Отчет по лабораторной работе №6
по курсу "Анализ алгоритмов"
по теме "Муравьиный алгоритм"

Студент: Барсуков Н.М. ИУ7-56
Преподаватель: Волкова Л.Л., Строганов Ю.В.

Содержание

Введение

Задача коммивояжёра - одна из самых известных задач транспортной логистики. Одно из первых её решений было предложено У.Гамильтоном в XIX веке. Суть задачи заключается в следующем:
необходимо найти оптимальный (кратчайший) путь через некоторые пункты, проходя каждый по одному разу. Мерой выгодности маршрута могут быть минимальное время, минимальные расходы на дорогу или минимальная длина пути. [**TSP**]

В последние годы интенсивно развивается научное направление, называемое природными вычислениями, объединяющее математические методы, в основе которых заложены природные механизмы и принципы принятия решений, сформированные во флоре и фауне на протяжении миллионов лет. Имитация муравьиной колонии составляет основу муравьиных алгоритмов. Колония муравьёв рассматривается как многоагентная система, в которой каждый муравей рассматривается как отдельный агент, функционирующий автономно по простым правилам. [**AntAlg**]

1 Аналитический раздел

В данном разделе описана поставленная цель и описаны задачи, необходимые для выполнения поставленной цели. Детально рассмотрены алгоритмы

1.1 Постановка задачи

Цель: реализовать и изучить алгоритм муравьинной колонии, решающий задачу коммивояжера. Сравнить с решением методом перебора
Для выполнения поставленной цели необходимо выполнить следующие задачи:

1) изучить:
   (a) алгоритм муравьиной колонии
   (b) метод полного перебора
2) реализовать выше перечисленные методы;
3) провести параметризацию алгоритма;
4) исследовать зависимость работы от регулирующих параметров;
5) провести сравнение с решением, найденным методом перебора.

1.2 Описание алгоритмов

Классический муравьиный алгоритм - один из вариантов решения задачи коммивояжёра. Настоящие муравьи способны находить кратчайший путь между своим гнездом и источников пищи, используя не видимые глазу знаки, а с помощью информации из феромона. Когда муравей идёт, он оставляет на своей дороге феромон и ориентируется на путевые феромоны, оставленные другими муравьями. Если муравей подходит к точке выбора пути, где ни на одной дороге ещё нет феромона, то муравей выберет рандомно путь из возможных. Если путей в такой точке выбора было два, то половина муравьёв пойдёт налево, другая половина - направо.

На рисунке 1 представлен выбор путей муравьями при движении из двух вершин при отсутствии путевых феромонов других муравьёв. Красным обозначены точки выбора пути.

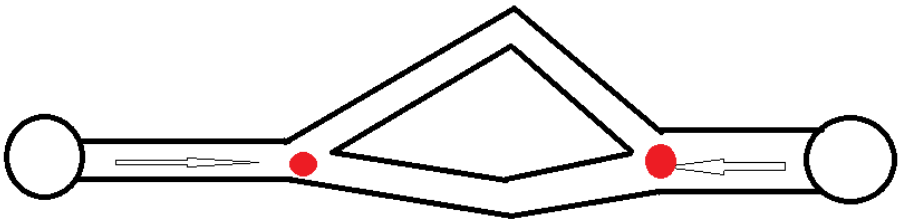

Рис. 1: Выбор пути муравьём при отсутствии информации путевых феромонов

Допустим, что исходно из каждой вершины вышло по 2 муравья. На рисунке 2 показано, как распределятся муравьи по путям при отсутствии феромона.

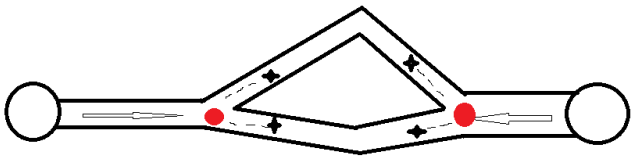

Рис. 2: Выбор пути муравьём при отсутствии информации путевых феромонов

Допустим, что все муравьи движутся примерно с одной скоростью. Если дороги были разной длины, то через некоторое время будет оставлено достаточно феромона, чтобы каждый муравей в точке выбора мог однозначно идентифицировать, какой путь был более популярен. Таким путём будет более короткий, поскольку муравьи будут проходить его быстрее, следовательно, будет оставлено больше феромона (рисунок 3). По мере роста количества феромона у более короткого пути уменьшается количество муравьёв, которые будут выбирать длинную дорогу.

