Государственное образовательное учреждение высшего профессионального образования
“Московский государственный технический университет имени Н.Э.Баумана”

Дисциплина: Анализ алгоритмов

Лабораторная работа 6

Решение задачи коммивояжера с помощью муравьиного алгоритма

Студент группы ИУ7-55,
Шестовских Н.А.
Преподаватели: Волкова Л.Л., Строганов Ю.В.

2019 г.

Содержание

Введение

Муравьиный алгоритм — один из эффективных полиномиальных алгоритмов для нахождения приближённых решений задачи коммивояжёра, а также решения аналогичных задач поиска маршрутов на графах.

Целью данной лабораторной работы является изучение муравьиных алгоритмов и приобретение навыков параметризации методов на примере муравьиного алгоритма, примененного к задаче коммивояжера.

Задачи данной лабораторной работы:

- рассмотрентть муравьиный алгоритм и алгоритм полного перебора в задаче коммивояжера;
- провести сравнение потребляемого процессорного времени этих алгоритмов на одном из языков программирования;
- провести параметризацию муравьиного алгоритма.

1 Аналитическая часть

В данной части будут рассмотрены теоретические основы задачи коммивояжера и муравьиного алгоритма.

1.1 Постановка задачи

Имеется сильно связный взвешенный ориентированный граф [?] с положительными весами, заданный в виде матрицы смежностей. Количество вершин в нем лежит в диапазоне от 5 до 20.Требуется решить задачу коммивояжера для этого графа.

1.2 Задача коммивояжера

Коммивояжёр (фр. commis voyageur) — бродячий торговец. Задача коммивояжёра — важная задача транспортной логистики, отрасли, занимающейся планированием транспортных перевозок. Коммивояжёру, чтобы распродать нужные и не очень нужные в хозяйстве товары, следует объехать n пунктов и в конце концов вернуться в исходный пункт. Требуется определить наиболее выгодный маршрут объезда. В качестве меры выгодности маршрута (точнее говоря, невыгодности) может служить суммарное время в пути, суммарная стоимость дороги, или, в простейшем случае, длина маршрута [?].

1.3 Решение полным перебором

Совершенно очевидно, что задача может быть решена перебором всех вариантов объезда и выбором оптимального. Беда в том, что количество возможных маршрутов очень быстро возрастает с ростом n (оно равно n! — количеству способов упорядочения пунктов). К примеру, для 100 пунктов количество вариантов будет представляться 158-значным числом — не выдержит ни один калькулятор! Мощная ЭВМ, способная перебирать миллион вариантов в секунду, будет биться с задачей на протяжении примерно 3·10144 лет. Увеличение производительности ЭВМ в 1000 раз даст хоть и меньшее в 1000 раз, но по-прежнему чудовищное время перебора вариантов. Не спасает ситуацию даже то, что для каждого варианта маршрута имеется 2n равноценных, отличающихся выбором начального пункта (n вариантов) и направлением обхода (2 варианта). Перебор с учётом этого наблюдения сокращается незначительно — до $\frac{n!}{2n} = \frac{(n-1)!}{2}$ вариантов [?].

1.4 Муравьиные алгоритмы

Все муравьиные алгоритмы базируются на моделировании поведения колонии муравьев. Колония муравьев может рассматриваться как многоагентная система, в которой каждый агент (муравей) функционирует автономнопо очень простым правилам. В противовес почтипримитивному поведению агентов, поведениевсей системы получается на удивление разумным.

Муравьиные алгоритмы представляют собой вероятностную жадную эвристику, где вероятности устанавливаются, исходя из информации о качестве решения, полученной из предыдущих решений.

Идея муравьиного алгоритма - моделирование поведения муравьёв, связанного с их способностью быстро находить кратчайший путь от муравейника к источнику пищи и адаптироваться к изменяющимся условиям, находя новый кратчайший путь[?]. При своём движении муравей метит путь феромоном, и эта информация используется другими муравьями для выбора пути. Это элементарное правило поведения и определяет способность муравьёв находить новый путь, если старый оказывается недоступным.

