



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
ИМЕНИ Н.Э. БАУМАНА
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)
(МГТУ им. Н.Э. БАУМАНА)

ФАКУЛЬТЕТ _____ «Информатика и системы управления»

КАФЕДРА _____ «Программное обеспечение ЭВМ и информационные технологии»

НАПРАВЛЕНИЕ ПОДГОТОВКИ _____ «09.03.04 Программная инженерия»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

Название: _____ Параметризация задачи. Муравьиный алгоритм.

Дисциплина: _____ Анализ алгоритмов

Студент	ИУ7-56Б	_____	Т. А. Казаева
	Группа	Подпись, дата	И. О. Фамилия

Преподаватель	_____	Л. Л. Волкова
	Подпись, дата	И. О. Фамилия

Москва, 2021 г.

Содержание

	Страница
Введение	2
1 Аналитический раздел	3
1.1 Задача коммивояжера	3
1.2 Муравьиный алгоритм	4
Принципы поведения муравьиной колонии	4
Муравьиный алгоритм в решении задачи коммивояжера . .	5
Параметризация метода	5
1.3 Вывод	7
2 Конструкторский раздел	8
2.1 Описание структур данных	8
2.2 Описание памяти, используемой алгоритмом	8
Алгоритм полного перебора	8
Муравьиный алгоритм	9
2.3 Описание работы алгоритмов	9
2.4 Выделение классов эквивалентности	14
2.5 Структура ПО	14
Вывод	14
3 Технологический раздел	15
3.1 Требования к ПО	15
3.2 Средства реализации	15
3.3 Листинги кода	15
Муравьиный алгоритм	15
Алгоритм полного перебора	21

Автоматическая параметризация	22
3.4 Тестирование ПО	23
3.5 Вывод	23
4 Исследовательский раздел	24
4.1 Технические характеристики	24
4.2 Постановка эксперимента	24
4.3 Класс данных №1	25
4.4 Класс данных №2	26
4.5 Результаты эксперимента	27
4.6 Вывод	27
Заключение	28
Список литературы	29
Приложение А. Таблица параметризации для класса данных №1 . .	30
Приложение Б. Таблица параметризации для класса данных №2 . .	32

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. Целью данной работы является анализ предложенной оптимизации и параметризация алгоритма для двух классов данных. Для достижения поставленной цели необходимо выполнить следующие задачи:

- реализовать алгоритм полного перебора и алгоритм муравьиной колонии;
- разработать два класса эквивалентности для параметризации алгоритма муравьиной колонии;
- провести параметризацию для разработанных классов эквивалентности;
- сравнить алгоритмы, согласно следующим параметрам:
 - * качество полученного решения;
 - * 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

В данном разделе представлено описание задачи коммивояжера, сформулирована и формализована математическая модель для решения задачи коммивояжера методом "грубой силы" и методом муравьиной колонии.

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 симметричном неориентированном взвешенном графе согласно правилам, сформулированным ниже.

1. Город представлен вершиной графа.
2. Пути – это ребра графа.
3. Расстояние – это вес ребра графа.
4. Граф симметричен, значит, расстояние между двумя городами одинаково во всех противоположных направлениях.

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]:

- многократность;
- положительная обратная связь;
- отрицательная обратная связь;
- случайность.

Муравьи для обмена информацией используют разнесенный во времени тип взаимодействия, при котором один субъект изменяет некоторый параметр окружающей среды, а остальные 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. *Множественность.* Один муравей рассматривается как самостоятельный коммивояжер, решающий задачу. Одновременно несколько муравьев занимаются поиском одного маршрута.
2. *Положительная обратная связь.* Муравей оставляет 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, представляет 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:

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

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:

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

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 [5].

Испарение феромона обеспечивается правилом 1.3:

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

где m – количество муравьев в колонии.

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

1.3 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. Расстояние между двумя любыми городами симметрично во всех направлениях.

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. Конструкторский раздел

Раздел содержит описание работы алгоритмов и обоснование структур данных, выбранных при их реализации, оценку используемой памяти, описание системы тестирования и выделение классов эквивалентности для 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 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

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:

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

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.

Муравьиный алгоритм

Память, используемая муравьиным алгоритмом, складывается из содержимого структуры колонии и содержимого структуры каждого муравья. Таким образом, формула принимает вид 2.2:

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

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

2.3 Описание работы алгоритмов

На рисунке 2.1 представлена схема работы муравьиного алгоритма. Обращение к элементам массивов на схеме алгоритмов обозначено подстрочными индексами. Операция присваивания обозначена как " \leftarrow " операции "равно" и "не равно" как " $=$ " и " \neq " соответственно.

