Department of Economics and Finance

Chair of Gambling: Probability and Decision

PageRank Algorithm: Integrating Markov Chains and Computational Methods in Link Analysis

Supervisor:

Prof.

Hlafo Alfie Mimun

Candidate:

Marijana Pavlovska

273701

Academic Year 2023/2024

To my best friends Andrea, Anja and Tamara,

to my father, Dragan,

to my uncle, Mladen,

and to my mother, Nikoleta:

Thank you for your unconditional support and faith in me.

Contents

Introduction

In today's world, the ability to obtain information with just a single search and a click has become second nature. Whether one uses Google, Mozilla, Internet Explorer, or any other search engine, the results displayed are driven by sophisticated algorithms [4]. In this thesis, Google serves as the primary focus of discussion. For instance, if one searches on Google for a traditional ajvar recipe—a dish popular in the Balkans—the top result is likely from a reputable, well-known page. The algorithm can be understood through the lens of game theory: each web page has a certain probability of appearing at the top, similar to a tree where each branch carries a probability, and each outcome offers a payoff.

The core idea behind Google's PageRank algorithm is that a page's relevance is determined by the number and quality of links pointing to it [4]. The more reputable the links, the higher the payoff, and thus, the higher the page ranks. Since game theory is a branch of applied mathematics, the concepts described here can be analyzed through mathematical and computational methods.

That said, PageRank can be understood through Markov processes [7, 8]. Imagine a large restaurant with numerous tables, where a waiter serves customers. The waiter's movement or path from one table to another can be represented as a probability distribution between 0 and 1, visualized as a graph. If the waiter is at Table 1, there is a positive probability that he will eventually move to Table 2, and from there to Table 3, and so on. The ability of the waiter to move from any table to

any other table reflects the concept of irreducibility. Additionally, the waiter can decide when to return to a table, introducing the concept of aperiodicity.

These ideas are crucial when ranking web pages. Just as a waiter can move between tables with a positive probability, a user can navigate from one web page to another, and sometimes even to an unrelated page. This path resembles a directed graph created by a user "surfing" the internet [4]. These graphs can be visualized using Python, particularly with the NetworkX library [1].

Web surfing can be seen as a series of paths leading to specific results. Understanding the basis of these results and how to predict or influence them requires a deep dive into Markov chains[7, 8] and Python[1, 3]. The topic is particularly intriguing due to its diverse applications across various fields. The broad applicability of PageRank—ranging from ranking search results and social network analysis to understanding protein interactions, academic influence, and even everyday life—has greatly motivated my interest in this subject.

For instance, if a waiter frequents Table 3 more than others, it is likely because that table has influential guests, drawing attention from others. Similarly, a web page with a high PageRank is frequently visited and linked by other reputable sources. In essence, it has the highest probability of being visited, i.e., PageRank [4]. Markov chains and Python provide the tools to visualize, analyze, and investigate these concepts through computational and mathematical approaches. As mentioned, this has various applications, including the Twitter network [3, 5] examined in this thesis, which consists of four chapters, each exploring different aspects of the topic. More specifically:

1. **Chapter 1:** introduction Markov Chains' basics and key properties with examples.

2. **Chapter 2:** explanation using Markov Chains in the PageRank algorithm with practical examples.

3. **Chapter 3:** network theory, Python implementation, and PageRank calculation examples.

4. **Chapter 4:** application to a Twitter dataset, analyzing PageRank and user rankings.

Chapter 1

Markov Chains

1.1 Introductory Definitions and Properties

Definition 1.1. *V is a **state space**, which is a countable or finite set. $V = \{v_1, v_2, v_3, \ldots, v_n\}$, where n is the **number of elements** in V. If V is countable and infinite, then $n = \infty$.*
*An $n \times n$ matrix P is a **stochastic matrix** if the following two properties hold:*

***P1**: For $i, j = 1, \ldots, n$, $p_{ij} \in [0, 1]$.*

***P2**: For $i = 1, \ldots, n$, $\sum_{j=1}^{n} P_{i,j} = 1$.*

*The vector $\pi(0)$ of length n is a **probability distribution** on V if the following two properties hold:*

***D1** : For any $j = 1, \ldots, n$, $\pi_j(0) \in [0, 1]$.*

***D2**: $\sum_j \pi_j(0) = 1$.*

$P_{i,j}$ is the probability of moving from a state indexed by i to a state indexed by j. Finally we define $\pi_i(t) := P(X_t = v_i)$ and hence $\pi(t) := (\pi_1(t), \ldots, \pi_n(t))$ is the probability distribution of X_t.

