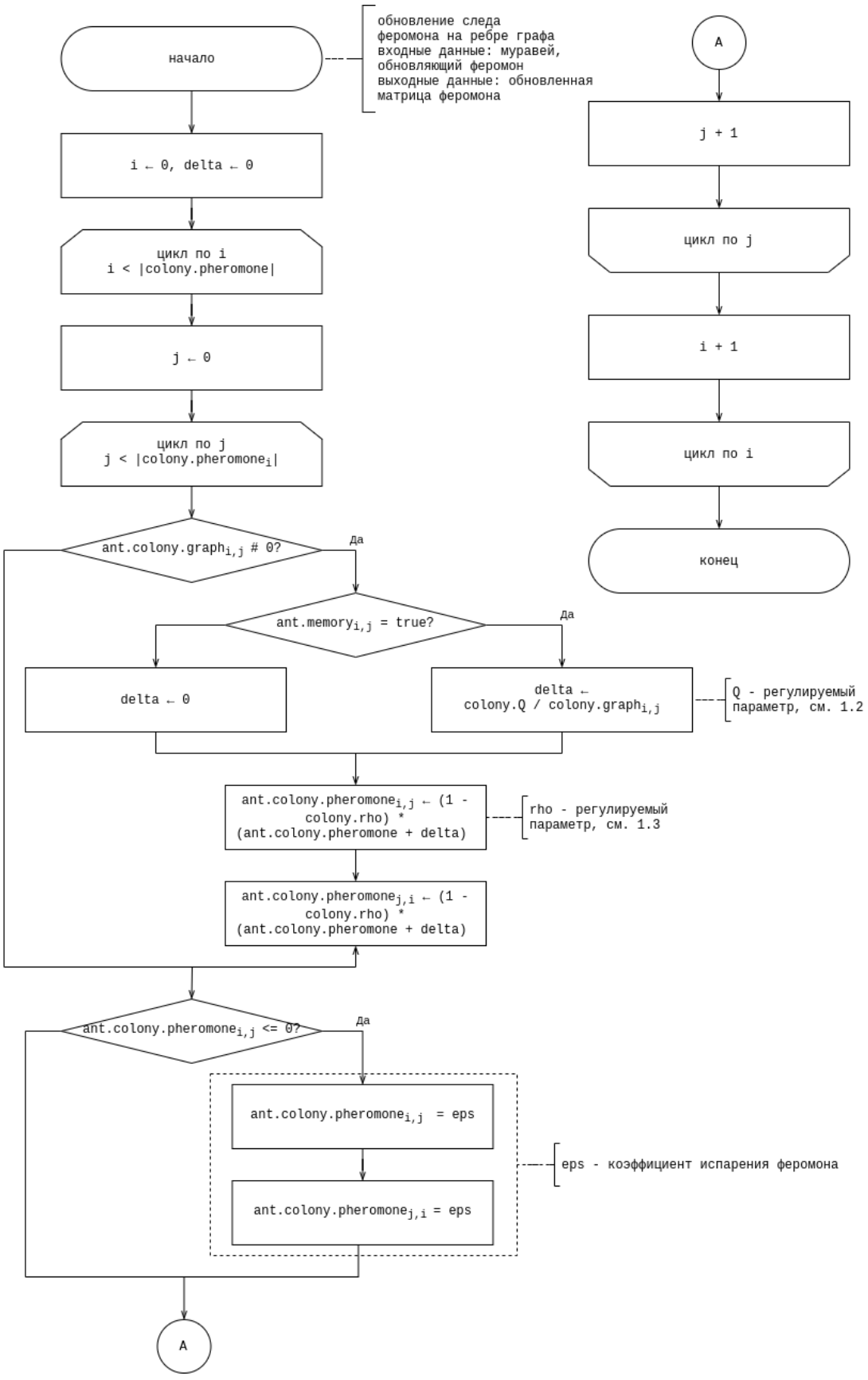

Рисунок 2.3 — Схема работы алгоритма обновления феромона

2.4 Структура ПО

Программа поделена на ряд смысловых модулей, описанных ниже:

— Модуль «brute», в котором содержится реализация алгоритма полного перебора;

— Модуль «aco», в котором содержится реализация алгоритма муравьиной колонии;

— Модуль «interf», в котором содержатся вспомогательные функции для пользовательского интерфейса.

Программа имеет консольный интерфейс.

Вывод

Были разработаны схемы алгоритмов, необходимых для решения задачи. Получено достаточно теоретической информации для написания программного обеспечения.

