Лабораторная работа №6
По курсу: "Анализ алгоритмов"
По теме: "Муравьиные алгоритмы"

Студент: Кондрашова О.П.
Группа: ИУ7-55Б
Преподаватели: Волкова Л.Л., Строганов Ю.В.

Москва, 2019

Содержание

Введение

Целью данной работы является изучение муравьиного алгоритма для решения задачи коммивояжера, а также приобретение навыка параметризации метода на примере решения задачи коммивояжера методом, основанным на муравьином алгоритме.

Задачи данной работы:

1. С помощью одного из муравьиных алгоритмов решить задачу коммивояжера;
2. организовать алгоритм полного перебора и провести сравнительные анализ муравьиного алгоритма и алгоритма полного перебора;
3. провести замеры скорости работы работы алгоритма при разных коэффициентах и количестве поколений.

1 Аналитическая часть

В данном разделе приведено описание оригинального муравьиного алгоритма и несколько способов его улучшения

1.1 Оригинальный муравьиный алгоритм

Каждый муравей хранит в памяти список пройденных им узлов. Этот список называют списком запретов или просто памятью муравья. Выбирая узел для следующего шага, муравей «помнит» об уже пройденных узлах и не рассматривает их в качестве возможных для перехода. На каждом шаге список запретов пополняется новым узлом, а перед новой итерацией алгоритма – то есть перед тем, как муравей вновь проходит путь – он опустошается.

Кроме списка запретов, при выборе узла для перехода муравей руководствуется «привлекательностью» ребер, которые он может пройти. Она зависит, во-первых, от расстояния между узлами (то есть от веса ребра), а во-вторых, от следов феромонов, оставленных на ребре прошедшими по нему ранее муравьями. Естественно, что в отличие от весов ребер, которые являются константными, следы феромонов обновляются на каждой итерации алгоритма: как и в природе, со временем следы испаряются, а проходящие муравьи, напротив, усиливают их.

Пусть муравей находится в узле, а узел – это один из узлов, доступных для перехода: Обозначим вес ребра, соединяющего узлы i и j, как $\tau_{i,j}$, а интенсивность феромона на нем – как $\eta_{i,j}$. Тогда вероятность перехода муравья из i в j будет равна:

$$P_{i,j} = \frac{(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})}{\sum(\tau_{i,j}^{\alpha})(\eta_{i,j}^{\beta})} \tag{1}$$

где

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

$\eta_{i,j}$ – 1/d(ij), где d(ij) - длина ребра ij;

α – коэффициент "стадности";

β – коэффициент "жадности".

Заметим, что $\alpha + \beta = const$. При $\beta = 0$ алгоритм всегда будет выбирать путь, на котором отложено больше феромона, а при $\alpha = 0$ алгоритм вырождается в “жадный”, то есть всегда выбирает самое короткое ребро.

Для того, чтобы повысить вариативность выбора (и не выбирать просто максимальную вероятность), можно "подбросить монетку использовать случайное действительное число от 0 до 1. Последовательно суммируя полученные вероятности $P_{i,j}$ пока не получим значение большее или равное случайному числу. Та вероятность, суммирование с которой дало такой результат и будет подходящей, значит муравей выберет город j, для которого была вычислена данная вероятность.

После того, как муравей успешно проходит маршрут, он оставляет на всех пройденных ребрах след, обратно пропорциональный длине пройденного пути:

$$\tau_{i,j} = (1 - \rho)\tau_{i,j} + \Delta\tau_{i,j}, \tag{2}$$

где

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

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

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

Количество феромона на дуге ij ($\tau_{i,j}$) обычно определяется как сумма значений $\Delta\tau^k_{i,j}$.

$$\Delta\tau^k_{i,j} = \begin{cases} Q/L_k & \text{Если k-ый муравей прошел по ребру ij;} \\ 0 & \text{Иначе} \end{cases} \tag{3}$$

где

Q – нормировочная константа, которая соразмерна длине лучшего пути;

L_k – длина пути k-го муравья.

Ниже приведены вариации муравьиного алгоритма.

1. **Элитарная муравьиная система**. Из общего числа муравьёв выделяются так называемые «элитные муравьи». По результатам каждой итерации алгоритма производится усиление лучших маршрутов путём прохода по данным маршрутам элитных муравьёв и, таким образом, увеличение количества феромона на данных маршрутах. В такой системе количество элитных муравьёв является дополнительным параметром, требующим определения. Так, для слишком большого числа элитных муравьёв алгоритм может «застрять» на локальных экстремумах.

