Титульный лист, титульный лист, титульный лист, титульный лист титульный лист титульный лист титульный лист титульный лист ТИТУЛЬНЫЙ ЛИСТ ТИТУЛЬНЫЙ ЛИСТ титульный Лист титульный Лист титульный Лист Лист Титульный Лист Титульный Лист Титульный

титульный лист ? титульный ЛИСТ?

титульный… с П и с о К?

Ясно, автор сошел с ума

Автор: Румянцев А. А.

Факультет: СУиР

Группа: R3241

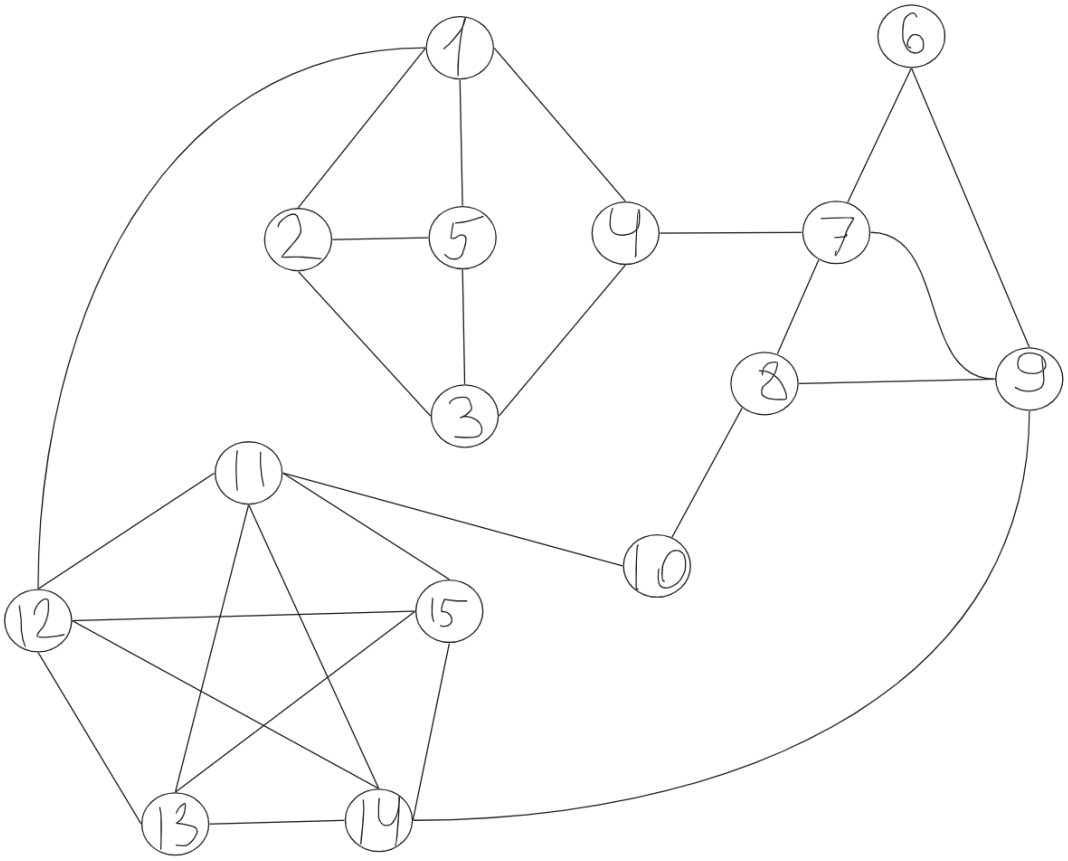
Поток: Прак.Лин.Ал. 1.3

Преподаватель: Перегудин А. А.