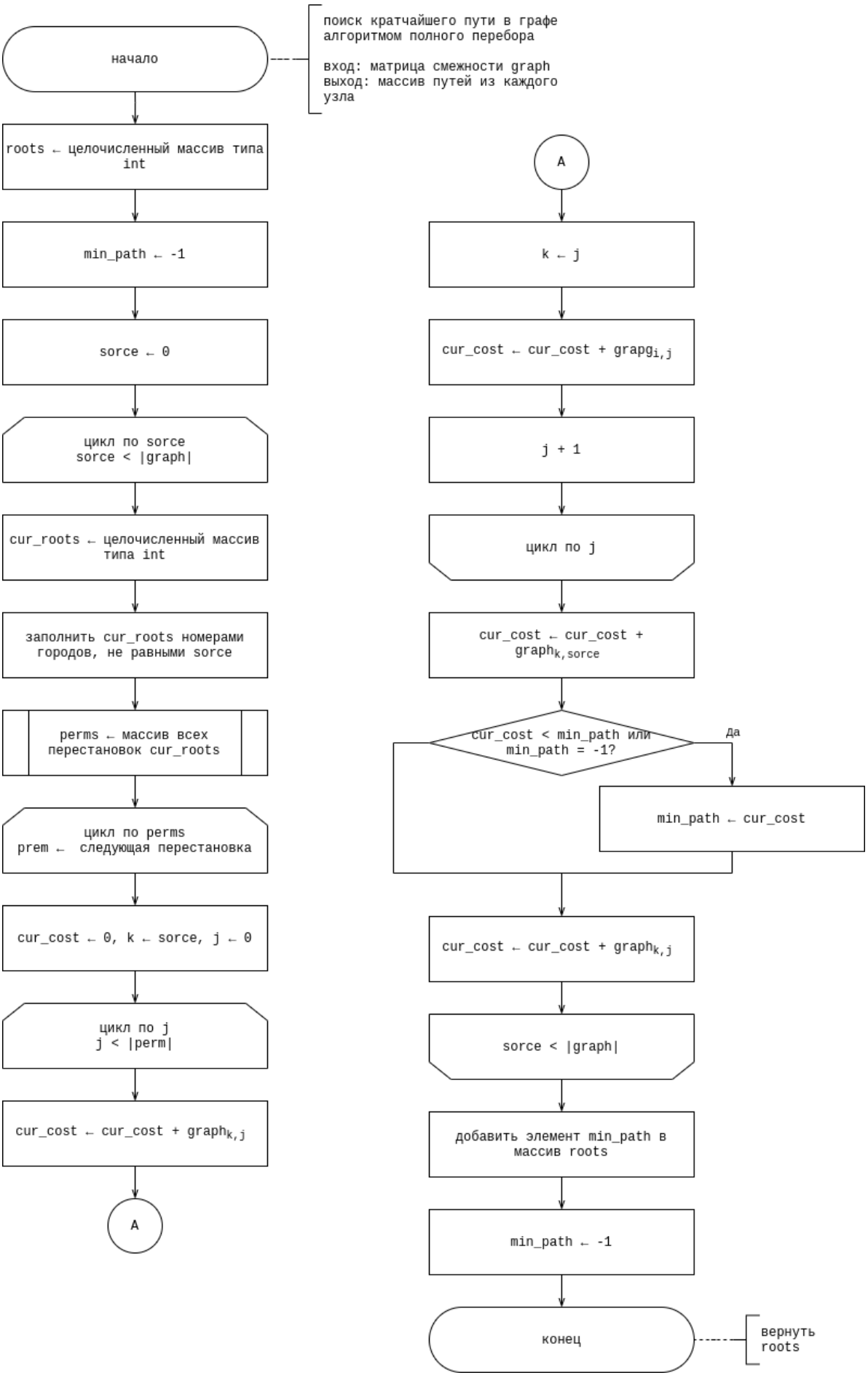

Рисунок 2.4 — Схема работы алгоритма "грубой силы"

3 Технологический раздел

В данном разделе представлены средства, использованные в процессе разработки для реализации задачи, а также листинг кода программы. Кроме того показаны результаты тестирования.

3.1 Требования к ПО

Программное обеспечение должно удовлетворять следующим требованиям:

— программа принимает на вход количество дней, матрицу смежности графа и регулируемые параметры;

— программа выдает массив кратчайших путей, выходящих из каждой вершины графа.

3.2 Средства реализации

Для реализации ПО был выбран компилируемый многопоточный язык программирования Golang, поскольку язык отличается легкой и быстрой сборкой программ и автоматическим управлением памяти.

3.3 Листинг программы

В приведенных ниже листингах представлены следующие реализации:

1) муравьиного алгоритма (листинги 3.1, 3.2, 3.3);
2) алгоритма полного перебора (листинги 3.7).