Какие же механизмы обеспечивают стольсложное поведение муравьев, и что можем мыпозаимствовать у этих крошечных существ длярешения своих глобальных задач? Основу «социального» поведения муравьев составляет самоорганизация — множество динамических механизмов, обеспечивающих достижение системой глобальной цели в результате низкоуровневого взаимодействия ее элементов. Принципиальнойособенностью такого взаимодействия являетсяиспользование элементами системы только локальной информации. При этом исключается любое централизованное управление и обращениек глобальному образу, репрезентирующему систему во внешнем мире. Самоорганизация является результатом взаимодействия следующих четырех компонентов [?] :

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

Рассмотрим случай, 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 обогащён феромоном. Поскольку движение муравьёв определяется концентрацией феромона, то следующие будут предпочитать именно этот путь, продолжая обогащать его феромоном до тех пор, пока этот путь по какой-либо причине не станет недоступен.

Очевидная положительная обратная связь быстро приведёт к тому, что кратчайший путь станет единственным маршрутом движения большинства муравьёв. Моделирование испарения феромона - отрицательной обратной 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.

Обобщим все выше сказанное. Любой муравьиный алгоритм, независимо от модификаций, представим в следующем виде:

- Создание муравьев;
- Поиск решения;
- Обновление феромона;
- Дополнительные действия (опционально).

Теперь рассмотрим каждый шаг в цикле более подробно:

1. Создание муравьев

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. Поиск решения

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

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

где $\tau_{i,j}$— расстояние от города i до j;
$\eta_{i,j}$—количество феромонов на ребре ij;
α— параметр влияния длины пути;
β— параметр влияния феромона.

3. Обновление феромона

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. Итого, новый след феромона вычисляется по формуле **??**:

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

где $\rho_{i,j}$ - доля феромона, который испарится;
$\tau_{i,j}$ - количество феромона на дуге ij;
$\Delta\tau_{i,j}$ - количество отложенного феромона, вычисляется по формуле **??**.

4. Дополнительные действия

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

1.5 Муравьиный алгоритм в задаче коммивояжера

Рассмотрим, как реализовать четыре составляющие самоорганизации муравьев при оптимизации маршрута коммивояжера. Многократностьвзаимодействия реализуется итерационным поиском маршрута коммивояжера одновременнонесколькими муравьями. При этом каждый муравей рассматривается как отдельный, независимыйкоммивояжер, решающий свою задачу. За однуитерацию алгоритма каждый муравей совершаетполный маршрут коммивояжера.Положительная обратная связь реализуетсякак имитация поведения

муравьев типа «оставление следов–перемещение по следам». Чем больше следов оставлено на тропе — ребре графа в задаче коммивояжера, тем больше муравьев будетпередвигаться по ней. При этом на тропе появляются новые следы, привлекающие дополнительных муравьев. Для задачи коммивояжера положительная обратная связь реализуется следующимстохастическим правилом: вероятность включения ребра графа в маршрут муравья пропорциональна количеству феромона на нем.

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. Тогда откладываемое количество феромона может быть задано в виде:

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

где Q - количество феромона, переносимого муравьем;

Тогда

$$\Delta\tau_{i,j} = \tau_{i,j}^0 + \tau_{i,j}^1 + ... + \tau_{i,j}^k \tag{4}$$

где k - количество муравьев в вершине графа с индексами i и j.

1.6 Вывод

В данном разделе были рассмотрены общие принципы муравьиного алгоритма и применение его к задаче коммивояжера.

2 Конструкторская часть

В данном разделе будет рассмотрены схемы алгоритмов.

2.1 Схема алгоритма полным перебором

На рисунке ?? приведена схема алгоритма полного перебора в задаче коммивояжера.