Содержание

Задание 1

Составим связный граф из 15 вершин, которые пронумеруем от 1 до 15. Можно увидеть, что на графе отчетливо видно 3 сообщества – ромбик, треугольничек и шестиугольник со звездочкой внутри



Составим матрицу Лапласа в соответствии с нумерацией вершин и найдем ее собственные числа и соответствующие им собственные вектора, используя язык программирования “python” и библиотеку “numpy”. Заодно напишем и разберем один из алгоритмов кластеризации – “k-means” или же «Метод k-средних», используя библиотеку “sklearn”

Конечно, вычислить собственные числа и собственные вектора можно было с помощью сайта matrixcalc.org, но с ним у меня не получалась хорошая кластеризация

Код

```
from sklearn.cluster import KMeans
import numpy as np


def get_eigen(matrix):
    eig_val, eig_vec = np.linalg.eig(matrix)
    idx = np.argsort(eig_val)
    eig_val = eig_val[idx]
    eig_vec = eig_vec[:, idx]
    return (eig_val, eig_vec)


laplacian = np.array(
    [
        [4, -1, 0, -1, -1, 0, 0, 0, 0, 0, 0, -1, 0, 0, 0],
        [-1, 3, -1, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, -1, 3, -1, -1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [-1, 0, -1, 3, 0, 0, -1, 0, 0, 0, 0, 0, 0, 0, 0],
        [-1, -1, -1, 0, 3, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 2, -1, 0, -1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, -1, 0, -1, 4, -1, -1, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, -1, 3, -1, -1, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, -1, -1, -1, 4, 0, 0, 0, 0, -1, 0],
        [0, 0, 0, 0, 0, 0, 0, -1, 0, 2, -1, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 5, -1, -1, -1, -1],
        [-1, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, 5, -1, -1, -1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, 4, -1, -1],
        [0, 0, 0, 0, 0, 0, 0, 0, -1, 0, -1, -1, -1, 5, -1],
        [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, -1, -1, -1, -1, 4],
    ]
)

eigens = get_eigen(laplacian)
clusters = 3
X = np.array(eigens[1][:, :clusters])

kmeans = KMeans(n_clusters=clusters, random_state=0, n_init="auto").fit(X)
print(kmeans.labels_)
```

Разберемся в том, что написано в коде:

> 1-2 строчка импортируют необходимые для работы кода библиотеки (в этой лабораторной достаточно “numpy” для операций с матрицами и “sklearn” для использования метода k-средних)

> С 5 по 10 строчку написана функция для поиска собственных чисел и собственных векторов передаваемой матрицы. Внутри метода сразу происходит сортировка собственных чисел от меньшего к большему и

соответствующих им собственных векторов. Возвращается кортеж, где на первом индексе находится список собственных чисел, а на втором список с векторами

> С 13 по 31 строчку в переменную “laplacian” записана построенная матрица Лапласа

> На 33 строчке в переменную “eigens” поступает кортеж после вызова функции “get_eigen(matrix)” с переданным параметром “laplacian”

> На 34 строчке задано количество кластеров, на которые программе необходимо будет поделить граф

> На 35 строчке составляется список «обрезанных векторов», и. с. список точек в пространстве $\mathbb{R}^k$, где k == clusters

> На 37 строчке в переменную “kmeans” записывается вычисленная кластеризация с помощью создания объекта “KMeans(…)” с переданными параметрами “n_clusters” – количество кластеров, “random_state” – детерминированная случайность при инициализации центроида, “n_init” – количество запусков алгоритма с разными начальными значениями центроида и вызова функции “fit(…)” для вычисления k-средних

> Наконец, 38 строчка выводит полученный список, где каждой позиции соответствует вершина на графе с индексом i+1

Выберем число желаемых компонент кластеризации графа k=4

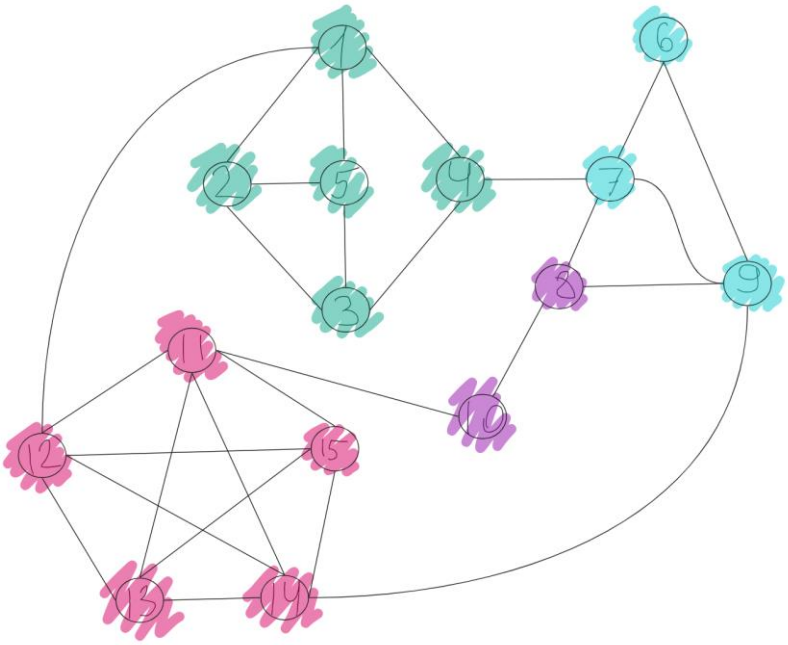
Осталось предоставить все программе – она возьмет k собственных векторов матрицы Лапласа, соответствующих самым маленьким собственным числам, составит набор точек пространства $\mathbb{R}^k$ и разобьет их на k кластеров

