Отчет о разработке программы, имитирующей случайные блуждания в графе

Пелагеев Д.И.

28 июня 2024 г.

Содержание

Введение

Цель данной работы – это изучить применения теории графов для моделирования и анализа электрических цепей, представленных в виде графов, а также определение среднего времени перемещения между двумя вершинами графа с использованием метода случайного блуждания. Мы рассматриваем однородное случайное блуждание, при котором каждая вершина выбирает одну из своих соседних вершин с равной вероятностью на каждом шаге.

Электрические сети могут быть эффективно представлены с помощью графов, где вершины соответствуют узлам сети, а ребра — проводникам или линиям связи между этими узлами. Одним из ключевых аспектов анализа таких сетей является расчет сопротивления между двумя узлами, что имеет важное значение для оценки характеристик сети и её надежности. Этот расчет можно провести с использованием законов Кирхгофа, но альтернативный подход предполагает использование методов теории графов и случайного блуждания.

В данной работе мы реализуем метод моделирования случайного блуждания по графу для вычисления среднего времени посещения указанной вершины и среднего времени обхода всего графа. Этот метод основывается на предположении, что время перемещения по каждому ребру равно единице, а выбор следующей вершины происходит случайно с равной вероятностью для всех соседей текущей вершины. Интересным является тот факт, что среднее время перемещения тесно связано с электрическим сопротивлением между узлами графа, если сопротивление каждого ребра также принять равным единице.

В данном отчете была разработана программа на Octave, которая имитирует случайные блуждания.

Теоретические основы

2.1 Основные понятия теории графов

Теория графов, изучает графы - абстрактные структуры, состоящие из вершин (узлов) и рёбер (связей между узлами). Основные понятия теории графов включают следующие элементы:

Граф $G = (V, E)$ состоит из множества вершин V и множества рёбер E, где каждое ребро представляет собой пару вершин. Граф может быть ориентированным (если рёбра имеют направление) и неориентированным (если рёбра не имеют направления).

Матрица смежности A графа G размером $n \times n$ (где $|V|$ обозначает количество вершин графа и равна n задается как квадратная матрица, в которой элемент a_{ij} равен 1, если существует ребро между вершинами v_i и v_j, и 0 в противном случае. И где Для ориентированного графа матрица смежности не обязательно симметрична[8].

Степень вершины v в графе G (обозначаемая как $\deg(v)$) - это количество рёбер, связанных с данной вершиной. В ориентированном графе различают входящую степень ($\deg_{in}(v)$) и исходящую степень ($\deg_{out}(v)$)[3].

Граф G называется связным, если существует путь между любой парой вершин. Для ориентированных графов используется понятие сильной связности, когда для любой пары вершин u и v существует ориентированный путь как от u к v, так и от v к u[2].

2.2 Связь между случайными блужданиями и электрическими сопротивлениями

Случайное блуждание на графе G представляет собой процесс, при котором на каждом шаге изменяется состояние случайного процесса, заключающееся в перемещении из текущей вершины в одну из соседних вершин, выбранную независимым образом с равной вероятностью. Этот процесс можно моделировать с помощью матрицы переходов P, где элемент p_{ij} представляет собой вероятность перехода из вершины v_i в вершину v_j.

Электрические сети могут быть представлены в виде графов, где вершины соответствуют узлам сети, а рёбра - проводникам или линиям связи. Для анализа таких сетей используется подход основаный на теории графов и модели случайных блужданий.

Сопротивление между двумя узлами i и j в электрической сети можно вычислить, моделируя сеть в виде графа и используя случайные блуждания. В частности, среднее время первого попадания из узла i в узел j тесно связано с электрическим сопротивлением между этими узлами.

Для расчета среднего времени первого попадания (hitting time) можно использовать следующую формулу. Пусть G - граф, где V - множество вершин, E - множество рёбер. Рассмотрим однородное случайное блуждание, при котором вероятность перехода из одной вершины в любую соседнюю вершину, одинакова и равно к $1/d(j)$. А среднее время случайного перемещения из i в j и обратно (commute time) будет тогда равно $1/d(j) + 1/d(i)$, где $d(x)$ - степень вершины x.

2.3 Время первого попадания

2.3.1 Теорема 1. О времени первого попадания

Среднее время первого попадания в вершину j из вершины i определяется как разность потенциалов в Сценарии А:

$$\phi_{ij} = H_{ij} \tag{1}$$

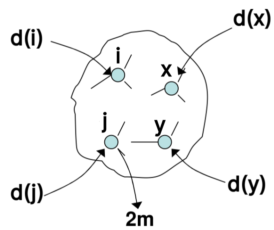

Рис. 1: Сценарий А

Теорема взята из [7], лемма 24.6.