2. **Max-Min муравьиная система**. Добавляются граничные условия на количество феромонов (τ_{min}, τ_{max}). Феромоны откладываются только на глобально лучших или лучших в итерации путях. Все рёбра инициализируются значением τ_{max}.

3. **Ранговая муравьиная система(ASrank)**. Все решения ранжируются по степени их пригодности. Количество откладываемых феромонов для каждого решения взвешено так, что более подходящие решения получают больше феромонов, чем менее подходящие.

4. **Длительная ортогональная колония муравьёв (COAC)**. Механизм отложения феромонов COAC позволяет муравьям искать решения совместно и эффективно. Используя ортогональный метод, муравьи в выполнимой области могут исследовать их выбранные области быстро и эффективно, с расширенной способностью глобального поиска и точностью.

1.2 Вывод

Был рассмотрен поверхностно стандартный муравьиный алгоритм и его оптимизации.

2 Конструкторская часть

В данной части будет разобран принцип работы алгоритма.

2.1 Разработка реализаций алгоритмов

Основное действие происходит в функции ant_colony, приведенной в листинге 1. На вход ей поступают следующие параметры:

1. cities_number - количество городов;
2. route - вектор, в который будет записан наилучший маршрут;
3. distance - матрица расстояний между городами;
4. alpha - коэффициент стадности;
5. beta -коэффициент жадности;
6. evaporation - процент распыления феромона;
7. Q - величина феромона обычного муравья;
8. t - время жизни колонии.

Псевдокод работы алгоритма можно представить следующим образом:

1. Инициализация параметров cities_number, alpha, beta, evaporation, Q, elit, elit;
2. генерация матрицы расстояний;
3. инициализация матрицы феромонов, присвоение начальной концентрации феромона на ребрах;
4. инициализация пустого маршрута и длины начального кратчайшего маршрута length = 0;
5. цикл по времени жизни колонии t;
6. инициализация матрицы феромонов для данного поколения муравьев;
7. цикл по муравьям;
8. построить маршрут, используя формула вероятности посещения города (1) и определить его длину;
9. обновить феромон на маршруте, используя формулу (3);
10. если полученная длина меньше текущей, примем ее за минимальную;
11. конец цикла по муравьям;
12. вычислить испарение феромона на матрице феромонов;
13. прибавить к общей матрице феромонов матрицу феромонов текущего поколения муравьев;
14. конец цикла по времени жизни колонии;
15. вернуть длину кратчайшего маршрута.

2.2 Схема алгоритма

На рис. 1 представлена схема муравьиного алгоритма.