Листинг 3.1 — Реализация муравьиного алгоритма

```
1  func ACOWrapper(gr_matrix [][]int, iters int, conf_filename string) *[]int {
2      shortest := make([]int, len(gr_matrix))
3
4      colony := makeColony(gr_matrix, iters, conf_filename)
5      if (colony == nil) {
6          return nil
7      }
8
9      shortest_tours := make([][]int, len(gr_matrix))
10     for i := range gr_matrix {
11         shortest_tours[i] = make([]int, len(gr_matrix[i]))
12     }
13
14     for i := 0; i < colony.iters; i++ {
15         for j := 0; j < len(colony.gr_matrix) - 5; j++ {
16             ant := colony.placeAnt(j)
17             ant.elit_ant = true;
```

Листинг 3.2 — Реализация муравьиного алгоритма

```
            ant.startTour()
            current := ant.tourLength()
            if (current < shortest[j] || shortest[j] == 0) && len(ant.memory_tour)
               == len(colony.gr_matrix){
                shortest[j] = current
                colony.shortest_tour = ant.memory_tour
                copy(shortest_tours[j], ant.memory_tour)
            }
        }
        for j := len(colony.gr_matrix) - 5; j < len(colony.gr_matrix); j++ {
            ant := colony.placeAnt(j)
            ant.elit_ant = false
            ant.startTour()
            current := ant.tourLength()
            if (current < shortest[j] || shortest[j] == 0) && len(ant.memory_tour)
               == len(colony.gr_matrix){
                shortest[j] = current
                colony.shortest_tour = ant.memory_tour
                copy(shortest_tours[j], ant.memory_tour)
            }
        }
    }
    return &shortest
}
```

Листинг 3.3 — Реализация перемещений муравья

```
func (ant *Ant) startTour() {
    next := -1

    for flag := true; flag; flag = (next != -1) {
        vis := ant.getVision()
        next = fortuneWheel(vis)
        if (next != -1) {
            ant.tourAnt(next)
            ant.updatePhTrace()
        }
    }
}
```

Листинг 3.4 — Реализация функций для рассчета перемещений муравья

```
func (ant *Ant) getVision() []float64 {
    var (
        vis = make([]float64, 0)
        denom = 0.0
    )
    ant.memory_tour = append(ant.memory_tour,ant.position)
    for i, l := range ant.tour[ant.position] {
        if l != 0 {
            term := math.Pow((1.0 / float64(l)), ant.colony.conf.TRACE_WEIGHT) * +
                math.Pow(ant.colony.phero_matrix[ant.position][i],
                    ant.colony.conf.TOUR_VISIB)
            denom += term
            vis = append(vis, term)
        } else {
            vis = append(vis, 0)
        }
    }

    for i := 0; i < len(vis); i++ {
        vis[i] /= denom
    }

    return vis
}

func fortuneWheel(vis []float64) int {
    var (
        cumul_sum = make([]float64, len(vis))
        coin = rand.New(rand.NewSource(time.Now().UnixNano())).Float64()
        chosen = -1
    )

    for i := 0; i < len(vis); i++ {
        for j := i; j < len(vis); j++ {
            cumul_sum[i] += vis[j]
        }
    }
    for i := 0; i < len(cumul_sum); i++ {
        if i == len(cumul_sum) - 1 {
            if 0.0 <= coin && coin <= cumul_sum[i] {
                chosen = i
            }
        } else {
            if cumul_sum[i + 1] < coin && coin <= cumul_sum[i] {
                    chosen = i
```

Листинг 3.5 — Реализация функций для рассчета перемещений муравья

```
            }
        }
    }

    return chosen
}
```

Листинг 3.6 — Реализация функции обновления феромона

```
func (ant *Ant) updatePhTrace() {
    delta := 0.0
    for i := 0; i < len(ant.colony.phero_matrix); i++ {
        for j, phero := range ant.colony.phero_matrix[i] {
            if ant.colony.gr_matrix[i][j] != 0 {
                if ant.memory[i][j] {
                    delta = ant.colony.conf.Q / float64(ant.colony.gr_matrix[i][j])
                } else {
                    delta = 0
                }
                ant.colony.phero_matrix[i][j] = (1 - ant.colony.conf.EVAP_RATE) *
                   (float64(phero) + delta)
                ant.colony.phero_matrix[j][i] = (1 - ant.colony.conf.EVAP_RATE) *
                   (float64(phero) + delta)
            }
            if ant.colony.phero_matrix[i][j] <= 0 {
                ant.colony.phero_matrix[i][j] = ant.colony.conf.PHERO_EPS
                ant.colony.phero_matrix[j][i] = ant.colony.conf.PHERO_EPS
            }
        }
    }
    if ant.elit_ant == true && len(ant.colony.shortest_tour) != 0 {
        L := 0
        for i:= 0; i < len(ant.colony.shortest_tour) - 1; i++ {
            L += ant.colony.gr_matrix[ant.colony.shortest_tour[i]]
            [ant.colony.shortest_tour[i+1]]
        }
        for i:= 0; i < len(ant.colony.shortest_tour) - 1; i++ {
            ant.colony.phero_matrix[ant.colony.shortest_tour[i]]
            [ant.colony.shortest_tour[i + 1]] += ant.colony.conf.Q / float64(L)
        }
        ant.colony.phero_matrix[ant.colony.shortest_tour[len(ant.colony.shortest_tour)
           - 1]][ant.colony.shortest_tour[0]] += ant.colony.conf.Q / float64(L)
    }
}
```

