БУ ВО «Сургутский государственный университет»

Политехнический институт

Кафедра автоматизированных систем обработки информации и управления

ОТЧЕТ

ПО ЛАБОРАТОРНОЙ РАБОТЕ №6

ПО ТЕМЕ «Реализация алгоритма PageRank »

ПО ДИСЦИПЛИНЕ «Поисковые алгоритмы в информационном пространстве»

Выполнил: студент группы №606-12,

Демьянцев Виталий Владиславович

Принял: ст. преподаватель,

Гавриленко Анна Владимировна

Сургут 2024

**Введение**

Данная лабораторная работа посвящена реализации и анализу алгоритма PageRank для оценки значимости узлов в графе. Основная цель — изучить влияние структуры графа и параметра затухания (α) на распределение весов, провести эксперименты с различными значениями α и визуализировать результаты.

PageRank — алгоритм, используемый для ранжирования веб-страниц, основанный на анализе ссылок. Он моделирует поведение случайного пользователя, переходящего по ссылкам с вероятностью α и случайным образом с вероятностью 1-α.

**Алгоритмы и методы**

**PageRank**

* **Описание:**
  + Матрица переходов M строится на основе графа: M[j, i] = 1 / num\_outgoing, если есть ссылка из i в j, иначе 0.
  + Учитывается damping factor: M = α \* M + (1-α) / N \* ones(N, N).
  + Итеративно вычисляется вектор PR: PR = M \* PR, пока не достигнута сходимость.
* **Временная сложность:** O(N² \* k), где N — число узлов, k — число итераций.

**Требования**

Для выполнения работы использован Python версии 3.x с библиотеками:

* numpy — для работы с матрицами;
* matplotlib и networkx — для визуализации.

**Реализация и эксперименты**

**Структура кода**

1. **Создание графа:**
   * create\_graph() — задает граф с 6 узлами (A–F).
2. **Матрица переходов:**
   * build\_transition\_matrix(graph, nodes, alpha) — строит матрицу M.
3. **PageRank:**
   * compute\_pagerank(M, nodes) — вычисляет веса узлов.
4. **Визуализация:**
   * plot\_graph(graph) — отображает граф.
   * plot\_history(nodes, results) — показывает изменение PR по итерациям.

**Тестовые данные**

* Граф:
  + A → B, C
  + B → C, F
  + C → A
  + D → A, B, E
  + E → F, D
  + F → E, C, B
* Значения α: 0.3, 0.7, 0.85, 0.95.

**Результаты**

**Пример для α=0.85:**

Сошлось на итерации 18

Итоговые PageRank:

A: 0.2623

B: 0.1938

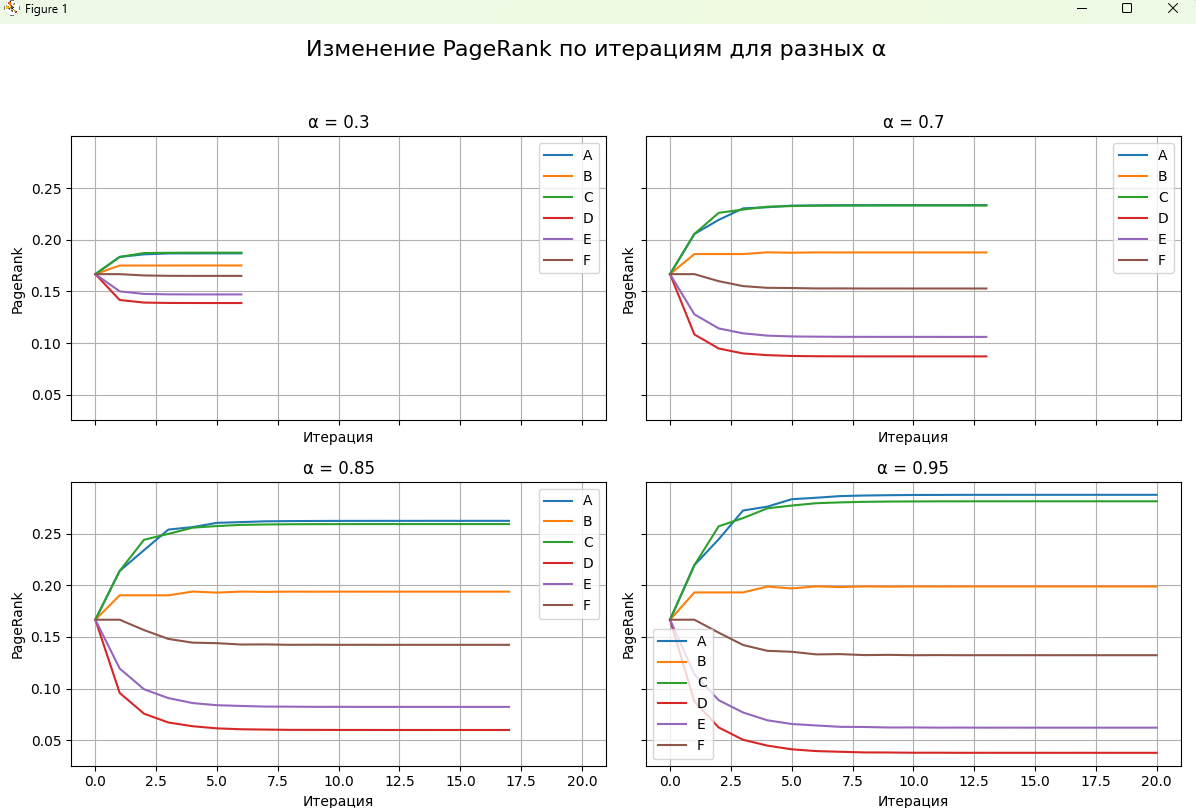
C: 0.2592

D: 0.0600

E: 0.0823

F: 0.1424

Сумма PR: 1.0000



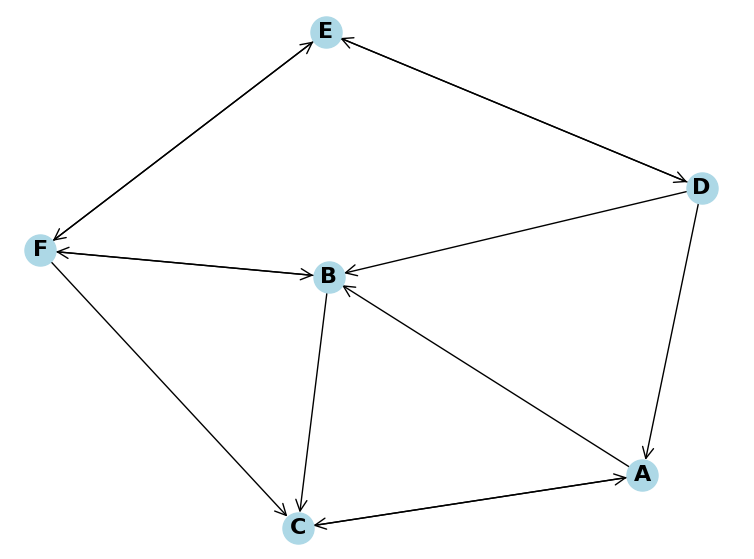


Таблица изменений для α=0.85 (первые 5 итераций):

Итерация | A | B | C | D | E | F

0 | 0.1667| 0.1667| 0.1667| 0.1667| 0.1667| 0.1667

1 | 0.2139| 0.1903| 0.2139| 0.0958| 0.1194| 0.1667

2 | 0.2340| 0.1903| 0.2440| 0.0758| 0.0994| 0.1566

3 | 0.2539| 0.1903| 0.2497| 0.0672| 0.0908| 0.1481

4 | 0.2563| 0.1939| 0.2557| 0.0636| 0.0860| 0.1445

**Анализ результатов**

**Зависимости по весам**

1. **Количество входящих ссылок:**
   * Узлы A (вх. от C, D) и C (вх. от A, B, F) имеют больше входящих ссылок и получают высокий PR (0.2623 и 0.2592 при α=0.85).
   * Узел D (вх. только от E) имеет низкий PR (0.0600), так как зависит от узла с малым весом.
2. **Значимость ссылающихся узлов:**
   * Ссылки от узлов с высоким PR увеличивают вес сильнее. Например, A получает вес от D (низкий PR) и C (высокий PR), но вклад C значительнее.
   * F (вх. от B, E) имеет умеренный PR (0.1424), так как B передает больше веса, чем E.
3. **Влияние α (damping factor):**
   * При α=0.3 (больше случайных переходов) PR распределяется равномернее: A=0.1868, C=0.1874, D=0.1387 (разница мала).
   * При α=0.95 (больше веса ссылкам) различия усиливаются: A=0.2875, D=0.0379 (разрыв увеличивается).
   * Сходимость замедляется с ростом α: 7 итераций при α=0.3 против 21 при α=0.95.

**Производительность и сходимость**

* Алгоритм сходится быстрее при низком α (меньше итераций), так как случайные переходы сглаживают распределение.
* Высокое α требует больше итераций, усиливая влияние структуры графа.