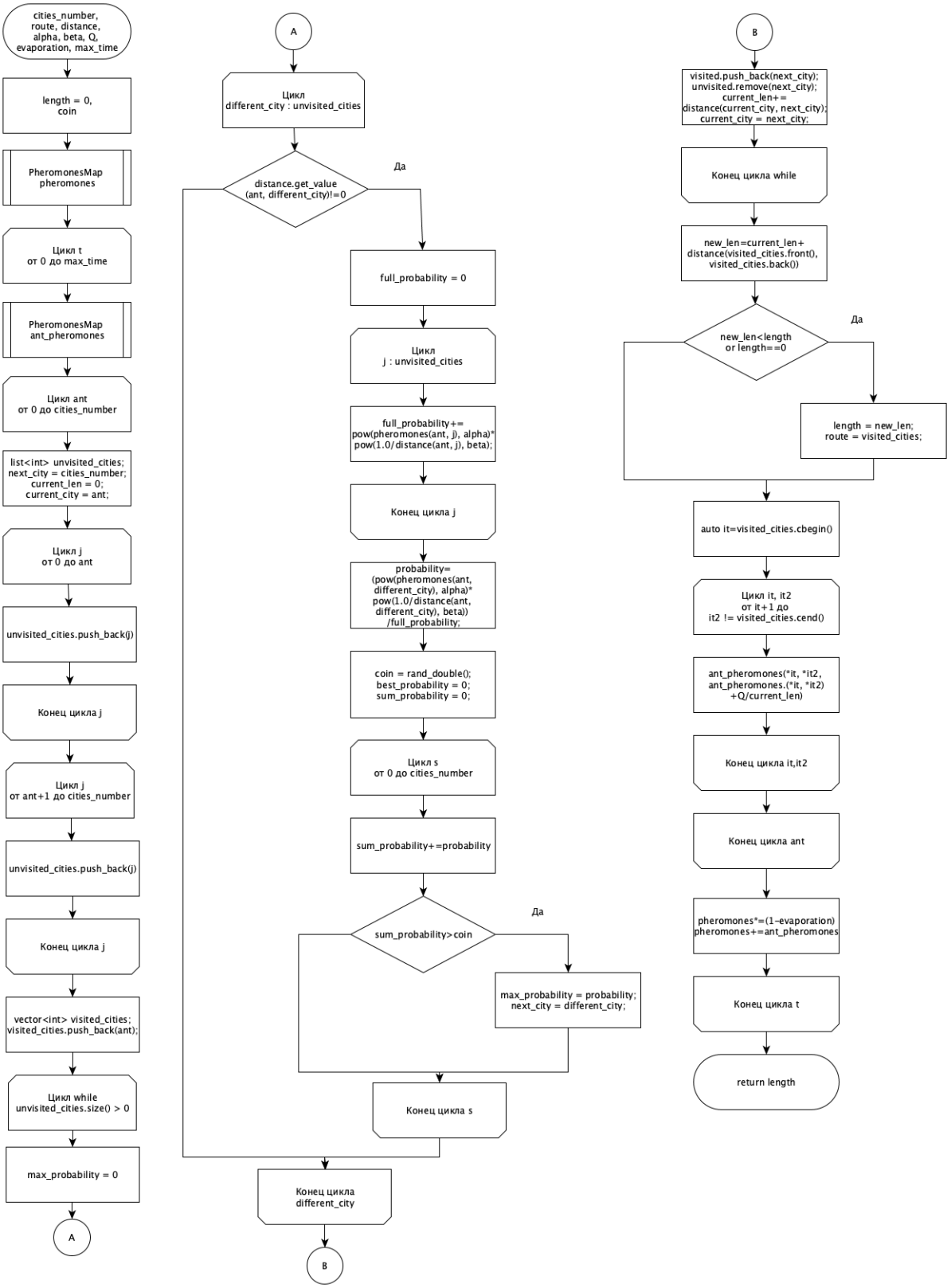

Рис. 1: Схема муравьиного алгоритма

2.3 Вывод

В данном разделе был кратко описан реализованный муравьиный алгоритм и представлена схема данного алгоритма.

3 Технологическая часть

В этом разделе будут изложены требования к программному обеспечению и листинги алгоритмов.

3.1 Средства реализации

Данная программа разработана на языке C++, поддерживаемом многими операционными системами. Проект выполнен в среде Xcode.

3.2 Листинги функций

В данном разделе представлены листинги реализованных алгоритмов.

В листинге 1 представлен муравьиный алгоритм.

В алгоритме используюся реализованные классы CitiesMap и PheromonesMap для работы с матрицей расстояний и матрицей феромона на ребрах.