Полученный вывод в консоль:

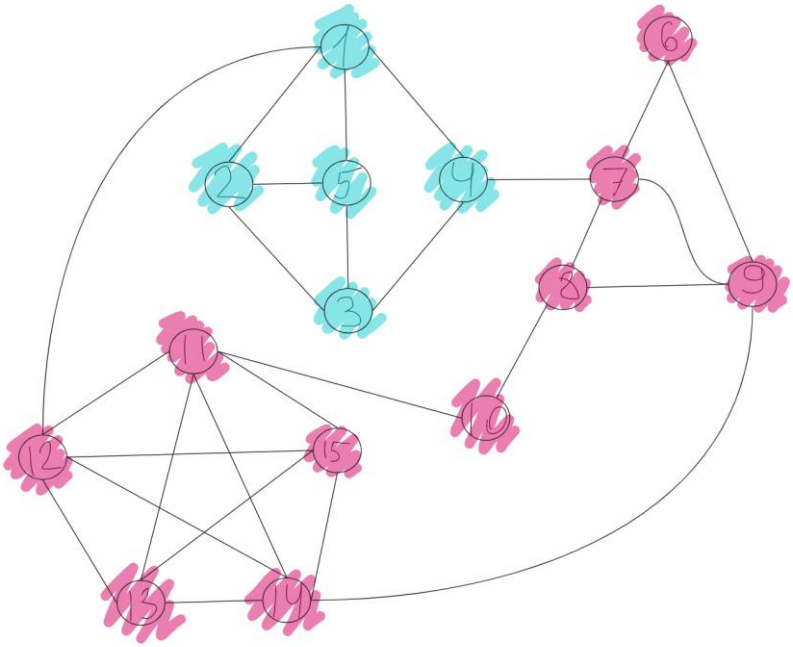
```
[2 2 2 2 2 0 0 3 0 3 1 1 1 1 1]
```

Теперь долгожданные рисуночки вместо душного текста – раскрасим вершины графа в соответствии с их принадлежностью к кластерам

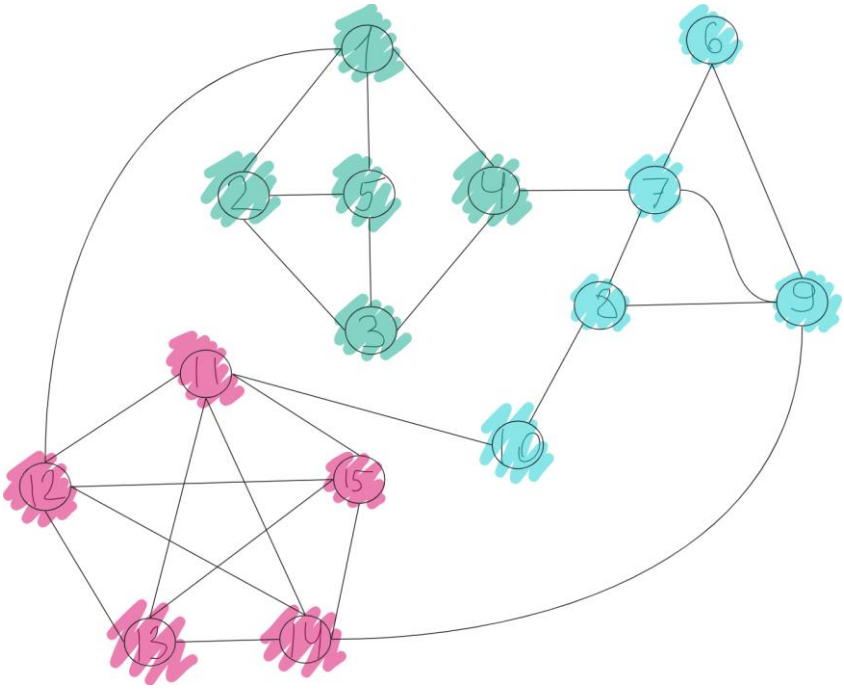
Получим следующий результат для k=4:



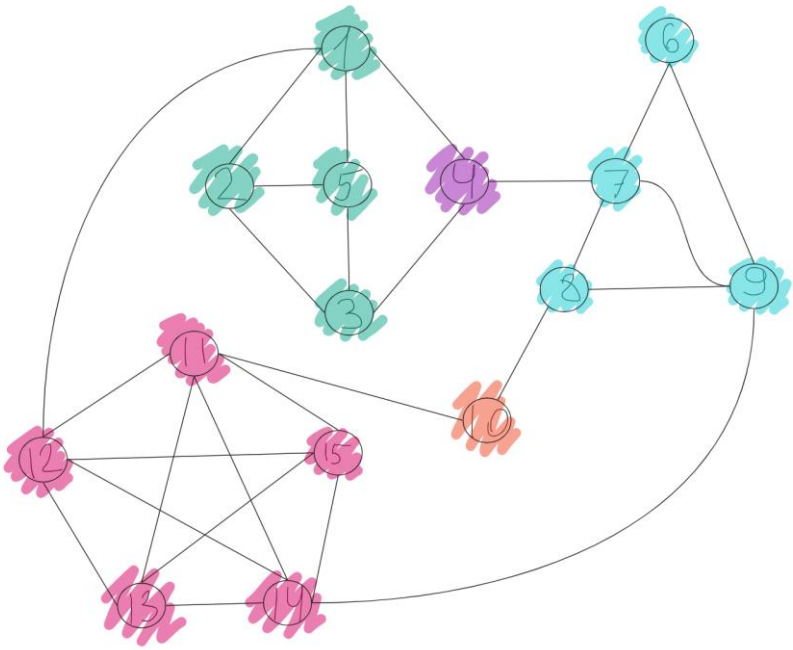
Пусть k=2, тогда кластеризация будет иметь вид:



А теперь для k=3!



А что, если k=5?



Почему это работает?

Рассмотрим нашу матрицу Лапласа. Она описывает простой конечный граф, она симметрична, ее определитель равен 0, сумма элементов каждой строки (столбца) равна 0. Получается, на самом деле мы работаем с матрицей Кирхгофа

Одним из собственных значений такой матрицы является 0. Его кратность равна числу связных компонент графа (в нашем случае кратность=1). Соответствующий собственный вектор состоит из единиц

Остальные собственные значения положительны. Второе по малости значение – индекс связности графа, а соответствующий собственный вектор – вектор Фидлера. Этот вектор можно использовать для разбиения графа

