*Федеральное государственное бюджетное образовательное учреждение* *высшего профессионального образования*

|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | ***«Московский государственный технический университет  имени Н.Э. Баумана (национальный исследовательский институт)»***  ***(МГТУ им. Н.Э. Баумана)*** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА ИУ7

**Отчёт**

**по лабораторной работе №6**

**Дисциплина: Анализ алгоритмов**

**Тема лабораторной работы: Муравьиный алгоритм**

Студент гр. ИУ7-51Б **\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Уласик Е.А.** (Подпись, дата) (И.О. Фамилия)

Преподаватель  **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Волкова Л.Л.**

(Подпись, дата) (И.О. Фамилия)

Москва, 2019г.

[**Введение**](#_cyz545g5oplz) **3**

[**Аналитическая часть**](#_6o42uljmvtdx) **4**

[Задача коммивояжера](#_8wu100awdpjd) 4

[Методы решения](#_lfxano8jem07) 5

[Вывод](#_a4v8ckpk8paa) 7

[**Конструкторская часть**](#_h4ikfgj8vpue) **9**

[Схемы алгоритмов](#_fkritgd7gzrs) 9

[Вывод](#_sizk0iqntltb) 11

[**Технологическая часть**](#_y4sqeieoa19l) **13**

[Требования к ПО](#_thw60gg9h0ab) 13

[Средства реализации](#_thw60gg9h0ab) 13

[Листинги кода](#_thw60gg9h0ab) 13

[Вывод](#_wjbe302psaxk) 16

[**Экспериментальная часть**](#_nlgadzhlt75v) **17**

[Примеры работы программы](#_d98br0de3z55) 17

[Постановка эксперимента](#_d98br0de3z55) 19

[Сравнительный анализ на материалах эксперимента](#_d98br0de3z55) 19

[Вывод](#_ykpishucd1vt) 22

[**Заключение**](#_q9dg95p5sseq) **24**

[**Список источников**](#_he5m7e4ehkml) **25**

### Введение

Цель данной лабораторной работы: изучить муравьиный алгоритм по материалам решения задачи коммивояжера.

Задачи лабораторной работы следующие.

1. Описать методы решения.
2. Реализовать алгоритм и описать полученные результаты.
3. Выбрать класс данных и составить набор данных.
4. Провести параметризацию метода на основе муравьиного алгоритма для выбранного класса данных.
5. Провести сравнительный анализ двух методов.
6. Дать рекомендации о применимости метода решения задачи коммивояжера на основе муравьиного алгоритма.

### 

### Аналитическая часть

В данном разделе будет описана задача коммивояжера и некоторые методы её решения.

#### Задача коммивояжера

В общем случае задача коммивояжера (странствующего торговца) может быть сформулирована следующим образом: найти самый выгодный (самый короткий, самый дешевый, и т.п.) маршрут, начинающийся в исходном городе и проходящий ровно один раз через каждый из указанных городов, с последующим возвратом в исходный город [1].

Проблему коммивояжера можно представить в виде модели на графе, то есть, используя вершины и ребра между ними. Таким образом, M вершин графа соответствуют M городам, а ребра (i, j) между вершинами i и j — пути сообщения между этими городами. Каждому ребру (i, j) можно сопоставить критерий выгодности маршрута cij ≥ 0,который можно понимать как, например, расстояние между городами, время или стоимость поездки. В целях упрощения задачи и гарантии существования маршрута обычно считается, что модельный граф задачи является полностью связным, то есть, что между произвольной парой вершин существует ребро.

Гамильтоновым циклом называется маршрут, включающий ровно по одному разу каждую вершину графа. Таким образом, решение задачи коммивояжёра — это нахождение гамильтонова цикла минимального веса в полном взвешенном графе.

#### Методы решения

Пусть нам дано M - число городов, D - матрица смежности, каждый элемент которой - вес пути из одного города в другой. Существует метод грубой силы решения поставленной задачи, а именно полный перебор всех возможных гамильтоновых циклов в заданном графе с нахождением минимального по весу. Этот метод гарантированно даст идеальное решение (глобальный минимум по весу). Однако стоит учитывать, что сложность такого алгоритма составляет M! и время выполнения программы, реализующий такой подход, будет расти экспоненциально в зависимости от размеров входной матрицы.