2.3.2 Доказательство

Для доказательства данной теоремы введем законы Кирхгофа и Ома:

- Суммарный ток в вершине и из нее равен нулю (К1) [5]
- Сумма разностей потенциалов в любом цикле равна нулю (К2) [5]
- Ток, протекающий по любому ребру, равен $\frac{\text{разности потенциалов}}{\text{сопротивление}}$ (Закон Ома) [6]

Теперь рассмотрим любую вершину $i \in V$. Используя законы Кирхгофа и Ома, мы имеем:

$$\begin{aligned} d(i) &= \sum_{(i,x)\in E} \text{текущая } i \to x (\text{К1}) \\ &= \sum_{(i,x)\in E} \phi_{ix} (\text{Закон Ома}) \\ &= \sum_{(i,x)\in E} (\phi_{ij} - \phi_{xj}) (\text{К2}) \\ &= d(i)\phi_{ij} - \sum_{(i,x)\in E} \phi_{xj} \end{aligned}$$

Преобразование даст:

$$\phi_{ij} = \begin{cases} 0 & i = j \\ 1 + \frac{1}{d(i)} \sum\limits_{(i,x)\in E} \phi_{xj} & i \neq j \end{cases} \tag{2}$$

С другой стороны, рассмотрим случайную прогулку, начинающуюся с i. Рассматривая только первый шаг этой прогулки, мы видим, что время попадания H_{uv} удовлетворяет следующим условиям:

$$H_{ij} = \begin{cases} 0 & i = j \\ 1 + \frac{1}{d(i)} \sum\limits_{(i,x)\in E} H_{xj} & i \neq j \end{cases} \tag{3}$$

Но это точно такая же система линейных уравнений, как и (2), удовлетворяемая ϕ_{ij}. Поскольку мы знаем, что эта система имеет единственное решение (потенциалы однозначно определяются потоками тока), мы выводим, что:

$$H_{ij} = \phi_{ij} \quad \forall i \in V$$

■

2.4 Эффективное сопротивление

Эффективное сопротивление (резистивное расстояние) $\Omega_{vw}(\mathcal{L})$ между вершинами u и v в графе с равным сопротивлением r на всех рёбрах определяется как сумма взвешенных квадратов разностей между компонентами собственных векторов графа для всех пар узлов v и w. В частности, для каждого узла v и w вычисляется разность компонент собственных векторов ψ_{jv} и ψ_{jw}, которые соответствуют ненулевым собственным значениям $\lambda_j(\mathcal{L})$ лапласиана. Затем, каждая разность возводится в квадрат, делится на соответствующее собственное значение, и все такие дроби суммируются для всех j от 2 до n. :

$$\Omega_{vw}(\mathcal{L}) = \sum_{j=2}^{n} \frac{1}{\lambda_j(\mathcal{L})}(\psi_{jv} - \psi_{jw})^2 \tag{4.1}$$

Данная запись присутствует в [4] на стр. 11, но для удобства, формула (4.1) будет видоизменена:

$$R_{ij} = \sum_{k=1}^{n-1} \frac{(u_{ki} - u_{kj})^2}{\lambda_k} \tag{4.2}$$

В формуле 4.2 λ_k – это собственные значения Лапласиана, где $\lambda_0 = 0$, u_{ki} и u_{kj} - i-я и j-я компоненты k-го собственного вектора Лапласиана, а n – это вершины. Лапласиан, или матрица Лапласа определяется как матрица степеней D минус матрица смежности A:

$$\mathcal{L} = D - A$$

2.5 Время возвращения

2.5.1 Теорема 2. О времени возвращения

Среднее время случайного перемещения из i в j и обратно определяется как среднее время первого попадания в вершину j из вершины i плюс среднее время первого попадания в вершину i из вершины j иле же как количество ребер m умноженное на эффективное сопротивление R_{ij} умноженное на два.

$$C_{ij} = H_{ij} + H_{ji} = 2mR_{ij} \tag{5}$$

Формула взята из [1], стр. 577.