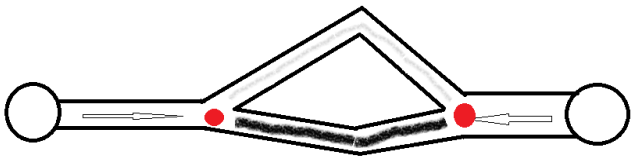

Рис. 3: Выбор пути муравьём в результате обработки информации путевых феромонов

Через некоторое время воздействие феромона у длинной дороги будет

настолько низким, что все муравьи будут выбирать короткий путь.[**ACS**]

Для описания поведения муравьёв в реальной жизни введена муравьиная система - алгоритм, в котором искусственно созданные "муравьи"совместно решают проблему обмена информацией через феромон при прохождении вершин графов. Основная идея данного генетического алгоритма состоит в следующем:

1) введём описание данных следующим образом:
   (a) существуют агенты, "муравьи";
   (b) каждый муравей обладает памятью - существует список городов
   $$J_{i,k}, \tag{1}$$
   которые нужно посетить k-ому муравью, если он находится в городе i;
   (c) каждый муравей обладает зрением - у муравья есть эвристическое желание посетить город j (находится он при этом в городе i) - это видимость
   $$n_{ij} = \frac{1}{D_{ij}}, \tag{2}$$
   где D - матрица расстояний (чем короче ребро, тем больше желание пройти по нему);
   (d) каждый муравей обладает обоняннием - муравей может улавливать феромон, оставленный другими муравьями;
   (e) происходит непрямой обмен - стигмерж - информацией между муравьями через окружающую среду;
   (f) муравей использует опыт колонии;
   (g) у колонии есть время жизни
   $$1 \leqslant t \leqslant t_{max} \tag{3}$$
   (h) на каждом шаге по времени все муравьи формируют по маршруту
   $$T_k(t), \tag{4}$$
   длины маршрутов
   $$L_k(t) \tag{5}$$
   можно вычислить;
2) в соответствии с длиной на рёбра маршрута накладывается феромон;
3) вместе с испарением феромона происходит переход на новый шаг по времени, при этом происходит обновление наилучшего маршрута
   $$T^* \tag{6}$$

и длины наилучшего маршрута

$$L^* \tag{7}$$

в случае, если один из k муравьёв нашёл лучшее решение;

4) если муравей k находится в городе i, то переход в другой город j происходит по вероятностному правилу

$$P_{ij,k}(t) = \begin{cases} 0, & j \notin J_{i,k} \\ \frac{(\tau_{ij}(t))^{\alpha} * (\eta_{ij})^{\beta}}{\sum\limits_{q \in J_{i,k}} (\tau_{iq}(t))^{\alpha} * (\eta_{iq})^{\beta}}, & j \in J_{i,k}, \end{cases} \tag{8}$$

где τ - матрица значений феромона, α и β - параметры, соблюдающие условие, что $\alpha + \beta = const$;

5) когда маршруты сформированы, каждый муравей оставляет на рёбрах своего маршрута $T_k(t)$ феромон

$$\Delta\tau_{ij,k}(t) = \begin{cases} 0, & (i,j) \notin T_k(t) \\ \frac{Q}{L_k(t)}, & (i,j) \in T_k(t), \end{cases} \tag{9}$$

где Q - параметр того же порядка, что и L^* - длина оптимального маршрута;

6) переход на следующий шаг по времени задействует ρ - коэффициент испарения феромона:

$$\tau_{ij}(t+1) = (1-\rho) * \tau_{ij}(t) + \sum_{k=1}^{m} \Delta\tau_{ij,k}(t), \tag{10}$$

где m - размер колонии.

[**RECA**]

В начале алгоритма количество феромона принимается равным небольшому положительному числу. Общее количество муравьёв остаётся постоянным и равным количеству городов, каждый муравей начинает маршрут из своего города. Данный генетический алгоритм можно модифицировать, введя e элитных муравьёв, которые усиливают рёбра лучшего маршрута T^*:

$$\Delta\tau_{ij,e}(t) = \frac{e * Q}{L^*} \tag{11}$$

[**RECA**]

Сложность алгоритма составляет $O(t_{max}\dot{m}ax(m, n^2))$, то есть сложность зависит от времени жизни колонии, количества городов и количества муравьёв в колонии. [**RECA**]

Algorithm 1 Муравьиный алгоритм

1: $D \leftarrow \_values\_$ ▷ Ввод матрицы расстояний
2: $Parameters \leftarrow \_values\_$ ▷ Инициализация параметров алгоритма
3: $Feromon \leftarrow \_values\_$ ▷ Инициализация начальной концентрации феромона
4: $\eta_{ij} \leftarrow \_values\_$ ▷ Инициализация видимости
5: $T^* \leftarrow \_values\_$
6: $L^* \leftarrow \_values\_$ ▷ Выбор начального текущего маршрута и определение длины лучшего текущего маршрута
7: **for** t=1; t_{max}; t++ **do** ▷ Цикл по времени жизни колонии
8: **for** k=1; m; k++ **do** ▷ Цикл по всем муравьям
9: Построить маршрут $T_k(t)$ по правилу 8 и рассчитать длину $L_k(t)$.
10: **end for**
11: **if** $L_k(t) < L^*$ **then** ▷ Если найден лучший путь
12: Обновить L^* и T^*
13: **end if**
14: **for** рёбра графа **do**
15: Обновить следы феромона на ребре по правилам 10 и 11
16: **end for**
17: **end for**
18: Print(T^* и L^*) ▷ Вывести кратчайший маршрут T^* и его длину L^*

В данном разделе поставлена задача. Детально расписаны алгоритмы.

2 Конструкторский раздел

В данном разделе приводятся схемы работы алгоритмов:

2.1 Алгоритмы

1) перебором 4 5;

2) муравьиный 6 7 8.