Поскольку на практике чаще всего необходимо получить решение как можно быстрее, при этом требуемое решение не обязательно должно быть наилучшем, был разработан ряд методов, называемых эвристическими, которые решают поставленную задачу за гораздо меньшее время, чем метод полного перебора. В основе таких методов лежат принципы из окружающего мира, которые в дальнейшем могут быть формализованы.

Одним из таких методов является муравьиный метод. Более подробное описание и возможные оптимизации алгоритма описаны в книге М.В. Ульянова [2]. Он применим к решению задачи коммивояжера и основан на идее муравейника. Модель данного метода: у муравья есть 3 чувства:

* зрение (муравей может оценить длину ребра);
* обоняние (муравей может унюхать феромон - вещество, выделяемое муравьем, для коммуникации с другими муравьями);
* память (муравей запоминает свой маршрут).

Благодаря введению обоняния между муравьями возможен непрямой обмен информацией.

Введем вероятность Pk, ij(t) выбора следующего города j на маршруте муравьем k, который в текущий момент времени t находится в городе i.

(1)

В формуле(1) τij(t) - феромон на ребре ij, ηij - “привлекательность” ребра для муравья ηij = 1/d(ij), где d(ij) - длина ребра ij, J - список городов, которые к моменту времени t муравей еще не посетил.

𝛼, 𝛽 - настроечные параметры, определяющие соотношение важности τ и η. 𝛼 + 𝛽 = const. 𝛼 - коэффициент “стадности”, если 𝛼 = 0, то алгоритм вырождается в “жадный” (т.е. всегда выбирает самое короткое ребро), 𝛽 - коэффициент “ жадности”, если 𝛽 = 0, то алгоритм всегда будет выбирать путь, на котором отложено больше феромона.

Чтобы повысить вариативность выбора (и не выбирать просто максимальную вероятность), можно использовать случайное действительное число от 0 до 1. Последовательно суммируя полученные вероятности Pij(t) пока не получим значение большее или равное случайному числу. Та вероятность, суммирование с которой дало такой результат и будет подходящей, следующим будет город j, для которого была вычислена эта вероятность.

За условный день каждый муравей k из колонии проходит один маршрут, откладывая на каждом ребре ij, которое принадлежит маршруту муравья, определенное количество феромона. Количество феромона вычисляется по следующей формуле:

(2), где Lk - длина пути k-го муравья; Q - нормировочная константа, которая задается изначально и соразмерна длине лучшего пути. После окончания условного дня наступает условная ночь, в течение которой феромон испаряется с ребер с коэффициентом 𝜌. Количество феромона на следующий день вычисляется по следующей формуле:

(3)

Таким образом, псевдокод муравьиного алгоритма можно представить так:

1. Ввод матрицы расстояний D, количества городов M;
2. Инициализация параметров алгоритма — 𝛼, 𝛽, Q, tmax, 𝜌;
3. Инициализация ребер — присвоение “привлекательности” ηij и начальной концентрации феромона τstart;
4. Размещение муравьев в случайно выбранные города без совпадений;
5. Инициализация начального кратчайшего маршрута Lp = null и определение длины кратчайшего маршрута Lmin = inf;
6. Цикл по времени жизни колонии t = 1, tmax;
   1. Цикл по всем муравьям k = 1, M;
      1. Построить маршрут Tk(t) по правилу (1) и рассчитать длину получившегося маршрута Lk(t);
      2. Обновить феромон на маршруте по правилу (2);
      3. Если Lk(t) < Lmin, то Lmin = Lk(t) и Lp = Tk(t);
   2. Конец цикла по муравьям;
   3. Цикл по всем ребрам графа;
      1. Обновить следы феромона на ребре по правилам (3);
   4. Конец цикла по ребрам;
7. Конец цикла по времени;
8. Вывести кратчайший маршрут Lp и его длину Lmin.

#### Вывод

Применение муравьиного алгоритма для решения задачи коммивояжера обосновано в тех случаях, когда необходимо быстро найти решение или когда для решения задачи достаточно получения первого приближения.