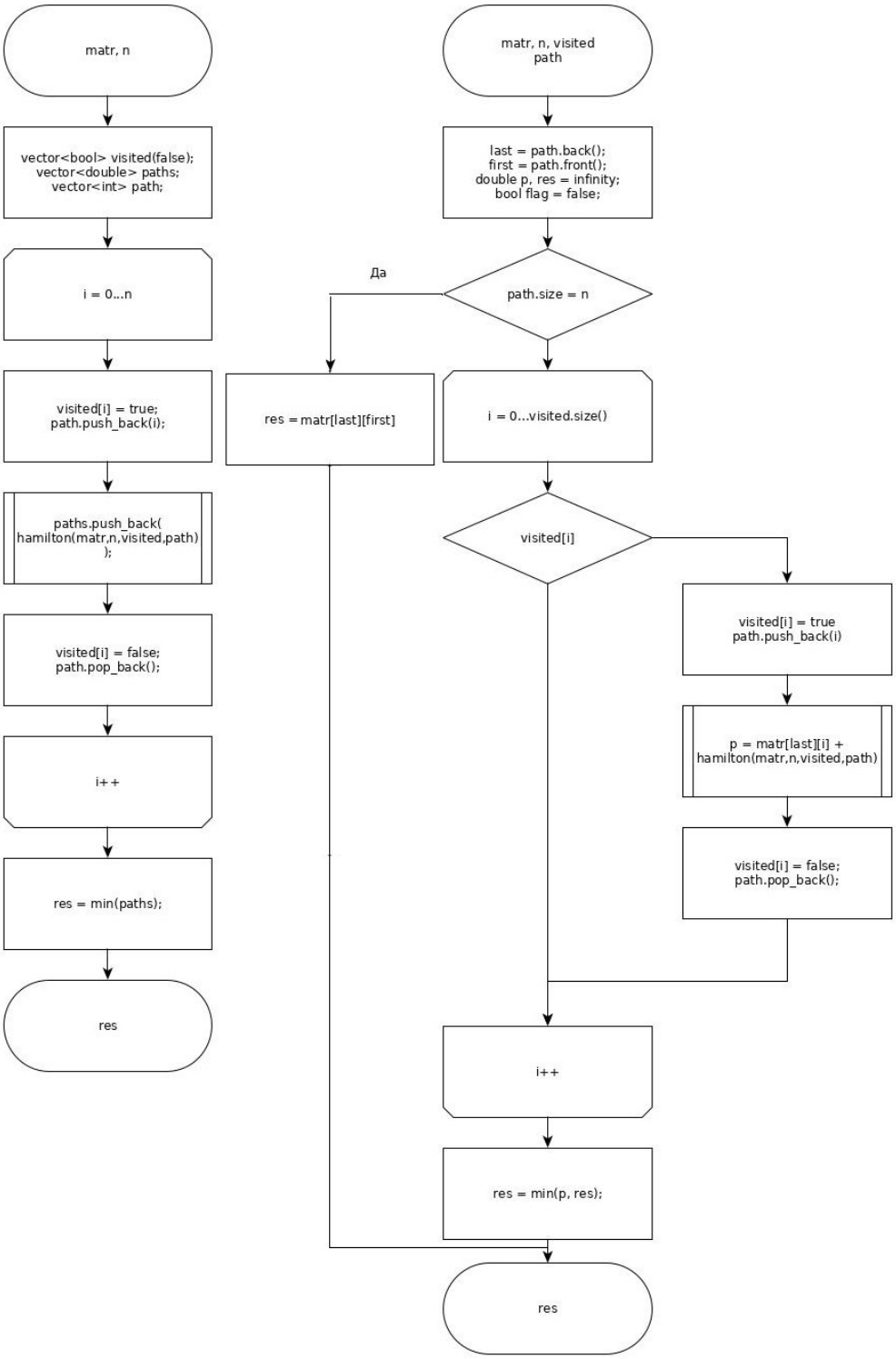

Рис. 1: Полный перебор в задаче коммивояжера

2.2 Схема муравьиного алгоритма

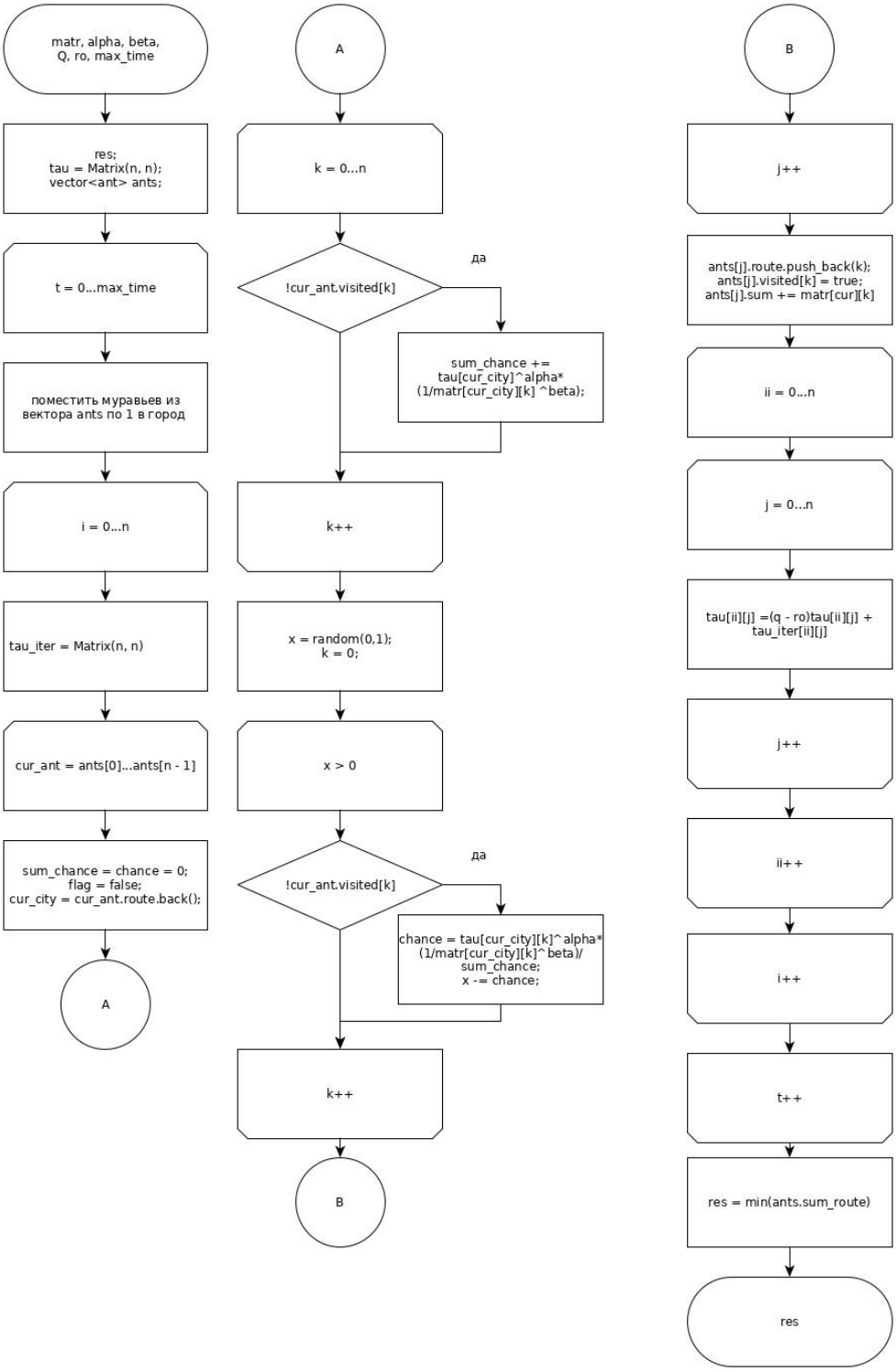

Рис. 2: Муравьиный агоритм в задаче коммивояжера

2.3 Вывод

В данном разделе были рассмотрены схемы алгоритмов.

3 Технологическая часть

В данной части будут рассмотрены средства для реализации муравьиного алгоритма в задаче коммивояжера.

3.1 Требования к программному обеспечению

Входные данные - матрица смежноти графа.