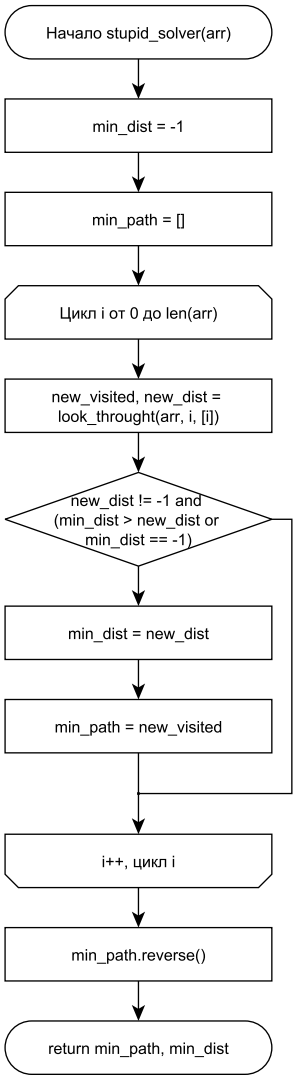

Рис. 4: Нахождение решения перебором

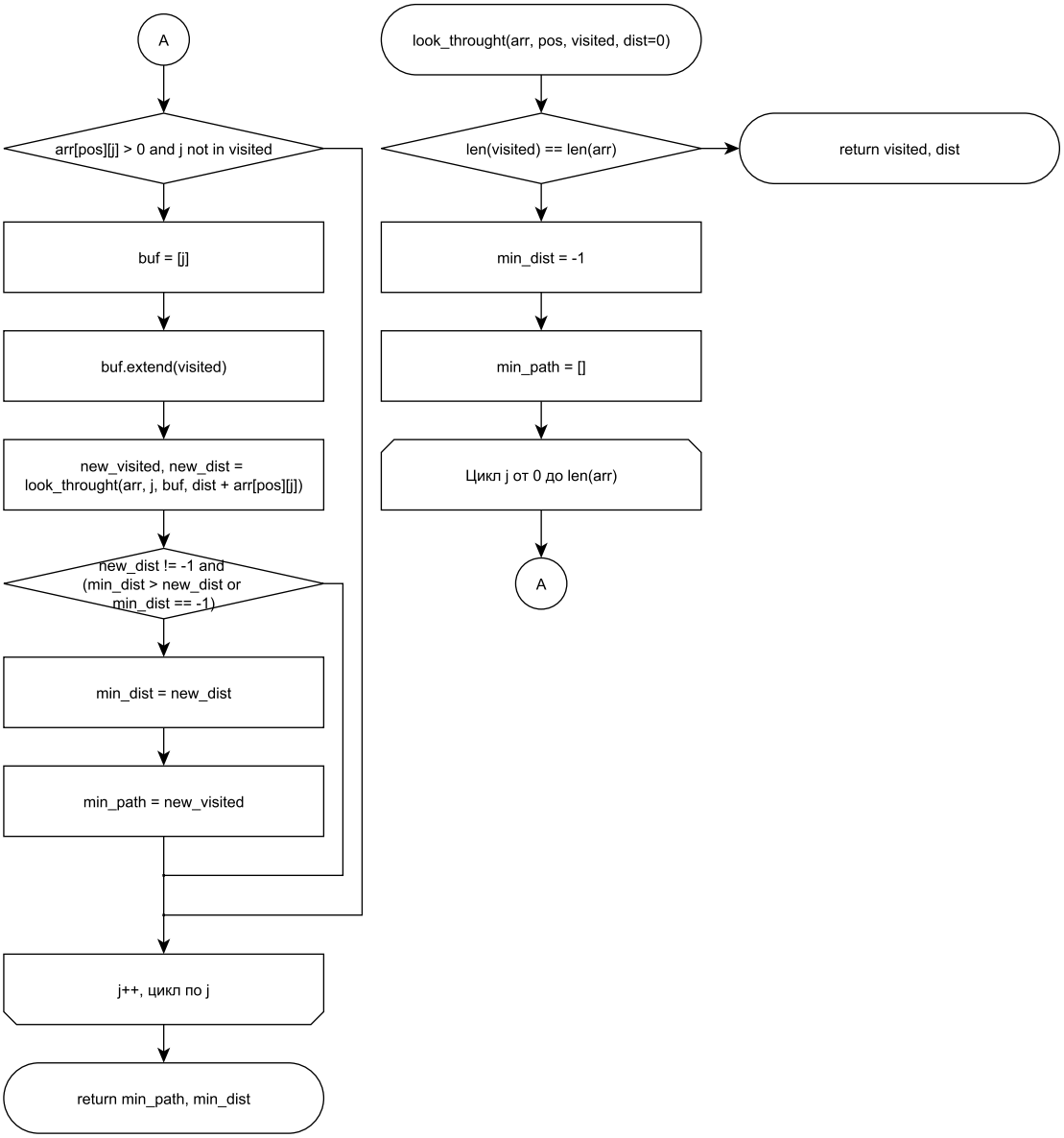

Рис. 5: Нахождение решения перебором

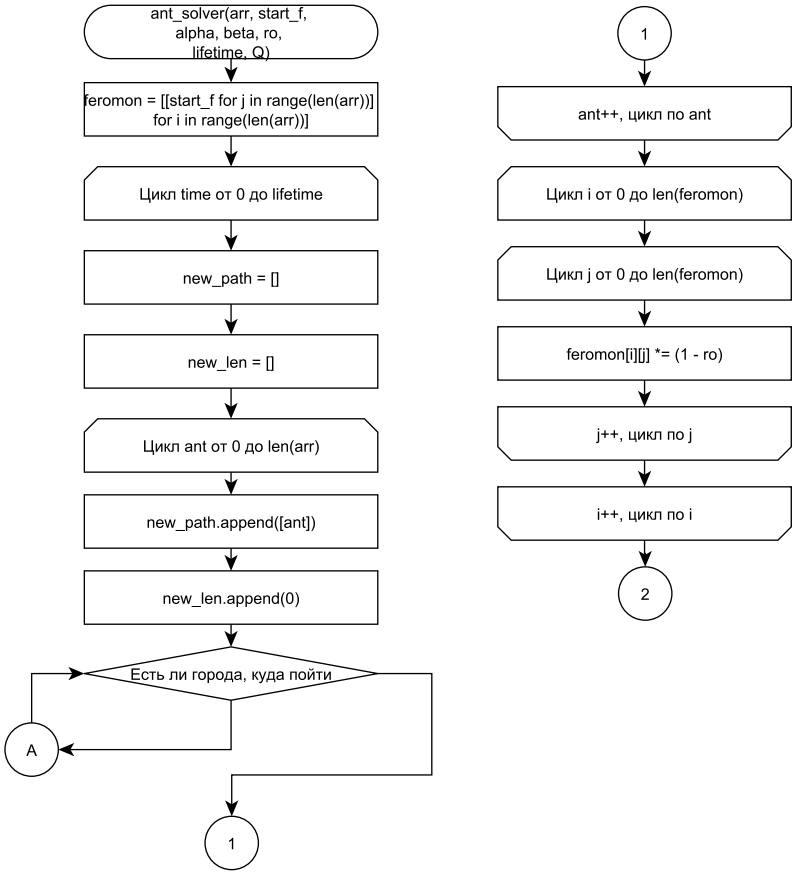

Рис. 6: Муравьиный алгоритм

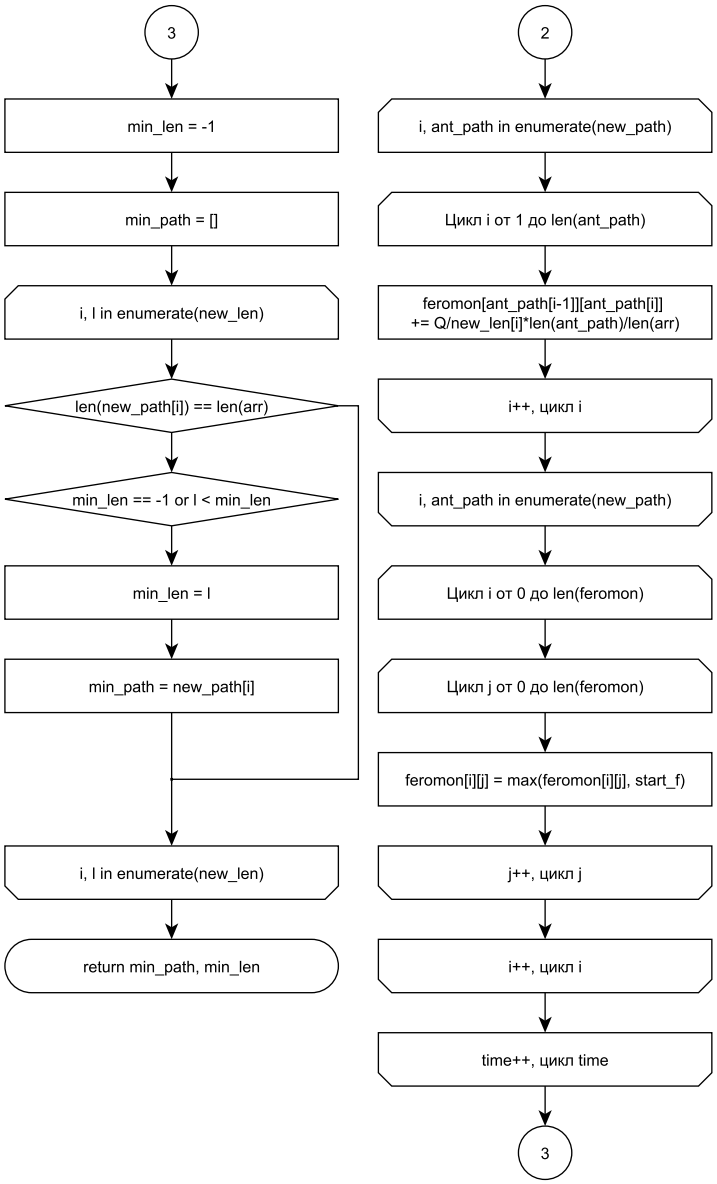

Рис. 7: Муравьиный алгоритм

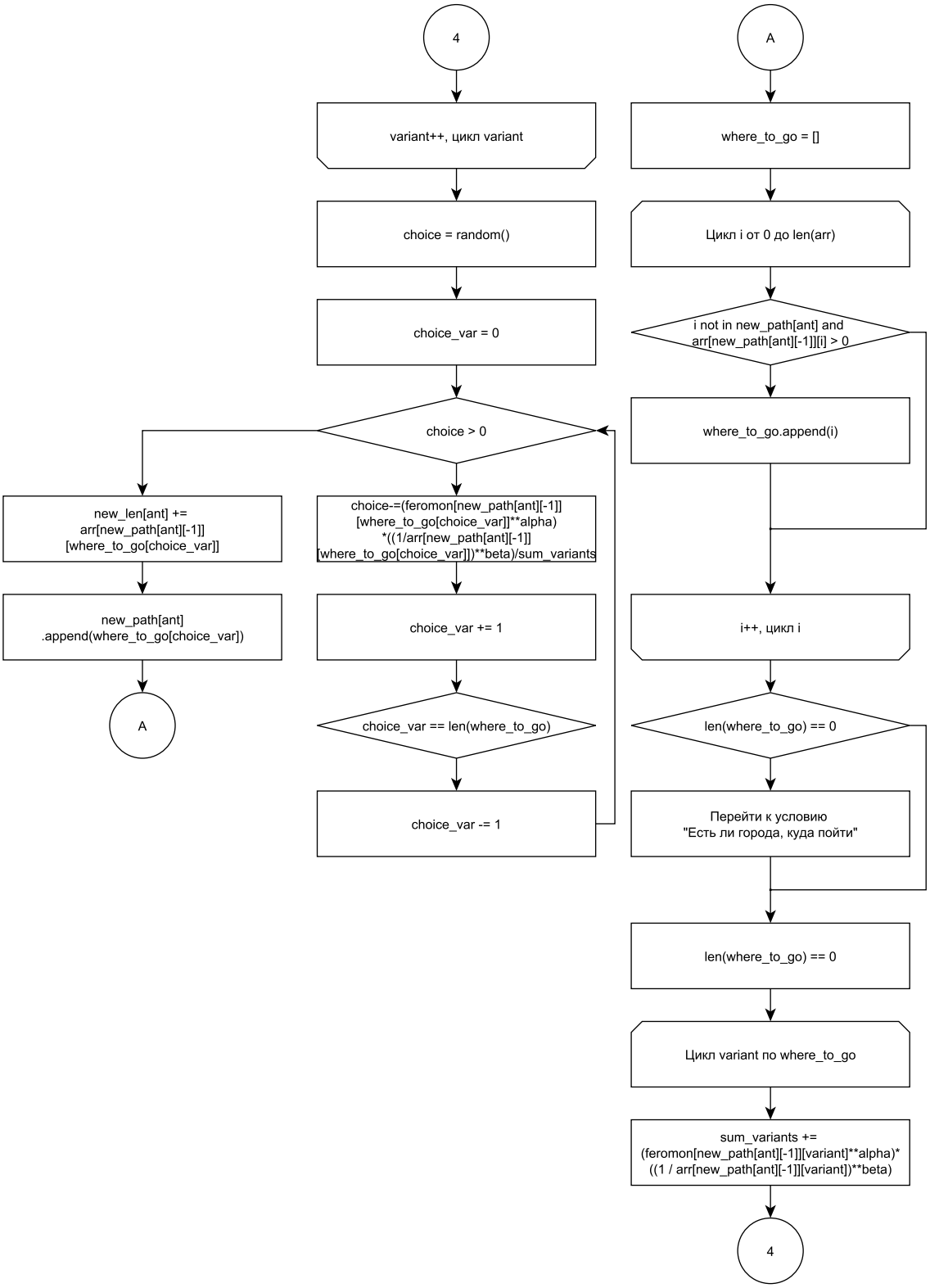

Рис. 8: Муравьиный алгоритм

В данном разделе приведены схемы алгоритмов

3 Технологический раздел

3.1 Требования к программному обеспечению

Минимальные системные требования: PC с операционной системой Windows версии XP/Vista/7/8/10. Требуются устройства ввода: клавиатура, мышь.

3.2 Средства реализации

Для выполнения работы был выбран язык программирования Python ввиду его простоты.

3.3 Интерфейс

Интерфейс из себя представляет простую консоль, где пользователю ничего делать ненужно.

3.4 Листинг

Листинг 1: Метод перебора

```
7
def look_throught(arr, pos, visited, dist=0):
    if len(visited) == len(arr):
        return visited, dist

    min_dist = -1
    min_path = []
    for j in range(0, len(arr)):
        if arr[pos][j] > 0 and j not in visited:
            buf = [j]
            buf.extend(visited)
            new_visited, new_dist = \
                    look_throught(arr, j, buf, dist +
                        arr[pos][j])
            if new_dist != -1 and (min_dist > new_dist or min_dist
                == -1):
                min_dist = new_dist
                min_path = new_visited

    return min_path, min_dist

def stupid_solver(arr):
    min_dist = -1
    min_path = []
    for i in range(0, len(arr)):
        new_visited, new_dist = look_throught(arr, i, [i])
        if new_dist != -1 and (min_dist > new_dist or min_dist ==
            -1):
            min_dist = new_dist
            min_path = new_visited

    min_path.reverse()
    return min_path, min_dist
```