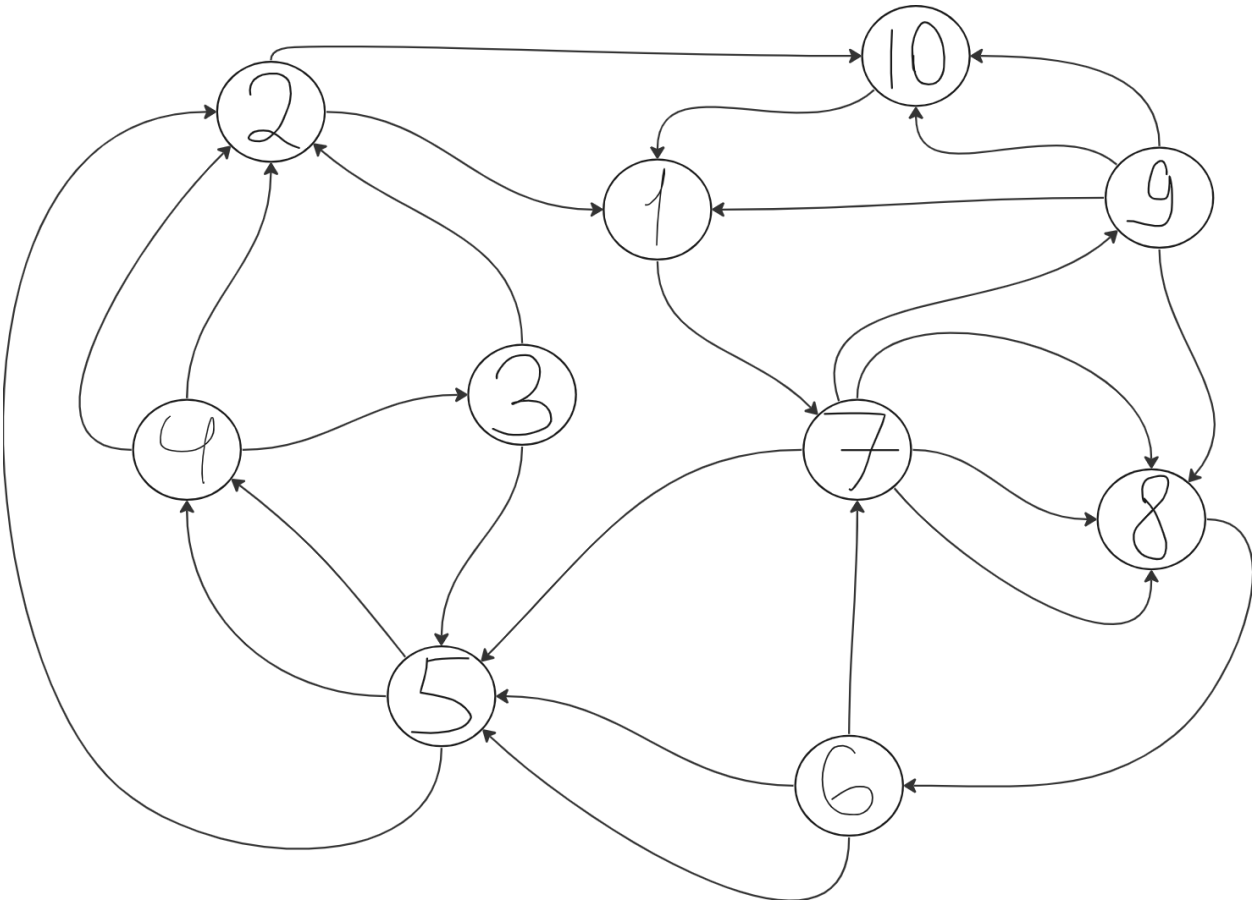
Пусть для некоторого конечного неориентированного графа вектор Фидлера имеет вид:

$$v = \begin{bmatrix} 0{,}415 \\ 0{,}309 \\ 0{,}069 \\ -0{,}221 \\ 0{,}221 \\ -0{,}794 \end{bmatrix}$$

Отрицательные значения на позициях 4 и 6 означают, что вершина 6 и соседняя точка сочленения 4 плохо связаны. Положительные же значения соответствуют остальным вершинам. Таким образом, знак элементов вектора Фидлера можно использовать для разбиения графа на две компоненты – {1, 2, 3, 5} и {4, 6}

Задание 2

Придумаем связный ориентированный граф из 10 вершин и 25 стрелочек, пронумеруем вершины от 1 до 10



Теперь составим матрицу M по следующему принципу:

$$m_{ij} = \frac{\text{число стрелочек, выходящих из } j-\text{й вершины и входящих в } i-\text{ю}}{\text{общее число стрелочек, выходящих из } j-\text{й вершины}}$$

Записывать в word матрицу размера 10x10 немного неудобно, так что я представлю ее вам уже записанной в переменную M в коде на следующей странице. Разобрать работу кода можно тут