Листинг 3.7 — Реализация алгоритма полного перебора

```
func BruteForce(gr_matrix [][]int) []int {
    var (
        roots = make([]int, 0)
        graph_len = len(gr_matrix)
        min_path = -1
    )

    for source := 0; source < graph_len; source++ {
        cur_roots := make([]int, 0)
        for i := 0; i < graph_len; i++ {
            if i != source {
                cur_roots = append(cur_roots, i)
            }
        }
        next_permutation := permutations(cur_roots)
        for _, perm := range next_permutation {
            curr_cost := 0
            k := source
            flag := true

            for _, j := range perm {
                curr_cost += gr_matrix[k][j]
                if gr_matrix[k][j] == 0{
                    flag = false
                }
                k = j
            }

            if gr_matrix[k][source] == 0 {
                flag = false
            }
            curr_cost += gr_matrix[k][source]

            if (curr_cost < min_path || min_path == -1) && curr_cost != 0 && flag ==
                true{
                min_path = curr_cost
            }
        }

        roots = append(roots, min_path)
        min_path = -1
    }
    return roots
}
```

Автоматическая параметризация

Скрипт автоматической перестановки генерирует все комбинации параметров в заданном диапазоне и помещает в конфигурационный файл. Функция генерации параметров имеет следующий вид (3.8):

Листинг 3.8 — Реализация перемещений муравья

```
def form_coefs():
    coefsgen = []
    for i in range(11):
        stack = []
        alpha = round(0.0 + 0.1 * i, 2)
        betta = round(1 - alpha, 2)
        # stack.append([alpha, betta])

        stack.append([round(0.0 + 0.1 * j, 2) for j in range(11)])
        stack.append([3])          # q
        stack.append([0.5])        # init
        stack.append([0.7])        # eps
    return stack
```

3.4 Тестирование ПО

Результаты тестирования ПО приведены в таблице 3.1.

Таблица 3.1 — Тестирование ПО

Матрица	*Полный перебор*	*Муравьиный алгоритм*
0 3 6 1 1 0 2 1 2 1 0 2 1 2 3 0	6 6 6 6	6 6 6 6
0 3 7 5 4 1 0 3 2 1 4 1 0 1 1 3 1 2 0 1 2 1 3 2 0	10 10 10 10 10	10 10 10 10 10

Вывод

Было написано и протестировано программное обеспечение для решения поставленной задачи.

4 Исследовательский раздел

Раздел содержит описание классов данных для которых проведена параметризация, технические характеристики устройства, на котором проведен эксперимент. Также раздел содержит результаты параметризации алгоритма муравьиной колонии на выбранных классах данных.

4.1 Технические характеристики

Тестирование выполнялось на устройстве со следующими техническими характеристиками:

— операционная система: macOS Ventura 13.0;
— оперативная память: 8 Gb;
— процессор: Apple M1.

4.2 Постановка эксперимента

Эксперимент проведен на матрицах 4.1 и 4.2 типа int. Количество муравьев фиксировано и равно размеру матрицы смежности. Проведенный эксперимент определяет комбинацию параметров, решающих задачу с наилучшим качеством. Качественная оценка работы алгоритма на определенном наборе зависит от двух критериев: количество дней жизни муравьиной колонии и погрешность результата. Для каждого набора параметров поведен один замер. Данные не усреднялись.

После параметризации были проведены временные замеры выполнения алгоритмов. Для каждого набора параметров было проведено 100 замеров.

Во время тестирования устройство было подключено к блоку питания и не нагружено никакими приложениями, кроме встроенных приложений окружения, окружением и системой тестирования. Оптимизация компилятора была отключена.