Листинг 2: Муравьиный метод

```
7
def ant_solver(arr, start_f, alpha, beta, ro, lifetime, Q):
    feromon = [[start_f for j in range(len(arr))] for i in
        range(len(arr))]

    for time in range(lifetime):
        new_path = []
        new_len = []

        for ant in range(len(arr)):
            new_path.append([ant])
            new_len.append(0)

            while True:
                where_to_go = []
                for i in range(len(arr)):
                    if i not in new_path[ant] and
                        arr[new_path[ant][-1]][i] > 0:
                        where_to_go.append(i)

                if len(where_to_go) == 0:
                    break
                sum_variants = 0

                #print(where_to_go)

                for variant in where_to_go:
                    #print(1 / arr[new_path[ant][-1]][variant])
                    sum_variants +=
                        (feromon[new_path[ant][-1]][variant] **
                        alpha) * \
                                ((1 /
                                    arr[new_path[ant][-1]][variant])
                                    ** beta)

                choice = random()
                choice_var = 0
                while choice > 0:
                    #print(choice, choice_var)
                    choice -=
                        (feromon[new_path[ant][-1]][where_to_go[choice_var]]
                        ** alpha) * \
                                ((1 /
                                    arr[new_path[ant][-1]][where_to_go[choice_var]])
                                    ** beta) / sum_variants
                    choice_var += 1
                    if choice_var == len(where_to_go):
                        choice_var -= 1
                        break

                new_len[ant] +=
                    arr[new_path[ant][-1]][where_to_go[choice_var]]
                new_path[ant].append(where_to_go[choice_var])

        for i in range(len(feromon)):
            for j in range(len(feromon)):
                feromon[i][j] *= (1 - ro)

        #print(new_len)
        #print(new_path)
        for i, ant_path in enumerate(new_path):
```

```
50                  for i in range(1, len(ant_path)):
51                      #feromon[ant_path[i-1]][ant_path[i]] += Q /
                            new_len[i]
52                      feromon[ant_path[i-1]][ant_path[i]] += Q /
                            new_len[i] * \
53                                                              len(ant_path)
                                                                    /
                                                                len(arr)
54
55              for i in range(len(feromon)):
56                  for j in range(len(feromon)):
57                      feromon[i][j] = max(feromon[i][j], start_f)
58
59          min_len = -1
60          min_path = []
61          for i, l in enumerate(new_len):
62              if len(new_path[i]) == len(arr):
63                  if min_len == -1 or l < min_len:
64                      min_len = l
65                      min_path = new_path[i]
66
67          return min_path, min_len
```

В данном разделе описаны минимальные системные требования для оборудования. Указаны средства реализации. Приведено описание интерфейса.

4 Иследовательский раздел

4.1 Исследование поведения алгоритма в зависимости от изменяемых параметров

Рассмотрим формулу

$$P_{ij,k}(t) = \begin{cases} 0, & j \notin J_{i,k} \\ \frac{(\tau_{ij}(t))^{\alpha} * (\eta_{ij})^{\beta}}{\sum\limits_{q \in J_{i,k}} (\tau_{iq}(t))^{\alpha} * (\eta_{iq})^{\beta}}, & j \in J_{i,k}. \end{cases}$$

Её результатом будет являться вероятность, которая не будет превышать значение, равное единице. Обеспечивается это тем, что внизу стоит сумма произведений для всех граней, а сверху - только для одной, входящей в сумму знаменателя.

Как было указано в аналитическом разделе, формула обладает двумя регулируемыми параметрами:
α - значимость для муравья феромона, оставленного на путях;
β - важность расстояния до конкретного города.
Если отрегулировать данные параметры, можно добиться разной эффективности работы алгоритма.

В частности, если, к примеру, $\alpha = 0$, то алгоритм из муравьиного вырождается в жадный алгоритм, то есть муравей игнорирует опыт предыдущих агентов и просто выбирает самое кратчайшее ребро из всех возможных. Минусом такого выбора может являться то, что нет гарантии, что такой путь даст наилучшее решение, поскольку у более длинного ребра дальнейший путь может оказаться короче, чем тот, который получится по сделанному муравьём выбору.

Если же принять $\beta = 0$, то муравей игнорирует расстояние и руководствуется лишь феромоном, оставленным на путях. В таком случае алгоритм не зависит от конкретного расположения городов вообще, то есть характер его работы случайный, а результат непредсказуемый, что может дать в конкретных ситуациях такое решение, которое окажется хуже результата в случае превращения алгоритма в жадный.

Важно понимать, что нельзя слишком большую роль отводить случайности, а именно это и регулирует параметр β. Случайность в алгоритме должна лишь давать шанс на то, что будет выбран не самый оптимальный на первый взгляд путь, который может быть самым кратчайшим в дальнейшей перспективе.

Из данных соображений можно сделать вывод, что необходимо всегда устанавливать более значимым параметр β, однако М.Дориго, автор алгоритма, утверждает, что наилучший результат достигается при $\alpha = \beta$.

