МГТУ им. Баумана

Лабораторная работа №5

По курсу: "Анализ алгоритмов"

Муравьиные алгоритмы

Работу выполнила: Лаврова Анастасия, ИУ7-55Б

Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Оглавление

Введение

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах. Суть подхода заключается в анализе и использовании модели поведения муравьёв, ищущих пути от колонии к источнику питания, и представляет собой метаэвристическую оптимизацию.

Целью данной лабораторной работы является изучение муравьиного алгоритма для решения задачи коммивояжера, анализ влияния параметров алгоритма на время его выполнения.

Задача: провести сравнение муравьиного алгоритма и полного перебора.

1 | Аналитическая часть

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной их предыдущих решений. Они могут использоваться как для статических, так и для динамических комбинаторных оптимизационных задач. Сходимость гарантирована, то есть в любом случае мы получим оптимальное решение, однако скорость сходимости неизвестна.

Идея муравьиного алгоритма - моделирование поведения муравьев, связанного с их способностью быстро находить кратчайший путь. При своем движении муравей метит путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьев находить новый путь, если старый оказывается недоступным. Рассмотрим случай, показанный на рисунке, когда на оптимальном доселе пути возникает преграда. В этом случае необходимо определение нового оптимального пути. Дойдя до преграды, муравьи с равной вероятностью будут обходить её справа и слева. То же самое будет происходить и на обратной стороне преграды. Однако, те муравьи, которые случайно выберут кратчайший путь, будут быстрее его проходить, и за несколько передвижений он будет более обогащен феромоном. Поскольку движение муравьев определяется концентрацией феромона, то следующие будут прдпочитать именно этот путь, продолжая обогощать его феромоном до тех пор, пока этот путь по какой-либо причине не станет недоступен.

Очевидная положительная обратная связь приведет к тому, что кратчайший путь станет единственным маршрутом движения большинства муравьев. Моделирование испарения феромона - отрицательной обрат-

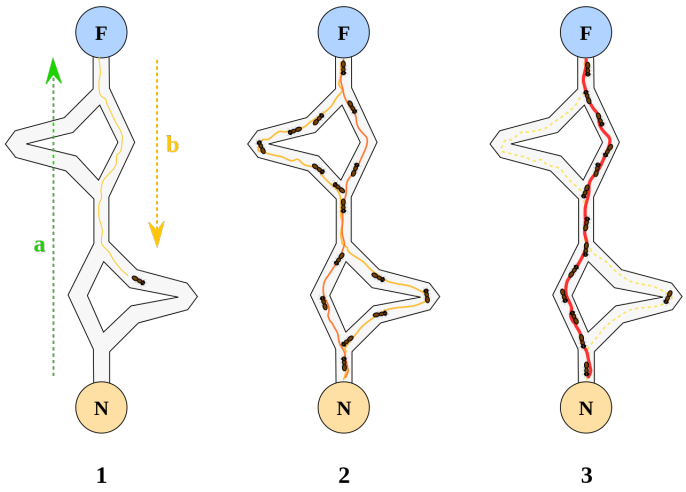


Рис. 1.1: Принцип работы муравьиного алгоритма

ной связи - гарантирует нам, что найденное локально оптимальное решение не будет единственным - муравьи будут искать и другие пути. Если мы моделируем процесс такого поведения на некотором графе, рёбра которого представляют собой возможные пути перемещения муравьёв, в течение определённого времени, то наиболее обогащённый феромоном путь по рёбрам этого графа и будет являться решением задачи, полученным с помощью муравьиного алгоритма.

1.1 Обобщенный алгоритм

Любой муравьиный алгоритм, независимо от модификаций, представим в следующем виде:

- Создаем муравьев
- Ищем решения
- Обновляем феромон
- Дополнительные действия (опционально)

1.2 Применение для задачи коммивояжёра

Теперь рассмотрим каждый шаг в цикле более подробно:

1. Создаем муравьёв

Стартовая точка, куда помещается муравей, зависит ограничений, накладываемых условиями задачи. Потому что для каждой задачи способ размещения муравьёв является определяющим. Либо все они помещаются в одну точку, либо в разные с повторения, либо без повторений.
На этом же этапе задается начальный уровень феромона. Он инициализируется небольшим положительным числом для того, чтобы на начальном шаге вероятности перехода в следующую вершину не были нулевыми.