2.5.2 Доказательство

Для доказательства используем сценарий B, который похож на сценарий A, за исключением того, что мы удаляем $2m$ единиц тока с узла i, а не с узла j. Обозначая разность потенциалов в сценарии B через $\phi^{'}$, мы имеем, согласно теореме 1:

$$\phi^{'}_{ji} = H_{ji} \tag{6}$$

Теперь рассмотрим сценарий C, который похож на сценарий B, но с обратными токами. Обозначив разность потенциалов обозначив разность потенциалов в этом сценарии через $\phi^{''}$, мы получим:

$$\phi^{''}_{ij} = \phi^{'}_{ji} = H_{ji} \tag{7}$$

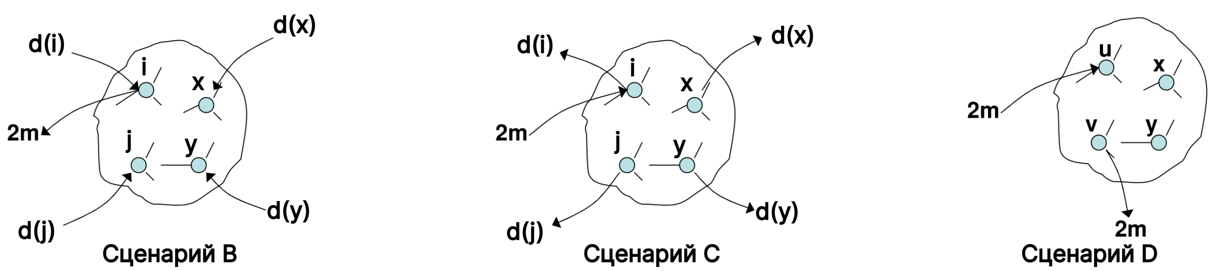

Рис. 2: Сценарий B,C и D

Наконец, рассмотрим сценарий D, который представляет собой сумму сценариев A и C. В силу линейности и обозначения потенциальные различия в сценарии D через $\phi^{'''}$, мы имеем:

$$\phi^{'''}_{ij} = \phi_{ij} + \phi^{''}_{ij} = H_{ij} = H_{ji} \tag{8}$$

Так как $\phi^{''}_{ij}$ – это разность потенциалов, необходимая для перемещения $2m$ единиц тока из i в j, поэтому по закону Ома она равна $2mR_{ij}$.

Реализация программы

3.1 Переменные

3.1.1 Входные переменные

- `adj` - матрица смежности графа.
- `start` - индекс начального узла.
- `end_` - индекс конечного узла.
- `num_sim` - количество симуляций.

3.1.2 Выходные переменные

- `fht` - вектор, содержащий время первого попадания в конечный узел для каждой симуляции.
- `ct` - вектор, содержащий время полного обхода графа для каждой симуляции.
- `cmt` - вектор, содержащий время перемещения от начального узла к конечному и обратно для каждой симуляции.
- `sfh` - отсортированный вектор времени первого попадания и их количества.
- `sct` - отсортированный вектор времени полного обхода графа и их количества.
- `scmt` - отсортированный вектор времени перемещения и их количества.
- `mfht` - среднее время первого попадания в конечный узел.
- `mct` - среднее время полного обхода графа.
- `mcmt` - среднее время перемещения от начального узла к конечному и обратно.
- `eff_res` - эффективное сопротивление графа, рассчитанное как среднее время перемещения, деленное на удвоенное количество ребер.

3.1.3 Промежуточные переменные

- `n` - количество узлов в графе.
- `m` - количество ребер в графе, рассчитанное как половина суммы всех элементов матрицы смежности.
- `curr` - текущий узел в процессе блуждания.
- `visited` - булевый вектор, указывающий, были ли посещены узлы.
- `steps` - количество шагов, сделанных в текущей симуляции.
- `first_hit` - булева переменная, указывающая, было ли достигнуто первое попадание в конечный узел.
- `visit_count` - количество посещенных узлов.
- `neighbors` - вектор соседних узлов для текущего узла.
- `next` - следующий узел для перехода.
- `cms` - количество шагов для перемещения от начального узла к конечному и обратно в текущей симуляции.
- `file_path` - путь к файлу с матрицей смежности.
- `elapsed_time` - время, затраченное на выполнение симуляций.

3.2 Скрипт random_walk.m

В данной работе была разработана программа на языке Octave, которая имитирует случайные блуждания по графу, заданному матрицей смежности. Программа рассчитывает статистические данные. Об этих данных ниже рассказано более подробно.

⟨ *random_ walk* 8a ⟩ ≡

```
function [fht, ct, sfh, sct, mfht, mct, eff_res, ...
mcmt, scmt] = random_walk(adj, start, end_, num_sim)
```
⟨ *initialization of variables* 8c ⟩
⟨ *simulation loop* 8d ⟩
⟨ *calculation* 10b ⟩
⟨ *sort function* 10c ⟩
◇

Fragment referenced in 8b.