При этом ряд экспериментов показывает, что эффективность муравьиных алгоритмов растет с ростом размерности решаемых задач оптимизации [2], что также является преимуществом этого метода.

### Конструкторская часть

В данном разделе будут представлены схемы алгоритмов.

#### Схемы алгоритмов

На рисунках 2.1.1 - 2.1.2 будет представлена схема муравьиного алгоритма. На рисунках 2.1.3 - 2.1.4 будет представлена схема алгоритма полного перебора.

|  |  |
| --- | --- |
|  |  |
| *Рис. 2.1.1 - муравьиный алгоритм (1)* | *Рис. 2.1.2 - муравьиный алгоритм (2)* |

|  |
| --- |
|  |
| *Рис. 2.1.3 - алгоритм полного перебора(1)* |

|  |
| --- |
|  |
| *Рис. 2.1.4 - алгоритм полного перебора(2)* |

#### Вывод

Как видно из схем алгоритмов, количество блоков операций над данным в алгоритме полного перебора меньше, чем в муравьином. Это объясняется более сложной моделью данных в случае муравьиного метода. При этом стоит отметить, что в стандартный алгоритм полного перебора внесено одно изменение, а именно проверка текущей длины пути: если она уже больше длины минимального на текущий момент маршрута, то дальше маршрут по этому пути не строится. Такая модификация позволяет сократить время выполнения программы в несколько раз, и необходима, чтобы провести последующие эксперименты (см. экспериментальную часть отчета).

### Технологическая часть

В этом разделе будут описаны требования к программе, средства реализации и представлены листинги кода.

#### Требования к ПО

Программа на вход получает матрицу смежности графа: двумерный массив целых чисел. Результат работы программы: последовательность чисел - минимальный по стоимости маршрут и одно целое число - суммарная стоимость этого маршрута (включая возвращение в исходный город).

#### Средства реализации

Для реализации вышеописанных алгоритмов был выбран язык программирования C++. Были использованы классы vector и pair из стандартной библиотеки. Среда разработки: CLion.

#### Листинги кода

В процессе разработки программы был реализован шаблон класса Matrix, облегчающий работу с матрицей смежности, а также с матрицами для хранения количества феромона на ребрах.

В листинге 1 будет представлена реализация муравьиного алгоритма, основная часть которого представлена функцией aco. Функция range(...) добавляет во второй вектор те значения, которых нет в первом, значения - целые числа от 0 до переданного count. Функция index(...) производит поиск в заданном векторе указанного значения и возвращает индекс элемента при первом совпадении. Функция rand\_double() возвращает случайное вещественное число от 0 до 1 включительно.

Листинг 1: муравьиный алгоритм.