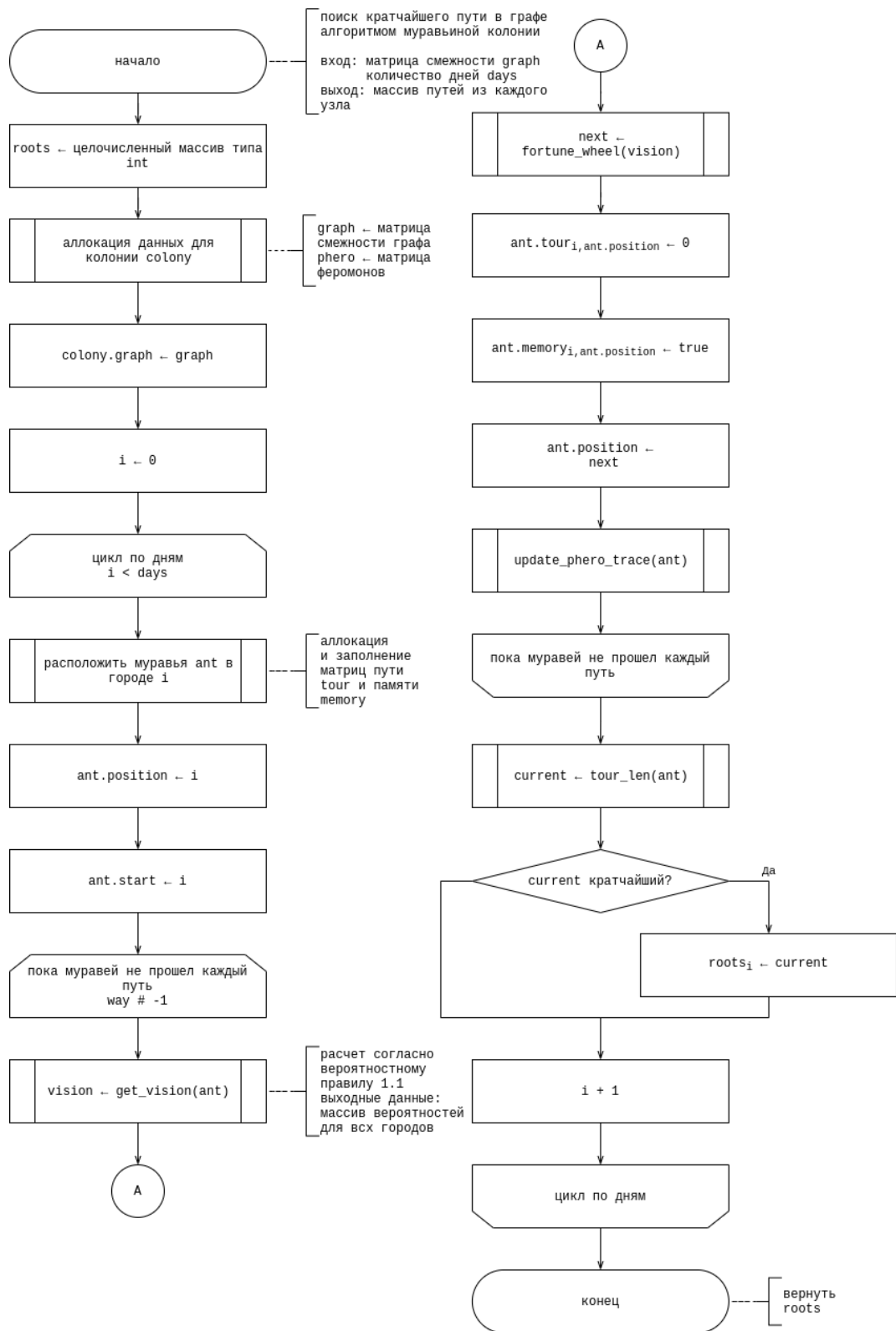


Рисунок 2.1 – Схема работы муравьиного алгоритма

На рисунке 2.1 представлена схема работы алгоритма случайного выбора города. Оператор взятия длины массива обозначен как $|array|$. Остальные обозначения аналогично 2.1.