На феромон оказывает влияние параметр ρ - коэффициент испарения феромона:

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

Если $\rho = 0$, то феромон не испаряется вообще. Построим рассуждения следующим образом:

1. в самом начале количество феромона везде одинаково, вероятнее всего то, что муравей пойдёт к самому ближайшему городу;

2. так как феромон не испарялся, перевес ближайшего города начинает увеличиваться;

3. как итог, все муравьи будут выбирать кратчайшее ребро из доступных, следовательно, будет получена версия жадного алгоритма.

Если $\rho = 1$, то испаряется весь феромон, а это значит, что роль предыдущих поколений сводится к нулю. Теряется сам смысл муравьиного алгоритма - опора на опыт предыдущих агентов.

В результате, можно прийти к выводу, что испарение не должно принимать крайних значений, находясь между ними. Вероятно, оптимальным значением может быть $\rho \approx 0.5$, которое может несколько увеличиваться или уменьшаться, но при большом отклонении значения к крайним есть потенциальная вероятность вырождения алгоритма в жадный или вероятность потери опоры на опыт других поколений.

4.2 Пример работы

Для тестирования и оценки параметров был выбран граф, представленный на рисунке **??** и заданный следующей матрицей (первая строка и первый столбец задают города, пересечения их элементов задают расстояние между городами, если между ними есть прямой путь):

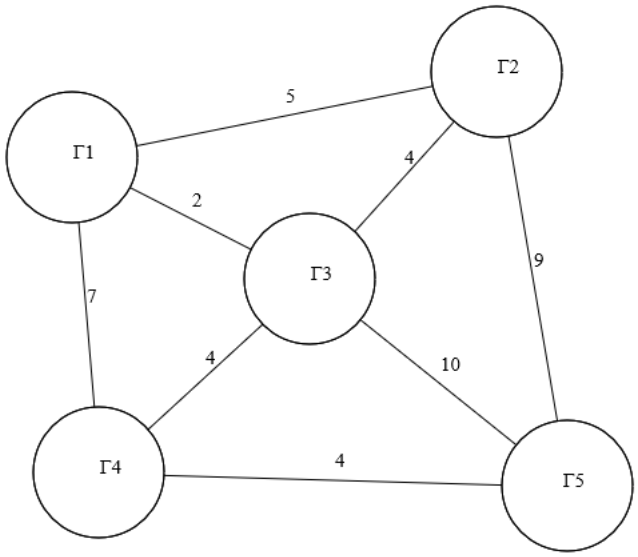

Рис. 9:

Таблица 1: Матрица расстояний между городами

	Г1	Г2	Г3	Г4	Г5
Г1	0	5	2	7	0
Г2	5	0	4	0	9
Г3	2	4	0	4	10
Г4	7	0	4	0	4
Г5	0	9	10	4	0

Для данной матрицы лучший путь, обнаруженный муравьиной колонией, составлял 17.

Нахождение решение перебором обнаружило результат, равный 15.