1. pair<int, vector<int>> aco(const Matrix<int> &distances, const int &t\_max, const double &alpha, const double &ro)
2. {
3. const unsigned int town\_number = distances.n;
4. const double Q = distances.compute\_avg() \* town\_number;
5. const double betta = ALPHA\_BETTA\_CONST - alpha;
6. Matrix <double> town\_attraction(town\_number); // inversed distances
7. town\_attraction.reverse\_values(distances);
8. Matrix<double> teta(town\_number); // amount of pheromone on the paths
9. teta.fill\_value(teta\_start);
10. Matrix<double> delta\_teta(town\_number); // amount of pheromone added
11. int min\_path\_length = -1;
12. vector<int> min\_path;
13. vector<double> probabilities(town\_number, 0.0);
14. double coin;
15. for (int t = 0; t < t\_max; t++)
16. {
17. delta\_teta.zero();
18. // loop over each ant
19. for (int k = 0; k < town\_number; k++)
20. {
21. vector<int> current\_path = {k};
22. int current\_path\_length = 0;
23. int i = k; // current town
24. // loop to get the path for the current ant
25. while (current\_path.size() != town\_number)
26. {
27. vector<int> not\_visited;
28. range(not\_visited, current\_path, town\_number);
29. fill(probabilities.begin(), probabilities.end(), 0.0);
30. // calculating P
31. for (auto j : not\_visited)
32. {
33. int cur\_index = index(not\_visited, j);
34. if (distances[i][j] != 0)
35. {
36. double sum = 0;
37. for (auto n : not\_visited)
38. sum += pow(teta[i][n], alpha) \* pow(town\_attraction[i][n], betta);
39. probabilities[cur\_index] = pow(teta[i][j], alpha) \* pow(town\_attraction[i][j], betta) / sum;
40. }
41. else
42. probabilities[cur\_index] = 0;
43. }
44. // toss a 'coin' to choose which town is next
45. coin = rand\_double();
46. unsigned int best\_p = 0;
47. double sum\_p = 0;
48. // calculating best\_p...
49. // when the sum of probabilities will be bigger than coin value than we found best\_p - index of next town in not\_visited
50. for (unsigned int s = 0; s < town\_number; s++)
51. {
52. sum\_p += probabilities[s];
53. if (coin < sum\_p)
54. {
55. best\_p = s;
56. break;
57. }
58. }
59. int to\_add = not\_visited[best\_p]; // new town in the path
60. // adding new town to the total path
61. current\_path.push\_back(to\_add);
62. // updating total path length
63. current\_path\_length += distances[i][to\_add];
64. // updtaing pheromone on the current edge
65. delta\_teta[i][to\_add] += Q / current\_path\_length;
66. i = to\_add; // current town is changed for next iteration
67. not\_visited.erase(not\_visited.begin() + best\_p);
68. }
69. // counting way back
70. current\_path\_length += distances[current\_path[current\_path.size() - 1]][current\_path[0]];
71. // updating minimum path and it's length
72. if (min\_path\_length == -1 || (current\_path\_length < min\_path\_length))
73. {
74. min\_path\_length = current\_path\_length;
75. min\_path = current\_path;
76. }
77. }
78. // updating pheromone on all edges during night
79. teta \*= (1.0 - ro);
80. teta += delta\_teta;
81. }
82. return pair<int, vector<int>>(min\_path\_length, min\_path);
83. }

В листинге 2 будет представлена реализация алгоритма полного перебора. Основная функция вызывает в цикле для каждой вершины вспомогательную функцию hamilton(...), в которой проверяются все непосещенные вершины и если такие есть, то эта же функция вызывается рекурсивно для следующих вершин и так пока не будет пройден полноценный маршрут.

Листинг 2: полный перебор.

1. pair<int, vector<int> > brute\_force(const Matrix<int> &distances)
2. {
3. int n = distances.n;
4. vector<bool> visited(n, 0);
5. vector<int> cur\_path;
6. vector<int> min\_path;
7. int cur\_len = 0;
8. int min\_path\_len = INT\_MAX;
9. for (int i = 0; i < n; i++)
10. {
11. cur\_path.clear();
12. cur\_path.push\_back(i);
13. fill(visited.begin(), visited.end(), 0);
14. visited[i] = 1;
15. cur\_len = 0;
16. hamilton(distances, min\_path, min\_path\_len, cur\_path, visited, cur\_len);
17. }
18. cout << "brute force method found " << s << " routes" << endl;
19. return pair<int, vector<int>>(min\_path\_len, min\_path);
20. }
21. void hamilton(const Matrix<int> &distances, vector<int> &min\_path, int &min\_distance, vector<int> &cur\_path, vector<bool> &visited, int &cur\_len)
22. {
23. if (cur\_path.size() == distances.n)
24. {
25. s++;
26. int tmp = distances[cur\_path.back()][cur\_path[0]];
27. if (cur\_len + tmp < min\_distance)
28. {
29. min\_path = cur\_path;
30. min\_distance = cur\_len + tmp;
31. }
32. return;
33. }
34. for (int i = 0; i < distances.n; i++)
35. {
36. if (!visited[i])
37. {
38. int tmp = distances[cur\_path.back()][i];
39. if (cur\_len + tmp > min\_distance)
40. continue;
41. cur\_len += tmp;
42. cur\_path.push\_back(i);
43. visited[i] = 1;
44. hamilton(distances, min\_path, min\_distance, cur\_path, visited, cur\_len);
45. visited[i] = 0;
46. cur\_path.pop\_back();
47. cur\_len -= tmp;
48. }
49. }
50. }

#### Вывод

Были реализованы два алгоритма, решающие задачу коммивояжера, для дальнейшего сравнительного анализа точности и скорости вычислений.