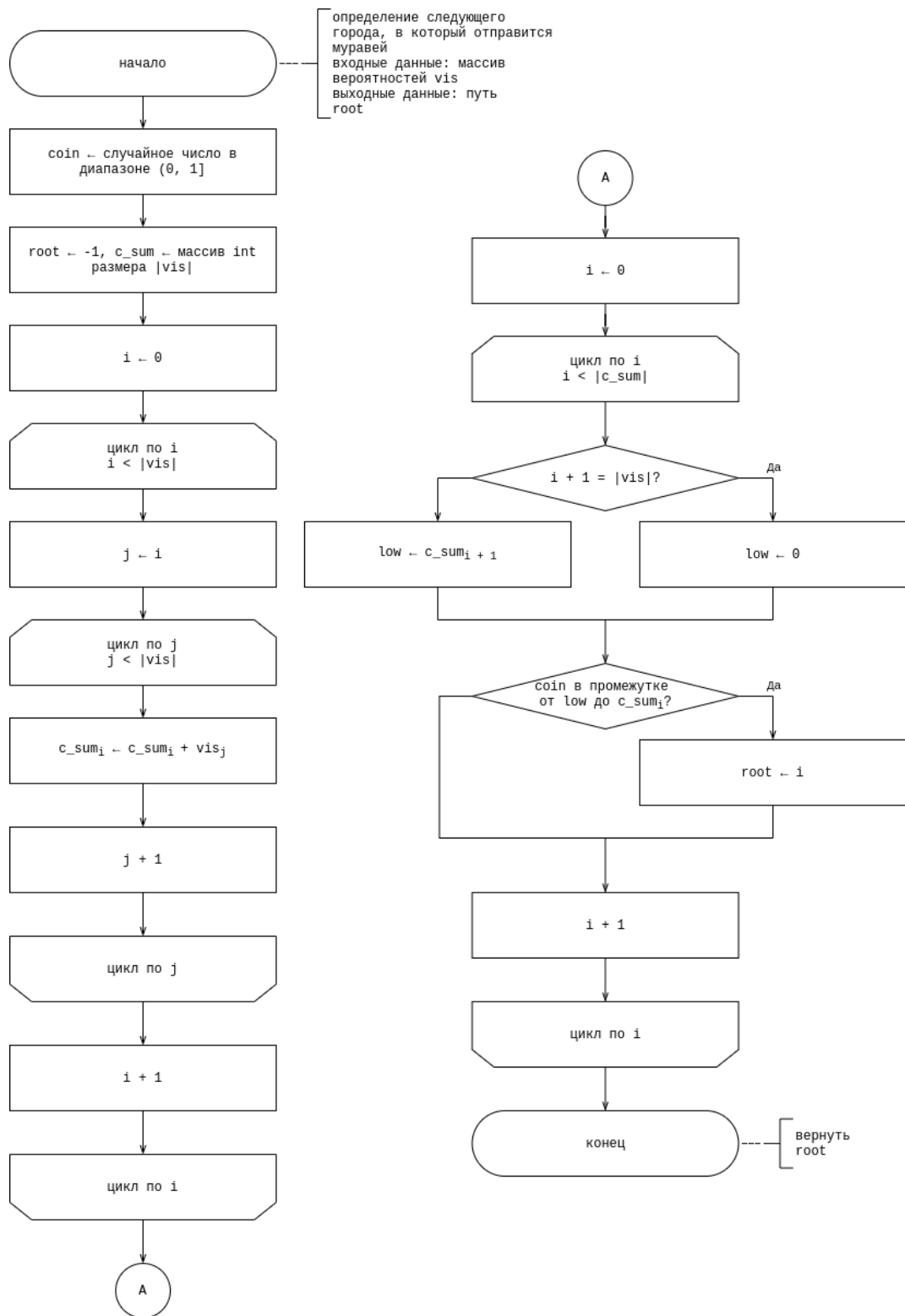


Рисунок 2.2 – Схема работы алгоритма случайного выбора города

Рисунок 2.3 демонстрирует работу алгоритма обновления феромона. Обозначения аналогично 2.1.