2. Ищем решения

Вероятность перехода из вершины i в вершину j определяется по следующей формуле:

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

где

$\tau_{i,j}$ – расстояние от города i до j;

$\eta_{i,j}$ – количество феромонов на ребре ij;

α – параметр влияния длины пути;

β – параметр влияния феромона.

3. Обновляем феромон

Уровень феромона обновляется в соответствии с приведённой формулой:
После того, как муравей успешно проходит маршрут, он оставляет на всех пройденных ребрах след, обратно пропорциональный длине пройденного пути:

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j}, \tag{1.2}$$

где

$\rho_{i,j}$ – доля феромона, который испарится;

$\tau_{i,j}$ – количество феромона на дуге ij;

$\Delta\tau_{i,j}$ – количество отложенного феромона.

Также нужно заметить, что количество отложенного феромона ($\tau_{i,j}$) является суммой всех $\Delta\tau^k_{i,j}$:

$$\Delta\tau^k_{i,j} = \begin{cases} Q/L_k & \text{Если k-ый мурваей прошел по ребру ij;} \\ 0 & \text{Иначе} \end{cases} \tag{1.3}$$

где

Q – количество феромона, переносимого муравьем;

L_k – стоимость k-го пути муравья (обычно длина).

4. Дополнительные действия
Обычно здесь используется алгоритм локального поиска, однако он может также появиться и после поиска всех решений.

Теперь с учетом особенностей задачи коммивояжёра, мы можем описать локальные правила поведения муравьев при выборе пути.
1. Муравьи имеют собственную «память». Поскольку каждый город может быть посещён только один раз, то у каждого муравья есть список уже посещенных городов - список запретов. Обозначим через J список городов, которые необходимо посетить муравью k , находящемуся в городе i .
2. Муравьи обладают «зрением» - видимость есть эвристическое желание посетить город j , если муравей находится в городе i . Будем считать, что видимость обратно пропорциональна расстоянию между городами.
3. Муравьи обладают «обонянием» - они могут улавливать след феромона, подтверждающий желание посетить город j из города i на основании опыта других муравьёв. Количество феромона на ребре (i, j) в момент времени t обозначим через $tau_{i,j}(t)$
4. На этом основании мы можем сформулировать вероятностнопропорциональное правило, определяющее вероятность перехода k-ого муравья

из города i в город j.
5. Пройдя ребро (i, j) , муравей откладывает на нём некоторое количество феромона, которое должно быть связано с оптимальностью сделанного выбора. Пусть $T_k(t)$ есть маршрут, пройденный муравьем k к моменту времени t , $L_k(t)$ - длина этого маршрута, а Q - параметр, имеющий значение порядка длины оптимального пути.

1.3 Алгоритм перебора

Задача коммивояжёра в общем случае гарантированно решается оптимально только полным перебором всех вариантов. И точный переборный алгоритм её решения имеет факториальную сложность. Поэтому алгоритм практически бесполезен при N $>$ 15.

2 | Конструкторская часть

Требования к вводу:
Коэффициенты, влияющие на веса следа феромона (α и β), коэффициент испарения ρ, время жизни колонии t_{max}, матрица смежности
Требования к программе:
Длина кратчайшего пути

2.1 Разработка реализации программы

Рассмотрим псевдокод муравьиного алгоритма:

Обозначим через $T*$ наилучший текущий маршрут, через $L*$ — его длину.
1. Ввод матрицы расстояний D.
2. Инициализация параметров алгоритма — α, ρ, t_{max}.
3. Инициализация ребер — присвоение видимости n_{ij} и начальной концентрации феромона.
4. Размещение муравьев в случайно выбранные города без совпадений.
5. Выбор начального кратчайшего маршрута и определение $L*$.
6. Цикл по времени жизни колонии $t = 1, t_{max}$.
    7. Цикл по всем муравьям $k = 1, m$.
        8. Построить маршрут T_k(t) и рассчитать длину L_k(t).
    9. конец цикла по муравьям.
    10. Проверка всех $L_k(t)$ на лучшее решение по сравнению с $L*$.
    11. Если да, то обновить $L*$ и $T*$.
    12. Цикл по всем ребрам графа.
        13. Обновить следы феромона на ребре.
    14. конец цикла по ребрам.