"random_walk.oc" 8b≡
⟨ *random_ walk* 8a ⟩◇

В этом блоке инициализируются переменные для количества узлов и ребер в графе, векторов для хранения времени первого попадания, времени полного обхода, и времени перемещения.
⟨ *initialization of variables* 8c ⟩ ≡

```
    n = size(adj, 1);
    m = sum(adj(:)) / 2;
    fht = zeros(num_sim, 1);
    ct = zeros(num_sim, 1);
    cmt = zeros(num_sim, 1);
```
◇

Fragment referenced in 8a.

Этот блок выполняет цикл по числу симуляций для выполнения случайных блужданий.
⟨ *simulation loop* 8d ⟩ ≡

⟨ *initialising the current state* 8e ⟩
⟨ *graph wandering cycle* 9a ⟩
⟨ *hit check* 9b ⟩
⟨ *update visited nodes* 9c ⟩
⟨ *record round-trip time* 9d ⟩
⟨ *calculation of travel time* 10a ⟩
◇

Fragment referenced in 8a.

Этот блок начинает наш основной цикл, инициализируюя текущий узел как начальный узел , а также массивы для отслеживания посещенных узлов и количества шагов. Также инициализируются переменные для отслеживания первого попадания в конечный узел и количества посещенных узлов.
⟨ *initialising the current state* 8e ⟩ ≡

```
    for sim = 1:num_sim
        curr = start;
        visited = false(n, 1);
        visited(start) = true;
        steps = 0;
        first_hit = false;
        visit_count = 1;
```
◇

Fragment referenced in 8d.

Этот цикл продолжается до тех пор, пока не будут посещены все узлы графа. Внутри цикла определяется следующий узел для перехода случайным образом из соседей текущего узла.

⟨ *graph wandering cycle* 9a ⟩ ≡

```
        while visit_count < n
            neighbors = find(adj(curr, :));
            next = neighbors(randi(length(neighbors)));
            steps = steps + 1;
            curr = next;
```

◇

Fragment referenced in 8d.

Если это первое попадание в конечный узел, сохраняется количество шагов до первого попадания.

⟨ *hit check* 9b ⟩ ≡

```
            if ~first_hit && curr == end_
                fht(sim) = steps;
                first_hit = true;
            end
```

◇

Fragment referenced in 8d.

Если узел еще не был посещен, он помечается как посещенный, и увеличивается счетчик посещений.

⟨ *update visited nodes* 9c ⟩ ≡

```
            if ~visited(curr)
                visited(curr) = true;
                visit_count = visit_count + 1;
            end
        end
```

◇

Fragment referenced in 8d.

После завершения обхода всех узлов записывается количество шагов.

⟨ *record round-trip time* 9d ⟩ ≡

```
        ct(sim) = steps;
```

◇

Fragment referenced in 8d.

Затем рассчитывается время перемещения от начального узла к конечному и обратно. Это выполняется двумя циклами: один до конечного узла, и один обратно к начальному узлу.

⟨ *calculation of travel time* 10a ⟩ ≡

```
            cms = 0;
            curr = start;
            while curr ~= end_
                neighbors = find(adj(curr, :));
                next = neighbors(randi(length(neighbors)));
                cms = cms + 1;
                curr = next;
            end
            while curr ~= start
                neighbors = find(adj(curr, :));
                next = neighbors(randi(length(neighbors)));
                cms = cms + 1;
                curr = next;
            end
            cmt(sim) = cms;
        end
    ◇
```

Fragment referenced in 8d.

Здесь сортируются и подсчитываются вхождений для времени первого попадания, времени обхода и времени перемещения, используя вспомогательную функцию сортировки. Далее рассчитывается среднее время первого попадания, среднее время обхода, среднее время перемещения и эффективное сопротивление.
⟨ *calculation* 10b ⟩ ≡

```
        sfh = sort_and_count(fht);
        sct = sort_and_count(ct);
        scmt = sort_and_count(cmt);
        mfht = mean(fht);
        mct = mean(ct);
        mcmt = mean(cmt);

        eff_res = mcmt / (2 * m);
    end
    ◇
```

Fragment referenced in 8a.

Определение вспомогательной функции, которая сортирует данные и подсчитывает вхождения. Возвращает матрицу с отсортированными данными и их количеством.
⟨ *sort function* 10c ⟩ ≡

```
    function sc = sort_and_count(data)
        [sorted_data, ~, idx] = unique(data);
        counts = accumarray(idx, 1);
        sc = [sorted_data, counts];
    end
    ◇
```

Fragment referenced in 8a.

3.3 Скрипт run_random_walk.m

Для запуска функции random_walk и получения результатов был разработан скрипт run_random_walk. Этот скрипт задаёт параметры графа, запускает функцию random_walk и выводит результаты на экран.