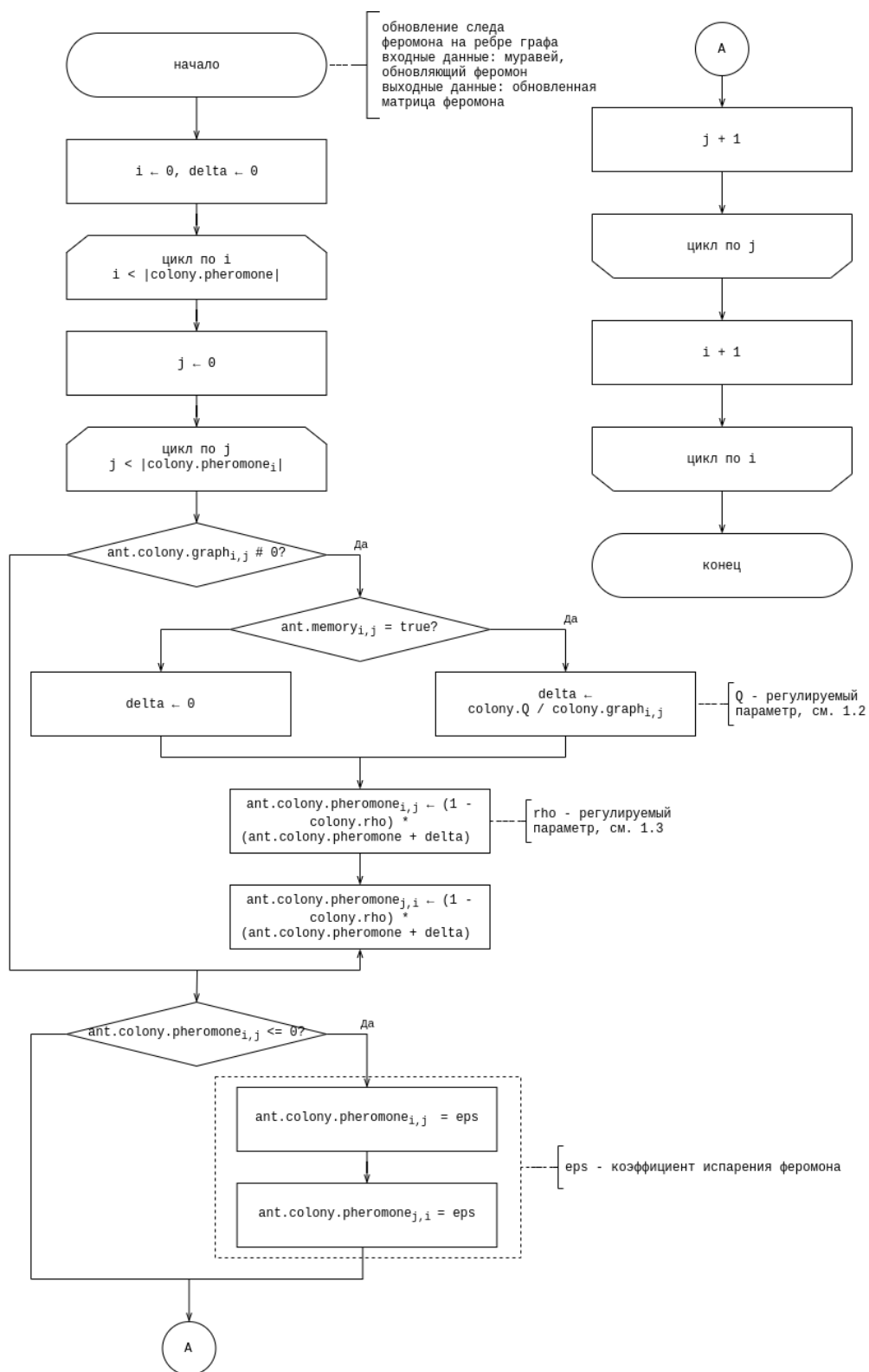


Рисунок 2.3 – Схема работы алгоритма обновления феромона

Рисунок 2.3 демонстрирует работу алгоритма "грубой силы". Обозначения аналогично 2.1.