Также найдем собственный вектор для λ_{max}

Заимствованный код

```python
import numpy as np


def pagerank(M, num_iterations: int = 100, d: float = 0.85):
    N = M.shape[1]
    v = np.ones(N) / N
    M_hat = d * M + (1 - d) / N
    for i in range(num_iterations):
        v = M_hat @ v
    return v


M = np.array(
    [
        [0, 1 / 2, 0, 0, 0, 0, 0, 0, 1 / 4, 1],
        [0, 0, 1 / 2, 2 / 3, 1 / 3, 0, 0, 0, 0, 0],
        [0, 0, 0, 1 / 3, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 2 / 3, 0, 0, 0, 0, 0],
        [0, 0, 1 / 2, 0, 0, 2 / 3, 1 / 5, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 1, 0, 0],
        [1, 0, 0, 0, 0, 1 / 3, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 3 / 5, 0, 1 / 4, 0],
        [0, 0, 0, 0, 0, 0, 1 / 5, 0, 0, 0],
        [0, 1 / 2, 0, 0, 0, 0, 0, 0, 2 / 4, 0],
    ]
)
v = pagerank(M, 100, 1)
print(v)
```

$$\lambda_{max} = 1, \quad v = \begin{bmatrix} 1.88 \\ 1.52 \\ 0.38 \\ 1.14 \\ 1.71 \\ 1.56 \\ 2.4 \\ 1.56 \\ 0.48 \\ 1 \end{bmatrix}$$

Теперь ранжируем вершины нашего графа в соответствии с PageRank-алгоритмом при отсутствии затухания (d=1)

Получим следующий результат:

```
[0.13793103 0.11151871 0.02787968 0.08363903 0.12545855
0.11445341 0.17608217 0.11445341 0.03521643 0.07336757]
```

Это вектор с вероятностями пользователя добраться до какой-либо из страниц за 100 итераций без затухания

Результат выведен в порядке возрастания – найдем вероятно наиболее посещаемую вершину – вершина 7, вероятность посещения пользователем за 100 кликов по ссылкам составляет 0.17608217. Вероятно, наименее посещаемая вершина – 3, так как ей соответствует число 0.02787968 за 100 шагов

Сравним с собственным вектором, соответствующим наибольшему собственному числу. Можно заметить, что на позиции 3 находится самое маленькое число 0.38, а на позиции 7 самое большое число 2.4. Пускай числа отличаются от того, что мы получили с помощью алгоритма PageRank, однако они совпадают по характеру своей величины – наименьшее или наибольшее

Как работает этот алгоритм?

Перед прохождением итераций задается вектор, состоящий из единиц, после чего делится на число страниц (вершин). Таким образом, сумма координат этого вектора будет равна 1. Короче говоря, это вектор вероятностей, и мы считаем, что пользователь равновероятно может оказаться на одной из N страниц. Вероятность не может превышать 1. После прохождения num_it итераций мы имеем вектор вероятностей, каждая координата которого показывает, на сколько высока вероятность того, что пользователь через num_it кликов по ссылкам попадет именно на эту страницу (координату)

Параметр d играет роль коэффициента затухания. Пользователь, нажимающий на ссылки, в итоге перестанет нажимать на них. Таким образом d является вероятностью

того, что человек на любом этапе продолжит переходить по ссылкам, а 1-d - вероятность того, что пользователь перейдет на случайную страницу

Значения PageRank — это элементы доминирующего правого собственного вектора модифицированной матрицы смежности, масштаб которых изменен так, что сумма каждого столбца равна единице. Этот вектор выглядит так:

$$\mathbf{R} = \begin{bmatrix} PR(p_1) \\ PR(p_2) \\ \vdots \\ PR(p_N) \end{bmatrix}$$

Этот вектор - решение следующего уравнения:

$$\mathbf{R} = \begin{bmatrix} (1-d)/N \\ (1-d)/N \\ \vdots \\ (1-d)/N \end{bmatrix} + d \begin{bmatrix} \ell(p_1,p_1) & \ell(p_1,p_2) & \cdots & \ell(p_1,p_N) \\ \ell(p_2,p_1) & \ddots & & \vdots \\ \vdots & & \ell(p_i,p_j) & \\ \ell(p_N,p_1) & \cdots & & \ell(p_N,p_N) \end{bmatrix} \mathbf{R}$$

где функция смежности l — это отношение количества исходящих ссылок со страницы j на страницу i к общему количеству исходящих ссылок страницы j. Функция смежности равна 0, если страница p_j не связана с p_i

Из-за большого собственного зазора модифицированной матрицы смежности, значения собственного вектора PageRank могут быть аппроксимированы с высокой степенью точности всего за несколько итераций