4.3 Класс данных №1

Класс данных №1 описан согласно следующей матрице смежности (4.1):

$$\begin{Vmatrix} 0 & 5 & 8 & 2 & 6 & 9 & 6 & 7 & 8 & 9 \\ 2 & 0 & 7 & 5 & 7 & 8 & 7 & 3 & 4 & 3 \\ 4 & 5 & 0 & 7 & 1 & 3 & 1 & 4 & 2 & 5 \\ 5 & 8 & 9 & 0 & 6 & 9 & 6 & 5 & 8 & 3 \\ 4 & 3 & 3 & 3 & 0 & 3 & 3 & 2 & 7 & 2 \\ 7 & 5 & 5 & 6 & 7 & 0 & 7 & 5 & 8 & 9 \\ 3 & 2 & 3 & 2 & 1 & 3 & 0 & 1 & 2 & 4 \\ 4 & 1 & 6 & 3 & 5 & 2 & 3 & 0 & 3 & 4 \\ 1 & 1 & 2 & 4 & 2 & 5 & 1 & 7 & 0 & 3 \\ 5 & 6 & 1 & 2 & 3 & 4 & 2 & 1 & 1 & 0 \end{Vmatrix} \tag{4.1}$$

Результаты работы алгоритма с различными комбинациями представлены в приложении А. Таблица 4.1 содержит выборку параметров, решающих задачу с наилучшим, насколько это возможно, качеством.

Таблица 4.1 — Комбинации параметров

α	β	ρ	N	рез.	погр.
0.0	1.0	0.9	11	21	0
0.1	0.9	0.6	11	21	0
0.0	1.0	0.0	11	21	0
0.3	0.7	0.0	11	21	0
0.7	0.3	1.0	11	21	0
0.9	0.1	0.0	11	21	0
0.9	0.1	1.0	11	21	0
1.0	0.0	0.8	11	21	0
0.1	0.9	0.0	12	21	0
0.8	0.2	0.0	12	21	0
0.8	0.2	0.1	13	21	0
0.9	0.1	0.2	13	21	0
1.0	0.0	0.4	13	21	0
0.4	0.6	1.0	14	21	0
0.4	0.6	0.0	15	21	0

4.4 Класс данных №2

Класс данных №1 описан согласно следующей матрице смежности (4.2):

$$\begin{Vmatrix} 0 & 5 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 1 & 0 & 6 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 1 & 2 & 0 & 7 & 8 & 9 & 10 & 11 & 12 & 13 \\ 1 & 2 & 3 & 0 & 8 & 9 & 10 & 11 & 12 & 13 \\ 1 & 2 & 3 & 4 & 0 & 9 & 10 & 11 & 12 & 13 \\ 1 & 2 & 3 & 4 & 5 & 0 & 10 & 11 & 12 & 13 \\ 1 & 2 & 3 & 4 & 5 & 6 & 0 & 11 & 12 & 13 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 0 & 12 & 13 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 0 & 13 \\ 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 0 \end{Vmatrix} \tag{4.2}$$

Результаты работы алгоритма с различными комбинациями представлены в приложении Б. Таблица 4.2 содержит выборку параметров, решающих задачу с наилучшим, насколько это возможно, качеством.

Таблица 4.2 — Комбинации параметров

α	β	ρ	N	рез.	погр.
0.2	0.8	0.6	12	58	0
1.0	0.0	0.0	12	58	0
0.1	0.9	0.0	13	58	0
0.9	0.1	0.0	13	58	0
0.4	0.6	0.0	15	58	0
0.5	0.5	1.0	15	58	0
0.7	0.3	1.0	15	58	0
0.8	0.2	0.0	15	58	0
0.3	0.7	0.0	16	58	0
0.6	0.4	0.0	16	58	0
1.0	0.0	0.4	16	58	0
0.2	0.8	0.0	17	58	0
1.0	0.0	0.5	17	58	0
0.7	0.3	0.9	18	58	0
1.0	0.0	0.9	19	58	0

4.5 Результаты эксперимента

Ниже приведена выборка значений, решающих задачу на двух классах данных с наилучшим качеством (4.3):

Таблица 4.3 — Комбинации параметров

α	β	ρ	N		рез.		погр.	
	класс данных		1	2	1	2	1	2
0.3	0.7	0.0	11	16	21	58	0	0
0.7	0.3	1.0	11	15	21	58	0	0
0.9	0.1	0.0	11	13	21	58	0	0
0.1	0.9	0.0	12	13	21	58	0	0
0.8	0.2	0.0	12	15	21	58	0	0
0.8	0.2	0.1	13	12	21	58	0	0
1.0	0.0	0.4	13	16	21	58	0	0
0.4	0.6	0.0	15	15	21	58	0	0

Для временных замеров была взята следующая конфигурация: $\alpha = 0.8$, $\beta = 0.2$, $\rho = 0.1$, $N = 15$. Замеры показали, что на матрицах 4.1 и 4.2 алгоритм муравьиной колонии решает поставленную задачу примерно в 400 раз быстрее, чем алгоритм полного перебора.