### Экспериментальная часть

В данном разделе будут приведены примеры работы программы, а также будет проведена параметризация метода решения задачи коммивояжера на основе муравьиного алгоритма, будет приведен сравнительный анализ двух алгоритмов.

#### Примеры работы программы

На рисунках 4.1.1 - 4.1.3 будут приведены примеры работы программы. В консоль выводится входная матрица смежности, для двух алгоритмов: длина кратчайшего маршрута и сам маршрут.

|  |
| --- |
|  |
| *Рис. 4.1.1 - размер входной матрицы 3* |

|  |
| --- |
|  |
| *Рис. 4.1.2 - размер входной матрицы 5* |

|  |
| --- |
|  |
| *Рис. 4.1.3 - размер входной матрицы 10* |

Как видно из приведенных примеров, на бòльших размерностях, муравьиный алгоритм находит не идеальное решение, в то время как полный перебор находит наилучший вариант в любом случае. При функциональном тестировании все тесты были пройдены и результаты совпали с ожидаемыми.

#### Постановка эксперимента

Для проведения экспериментов была использована матрица расстояний 10x10, изображенная на рисунке 4.2.1. В каждом эксперименте фиксировались значения 𝛼, 𝛽, ρ и tmax. В течение экспериментов значения 𝛼, 𝛽, ρ менялись от 0 до 1 включительно с шагом 0.25, tmax от 10 до 300 с шагом 10. Результатом проведения эксперимента считалась разница между длинной маршрута, рассчитанного алгоритмом полного перебора и муравьиным алгоритмом с текущими параметрами. Количество повторов каждого эксперимента = 100. Результат одного эксперимента рассчитывается как средний из результатов проведенных испытаний с одинаковыми входными данными.

|  |
| --- |
|  |
| *Рис. 4.2.1 - матрица расстояний для экспериментов.* |

#### Сравнительный анализ на материалах эксперимента

В таблице 4.3.1 представлены результаты проведенных экспериментов (первые 25 строк) для параметров 𝛼, ρ и tmax (𝛽 вычисляется непосредственно в функции поиска кратчайшего расстояния в зависимости от значения 𝛼). Первый столбец представляет собой значения параметра ρ, второй - 𝛼 и третий - tmax. В четвертом столбце (difference) представлено среднее отклонение от длины идеального маршрута, вычисленной алгоритмом полного перебора. Значения в таблице отсортированы по возрастанию как раз по четвертому столбцу.

*Таблица 4.3.1: результаты параметризации муравьиного алгоритма.*

|  |  |  |  |
| --- | --- | --- | --- |
| ρ | α | tmax | difference |
| 0.5 | 0.5 | 290 | 0.00040302 |
| 0.25 | 0.75 | 310 | 0.010204 |
| 0.75 | 0.5 | 310 | 0.0106051 |
| 0.5 | 0.5 | 260 | 0.020207 |
| 0.5 | 0.5 | 310 | 0.0203 |
| 0.25 | 0.75 | 300 | 0.020404 |
| 0.5 | 0.5 | 250 | 0.0207041 |
| 0.75 | 0.5 | 250 | 0.0209101 |
| 0.5 | 0.5 | 300 | 0.030004 |
| 0.5 | 0.5 | 270 | 0.0302021 |
| 0.75 | 0.5 | 270 | 0.0304021 |
| 0.25 | 0.5 | 310 | 0.0305061 |
| 0.75 | 0.5 | 260 | 0.0402091 |
| 0.5 | 0.5 | 280 | 0.040302 |
| 0.25 | 0.75 | 290 | 0.0404041 |
| 0.25 | 0.75 | 280 | 0.0404051 |
| 0.75 | 0.25 | 310 | 0.0404071 |
| 0.25 | 0.75 | 240 | 0.0404081 |
| 0.25 | 0.75 | 270 | 0.040505 |
| 0.75 | 0.25 | 300 | 0.0407081 |
| 0.25 | 0.75 | 230 | 0.0408051 |
| 0.5 | 0.5 | 230 | 0.0410121 |
| 0.25 | 0.75 | 250 | 0.0504041 |
| 0.25 | 0.75 | 260 | 0.050504 |
| 0.25 | 0.5 | 300 | 0.0506091 |