Выходные данные - длина самого выгодного пути.
На рисунке ?? дана функциональная схема решения задачи коммивояжера в нотации IDEF0.

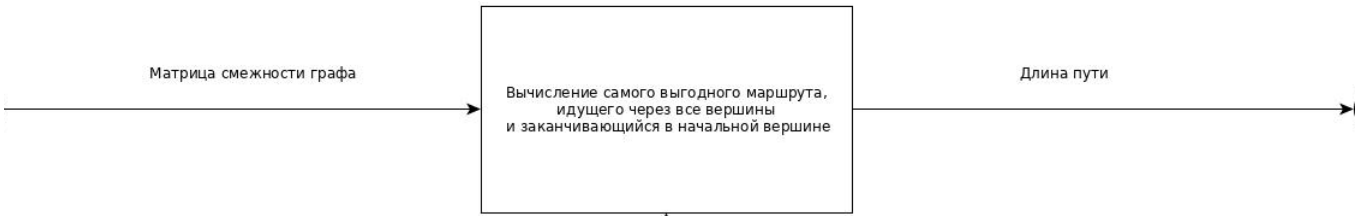

Рис. 3: Функциональная схема решения задачи коммивояжера

3.2 Средства реализации

В качестве языка программирования был выбран c++ [?] из-за большого числа библиотек, в качестве фреймворка для разработки был выбран QT [?] по тем же причинам, среда разработки - QTCreator [?]. Для тестов по процессорному времени использовалась функция clock().