4.6 Вывод

Была проведена параметризация алгоритма для двух классов данных. Оптимизация задачи коммивояжера алгоритмом муравьиной колонии показывает следующий результат: при $\alpha = 0.8$, $\beta = 0.2$, $\rho = 0.1$, $N = 15$ поставленная задача будет решена в 400 раз быстрее.

Заключение

В ходе лабораторной работы были реализованы алгоритмы полного перебора и муравьиной колонии, были разработаны два класса эквивалентности для параметризации муравьиного алгоритма и была проведена параметризация. Был проведен сравнительный анализ алгоритмов.

Муравьиные алгоритмы обеспечивают качественное, насколько это возможно, решение комбинаторных задач, для которых не существует быстрых полиномиальных решений. На исследуемых классах данных алгоритм показал оптимизацию в 400 раз. Проведенное исследование позволяет рекомендовать применение муравьиного алгоритма вместо алгоритма полного перебора для решения задачи коммивояжера, хотя алгоритм полного перебора является более универсальным — для него не нужно проводить параметризацию, он дает гарантированно точный результат, хоть и за большее время. Опираясь на проведенное исследование, можно сделать вывод, что если критерием оценки задачи коммивояжера является универсальность, то следует использовать алгоритм полного перебора. Однако, если задача должна быть решена на определенном классе данных и критерием оценки является скорость выполнения, то для решения стоит выбрать алгоритм муравьиной колонии.

Список источников

1) В.С. Титов Э.И. Ватутин. “Анализ результатов применения алгоритма муравьиной колонии в задаче поиска пути в графе при наличии ограничений”. в: Математическое и программное обеспечение суперкомпьютеров;

2) Акименко А. С. “Метод ACO для решения задач оптимизации”. в: Вестник науки и образования (2018).;

3) E. L. Lawler. The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization. 1985.

4) Штовба С.Д. “Муравьиные алгоритмы”. в: Exponenta Pro. Математика в приложениях (2003).

5) ДМ. В. Ульянов. Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ. под ред. В. С. Аролович. Физматлит, 2008.

6) Rune in Golang. [Электронный ресурс]. Режим доступа: https://www.geeksforgeeks.org/rune-in-golang/ (дата обращения: 10.12.2022).

Приложение А

Таблица 1 – Параметризация класса данных №1

α	β	ρ	N	рез.	погр.
0.0	1.0	0.0	11	39	0
0.0	1.0	0.1	196	43	4
0.0	1.0	0.2	188	43	4
0.0	1.0	0.3	160	43	4
0.0	1.0	0.4	416	43	4
0.0	1.0	0.5	145	39	0
0.0	1.0	0.6	106	39	0
0.0	1.0	0.7	208	39	0
0.0	1.0	0.8	96	39	0
0.0	1.0	0.9	78	39	0
0.0	1.0	1.0	76	39	0
0.1	0.9	0.0	12	39	0
0.1	0.9	0.1	224	42	3
0.1	0.9	0.2	200	43	4
0.1	0.9	0.3	208	39	0
0.1	0.9	0.4	285	42	3
0.1	0.9	0.5	274	39	0
0.1	0.9	0.6	71	39	0
0.1	0.9	0.7	175	39	0
0.1	0.9	0.8	68	39	0
0.1	0.9	0.9	94	39	0
0.1	0.9	1.0	116	39	0
0.2	0.8	0.0	19	39	0
0.2	0.8	0.1	202	40	1
0.2	0.8	0.2	380	40	1
0.2	0.8	0.3	146	43	4
0.2	0.8	0.4	22	41	2
0.2	0.8	0.5	99	39	0
0.2	0.8	0.6	58	39	0
0.2	0.8	0.7	212	39	0
0.2	0.8	0.8	88	39	0
0.2	0.8	0.9	78	39	0
0.2	0.8	1.0	20	39	0