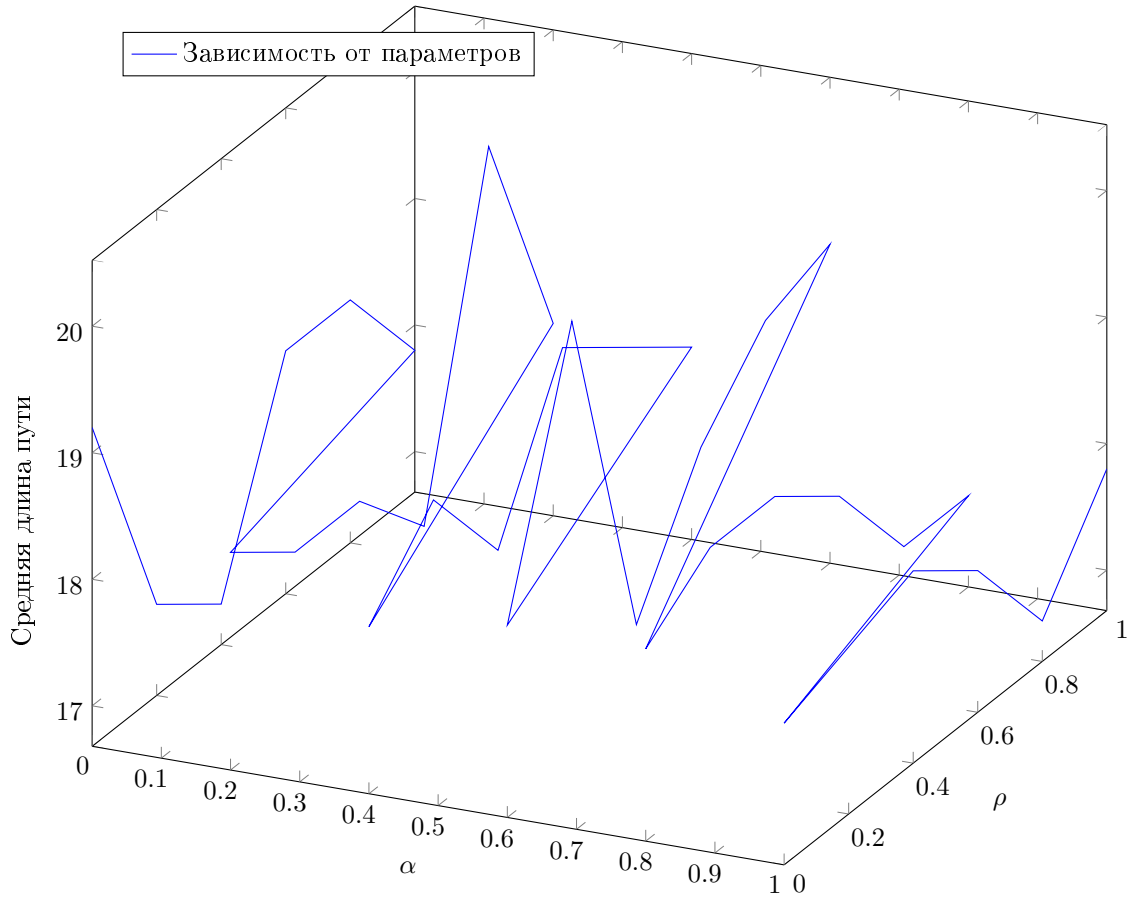

Рис. 10: График зависимостей параметров α, ρ и среднего значения найденного пути за всё время жизни колонии при данных параметрах

Таблица 2: Время работы алгоритмов

Время жизни колонии	Перебор	Муравьи	Разница времени работы в процентах
1	0.000223	0.000168	-33%
10	0.000249	0.001213	+21%
100	0.000282	0.012529	+98%
1000	0.000238	0.127736	+193%
10000	0.000222	1.216264	+292%

На графике 10 представлена зависимость средних значений длины пути при времени жихни колонии на выборках от 1 до 10^4 от параметров α и ρ (параметр β связан со значением α, поэтому на графике не рассматривается отдельно).

На рисунке **??** представлены средние значения с помощью поверхностной диаграммы.

Наборы параметров, при которых алгоритм постепенно увеличивает длину найденного пути:

Таблица 3: Наборы параметров при стабильном результате

Набор	α	β	ρ
1	0	1	0.4
2	1	0	0.8

Таблица 4: Наборы параметров при увеличении длины пути

Номер набора	α	β	ρ
1	0.2	0.8	0.2
2	0.2	0.8	0.4
3	1	0	0.6

Наборы параметров, при которых алгоритм позволяет постепенно снижать длину найденного пути:

Таблица 5: Наборы параметров при уменьшении длины пути

Номер набора	α	β	ρ
1	0	1	0
2	0.2	0.8	1
3	0.4	0.6	0
4	0.4	0.6	1
5	0.6	0.4	0
6	0.8	0.2	1

Набор параметров, рассматриваемых в гипотезе, заявленной в аналитическом разделе ($\alpha = 0.5, \beta = 0.5, \rho = 0.5$), при любом времени жизни колонии из рассматриваемого промежутка (от 1 до 10000) даёт минимальный путь из находимых муравьиным алгоритмом для данного примера, то есть 17. Результат остаётся аналогичным и в том случае, если при том же значении коэффициента испарения феромона параметры $\alpha = [0.4..0.5], \beta = [0.5..0.6]$.

4.3 Вывод

В результате экспериментов подтвердилось то, что оптимальной является ситуация, когда $\alpha = 0.5, \beta = 0.5, \rho = 0.5$. Если параметры α и β позволяли изменить их в противоположные стороны вплоть до $\Delta 0.1$ относительно уравновешенного значения, то параметр ρ увеличивал найденную длину пути даже при изменении на 10% в любую из сторон.

Стабильное изменение результатов работы (в том числе, постоянный результат при различных значениях времени жизни из выборки) муравьиной колонии на неориентированном графе сопровождалось тем, что параметры начинали вырождаться.

5 Заключение

В результате выполнения лабораторной работы была изучены задача коммивояжёра и способы её решения путём использования алгоритмов природных вычислений, получены навыки реализации муравьиного алгоритма, проведена его параметризация. Аналитическая параметризация, приведённая в конструкторском разделе, была подкреплена экспериментами. Было выявлено, что наилучший результат муравьиный алгоритм получает при $\alpha = 0.5, \beta = 0.5, \rho = 0.5$. Для коэффициента испарения феромона критичным оказалось даже изменение на 10%. Параметры α и β можно изменить до 10% от этих значений при преобладании коэффициента β (то есть 0.4 и 0.6 соответственно) без потери результата. В случае достижения параметрами граничных значений муравьиный алгоритм вырождается. Если $\alpha = 0$, то алгоритм становится жадным, а в противоположном случае перестаёт зависеть от расположения городов и начинает работать случайным образом. Если $\rho = 0$, то всё сведётся к жадному алгоритму, а если $\rho = 1$, то теряется зависимость от опыта предыдущих поколений колонии.

Список литературы

[1] Задача коммивояжера - метод ветвей и границ. – URL: *http : //galyautdinov.ru/post/zadacha – kommivoyazhera*

[2] Штовба С. Д. Муравьиные алгоритмы, Exponenta Pro. Математика в приложениях. 2004. № 4

[3] M. Dorigo & L. M. Gambardella, 1997. «Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem». IEEE Transactions on Evolutionary Computation, 1 (1): 53-66.

[4] М.В.Ульянов. Ресурсно-эффективные компьютерные алгоритмы // 2007. - Раздел III. - Глава 7. - С.195-206.