3.3 Листинг кода

На листинге ?? представлены две функции, в совокупности реализующих метод полного перебора.

Листинг 1: Метод полного перебора в задаче коммивояжера

```
double exhaustive_search(Matrix &m)
{
    int n = m.GetN();
    std::vector<bool> visited;
    std::vector<double> paths;
    std::list<int> path;
    for(int i = 0; i < n; i++)
        visited.push_back(false);
    for(int i = 0; i < n; i++)
    {
        visited[i] = true;
        path.push_back(i);
        paths.push_back(search_for_node(m, visited, path));
        path.pop_back();
        visited[i] = false;
    }
    double res = paths[0];
    for(int i = 0; i < n; i++)
        res = std::min(res, paths[i]);
    return res;
}

double search_for_node(Matrix &m, std::vector<bool> visited, std::list<int> path)
{
    int &cur_node = path.back();
    double p;
    double res;
    bool flag = false;
    if (path.size() ==(unsigned int) m.GetN())
    {
```

```
        int first = path.front();
        int last = path.back();
        return m[last][first];
    }
    for(unsigned int i = 0; i < visited.size(); i++)
    {
        if (!visited[i])
        {
            p = m[cur_node][i];
            visited[i] = true;
            path.push_back(i);
            p += search_for_node(m, visited, path);
            if (!flag)
            {
                res = p;
                flag = true;
            }
            else
                res = std::min(res, p);
            visited[i] = false;
            path.pop_back();
        }
    }
    return res;
}
```

На листинге ?? представлена структура для хранения информации о муравье.

Листинг 2: Структура для хранения информации о муравье

```
struct ant
{
    std::vector<int> route;
    std::vector<bool> visited;
    double sum_route;
};
```

На листинге ?? представлен код муравьиного алгоритма.

Листинг 3: Муравьиный алгоритм в задаче коммивояжера

```
double ant_alg(Matrix &m, double alpha, double beta, double Q, double ro, int max_time)
{
    double res;
    unsigned int n = m.GetN();
    Matrix pheromones = Matrix(n, n, 0.1);
    std::vector<ant> ants;
    for(int t = 0; t < max_time; t++)
    {
        ants.clear();
        for(unsigned int i = 0; i < n; i++)
        {
            ant cur_ant;
            cur_ant.route.push_back(i);
            cur_ant.sum_route = 0;
            for(unsigned int j = 0; j < n; j++)
                cur_ant.visited.push_back((j == i) ? true : false);
            ants.push_back(cur_ant);
        }
        for (int i = 0; i < n; i++)
        {
            Matrix pheromones_iter(n, n, 0);
            for(unsigned int j = 0; j < ants.size(); j++)
            {
                double sum_chance = 0, chance = 0;
                bool flag = false;
                ant cur_ant = ants[j];
                int cur_city = cur_ant.route.back();
                for(unsigned int k = 0; k < n; k++)
                {
                    if (cur_ant.visited[k] == false)
```

```
31                      {
32                          sum_chance += pow(pheromones[cur_city][k], alpha) * pow(1/(m[
                                cur_city][k]), beta);
33                          flag = true;
34                      }
35                  }
36                  if (flag)
37                  {
38                      srand(time(NULL));
39                      float x = (rand()%1000)/((double)1000);
40                      unsigned int k = 0;
41                      for(; (x > 0) && (k < n); k++)
42                      {
43                          if (cur_ant.visited[k] == false)
44                          {
45                              chance = pow(pheromones[cur_city][k], alpha) * pow(1/(m[
                                    cur_city][k]), beta);
46                              chance /= sum_chance;
47                              x -= chance;
48                          }
49                      }
50                      k--;
51                      ants[j].route.push_back(k);
52                      ants[j].visited[k] = true;
53                      ants[j].sum_route += m[cur_city][k];
54                      pheromones_iter[cur_city][k] += Q/(m[cur_city][k]);
55                  }
56              }
57              for(unsigned int ii = 0; ii < n; ii++)
58                  for(unsigned int j = 0; j < n; j++)
59                      pheromones[ii][j] = (1 - ro) * pheromones[ii][j] + pheromones_iter[ii
                            ][j];
60          }
61
62          res = ants[0].sum_route;
63          for(auto iter = ants.cbegin(); iter != ants.cend(); ++iter)
64          {
65              res = std::min(res, iter->sum_route);
66          }
67      }
68      return res;
69  }
```