⟨ *run_ random_ walk* 11a ⟩ ≡

⟨ *parameters set-up* 11c ⟩
⟨ *running simulation* 11d ⟩
⟨ *results output* 12 ⟩
◇

Fragment referenced in 11b.

"run_random_walk.oc" 11b≡
⟨ *run_ random_ walk* 11a ⟩◇

Указание пути к файлу с матрицей смежности, начального узла, конечного узла и числа симуляций. Затем чтение матрицы смежности из указанного файла с помощью функции `dlmread` и сохранение в переменную матрицы смежности.

⟨ *parameters set-up* 11c ⟩ ≡

```
file_path = 'adjacency_matrix.txt';
start = 1;
end_ = 442;
num_sim = 1000;

adj = dlmread(file_path);
```
◇

Fragment referenced in 11a.

Выполнение функции случайного блуждания с заданными параметрами и измерение затраченного времени с помощью `tic` и `toc`. Результаты сохраняются в соответствующих переменных, а время выполнения - в переменной `elapsed_time`.

⟨ *running simulation* 11d ⟩ ≡

```
tic;
[fht, ct, sfh, sct, mfht, mct, eff_res, mcmt, scmt] = random_walk(adj, start, end_, num_sim);
elapsed_time = toc;
```
◇

Fragment referenced in 11a.

Вывод результатов в консоль.

⟨ *results output* 12 ⟩ ≡

```
fprintf('Среднее время первого попадания в вершину %d из вершины %d: %f шага.\n', end_, start, mfht);
fprintf('Среднее время прохода из вершины %d в вершину %d и обратно: %f шага.\n', start, end_, mcmt);
fprintf('Среднее время обхода всего графа: %f шага.\n', mct);
fprintf('Эффективное сопротивление: %f.\n', eff_res);
fprintf('Время выполнения программы: %f секунд.\н', elapsed_time);



fprintf('Повторения времени первого попадания в вершину (в формате [время-количество]):\n');
    for i = 1:size(sfh, 1)
       fprintf('[%d-%d]', sfh(i, 1), sfh(i, 2));
       if i ~= size(sfh, 1)
           fprintf(', ');
       else
           fprintf('.\n');
       end
end

fprintf('Повторения времени прохода из вершины %d в вершину %d и обратно (в формате [время-количество]):\n'
    for i = 1:size(sfh, 1)
       fprintf('[%d-%d]', scmt(i, 1), scmt(i, 2));
       if i ~= size(sfh, 1)
           fprintf(', ');
       else
           fprintf('.\n');
       end
end

fprintf('Повторения времени обхода всего графа (в формате [время-количество]):\n');
    for i = 1:size(sfh, 1)
       fprintf('[%d-%d]', sct(i, 1), sct(i, 2));
       if i ~= size(sfh, 1)
           fprintf(', ');
       else
           fprintf('.\n');
       end
end
◇
```

Fragment referenced in 11a.

Результаты и их анализ

4.1 Проведение тестов

В качестве примеров мы рассмотрим два графа. Первый граф будет тестовым для понимания методологии решения, а вторым графом является Московское метро.

4.1.1 Практический пример 1

$$\text{adj\_matrix} = \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix}$$

Начальная вершина была выбрана как вершина 1, а конечная вершина как вершина 4. Количество симуляций было установлено на 1000. Результаты симуляций показали следующее:

- Среднее время первого попадания в вершину 4 из вершины 1: 5.047200 шага.
- Среднее время прохода из вершины 1 в вершину 4 и обратно: 10.004400 шага.
- Среднее время обхода всего графа: 5.934400 шага.
- Эффективное сопротивление: 1.000440.
- Время выполнения программы: 7.940940 секунд.
- Повторения времени первого попадания в вершину (в формате [время-количество]): [2-3277], [3-1110], [4-1476], [5-881], [6-770], [7-578], [8-460], [9-346], [10-252], [11-198], [12-157], [13-116], [14-79], [15-69], [16-54], [17-33], [18-30], [19-27], [20-11], [21-8], [22-15], [23-7], [24-11], [25-11], [26-7], [27-4], [28-4], [29-2], [30-1], [31-1], [32-3], [36-1], [39-1].
- Повторения времени прохода из вершины 1 в вершину 4 и обратно (в формате [время-количество]): [4-1141], [5-754], [6-1116], [7-857], [8-930], [9-813], [10-701], [11-579], [12-551], [13-458], [14-388], [15-340], [16-241], [17-193], [18-172], [19-149], [20-126], [21-102], [22-66], [23-72], [24-46], [25-50], [26-34], [27-24], [28-25], [29-14], [30-15], [31-8], [32-6], [33-8], [34-6], [35-3], [36-2], [38-1], [39-3], [40-2], [41-1], [43-1], [47-1], [54-1].
- Повторения времени обхода всего графа (в формате [время-количество]): [3-2768], [4-1493], [5-1645], [6-959], [7-896], [8-538], [9-479], [10-288], [11-238], [12-175], [13-126], [14-86], [15-77], [16-54], [17-33], [18-30], [19-28], [20-11], [21-8], [22-15], [23-7], [24-11], [25-11], [26-7], [27-4], [28-4], [29-2], [30-1], [31-1], [32-3], [36-1], [39-1].