Листинг 1: Муравьиный алгоритм

```
1  int ant_colony(const int cities_number, vector<int>& route,
       const CitiesMap& distance, const double alpha, const
       double beta, const double Q, const double evaporation,
       const int max_time)
2  {
3      int length = 0;
4      route.clear();
5
6      PheromonesMap pheromones(cities_number, 10);
7
8      double coin;
9
10     for (int time = 0; time < max_time; time++)
11     {
12         PheromonesMap ant_pheromones(cities_number, 10);
13
14         for (int ant = 0; ant < cities_number; ant++)
15         {
16             list<int> unvisited_cities;
17             int next_city = cities_number;
18             int current_len = 0;
19
20             int current_city = ant;
21
22             for (int j = 0; j < ant; j++)
23             {
24                 unvisited_cities.push_back(j);
25             }
26             for (int j = ant + 1; j < cities_number; j++)
27             {
28                 unvisited_cities.push_back(j);
29             }
30
```

```
31              vector<int> visited_cities;
32              visited_cities.push_back(ant);
33
34              while (unvisited_cities.size() > 0)
35              {
36                  double max_probability = 0;
37
38                  for (int different_city : unvisited_cities)
39                  {
40                      if (distance.get_value(ant, different_city
                            ) != 0)
41                      {
42                          double full_probability = 0;
43                          for (int j : unvisited_cities)
44                          {
45                              full_probability += pow(pheromones
                                    .get_value(ant, j), alpha) *\
46                                      pow(1.0 / distance.
                                            get_value(ant, j),
                                            beta);
47
48                          }
49
50                          double probability = (pow(pheromones.
                                get_value(ant, different_city),
                                alpha) *\
51                                  pow(1.0 / distance.get_value(
                                        ant, different_city), beta
                                        )) / full_probability;
52
53                          coin = rand_double();
54                          int best_probability = 0;
55                          double sum_probability = 0;
56
57                          for (int s = 0; s < cities_number; s
                                ++)
58                          {
59                              sum_probability += probability;
60                              if (sum_probability > coin)
61                              {
62                                  max_probability = probability;
63                                  next_city = different_city;
64                                  break;
65                              }
66                          }
67                      }
68                  }
69
70                  visited_cities.push_back(next_city);
71                  unvisited_cities.remove(next_city);
72                  current_len += distance.get_value(current_city
                        , next_city);
73
74                  current_city = next_city;
```

```
75              }
76
77              int new_len = current_len + distance.get_value(
                    visited_cities.front(), visited_cities.back())
                    ;
78
79              if (new_len < length || length == 0)
80              {
81                  length = new_len;
82                  route = visited_cities;
83              }
84
85              auto it = visited_cities.cbegin();
86              for (auto it2 = it + 1; it2 != visited_cities.cend
                    (); ++it, ++it2)
87              {
88                  ant_pheromones.set_value(*it, *it2,
                        ant_pheromones.get_value(*it, *it2) + Q /
                        current_len);
89              }
90          }
91
92          pheromones *= (1 - evaporation);
93          pheromones += ant_pheromones;
94      }
95
96      return length;
97  }
```

В листинге 2 представлен алгоритм полного перебора.

Во вложенной функции hamilton 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: Алгоритм полного перебора

```
1  int full_search(const CitiesMap& distance)
2  {
3      int n = distance.get_cities_number();
4      vector<bool> visited(n, 0);
5      vector<int> current_route;
6      vector<int> min_route;
7      int current_len = 0;
8      int min_route_len = INT_MAX;
9      for (int i = 0; i < n; i++)
10     {
11         current_route.clear();
12         current_route.push_back(i);
13         fill(visited.begin(), visited.end(), 0);
14         visited[i] = 1;
15         current_len = 0;
16         hamilton(distance, min_route, min_route_len,
               current_route, visited, current_len);
17     }
```

```
18      return min_route_len;
19  }
20
21  void hamilton(const CitiesMap& distance, vector<int> &
        min_route, int &min_distance, vector<int> &current_route,
        vector<bool> &visited, int &current_len)
22  {
23      if (current_route.size() == distance.get_cities_number())
24      {
25          s++;
26          int tmp = distance.get_value(current_route.back(),
                current_route[0]);
27
28          if (current_len + tmp < min_distance)
29          {
30              min_route = current_route;
31              min_distance = current_len + tmp;
32          }
33
34          return;
35      }
36      for (int i = 0; i < distance.get_cities_number(); i++)
37      {
38          if (!visited[i])
39          {
40              int tmp = distance.get_value(current_route.back(),
                    i);
41              if (current_len + tmp > min_distance)
42                  continue;
43              current_len += tmp;
44              current_route.push_back(i);
45              visited[i] = 1;
46              hamilton(distance, min_route, min_distance,
                    current_route, visited, current_len);
47              visited[i] = 0;
48              current_route.pop_back();
49              current_len -= tmp;
50          }
51      }
52  }
```

3.3 Вывод

В данном разделе были представлены листинги функций муравьиного алгоритма и алгоритма полного перебора.

4 Исследовательская часть

В данном разделе будет проведена параметризация муравьиного алгоритма, а также будет произведен сравнительный анализ муравьиного алгоритма и алгоритма полного перебора.

4.1 Постановка эксперимента

Для проведения экспериментов была использована рандомно сгенерированная матрица расстояний 10×10. Элементы матрицы - целые числа в диапазоне от 1 до 10. В каждом эксперименте фиксировались значения α, β, ρ и t_max. В течение экспериментов значения α, β, ρ менялись от 0 до 1 включительно с шагом 0.1, t_{max} от 10 до 2000 с шагом 10.

4.2 Сравнительный анализ на материалах эксперимента

В таблице 1 представлены наилучшие результаты, полученные в ходе проведенной параметризации.