Notation 1.1. *In the entire chapter, we will write $P^t_{i,j}$ instead of $(P^t)_{i,j}$. So in general, $P^t_{i,j} \neq (P_{i,j})^t$.*

Definition 1.2. *The process $\{X_t\}_{t\in\mathbb{N}}$ is a **Markov chain** on the state space V with **transition matrix** P and **initial probability distribution** $\pi(0)$ if:*

***M1**: The **Markov property** holds, that is, for any $t \geq 1$ and $i = 1, \ldots, n$, we have:*

$$P(X_t = v_i \mid X_1, \ldots, X_{t-1}) = P(X_t = v_i \mid X_{t-1})$$

The Markov property tells us that the future, knowing the present, is independent of the past.

***M2**: For any $i = 1, \ldots, n$, $P(X_0 = v_i) = \pi_i(0)$.*

***M3**: For any $i, j = 1, \ldots, n$ and $t \geq 1$,*

$$P(X_t = v_j \mid X_{t-1} = v_i) = P_{i,j}$$

*The Markov chain is **time-homogeneous** since the matrix P does not depend on the time t.*

Proposition 1.1.1. *If $\{X_t\}$ is a Markov chain with transition matrix P, then*

$$\pi(t) = \pi(t-1) \cdot P \quad \text{for any } t \geq 1.$$

By iterating this formula, we get:

$$\pi(t) = \pi(t-1) \cdot P = \pi(t-2) \cdot P \cdot P = \pi(t-2) \cdot P^2 = \ldots = \pi(0) \cdot P^t.$$

So,

$$\pi(t) = \pi(0) \cdot P^t.$$

Example 1.1. *Consider the graph $G = (V, E)$, where $V = \{v_1, v_2, v_3, v_4\}$ and $E = \{(v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1)\}$. G is the graph, V are the vertices, and E are the edges of the "square graph".*

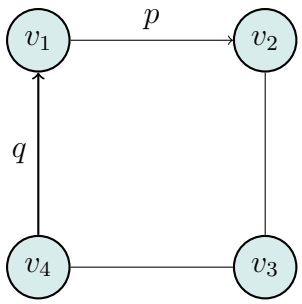


Fig. 1.1: Square Layout of Node Transitions

- *We start at vertex v_1.*
- *At each round, we have the probability p of moving clockwise and the probability q of moving counterclockwise.*

Let X_t be the position at time t. Since we start from vertex v_1, we have $P(X_0 = v_1) = 1$ and $P(X_0 = v_i) = 0$ for $\forall i = 2, 3, 4$. The probability distribution at round t is:

$$\pi(t) = (\pi_1(t), \pi_2(t), \pi_3(t), \pi_4(t)),$$

where, $\pi_i(t) := P(X_t = v_i)$. In our example, the initial probability distribution is:

$$\pi(0) = (1, 0, 0, 0).$$

Note that, $\sum_{i=1}^{4} \pi_i(0) = \sum_{i=1}^{4} P(X_0 = v_i) = 1$. The set V is called the state space, which contains 4 elements. Define P as the 4×4 matrix such that for $i, j = 1, 2, 3, 4$,

$$P_{i,j} = P(X_1 = v_j \mid X_0 = v_i).$$

Hence,

$$P = \begin{pmatrix} 0 & p & 0 & q \\ q & 0 & p & 0 \\ 0 & q & 0 & p \\ p & 0 & q & 0 \end{pmatrix}$$