Продолжение таблицы №1

α	β	ρ	N	рез.	погр.
0.3	0.7	0.0	11	39	0
0.3	0.7	0.1	244	41	2
0.3	0.7	0.2	326	41	2
0.3	0.7	0.3	169	42	3
0.3	0.7	0.4	128	42	3
0.3	0.7	0.5	133	39	0
0.3	0.7	0.6	89	39	0
0.3	0.7	0.7	31	39	0
0.3	0.7	0.8	127	39	0
0.3	0.7	0.9	40	39	0
0.3	0.7	1.0	76	39	0
0.4	0.6	0.0	15	39	0
0.4	0.6	0.1	72	39	0
0.4	0.6	0.2	14	41	2
0.4	0.6	0.3	142	41	2
0.4	0.6	0.4	236	41	2
0.4	0.6	0.5	93	39	0
0.4	0.6	0.6	52	39	0
0.4	0.6	0.7	116	39	0
0.4	0.6	0.8	72	39	0
0.4	0.6	0.9	120	39	0
0.4	0.6	1.0	14	39	0
0.5	0.5	0.0	26	39	0
0.5	0.5	0.1	233	39	0
0.5	0.5	0.2	427	40	1
0.5	0.5	0.3	88	40	1
0.5	0.5	0.4	395	40	1
0.5	0.5	0.5	63	39	0
0.5	0.5	0.6	83	39	0
0.5	0.5	0.7	103	39	0
0.5	0.5	0.8	113	39	0
0.5	0.5	0.9	58	39	0
0.5	0.5	1.0	26	39	0

Продолжение таблицы №1

α	β	ρ	N	рез.	погр.
0.6	0.4	0.0	26	39	0
0.6	0.4	0.1	267	39	0
0.6	0.4	0.2	228	39	0
0.6	0.4	0.3	316	39	0
0.6	0.4	0.4	14	41	2
0.6	0.4	0.5	143	39	0
0.6	0.4	0.6	135	39	0
0.6	0.4	0.7	55	39	0
0.6	0.4	0.8	88	39	0
0.6	0.4	0.9	108	39	0
0.6	0.4	1.0	22	39	0
0.7	0.3	0.0	16	39	0
0.7	0.3	0.1	39	39	0
0.7	0.3	0.2	427	39	0
0.7	0.3	0.3	283	39	0
0.7	0.3	0.4	247	40	1
0.7	0.3	0.5	98	39	0
0.7	0.3	0.6	44	39	0
0.7	0.3	0.7	48	39	0
0.7	0.3	0.8	82	39	0
0.7	0.3	0.9	26	39	0
0.7	0.3	1.0	11	39	0
0.8	0.2	0.0	12	39	0
0.8	0.2	0.1	13	39	0
0.8	0.2	0.2	65	39	0
0.8	0.2	0.3	366	39	0
0.8	0.2	0.4	41	41	2
0.8	0.2	0.5	75	39	0
0.8	0.2	0.6	20	39	0
0.8	0.2	0.7	58	39	0
0.8	0.2	0.8	99	39	0
0.8	0.2	0.9	37	39	0
0.8	0.2	1.0	28	39	0

Продолжение таблицы №1

α	β	ρ	N	рез.	погр.
0.9	0.1	0.0	11	39	0
0.9	0.1	0.1	10	39	0
0.9	0.1	0.2	13	39	0
0.9	0.1	0.3	32	39	0
0.9	0.1	0.4	10	39	0
0.9	0.1	0.5	17	39	0
0.9	0.1	0.6	106	39	0
0.9	0.1	0.7	16	39	0
0.9	0.1	0.8	58	39	0
0.9	0.1	0.9	48	39	0
0.9	0.1	1.0	11	39	0
1.0	0.0	0.0	41	39	0
1.0	0.0	0.1	18	39	0
1.0	0.0	0.2	22	39	0
1.0	0.0	0.3	18	39	0
1.0	0.0	0.4	13	39	0
1.0	0.0	0.5	18	39	0
1.0	0.0	0.6	23	39	0
1.0	0.0	0.7	27	39	0
1.0	0.0	0.8	11	39	0
1.0	0.0	0.9	26	39	0
1.0	0.0	1.0	25	39	0

Приложение Б

Таблица 1 – Параметризация класса данных №2

α	β	ρ	N	рез.	погр.
0.0	1.0	0.0	42	300	0
0.0	1.0	0.1	59	323	23
0.0	1.0	0.2	249	326	26
0.0	1.0	0.3	172	312	12
0.0	1.0	0.4	199	351	51
0.0	1.0	0.5	186	300	0
0.0	1.0	0.6	159	300	0
0.0	1.0	0.7	152	300	0
0.0	1.0	0.8	199	300	0
0.0	1.0	0.9	132	300	0
0.0	1.0	1.0	214	300	0
0.1	0.9	0.0	13	300	0
0.1	0.9	0.1	43	323	23
0.1	0.9	0.2	96	313	13
0.1	0.9	0.3	90	307	7
0.1	0.9	0.4	179	325	25
0.1	0.9	0.5	227	300	0
0.1	0.9	0.6	251	300	0
0.1	0.9	0.7	148	300	0
0.1	0.9	0.8	196	300	0
0.1	0.9	0.9	66	300	0
0.1	0.9	1.0	65	300	0
0.2	0.8	0.0	17	300	0
0.2	0.8	0.1	289	323	23
0.2	0.8	0.2	419	317	17
0.2	0.8	0.3	98	323	23
0.2	0.8	0.4	270	324	24
0.2	0.8	0.5	234	300	0
0.2	0.8	0.6	248	300	0
0.2	0.8	0.7	267	300	0
0.2	0.8	0.8	320	300	0
0.2	0.8	0.9	53	300	0
0.2	0.8	1.0	238	300	0