Результатом проведения эксперимента (4 столбец) считалась разница между длиной маршрута, рассчитанного алгоритмом полного перебора, и длиной маршрута, рассчитаной с помощью муравьиного алгоритма с текущими параметрами.

Таблица 1: Таблица результатов параметризации

ρ	α	t_{max}	difference
0.00	0.10	10	0.01556
0.00	0.50	10	0.01405
0.00	0.60	10	0.01263
0.00	0.60	20	0.01556
0.00	0.70	10	0.01263
0.00	0.70	20	0.0156
0.00	0.70	30	0.01556
0.00	0.70	40	0.01405
0.00	0.70	50	0.01556
0.00	0.80	10	0.01263
0.00	0.90	200	0.01263
0.10	0.40	60	0.01556
0.10	0.80	10	0.01263
0.20	0.60	30	0.01263
0.20	0.70	40	0.01263
0.20	0.70	50	0.01556
0.20	0.80	10	0.01263
0.40	0.80	170	0.01263
0.40	0.80	180	0.01405
0.40	0.80	200	0.01263
0.60	0.90	200	0.01263
0.70	0.80	10	0.01263
0.70	0.90	200	0.01263
0.80	0.50	10	0.01263
0.80	0.60	10	0.01263
0.80	0.60	20	0.01556
0.80	0.70	200	0.01556
0.80	0.80	10	0.01263
0.80	0.80	180	0.01263
0.80	0.80	200	0.01405
0.80	0.90	10	0.01263
0.80	0.90	200	0.01263
0.90	0.40	20	0.01556
0.90	0.40	120	0.01556
0.90	0.50	10	0.01263
0.90	0.60	10	0.01556
0.90	0.60	200	0.01556
0.90	0.70	20	0.01263
0.90	0.70	30	0.01556
0.90	0.70	200	0.01556
0.90	0.80	10	0.01263
0.90	0.90	200	0.01263

Как видно из таблицы 1, наиболее точное значение чаще всего получается при возрастания параметра α, но при этом ρ и t_{max} могут принимать любые значения в указанном диапазоне.

4.3 Анализ по времени

Проведем сравнительный анализ по времени алгоритма полного перебора и муравьиного алгоритма на матрицах различной размерности от 2 до 10.

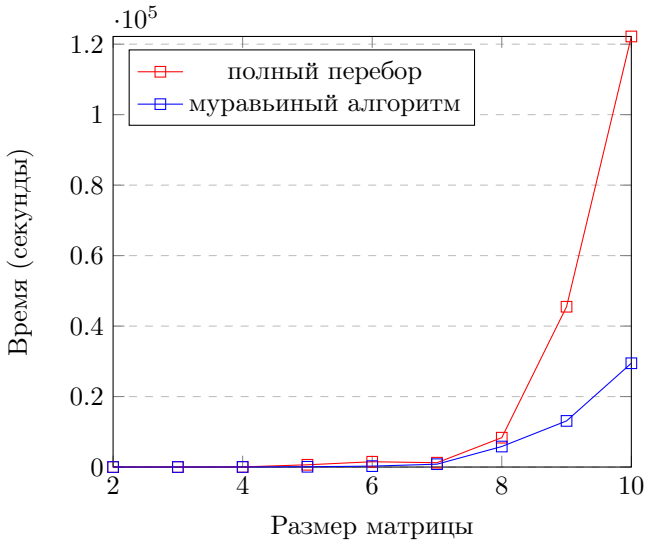

Рис. 2: График времени работы алгоритмов

Как видно из графика на рис. 2, на матрицах маленькой размерности алгоритмы работают примерно одинаково, но при возрастании размерности матрицы, время выполнения полного перебора начинает расти быстрее времени выполнения муравьиного алгоритма.

4.4 Вывод

Муравьиный алгоритм выигрывает по времени у алгоритма полного перебора при возрастании размерностей матрицы.

Заключение

В ходе работы был изучаен и реализован муравьиный алгоритм с введением элитных муравьей. Были получены наиболее оптимальные параметры для быстрого и правильного решения задачи коммивояжера на матрице из десяти городов.

Список литературы

[1] Штовба С. Д. Муравьиные алгоритмы, Exponenta Pro. Математика в приложениях. 2004

[2] Формулы выбора города, обновления феромонов https://habr.com/post/105302/