Как видно из представленной выше таблицы, наилучшим значение настроечного параметра 𝛼 является значение, равное 0.5. При этом коэффициент испарения ρ также равен 0.5. Наилучший результат достигается при tmax = 290 дням. Если рассмотреть всю получившуюся таблицу, то для различных tmax наилучшими вариантами сочетаний параметров ρ и 𝛼 будут соответственно 0.5 и 0.5, 0.75 и 0.5, 0.25 и 0.75.

Далее сравним муравьиный алгоритм с подобранными параметрами (0.5, 0.5, 290) и алгоритм полного перебора для решения задачи коммивояжера. На рисунках 4.3.2 и 4.3.3 представлены графики зависимости времени работы каждого из реализованных алгоритмов в зависимости от размера входной матрицы.

|  |
| --- |
|  |
| *Рис. 4.3.2 - график зависимости времени выполнения алгоритмов в зависимости от размера входной матрицы (n ∊ [2, .., 14]).* |

|  |
| --- |
|  |
| *Рис. 4.3.3 - график зависимости времени выполнения алгоритмов в зависимости от размера входной матрицы (n ∊ [2, .., 15]).* |

Как можно наблюдать на графиках, время работы алгоритма полного перебора растет экспоненциально в зависимости от размера входной матрицы, в то время как время выполнения муравьиного алгоритма изменяется практически линейно в зависимости от размера входной матрицы.

#### Вывод

В результате проведенных экспериментов были выявлены следующие оптимальные параметры для муравьиного алгоритма (представлены в таблице 4.3.4. Однако стоит учитывать, что чем больше значение tmax, тем больше вероятность того, что будет найден идеальный маршрут. При этом с увеличение tmax будет возрастать время выполнения программы.

*Таблица 4.3.4: оптимальные параметры для муравьиного алгоритма.*

|  |  |  |
| --- | --- | --- |
| ρ | 𝛼 | tmax |
| 0.5 | 0.5 | 290 |
| 0.75 | 0.5 | 290 |
| 0.25 | 0.75 | 290 |

Также был проведен сравнительный анализ муравьиного алгоритма и алгоритма полного перебора. Алгоритм полного перебора рационально использовать для матриц небольшого размера (n ≤ 10, 11) и когда необходимо найти гарантировано лучшее решение. Во всех остальных случаях муравьиный алгоритм работает эффективнее по времени, чем алгоритм грубой силы.

### Заключение

Цель данной лабораторной работы была достигнута. В ходе работы были изучены муравьиная модель для решения задачи коммивояжера и алгоритм полного перебора, решающий эту же задачу. Данные алгоритмы были реализованы, также был проведен их сравнительный анализ. Были экспериментально получены зависимости времени выполнения алгоритмов от размеров входной матрицы расстояний, кроме того, была проведена параметризация муравьиной модели. В результате: муравьиный алгоритм работает быстрее алгоритма полного перебора и линейно зависит от размеров входной матрицы. При проведении экспериментов были выявлены оптимальные параметры для муравьиного алгоритма, обеспечивающие минимальное отклонение длины вычисленного маршрута от длины идеального. Таким образом, муравьиный метод может быть применен для решения задачи коммивояжера с указанными настроечными параметрами.

### Список источников

1. Задача коммивояжёра. [Электронный ресурс] Режим доступа: <https://en.wikipedia.org/wiki/Travelling_salesman_problem>/ (дата обращения: 21.11.2019).
2. Ульянов М.В. РЕСУРСНО-ЭФФЕКТИВНЫЕ КОМПЬЮТЕРНЫЕ АЛГОРИТМЫ. РАЗРАБОТКА И АНАЛИЗ. [Текст] / М.В. Ульянов // Наука ФИЗМАТЛИТ. - 2007.