15. конец цикла по времени.
16. Вывести кратчайший маршрут $T*$ и его длину $L*$.

Сложность алгоритма — $\Theta(t_{max} \cdot max(m, n^2))$
Следовательно, сложность зависит от времени жизни колонии, количества городов и количества муравьев в колонии.

На рис. 2.2 представлена схема алгоритма полного перебора:
На рис. 2.2 представлена схема муравьиного алгоритма:

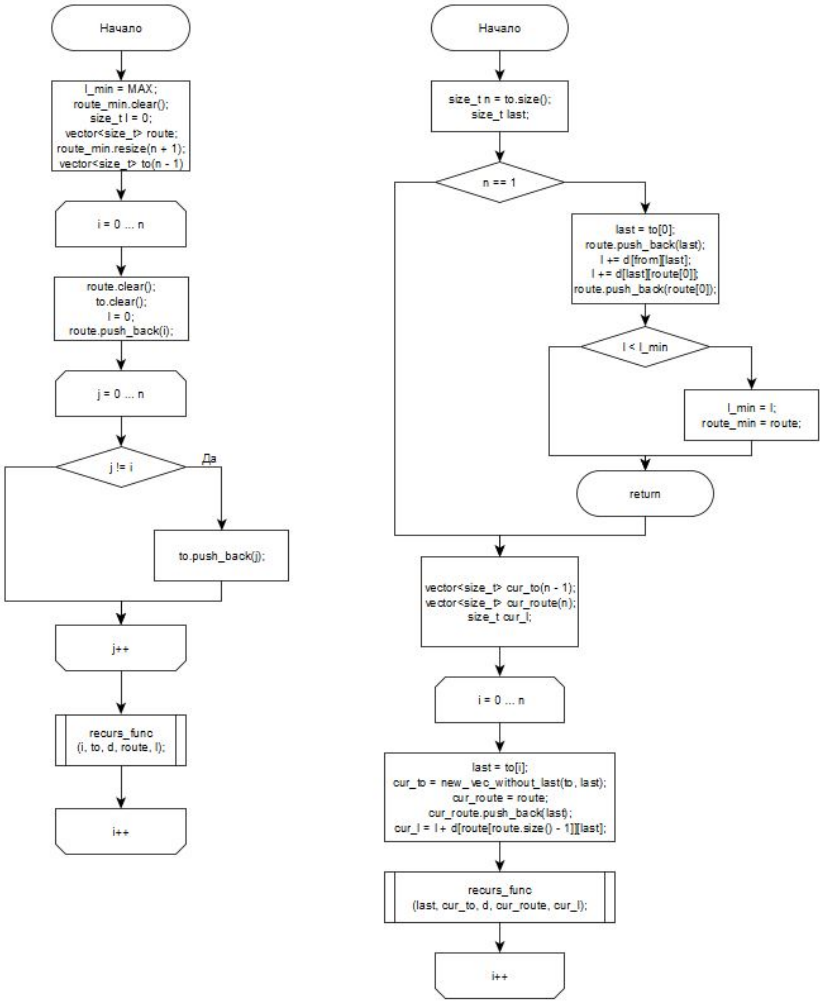


Рис. 2.1: Схема алгоритма полного перебора

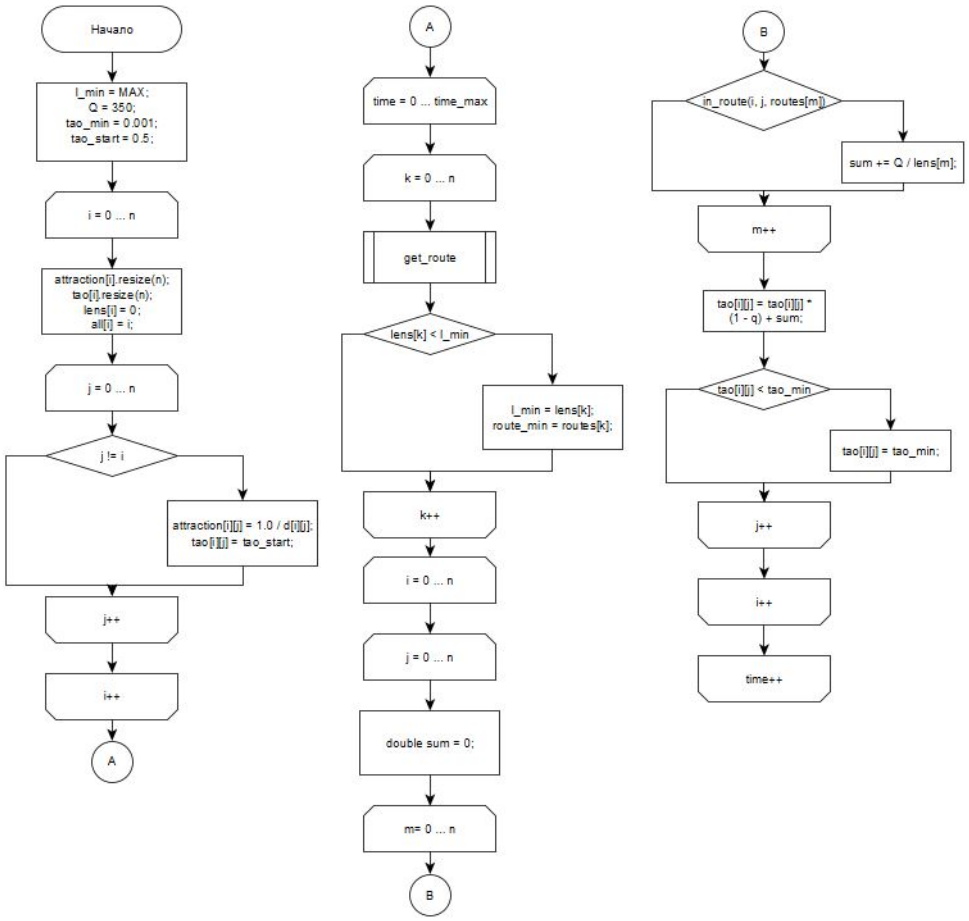


Рис. 2.2: Схема муравьиного алгоритма

3 | Технологическая часть

3.1 Выбор ЯП

Для реализации программ я выбрала язык программирования C++, так имею большой опыт работы с ним. Среда разработки - Visual Studio.

Для замера процессорного времени используется функция, возвращающая количество тиков (Листинг 3.1).

В листинге 3.2 рассмотрен алгоритм полного перебора, а в листинге 3.3 представлен муравьиный алгоритм.

Листинг 3.1: Функция получения тиков

```
unsigned __int64 tick()
{
  return (__rdtsc());
}
```