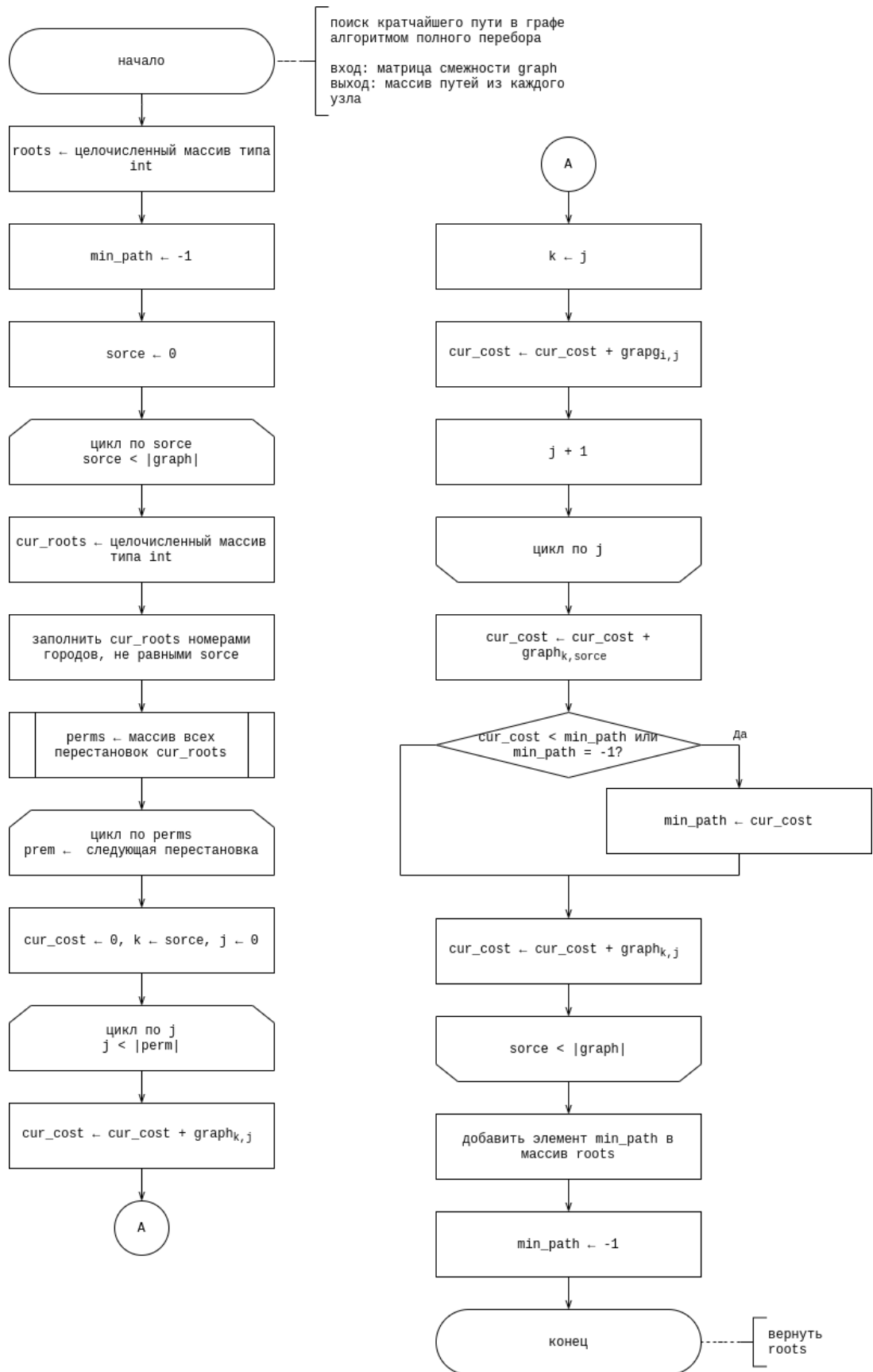


Рисунок 2.4 – Схема работы алгоритма "грубой силы"

2.4 Выделение классов эквивалентности

Алгоритм будет протестирован на матрицах, ограничения для которых были установлены в подразделе 1.3: матрицы симметричны, главная диагональ матриц нулевая, остальные элементы ненулевые.

2.5 Структура ПО

ПО имеет три независимых модуля:

- модуль `brute`, содержащий реализацию алгоритма полного перебора;
- модуль `aco`, содержащий реализацию алгоритма муравьиной колонии;
- модуль `interf`, содержащий вспомогательные функции для пользовательского интерфейса.

Вывод

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

3. Технологический раздел

Раздел содержит листинги реализованных алгоритмов, требование к ПО и тестирование реализованного программного обеспечения. Также раздел включает в себя описание системы автоматической параметризации.

3.1 Требования к ПО

Программное обеспечение должно удовлетворять следующим требованиям:

- программа принимает на вход количество дней, симметричную 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. В качестве среды разработки была выбрана среда VS Code, написание сценариев осуществлялось утилитой make. Скрипты для автоматизации были написаны на языке python.

3.3 Листинги кода

Муравьиный алгоритм

Листинг 3.1 демонстрирует типы данных для алгоритма муравьиной колонии.

Листинг 3.1 – Структуры данных

```
1 type Ant struct {
2     colony *Colony
3     tour    [][]int
4     memory  [][]bool
5     position int
6     start   int
7 }
8 type Colony struct {
9     gr_matrix [][]int
10    phero_matrix [][]float64
11    iters        int
12    conf         Config
13 }
14 type Config struct {
15     TRACE_WEIGHT float64 // alpha
16     TOUR_VISIB   float64 // beta
17     EVAP_RATE    float64 // p
18     Q            float64 // q
19     PHERO_INIT   float64 // start pheromone conc
20     PHERO_EPS    float64
21 }
```

Листинг 3.2 демонстрирует аллокацию структуры муравьиной колонии.

Листинг 3.2 – Аллокация колонии

```
1 func makeColony(gr_matrix [][]int, iters int,
2                conf_filename string) *Colony {
3     colony := new(Colony)
4     conf := ParseConfigFile(conf_filename)
5     if (conf == nil) {
6         return nil
7     }
8     colony.conf = *conf
9
10    colony.gr_matrix = gr_matrix
11    colony.iters = iters
12    colony.phero_matrix = make([][]float64, len(colony.gr_matrix))
13
14    for i := 0; i < len(colony.gr_matrix); i++ {
15        colony.phero_matrix[i] = make([]float64, len(colony.gr_matrix[i]))
16        for j := range colony.phero_matrix[i] {
17            colony.phero_matrix[i][j] = colony.conf.PHERO_INIT
18        }
19    }
20    return colony
21 }
```

Листинг 3.3 демонстрирует размещение муравья на точке в графе.