Стоит заметить, что при стократном увеличении симуляций, результаты изменятся лишь на немного, что можно посчитать как погрешность

4.1.2 Практический пример 2

В качестве второго примера будет взята матрица на 100 вершин. Начальная вершина была выбрана как вершина 1, а конечная вершина как вершина 442. Количество симуляций было установлено на 100000. Результаты симуляций показали следующее:

- Среднее время первого попадания в вершину 100 из вершины 1: 116.615000 шага.
- Среднее время прохода из вершины 1 в вершину 100 и обратно: 296.011000 шага.
- Среднее время обхода всего графа: 1463.259000 шага.
- Эффективное сопротивление: 0.564906.
- Время выполнения программы: 109.910745 секунд.
- Повторения времени первого попадания в вершину (в формате [время-количество]): [4-6], [5-4], [6-8], [7-7], [8-11], [9-8], [10-16], [11-12], [12-6], [13-10], [14-7], [15-11], [16-6], [17-7], [18-6], ... , [318-1], [319-1], [320-1], [321-2], [324-2], [325-1], [330-1], [331-1], [332-1], [337-1], [338-2], [344-2], [352-1], [353-2], [360-1], [363-1], [367-1], [369-1], [373-1], [377-1], [379-1], [384-2], [387-2], [392-1], [395-1], [408-1], [409-1], [415-1], [417-1], [418-1], [421-1], [425-1], [434-1], [435-1], [436-2], [437-2], [439-1], [441-1], [444-1], [446-2], [447-1], [449-1], [454-2], [462-1], [469-2], [478-1], [480-1], [483-1], [496-1], [498-1], [538-1], [551-1], [569-1], [610-1], [627-1], [652-1], [691-1], [813-1].
- Повторения времени прохода из вершины 1 в вершину 100 и обратно (в формате [время-количество]): [17-1], [18-1], [21-1], [22-1], [26-2], [27-1], [29-3], [30-1], [31-1], [32-2], [33-1], [36-4], [37-1], [38-1], [40-1], [42-1], [43-1], ... , [658-1], [675-1], [680-1], [681-2], [684-1], [685-1], [687-1], [691-1], [692-1], [693-1], [695-2], [702-1], [711-1], [727-1], [733-1], [737-1], [740-1], [743-1], [760-1], [764-1], [765-1], [773-1], [774-1], [789-1], [794-1], [796-1], [797-1], [806-1], [816-1], [819-2], [820-1], [822-1], [824-1], [845-1], [856-1], [870-1], [873-1], [899-1], [900-1], [921-2], [945-1], [949-1], [951-1], [963-1], [964-1], [1018-1], [1056-1], [1075-1], [1127-1], [1131-1], [1133-1], [1206-1], [1303-1], [1307-1], [1407-1], [1414-1], [1658-1].
- Повторения времени обхода всего графа (в формате [время-количество]): [438-1], [440-1], [482-1], [490-1], [496-1], [497-1], [500-1], [503-2], [541-2], [544-1], ... , [2741-1], [2746-1], [2753-1], [2766-1], [2775-1], [2777-2], [2788-1], [2807-1], [2808-1], [2814-1], [2816-1], [2819-1], [2852-1], [2867-1], [2881-1], [2887-1], [2895-1], [2910-1], [2949-1], [2953-1], [3011-1], [3049-1], [3072-1], [3074-1], [3075-1], [3082-1], [3092-1], [3094-1], [3104-1], [3122-1], [3142-1], [3227-1], [3279-1], [3309-1], [3337-1], [3350-1], [3382-1], [3394-1], [3456-1], [3490-1], [3500-1], [3518-1], [3581-1], [3603-1], [3672-1], [3749-1], [3766-1], [3916-1], [4203-1], [4288-1], [4346-1], [4631-1], [4669-1], [4764-1], [5158-1], [5414-1], [6796-1].