Листинг 3.2: Реализация полного перебора

```
void recurs_func(size_t from, vector<size_t> to, vector<
    vector<size_t>> d, vector<size_t> route, size_t l) {

  size_t n = to.size();
  size_t last;
  if (n == 1) {
    last = to[0];
    route.push_back(last);
    l += d[from][last];
```

```
9       l += d[last][route[0]];
10      route.push_back(route[0]);
11      if (l < l_min) {
12        l_min = l;
13        route_min = route;
14      }
15      return;
16    }
17
18    vector<size_t> cur_to(n - 1);
19    vector<size_t> cur_route(n);
20    size_t cur_l;
21    for (size_t i = 0; i < n; i++) {
22      last = to[i];
23      cur_to = new_vec_without_last(to, last);
24      cur_route = route;
25      cur_route.push_back(last);
26      cur_l = l + d[route[route.size() - 1]][last];
27      recurs_func(last, cur_to, d, cur_route, cur_l);
28    }
29
30  }
31
32  void perebor(size_t n, vector<vector<size_t>> d) {
33    l_min = MAX;
34    route_min.clear();
35    size_t l = 0;
36    vector<size_t> route;
37    route_min.resize(n + 1);
38    vector<size_t> to(n - 1);
39
40    for (size_t i = 0; i < n; i++) {
41      route.clear();
42      to.clear();
43      l = 0;
44
45      route.push_back(i);
46      for (size_t j = 0; j < n; j++)
47        if (j != i)
48          to.push_back(j);
```

```
49      recurs_func(i, to, d, route, l);
50    }
51
52    cout << endl << "ROUTE: ";
53    print_arr(route_min);
54    cout << "LENGTH: " << l_min << endl << endl;
55  }
```

Листинг 3.3: Реализация муравьиного алгоритма

```
1  vector<double> get_probability(size_t from, vector<size_t>
       to, vector<vector<double>> tao, vector<vector<double>>
       attraction,
2    size_t alpha, size_t beta) {
3
4    double znam = 0, chisl = 0;
5    size_t n = to.size();
6    vector<double> result(n);
7    for (size_t i = 0; i < n; i++) {
8      znam += pow(tao[from][to[i]], alpha) * pow(attraction[
         from][to[i]], beta);
9    }
10   for (size_t j = 0; j < n; j++) {
11     chisl = pow(tao[from][to[j]], alpha) * pow(attraction[
         from][to[j]], beta);
12     result[j] = chisl / znam;
13   }
14   return result;
15 }
16
17 void get_route(vector<size_t> all, size_t start, vector<
       size_t> &route, size_t &len, vector<vector<size_t>> d,
18   vector<vector<double>> tao, vector<vector<double>>
       attraction,
19   size_t alpha, size_t beta) {
20
21   route.resize(0);
22   route.push_back(start);
23   vector<size_t> to = new_vec_without_last(all, start);
24   size_t n_1 = tao.size() - 2;
25   size_t from;
```

```
26    double coin, sum;
27    bool flag;
28
29    for (size_t i = 0; i < n_1; i++) {
30      sum = 0;
31      flag = true;
32      from = route[i];
33      vector<double> p = get_probability(from, to, tao,
          attraction, alpha, beta);
34      coin = double(rand() % 10000) / 10000;
35      for (size_t j = 0; j < p.size() && flag; j++) {
36        sum += p[j];
37        if (coin < sum) {
38          route.push_back(to[j]);
39          len += d[from][to[j]];
40          to = new_vec_without_last(to, to[j]);
41          flag = false;
42        }
43      }
44    }
45    len += d[route[route.size() - 1]][to[0]];
46    route.push_back(to[0]);
47    len += d[route[route.size() - 1]][route[0]];
48    route.push_back(route[0]);
49  }
50
51  void ant(size_t n, vector<vector<size_t>> d, size_t alpha,
      size_t beta, double q, size_t time_max, ofstream& file)
      {
52
53    l_min = MAX;
54    route_min.clear();
55
56    double tao_min, tao_start, Q;
57    vector<size_t> all(n);
58    Q = 350;
59    tao_min = 0.001;
60    tao_start = 0.5;
61
62    vector<vector<size_t>> routes(n);
```

```
63    vector<size_t> lens(n);
64
65    vector<vector<double>> attraction(n);
66    vector<vector<double>> tao(n);
67
68    for (size_t i = 0; i < n; i++) {
69      attraction[i].resize(n);
70      tao[i].resize(n);
71      lens[i] = 0;
72      all[i] = i;
73      for (size_t j = 0; j < n; j++) {
74        if (i != j) {
75          attraction[i][j] = 1.0 / d[i][j];
76          tao[i][j] = tao_start;
77        }
78      }
79    }
80
81    for (size_t time = 0; time < time_max; time++) {
82      for (size_t k = 0; k < n; k++) {
83        get_route(all, k, routes[k], lens[k], d, tao,
              attraction, alpha, beta);
84        if (lens[k] < l_min) {
85          l_min = lens[k];
86          route_min = routes[k];
87        }
88      }
89      for (size_t i = 0; i < n; i++)
90        for (size_t j = 0; j < n; j++) {
91          double sum = 0;
92          for (size_t m = 0; m < n; m++) {
93            if (in_route(i, j, routes[m]))
94              sum += Q / lens[m];
95          }
96
97          tao[i][j] = tao[i][j] * (1 - q) + sum;
98          if (tao[i][j] < tao_min)
99            tao[i][j] = tao_min;
100       }
101   }
```

```
102 }
```