Листинг 3.3 – Размещение муравья

```
1 func (col *Colony) placeAnt(start int) *Ant {
2     ant := new(Ant)
3     ant.colony = col
4     ant.tour = make([][]int, len(col.gr_matrix))
5     ant.memory = make([][]bool, len(col.gr_matrix))
6     ant.position = start
7     ant.start = start
8
9     for i := range col.gr_matrix {
10         ant.tour[i] = make([]int, len(col.gr_matrix[i]))
11         copy(ant.tour[i], col.gr_matrix[i])
12     }
13
14     for i := range ant.memory {
15         ant.memory[i] = make([]bool, len(col.gr_matrix))
16     }
17     return ant
18 }
```

Пользователь взаимодействует с функцией, продемонстрированной на листинге 3.4.

Листинг 3.4 – Алгоритм муравьиной колонии

```
1 func ACOWrapper(gr_matrix [][]int, iters int,
2                 conf_filename string) *[]int {
3     shortest := make([]int, len(gr_matrix))
4     colony := makeColony(gr_matrix, iters, conf_filename)
5     if (colony == nil) {
6         return nil
7     }
8
9     for i := 0; i < colony.iters; i++ {
10         for j := 0; j < len(colony.gr_matrix); j++ {
11             ant := colony.placeAnt(j)
12             ant.startTour()
13             current := ant.tourLength()
14
15             if current < shortest[j] || shortest[j] == 0 {
16                 shortest[j] = current
17             }
18         }
19     }
20     return &shortest
21 }
```

Начало перемещения муравья по графу происходит согласно функции 3.5:

Листинг 3.5 – Перемещение муравья по графу

```
1 func (ant *Ant) startTour() {
2     next := -1
3
4     for flag := true; flag; flag = (next != -1) {
5         vis := ant.getVision()
6
7         next = fortuneWheel(vis)
8
9         if (next != -1) {
10             ant.tourAnt(next)
11             ant.updatePhTrace()
12         }
13     }
14 }
```

Вероятностный массив, полученный согласно правилу 1.1, вычисляется в следующей функции(3.6)

Листинг 3.6 – Получение массива зрения для муравья

```
1 func (ant *Ant) getVision() []float64 {
2     var (
3         vis = make([]float64, 0)
4         denom = 0.0
5     )
6
7     for i, l := range ant.tour[ant.position] {
8         if l != 0 {
9             term := math.Pow((1.0 / float64(l)),
10                 ant.colony.conf.TRACE_WEIGHT) * math.Pow
11                 (ant.colony.phero_matrix[ant.position][i],
12                 ant.colony.conf.TOUR_VISIB)
13
14             denom += term
15             vis = append(vis, term)
16         } else {
17             vis = append(vis, 0)
18         }
19     }
20
21     for i := 0; i < len(vis); i++ {
22         vis[i] /= denom
23     }
24     return vis
25 }
```

Листинг 3.7 демонстрирует метод выбора следующего города.

Листинг 3.7 – Выбор следующего города

```
1 func fortuneWheel(vis []float64) int {
2     var (
3         cumul_sum = make([]float64, len(vis))
4         coin = rand.New(rand.NewSource(time.Now().UnixNano())).Float64()
5         chosen = -1
6     )
7
8     for i := 0; i < len(vis); i++ {
9         for j := i; j < len(vis); j++ {
10             cumul_sum[i] += vis[j]
11         }
12     }
13
14     for i := 0; i < len(cumul_sum); i++ {
15         if i == len(cumul_sum) - 1 {
16             if 0.0 <= coin && coin <= cumul_sum[i] {
17                 chosen = i
18             }
19         } else {
20             if cumul_sum[i + 1] < coin && coin <= cumul_sum[i] {
21                 chosen = i
22             }
23         }
24     }
25
26     return chosen
27 }
```

Поведение муравья после выбора города описано на листинге 3.8

Листинг 3.8 – Путь муравья в следующий город

```
1 func (ant *Ant) tourAnt(root int) {
2     for i := range ant.tour[ant.position] {
3         ant.tour[i][ant.position] = 0
4         ant.tour[ant.position][i] = 0
5     }
6
7     ant.memory[ant.position][root] = true
8     ant.position = root
9 }
```

Листинг 3.9 демонстрирует обновление феромона согласно формуле 1.2.