**Визуализация**

* Граф показывает связи, подтверждая высокие веса A и C.
* Графики истории демонстрируют стабилизацию PR, с более резкими изменениями при высоком α.

**Выводы**

1. **PageRank** эффективно оценивает значимость узлов, учитывая как количество входящих ссылок, так и их вес.
2. Узлы с большим числом входящих ссылок (A, C) и ссылками от значимых узлов получают высокий PR.
3. Параметр α определяет баланс между структурой графа и случайностью: низкий α сглаживает веса, высокий α усиливает различия.
4. Алгоритм полезен для анализа сетей, но требует настройки α в зависимости от задачи.

Работа подтверждает зависимость весов от структуры графа и параметра α, демонстрируя их влияние на итоговые результаты.

import numpy as np

import matplotlib.pyplot as plt

import networkx as nx

# 1. Подготовка данных: создание графа

def create\_graph():

    graph = {

        'A': ['B', 'C'],

        'B': ['C', 'F'],

        'C': ['A'],

        'D': ['A', 'B', 'E'],

        'E': ['D', 'F'],

        'F': ['E', 'C', 'B'],

    }

    nodes = list(graph.keys())

    return graph, nodes

# 2. Построение матрицы переходов

def build\_transition\_matrix(graph, nodes, alpha):

    N = len(nodes)

    M = np.zeros((N, N))

    for i, node in enumerate(nodes):

        outgoing = graph.get(node, [])

        num\_outgoing = len(outgoing)

        if num\_outgoing > 0:

            for target in outgoing:

                j = nodes.index(target)

                M[j, i] = 1.0 / num\_outgoing

        else:

            M[:, i] = 1.0 / N

    M = alpha \* M + (1 - alpha) / N \* np.ones((N, N))

    return M

# 3. Реализация PageRank

def compute\_pagerank(M, nodes, max\_iter=100, tol=1e-6):

    N = len(nodes)

    pr = np.ones(N) / N

    history = [pr.copy()]

    for iteration in range(max\_iter):

        pr\_new = M.dot(pr)

        if np.max(np.abs(pr\_new - pr)) < tol:

            print(f"Сошлось на итерации {iteration + 1}")

            break

        pr = pr\_new

        history.append(pr.copy())

    return pr, history

# 4. Эксперименты с разными alpha

def run\_experiments(graph, nodes):

    alphas = [0.3, 0.7, 0.85, 0.95]

    results = {}

    for alpha in alphas:

        print(f"\n=== Эксперимент с alpha = {alpha} ===")

        M = build\_transition\_matrix(graph, nodes, alpha)

        pr, history = compute\_pagerank(M, nodes)

        results[alpha] = {'pr': pr, 'history': history}

        print("Итоговые PageRank:")

        for node, score in zip(nodes, pr):

            print(f"  {node}: {score:.4f}")

        print(f"Сумма PR: {np.sum(pr):.4f}")

    return results

# 5. Визуализация графа

def plot\_graph(graph):

    G = nx.DiGraph()

    for node, targets in graph.items():

        for target in targets:

            G.add\_edge(node, target)

    plt.figure(figsize=(8, 6))

    pos = nx.spring\_layout(G)

    nx.draw(G, pos, with\_labels=True, node\_color='lightblue',

            node\_size=500, font\_size=16, font\_weight='bold',

            arrows=True, arrowstyle='->', arrowsize=20)

    plt.title("Граф связей между страницами")

    plt.show()

# 6. Визуализация PageRank с субплотами

def plot\_history(nodes, results):

    alphas = list(results.keys())

    fig, axes = plt.subplots(2, 2, figsize=(12, 8), sharex=True, sharey=True)

    axes = axes.flatten()

    for idx, alpha in enumerate(alphas):

        history = np.array(results[alpha]['history'])

        ax = axes[idx]

        for i, node in enumerate(nodes):

            ax.plot(history[:, i], label=f"{node}")

        ax.set\_title(f"α = {alpha}")

        ax.set\_xlabel("Итерация")

        ax.set\_ylabel("PageRank")

        ax.grid(True)

        ax.legend()

    plt.suptitle("Изменение PageRank по итерациям для разных α", fontsize=16)

    plt.tight\_layout(rect=[0, 0, 1, 0.95])

    plt.show()

graph, nodes = create\_graph()

results = run\_experiments(graph, nodes)

plot\_history(nodes, results)

alpha = 0.85

print(f"\nТаблица изменений PageRank для α={alpha}:")

history = results[alpha]['history']

print("Итерация | " + " | ".join(nodes))

for i, pr in enumerate(history[:10]):

    print(f"{i:8d} | " + " | ".join(f"{x:.4f}" for x in pr))

plot\_graph(graph)