4.2 Теоретические расчеты

4.2.1 Теоретический пример 1

Теперь с помощью теории рассчитаем количество шагов до попадания в вершину 4 из вершины 1. Составим матрицу степеней из матрицы смежности, используемой выше:

$$\mathrm{D} = \begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix}$$

Из этих двух матриц составим матрицу Лапласа (Лапласиан), используя данную формулу: L = D - adj_matrix

$$\begin{pmatrix} 2 & 0 & 0 & 0 \\ 0 & 3 & 0 & 0 \\ 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 \end{pmatrix} - \begin{pmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} = \begin{pmatrix} 2 & -1 & -1 & 0 \\ -1 & 3 & -1 & -1 \\ -1 & -1 & 3 & -1 \\ 0 & -1 & -1 & 2 \end{pmatrix} = \mathrm{L}$$

Далее мы получаем собственные значения и собственные вектора матрицы Лапласа, чтобы их получить я воспользуюсь Octave:

⟨ *laplacian* 14a ⟩ ≡

```
laplacian_matrix = [2 -1 -1 0;
                   -1 3 -1 -1;
                   -1 -1 3 -1;
                    0 -1 -1 2];


[eigenvectors, eigenvalues_matrix] = eig(laplacian_matrix);

eigenvalues = diag(eigenvalues_matrix);

disp('Собственные значения:');
disp(eigenvalues);

disp('Собственные вектора:');
disp(eigenvectors);
◇
```

Fragment referenced in 14b.

`"laplacian.oc"` 14b≡

⟨ *laplacian* 14a ⟩◇

Где eigenvalues – это λ_k, а eigenvectors – это u_i

$$\lambda_1 = 0, \quad \lambda_2 = 2, \quad \lambda_3 = 4, \quad \lambda_4 = 4$$

$$\mathrm{U} = \begin{pmatrix} -0.50 & -0.71 & 0.49 & 0.09 \\ -0.50 & 0.00 & -0.62 & 0.60 \\ -0.50 & 0.00 & -0.36 & -0.79 \\ -0.50 & 0.71 & 0.49 & 0.09 \end{pmatrix}$$

У собственных значений убираем нулевое значения и берем нулевую и третью строки из матрицы U для расчета R_{14}. Получаем следующее:

$$\lambda_2 = 2, \quad \lambda_3 = 4, \quad \lambda_4 = 4$$

$$u_0 = (-0.50, -0.71, 0.49, 0.09)$$

$$u_3 = (-0.50, 0.71, 0.49, 0.09)$$

Теперь нам нужны значения n и m. У нас n уже известна, она равна 4, поэтому осталось найти m. Для этого находим сумму всех элементов матрицы смежности и делим их на 2.

$$\sum_{i,j} A_{ij} = 0 + 1 + 1 + 0 + 1 + 0 + 1 + 1 + 1 + 1 + 0 + 1 + 0 + 1 + 1 + 0 = 10$$

$$E = \frac{10}{2} = 5$$

В итоге получаем, что $m = 5$ и $n = 4$. Получив эти значения, мы можем посчитать эффективное сопротивление, воспользуемся формулой (4.2):

$$R_{14} = \sum_{k=1}^{n-1} \frac{(u_{k1} - u_{k4})^2}{\lambda_k} = \frac{(0.71 - (-0.71))^2}{2} + \frac{(0.49 - 0.49)^2}{4} + \frac{(0.09 - 0.09)^2}{4} \approx 1.0082$$

Теперь нужно посчитать commute time, для этого воспользуемся формулой (5):

$$C_{14} = 2mR_{14} = 2 * 5 * 1.0083 \approx 10.0082$$

Нам осталось посчитать только hitting time, но так как считая эту форуму в ручную мы получим не совсем точные значения, то предлагаю воспользоваться формулой (5):

$$C_{14} = H_{14} + H_{41}$$

Для нашего маленького и симметричного графа, это равносильно что:

$$C_{14} \approx 2H_{14}$$

Следовательно H_{14} равен:

$$H_{14} \approx \frac{C_{14}}{2} \approx \frac{10.0082}{2} \approx 5.0041$$

4.2.2 Теоретические пример 2

Пример 1 был приведен для демонстрации методологии. Так как решение данного графа на 100 вершин в ручную может затянуться надолго, воспользуемся программным методом, в котором будет реализованы математические формулы описанные выше.