3.4 Вывод

В данном разделе была представлена реализация муравьиного алгоритма и алгоритма полного перебора для поставленной в разделе ??.

4 Экспериментальная часть

Требуется сравнить эффективность муравьиного алгоритма и полного перебора в задаче коммивояжера по процессорному времени. Помимо этого, требуется провести параметризацию муравьиного алгоритма для нахождения лучших сочетаний параматров для графов, имеющих кличество вершин от 3 до 9 и веса от 10 до 100.

4.1 Постановка эксперимента

Требуется сравнить эффективность муравьиного алгоритма и полного перебора в задаче коммивояжера по процессорному времени. Помимо этого, требуется провести параметризацию муравьиного алгоритма для нахождения лучших сочетаний параматров. Графы в данном эксперименте имели кличество вершин от 3 до 9 с шагом 1 и веса - произвольные числа от 10 до 100 с шагом 1. Тестовый компьютер:

- ЦПУ -INTEL® PENTIUM® N5000 (4 физических ядра, 4 логических ядра, базовая тактовая частота - 1.1 ГГц, максимальная - 2.7 ГГц, в большинстве случаев тактовая частота $\approx$ 2.3 ГГц) [?];
- ОЗУ - 8 ГБ 2400 МГц;
- ОС - Ubuntu Mate 18.04.

4.2 Тест на процессорное время

Для данного теста константы в муравьином алгоритме были зафиксированы следующим образом: $\alpha = 2$, $t_{max} = 5$, $\rho = 0.2$, $\beta = 2$. Результаты теста представлены на графике **??**