P is the transition matrix, and it allows us to obtain the probability distribution at time t from the probability distribution at time $t-1$. To find the probability of being in the state j at time 1, we multiply the probability of being in the state i by the probability of moving from i to j for all i, and then sum all of these results over all i's, that is

$$\pi_j(1) = P(X_1 = v_j) = \sum_{i=1}^{4} P(X_1 = v_j \mid X_0 = v_i) P(X_0 = v_i)$$

$$= \sum_{i=1}^{4} P_{i,j}\pi_i(0) = \sum_{i=1}^{4} \pi_i(0) P_{i,j} = (\pi(0) \cdot P)_j.$$

The universal formula, that is for any $t > 0$, for $j = 1, 2, 3, 4$ is:

$$\pi_j(t) = P(X_t = v_j) = \sum_{i=1}^{4} P(X_t = v_j \mid X_{t-1} = v_i) P(X_{t-1} = v_i)$$

$$= \sum_{i=1}^{4} P_{i,j}\pi_i(t-1) = (\pi(t-1) \cdot P)_j.$$

Hence,

$$\pi(t) = \pi(t-1) \cdot P.$$

1.2 Transience and Recurrence

Definition 1.3. *T_j is the the **Hitting time** of the state $v_j \in V$ as the first time $t \geq 1$ in which the chain X_t visits the state v_j (excluding time 0).*

$$T_j := \min\{t \geq 1 \mid X_t = v_j\},$$

with the convention that $\min \emptyset = \infty$.

Definition 1.4. *N_j is the the **number of visits of the state** $v_j \in V$ as the number of times that the chain visits v_j (excluding time 0).*

$$N_j := \sum_{t=1}^{\infty} \mathbf{1}_{\{X_t = v_j\}}.$$

Definition 1.5. *$f_{i,j}$ is the **probability** that, starting from state v_i, the Markov chain will eventually reach state v_j at least once.*

$$f_{i,j} = \mathbb{P}(T_j < \infty \mid X_0 = v_i) = \mathbb{P}(N_j \geq 1 \mid X_0 = v_i).$$

Definition 1.6. *We say that state v_j is*

- ***transient** if $f_{j,j} < 1$;*
- ***recurrent** if $f_{j,j} = 1$.*

Let $\{X_t\}_{t\in\mathbb{N}}$ be a Markov chain defined on the state space $V = \{v_1, \ldots, v_n\}$, where n could be finite or infinite. If $N_j \geq m$ for some $m \geq 1$, it means the chain visits state v_j at least m times. Let $t_0 = 0$ denote the starting time of our chain. There exist times $t_1, t_2, \ldots, t_m$ such that $1 \leq t_1 < t_2 < \ldots < t_m$. Each t_i signifies the moment when the chain reaches state v_j for the i-th time. By definition at these times, we have:

$$X_{t_1} = X_{t_2} = \ldots = X_{t_m} = v_j\,.$$

Moreover, between any consecutive occurrences t_{i-1} and t_i, the chain does not visit v_j, that is:

$$X_t \neq v_j \quad \text{for all } t \in (t_{i-1}, t_i)\,.$$

This property ensures that the Markov chain $\{X_t\}$ precisely hits state v_j at time t_i, and during the interval (t_{i-1}, t_i), it visits in other states.

For $i = 1, 2, \ldots, m$, let us define A_i an event that describes the chain from the time it left state v_j until it reaches v_j again.

$$A_i = \{X_t \neq v_j \text{ for all } t \in (t_{i-1}, t_i),\ X_{t_i} = v_j\}\,.$$

Alternatively,

$$A_i = \{X_{t_{i-1}+1} \neq v_j,\ X_{t_{i-1}+2} \neq v_j,\ \ldots,\ X_{t_i-1} \neq v_j,\ X_{t_i} = v_j\}$$

Note that the event $\{N_j \geq m\}$ occurs if there exists a sequence of times $t_1, \ldots, t_m$ such that $A_1, \ldots, A_m$ occur, that is $\{N_j \geq m\} = \bigcap_{i=1}^{m} A_i$. Hence,

$$P(N_j \geq m \mid X_0 = v_i) = P\left(\bigcap_{i=1}^{m} A_i \mid X_0 = v_i\right)$$

is the probability of hitting v_j at least m times given that $X_0 = v_i$. Note that

$$P\left(\bigcap_{i=1}^{m} A_i \mid X_0 = v_i\right) = \prod_{k=1}^{m} P\left(A_k \mid \bigcap_{i=1}^{k-1} A_i \cap \{X_0 = v_i\}\right).$$