⟨ *theoretical_ random_ walk* 16 ⟩ ≡

```
file_path = '1adjacency_matrix.txt';
adj_matrix = dlmread(file_path);

n = size(adj_matrix, 1);
m = sum(sum(adj_matrix)) / 2;

degree_matrix = sum(adj_matrix, 2); % Определение переменной degrees

degrees = diag(degree_matrix);

laplacian_matrix = degrees - adj_matrix;

[eigenvectors, eigenvalues_matrix] = eig(laplacian_matrix);

eigenvalues = diag(eigenvalues_matrix);

eigenvalues_nonzero = eigenvalues(2:end);
eigenvectors_nonzero = eigenvectors(:, 2:end);

H = zeros(n, n);

max_iter = 20000;
tol = 1e-16;

for iteration = 1:max_iter
    H_prev = H;
    for u = 1:n
        for v = 1:n
            if u ~= v
                H(u, v) = 1 + (1 / degree_matrix(u)) * sum(H(adj_matrix(u, :) == 1, v));
            else
                H(u, v) = 0;
            end
        end
    end

    if mod(iteration, 1) == 0
        fprintf('Iteration %d, max change: %e\n', iteration, max(max(abs(H - H_prev))));
    end

    if max(max(abs(H - H_prev))) < tol
        break;
    end
end

i = 1;
j = 14;

H_ij = H(i, j);

R_ij = sum((eigenvectors_nonzero(i, :) - eigenvectors_nonzero(j, :)).^2 ./ eigenvalues_nonzero');

C_ij = 2 * m * R_ij;

fprintf('Среднее время прохода из вершины %d в вершину %d: %f.\n', i, j, H_ij);
fprintf('Среднее время прохода из вершины %d в вершину %d и обратно: %f.\n', i, j, C_ij);
fprintf('Эффективное сопротивление: %f.\n', R_ij);
```
◇

Fragment referenced in 17.

"theoretical_random_walk.oc" 17≡

⟨ *theoretical_ random_ walk* 16 ⟩◇

Результаты данных вычислений:

- Среднее время прохода из вершины 1 в вершину 100: 116.068140.
- Среднее время прохода из вершины 1 в вершину 100 и обратно: 298.611498.
- Эффективное сопротивление: 0.569869

Заключение

Результаты вычислительных экспериментов и теоретических расчетов для графа показывают высокую степень согласованности, что свидетельствует о правильности и эффективности используемых моделей и алгоритмов. Тестирование проводилось на двух графах с 4 вершинами и с 100 вершинами, и в обоих случаях теория совпала с практикой, что подтверждает правильность выбранных методов и их реализацию. При этом стоит отметить, что cover time, полученный из тестов, действительно меньше, чем commute time, что соответствует теоретическим ожиданиям и подтверждает правильность работы программ. Такая высокая степень совпадения результатов теории и практики говорит о том, что даже при увеличении размера графа, выбранные алгоритмы остаются корректными и эффективными. Эти результаты позволяют сделать вывод о возможности масштабирования используемых методов на более крупные графы, сохраняя при этом их точность и надежность.

Список литературы

[1] Ashok K. Chandra и др. “The Electrical Resistance of a Graph Captures its Commute and Cover Times”. В: (1989). URL: https://www.researchgate.net/publication/220268346_The_Electrical_Resistance_of_a_Graph_Captures_its_Commute_and_Cover_Times_Detailed_Abstract.

[2] Gary Chartrand и Ping Zhang. “A First Course in Graph Theory”. В: (2005). URL: http://lib.ysu.am/disciplines_bk/86ed8ab971105564c1b66357510f992a.pdf.

[3] Reinhard Diestel. “Graph Theory”. В: 173 (2000). URL: https://www.emis.de/monographs/Diestel/en/GraphTheoryII.pdf.

[4] Ernesto Estrada. “Path Laplacians versus fractional Laplacians as nonlocal operators on networks”. В: *New Journal of Physics* 23 (2021). URL: https://www.researchgate.net/publication/353256249_Path_Laplacians_versus_fractional_Laplacians_as_nonlocal_operators_on_networks.

[5] “Kirchhoff’s Rules”. В: (). URL: https://www.theexpertta.com/book-files/OpenStaxUniversityPhysicsVol2/UP2_10.3.%20Kirchhoff_s%20Rules_pg453-465.pdf.

[6] “Resistance and Ohms Law”. В: (). URL: https://www.salfordphysics.com/gsmcdonald/E_02_Lecture02.pdf.

[7] Alistair Sinclair. “Random Walks”. В: *CS271 Randomness & Computation* (2022). Lecture 24: November 15, University of California, Berkeley. URL: https://people.eecs.berkeley.edu/~sinclair/cs271/n24.pdf.

[8] Douglas B. West. “Introduction to Graph Theory”. В: (2001). URL: https://athena.nitc.ac.in/summerschool/Files/West.pdf.