Продолжение таблицы №1

α	β	ρ	N	рез.	погр.
0.3	0.7	0.0	16	300	0
0.3	0.7	0.1	17	312	12
0.3	0.7	0.2	82	333	33
0.3	0.7	0.3	288	316	16
0.3	0.7	0.4	401	320	20
0.3	0.7	0.5	276	300	0
0.3	0.7	0.6	105	300	0
0.3	0.7	0.7	192	300	0
0.3	0.7	0.8	274	300	0
0.3	0.7	0.9	83	300	0
0.3	0.7	1.0	67	300	0
0.4	0.6	0.0	15	300	0
0.4	0.6	0.1	23	304	4
0.4	0.6	0.2	47	307	7
0.4	0.6	0.3	164	313	13
0.4	0.6	0.4	439	323	23
0.4	0.6	0.5	345	300	0
0.4	0.6	0.6	213	300	0
0.4	0.6	0.7	74	300	0
0.4	0.6	0.8	131	300	0
0.4	0.6	0.9	118	300	0
0.4	0.6	1.0	128	300	0
0.5	0.5	0.0	29	300	0
0.5	0.5	0.1	236	300	0
0.5	0.5	0.2	486	310	10
0.5	0.5	0.3	457	314	14
0.5	0.5	0.4	428	304	4
0.5	0.5	0.5	340	300	0
0.5	0.5	0.6	144	300	0
0.5	0.5	0.7	236	300	0
0.5	0.5	0.8	54	300	0
0.5	0.5	0.9	127	300	0
0.5	0.5	1.0	15	300	0

Продолжение таблицы №1

α	β	ρ	N	рез.	погр.
0.6	0.4	0.0	16	300	0
0.6	0.4	0.1	207	300	0
0.6	0.4	0.2	105	304	4
0.6	0.4	0.3	94	311	11
0.6	0.4	0.4	148	312	12
0.6	0.4	0.5	178	300	0
0.6	0.4	0.6	182	300	0
0.6	0.4	0.7	23	300	0
0.6	0.4	0.8	77	300	0
0.6	0.4	0.9	53	300	0
0.6	0.4	1.0	57	300	0
0.7	0.3	0.0	37	300	0
0.7	0.3	0.1	42	300	0
0.7	0.3	0.2	303	300	0
0.7	0.3	0.3	98	308	8
0.7	0.3	0.4	492	307	7
0.7	0.3	0.5	76	300	0
0.7	0.3	0.6	89	300	0
0.7	0.3	0.7	85	300	0
0.7	0.3	0.8	107	300	0
0.7	0.3	0.9	18	300	0
0.7	0.3	1.0	15	300	0
0.8	0.2	0.0	15	300	0
0.8	0.2	0.1	12	300	0
0.8	0.2	0.2	80	300	0
0.8	0.2	0.3	65	304	4
0.8	0.2	0.4	130	315	15
0.8	0.2	0.5	141	300	0
0.8	0.2	0.6	154	300	0
0.8	0.2	0.7	94	300	0
0.8	0.2	0.8	58	300	0
0.8	0.2	0.9	56	300	0
0.8	0.2	1.0	58	300	0

Продолжение таблицы №1

α	β	ρ	N	рез.	погр.
0.9	0.1	0.0	13	300	0
0.9	0.1	0.1	25	300	0
0.9	0.1	0.2	36	300	0
0.9	0.1	0.3	20	300	0
0.9	0.1	0.4	42	300	0
0.9	0.1	0.5	41	300	0
0.9	0.1	0.6	32	300	0
0.9	0.1	0.7	21	300	0
0.9	0.1	0.8	103	300	0
0.9	0.1	0.9	56	300	0
0.9	0.1	1.0	57	300	0
1.0	0.0	0.0	12	300	0
1.0	0.0	0.1	82	300	0
1.0	0.0	0.2	37	300	0
1.0	0.0	0.3	40	300	0
1.0	0.0	0.4	16	300	0
1.0	0.0	0.5	17	300	0
1.0	0.0	0.6	48	300	0
1.0	0.0	0.7	30	300	0
1.0	0.0	0.8	23	300	0
1.0	0.0	0.9	19	300	0
1.0	0.0	1.0	30	300	0