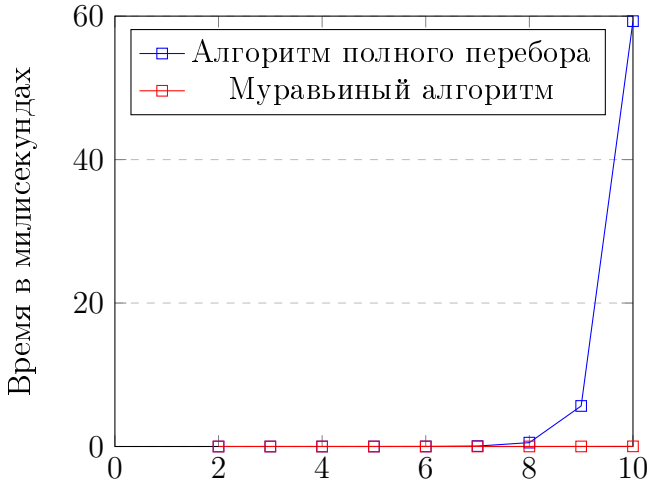
4 Исследовательская часть

4.1 Постановка эксперимента

Был проведен сравнительный анализ реализаций муравьиного алгоритма и полного перебора. Замеры времени проводились для графов с количеством вершин от 2 до 10 с шагом 1. Значения коэффциентов составили $\alpha = 0$, $t_{max} = 5$, $\rho = 0.1$.
С результатами можно ознакомиться в таблице 4.1:

Таблица 4.1: Результаты рамеров времени для алгоритма полного перебора и муравьиного алгоритма

Количество вершин	Полный перебор	Муравьиный алгоритм
2	0.00005	0.0002
3	0.0002	0.0007
4	0.0005	0.0011
5	0.002	0.0017
6	0.01	0.002
7	0.06	0.003
8	0.54	0.005
9	5.65	0.008
10	59.3	0.02



4.2 Параметризация муравьиного алгоритма на основании проведенного эксперимента

Для различных значений параметров α, β, ρ и t_{max} для каждой из нескольких матриц смежности с помощью муравьиного алгоритма и перебора была найдена некоторая длина маршрута. Далее выбраны наилучшие сочетания параметров муравьиного алгоритма на этих данных.
Параметр α менялся от 0 до 10, параметр ρ менялся от 0.1 до 0.9, параметр t_{max} менялся от 5 до 100.
Итого были выявлены оптимальные сочетания параметров (представлены в таблице 4.2):

Таблица 4.2: Результаты решения задачи параметризации

α	β	ρ	t_{max}
4	6	0.6	20
6	4	0.3	40
1	9	0.7	50
6	4	0.9	70
3	7	0.6	80
6	4	0.25	90

4.3 Вывод

Муравьиный алгоритм при количестве вершин больше 5 выигрывает во времени выполнения у алгоритма полного перебора, так как не смотря на то, что на 2-5 вершинах перебор выигрывает, его время выполнения очень быстро растет при увеличении числа вершин.

Также, для заданного класса данных были найдены параметры, которые обеспечивают наиболее оптимальное решение.

Заключение

В ходе работы был изучен и реализован оригинальный муравьиный алгоритм, а также простой перебор. Были получены наиболее оптимальные параметры для быстрого и правильного решения задачи коммивояжера на матрице из десяти "городов".