Листинг 3.9 – Обновление феромона

```
1 func (ant *Ant) updatePhTrace() {
2     delta := 0.0
3
4     for i := 0; i < len(ant.colony.phero_matrix); i++ {
5         for j, phero := range ant.colony.phero_matrix[i] {
6             if ant.colony.gr_matrix[i][j] != 0 {
7                 if ant.memory[i][j] {
8                     delta = ant.colony.conf.Q /
9                         float64(ant.colony.gr_matrix[i][j])
10                } else {
11                    delta = 0
12                }
13                ant.colony.phero_matrix[i][j] =
14                    (1 - ant.colony.conf.EVAP_RATE) * (float64(phero) + delta)
15                ant.colony.phero_matrix[j][i] =
16                    (1 - ant.colony.conf.EVAP_RATE) * (float64(phero) + delta)
17            }
18            if ant.colony.phero_matrix[i][j] <= 0 {
19                ant.colony.phero_matrix[i][j] = ant.colony.conf.PHERO_EPS
20                ant.colony.phero_matrix[j][i] = ant.colony.conf.PHERO_EPS
21            }
22        }
23    }
24 }
```

После прохождения маршрута муравей вычисляет длину пройденного пути следующим образом (3.10):

Листинг 3.10 – Длина маршрута

```
1 func (ant *Ant) tourLength() int {
2     var (
3         length = 0
4         last   = ant.position
5     )
6
7     for i, r := range ant.memory {
8         for root, passed := range r {
9             if passed {
10                 length += ant.colony.gr_matrix[i][root]
11             }
12         }
13     }
14     length += ant.colony.gr_matrix[ant.start][last]
15
16     return length
17 }
```

Алгоритм полного перебора

Листинг 3.3 демонстрирует алгоритм полного перебора.

```
1 func BruteForce(gr_matrix [][]int) []int {
2     var (
3         roots = make([]int, 0)
4         graph_len = len(gr_matrix)
5         min_path = -1
6     )
7     for source := 0; source < graph_len; source++ {
8         cur_roots := make([]int, 0)
9         for i := 0; i < graph_len; i++ {
10             if i != source {
11                 cur_roots = append(cur_roots, i)
12             }
13         }
14         next_permutation := permutations(cur_roots)
15         for _, perm := range next_permutation {
16             curr_cost := 0
17             k := source
18
19             for _, j := range perm {
20                 curr_cost += gr_matrix[k][j]
21                 k = j
22             }
23             curr_cost += gr_matrix[k][source]
24             if curr_cost < min_path || min_path == -1 {
25                 min_path = curr_cost
26             }
27         }
28         roots = append(roots, min_path)
29         min_path = -1
30     }
31     return roots
32 }
```

Получение всех перестановок происходит согласно функциям, демонстрируемым на листингах 3.11 и 3.12.

Листинг 3.11 – Получение всех перестановок

```
1 func permutations(path []int) [][]int {
2     var res [][]int
3     __permutationHelper(path, &res, 0)
4     return res
5 }
```

Листинг 3.12 – Получение одной перестановки

```
1 func __permutationHelper(path []int, res *[][]int, k int) {
2     for i := k; i < len(path); i++ {
3         path[i], path[k] = path[k], path[i]
4         __permutationHelper(path, res, k + 1)
5         path[k], path[i] = path[i], path[k]
6     }
7     if (k == len(path) - 1) {
8         r := make([]int, len(path))
9         copy(r, path)
10        *res = append(*res, r)
11        return
12    }
13 }
```

Автоматическая параметризация

скрипт автоматической перестановки генерирует все комбинации параметров в заданном диапазоне и помещает в конфигурационный файл. Функция генерации параметров имеет следующий вид (3.13):

Листинг 3.13 – Генерация параметров

```
1 def form_coefs():
2     coefs_gen = []
3     for i in range(11):
4         stack = []
5         alpha = round(0.0 + 0.1 * i, 2)
6         betta = round(1 - alpha, 2)
7
8         stack.append([round(0.0 + 0.1 * j, 2) for j in range(11)])
9         stack.append([40]) # q
10        stack.append([0.1]) # init
11        stack.append([0.01]) # eps
12        cprod = list(itertools.product(*stack))
13        coefs_cur = []
14        for x in cprod:
15            x = list(x)
16            x.insert(0, alpha)
17            x.insert(1, betta)
18            coefs_cur.append(x)
19            coefs_gen.append(coefs_cur)
20
21    return coefs_gen
```


3.4 Тестирование ПО

Результаты тестирования ПО приведены в таблице 3.1.

Таблица 3.1 – Тестирование ПО

<i>Матрица</i>	<i>Полный перебор</i>	<i>Муравьиный алгоритм</i>
0 7 8 7 9 7 0 8 3 6 8 8 0 8 6 7 3 8 0 8 9 6 6 8 0	30 30 30 30 30	30 30 30 30 30
0 3 1 6 8 3 0 4 1 2 1 4 0 5 2 6 1 5 0 6 8 2 2 6 0	12 12 12 12 12	12 12 12 12 12
0 10 15 20 10 0 35 25 15 35 0 30 20 25 30 0	80 80 80 80	80 80 80 80

3.5 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 Ubuntu 20.04.1 LTS;
- память 7 GiB;
- процессор Intel(R) Core(TM) i3-8145U CPU @ 2.10GHz.