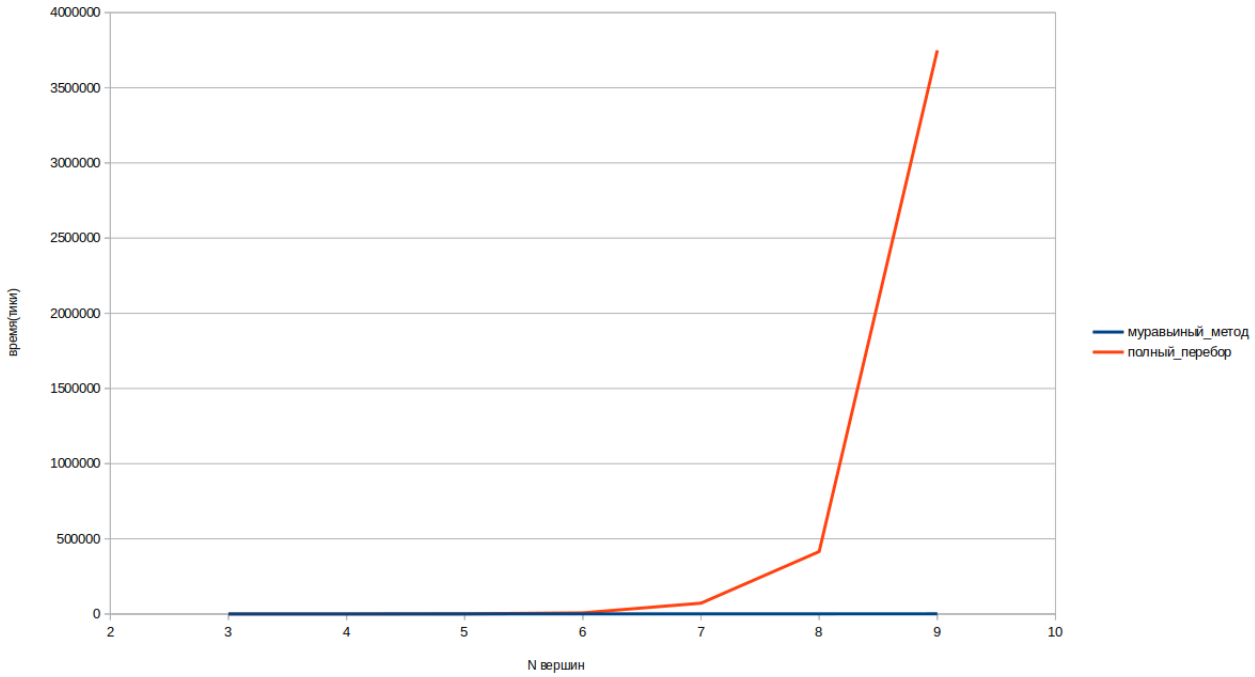

Рис. 4: Временной анализ

4.3 Параметризация

При графах, используемых в данном тесте, муравьиный алгоритм достаточно часто выдавал реальный результат(то есть результат совпадал с результатом при полном переборе). Это случалось примерно в 15 % случаев. Поэтому поиск лучшей комбинации не имел смысла, так как лучших комбинаций в таком случае получалось достаточно много. Был проведен тест, выявляющий, при каком фиксированном значении,например, α, и плавающих значениях других параметров количество лучших решений будет наивысшим. Такой тест был проведен по каждому из рассматриваемых параметров. В таблице 1 приведены найденные для каждого графа параметры, при которых метод решения дает наилучший результат.

Таблица 1: Результаты решения задачи параметризации

N	α	β	ρ	t_{max}
3	1	1	0.1	1
4	1	5	0.1	7
5	1	10	0.5	1
6	1	1	0.1	12
7	1	10	0.9	17
8	1	2	0.1	3
9	6	5	0.4	11

4.4 Вывод

По результатам тестирования для графов, описанных в разделе **??** по времени было выявлено, что при графах с количеством вершин до 4 включительно полный перебор работает быстрее, однако дальше муравьиный алгоритм становится на порядки более эффективным(до 10000 при N = 9)

По результатам параметризации были найдены лучшие коэффициенты при заданных графах.

Заключение

В ходе данной лабораторной работы был изучен муравьиный алгоритм и были приобретены навыки параметризации методов.

Были рассмотрены муравьиный алгоритм и алгоритм полного перебора в задаче коммивояжера. Было проведено сравнение потребляемого процессорного времени этих алгоритмов на языке c++ и проведена параметризация муравьиного алгоритма. По результатам тестирования для графов, описанных в разделе **??** по времени было выявлено, что при графах с количеством вершин до 4 включительно полный перебор работает быстрее, однако дальше муравьиный алгоритм становится на порядки более эффективным(до 10000 при N = 9). По результатам параметризации были найдены лучшие коэффициенты при заданных графах. Таким образом, муравьиный алгоритм является более предпочтительным для графов с количеством вершин 4 и более, так как он выдает приближенный ответ за гораздо меньшее время, чем полный перебор, который, в свою очередь, имеет сложность $O(n!)$, что делает его на практике неприменимым для большого количества вершин графа (от 20 и более).

Список литературы

[1] Белоусов А.И., Ткачев С.Б(2006). Дискретная математика, 4-е издание.

[2] Т.М. Товстик, Е.В. Жукова - Алгоритм приближенного решения задачи коммивояжера.

[3] Задача коммивояжера[Электронный ресурс] - режим доступа http://mech.math.msu.su/ shvetz/54/inf/perl-problems/chCommisVoyageur.xhtml

[4] Муравьиные алгоритмы[Электронный ресурс] - режим доступа http://www.machinelearning.ru/wiki/index.php?title=

[5] Штовба С.Д. - Муравьиные алгоритмы.

[6] И. В. Белоусов(2006), Матрицы и определители, учебное пособие по линейной алгебре, с. 1 - 16

[7] Документация по языку C++ [Электронный ресурс] - режим доступа https://devdocs.io/cpp/

[8] Документация по среде QTCreator [Электронный ресурс] - режим доступа http://doc.crossplatform.ru/qtcreator/1.2.1/

[9] Документация по фреймворку QT [Электронный ресурс] - https://doc.qt.io/

[10] Спецификации процессора INTEL PENTIUM 5000 [Электронный ресурс] - режим доступа https://www.intel.ru/content/www/ru/ru/products/processors/pentium/n5000.html