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):

$$\begin{pmatrix} 0 & 3 & 5 & 9 & 4 & 8 & 5 & 6 & 6 & 7 \\ 3 & 0 & 8 & 5 & 5 & 10 & 6 & 4 & 4 & 5 \\ 5 & 8 & 0 & 3 & 6 & 9 & 5 & 4 & 8 & 4 \\ 9 & 5 & 3 & 0 & 3 & 8 & 9 & 6 & 4 & 8 \\ 4 & 5 & 6 & 3 & 0 & 7 & 8 & 7 & 3 & 9 \\ 8 & 10 & 9 & 8 & 7 & 0 & 6 & 6 & 10 & 6 \\ 5 & 6 & 5 & 9 & 8 & 6 & 0 & 4 & 7 & 10 \\ 6 & 4 & 4 & 6 & 7 & 6 & 4 & 0 & 10 & 3 \\ 6 & 4 & 8 & 4 & 3 & 10 & 7 & 10 & 0 & 2 \\ 7 & 5 & 4 & 8 & 9 & 6 & 10 & 3 & 2 & 0 \end{pmatrix} \quad (4.1)$$

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.
0.9	0.1	0.1	10	39	0
0.9	0.1	0.4	10	39	0
0.0	1.0	0.0	11	39	0
0.3	0.7	0.0	11	39	0
0.7	0.3	1.0	11	39	0
0.9	0.1	0.0	11	39	0
0.9	0.1	1.0	11	39	0
1.0	0.0	0.8	11	39	0
0.1	0.9	0.0	12	39	0
0.8	0.2	0.0	12	39	0
0.8	0.2	0.1	13	39	0
0.9	0.1	0.2	13	39	0
1.0	0.0	0.4	13	39	0
0.4	0.6	1.0	14	39	0
0.4	0.6	0.0	15	39	0

4.4 Класс данных №2

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

$$\begin{array}{c}
 \left| \begin{array}{cccccccccc}
 0 & 34 & 27 & 82 & 46 & 25 & 72 & 30 & 47 & 59 \\
 34 & 0 & 38 & 22 & 37 & 59 & 42 & 21 & 44 & 68 \\
 27 & 38 & 0 & 43 & 32 & 23 & 62 & 70 & 23 & 48 \\
 82 & 22 & 43 & 0 & 69 & 19 & 43 & 95 & 67 & 66 \\
 46 & 37 & 32 & 69 & 0 & 55 & 94 & 50 & 33 & 51 \\
 25 & 59 & 23 & 19 & 55 & 0 & 61 & 30 & 74 & 49 \\
 72 & 42 & 62 & 43 & 94 & 61 & 0 & 46 & 52 & 33 \\
 30 & 21 & 70 & 95 & 50 & 30 & 46 & 0 & 30 & 86 \\
 47 & 44 & 23 & 67 & 33 & 74 & 52 & 30 & 0 & 76 \\
 59 & 68 & 48 & 66 & 51 & 49 & 33 & 86 & 76 & 0
 \end{array} \right|
 \end{array} \quad (4.2)$$

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.
0.8	0.2	0.1	12	300	0
1.0	0.0	0.0	12	300	0
0.1	0.9	0.0	13	300	0
0.9	0.1	0.0	13	300	0
0.4	0.6	0.0	15	300	0
0.5	0.5	1.0	15	300	0
0.7	0.3	1.0	15	300	0
0.8	0.2	0.0	15	300	0
0.3	0.7	0.0	16	300	0
0.6	0.4	0.0	16	300	0
1.0	0.0	0.4	16	300	0
0.2	0.8	0.0	17	300	0
1.0	0.0	0.5	17	300	0
0.7	0.3	0.9	18	300	0
1.0	0.0	0.9	19	300	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	39	300	0	0
0.7	0.3	1.0	11	15	39	300	0	0
0.9	0.1	0.0	11	13	39	300	0	0
0.1	0.9	0.0	12	13	39	300	0	0
0.8	0.2	0.0	12	15	39	300	0	0
0.8	0.2	0.1	13	12	39	300	0	0
1.0	0.0	0.4	13	16	39	300	0	0
0.4	0.6	0.0	15	15	39	300	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 литературы

- [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 АСО 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] М. В. Ульянов. *Ресурсно-эффективные компьютерные алгоритмы. Разработка и анализ*. под ред. В. С. Аролович. Физматлит, 2008.

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