This expression uses the multiplication rule, which states that the joint probability of a sequence of events can be computed by multiplying the conditional probabilities of each event given the previous ones. Moreover, we have:

$$\prod_{k=1}^{m} P\left(A_k \mid \bigcap_{i=1}^{k-1} A_i \cap \{X_0 = v_i\}\right) = \prod_{k=2}^{m} P\left(A_k \mid X_{t_{k-1}} = v_j\right) \cdot P(A_1 \mid X_0 = v_i).$$

The above formula follows from the Markov property. Indeed, the Markov property ensures that the probability of transitioning to the next state depends solely on the current state. Since each event A_k concludes when the chain reaches v_j, we only need to consider the current state v_j. Moreover, due to the time-homogeneous property, we have:

$$\begin{aligned}\prod_{k=2}^{m} P(A_k \mid X_{t_{k-1}} = v_j) \cdot P(A_1 \mid X_0 = v_i) &= P(A_1 \mid X_0 = v_j)^{m-1} \cdot P(A_1 \mid X_0 = v_i) \\ &= (f_{j,j})^{m-1} \cdot f_{i,j}.\end{aligned}$$

Indeed, the time-homogeneous property states that the probability of transitioning between states is independent of the specific time step. The f terms were derived using their definitions. Summarizing all the passages, we have:

$$P(N_j \geq m \mid X_0 = v_i) = (f_{j,j})^{m-1} \cdot f_{i,j}.$$

Let us now look at the expected number of state visits v_j given that the initial state is v_i, that is, $\mathbb{E}[N_j \mid X_0 = v_i]$. This expectation can be expressed as

$$\mathbb{E}[N_j \mid X_0 = v_i] = \sum_{m=1}^{\infty} \mathbb{P}(N_j \geq m \mid X_0 = v_i) = \sum_{m=1}^{\infty} (f_{jj})^{m-1} \cdot f_{ij}.$$

Let $k = m - 1$; then the above summation can be rewritten as

$$\sum_{k=0}^{\infty} (f_{jj})^k \cdot f_{ij}.$$

Using the geometric series identity

$$\sum_{k=0}^{\infty} a^k = \begin{cases} \frac{1}{1-a}, & \text{if } |a| < 1, \\ \infty, & \text{if } |a| \geq 1\,, \end{cases}$$

we derive

$$\mathbb{E}[N_j \mid X_0 = v_i] = \sum_{k=0}^{\infty} (f_{jj})^k \cdot f_{ij} = \begin{cases} \frac{f_{ij}}{1-f_{jj}}, & \text{if } v_j \text{ is transient,} \\ \infty, & \text{if } v_j \text{ is recurrent.} \end{cases}$$

This expression gives the expected number of state visits v_j starting from state v_i, considering whether v_j is transient or recurrent under the Markov chain dynamics.

Definition 1.7. *The sequence $\{X_t\}_{t\in\mathbb{N}}$ is an irreducible Markov chain if $f_{i,j} > 0$ for all $v_i, v_j \in S$. This means that for any two states in the state space S, there is a positive probability of transitioning from one state to the other. In terms of the associated graph G, this implies that there is a path connecting any pair of vertices. Therefore, saying that G is an irreducible graph is equivalent to stating that P is an irreducible matrix.*

Proposition 1.2.1. *The state v_j is*

- *transient if $\sum_{m=1}^{\infty} P_{j,j}^m < \infty$,*
- *recurrent if $\sum_{m=1}^{\infty} P_{j,j}^m = \infty$.*

Proof. We have

$$\begin{aligned}\mathbb{E}[N_j \mid X_0 = v_j] &= \mathbb{E}\left[\sum_{m=1}^{\infty} 1\{X_m = v_j\} \mid X_0 = v_j\right] \\ &= \sum_{m=1}^{\infty} \mathbb{P}(X_m = v_j \mid X_0 = v_j) \\ &= \sum_{m=1}^{\infty} P_{j,j}^m.\end{aligned}$$

Since

$$\mathbb{E}[N_j \mid X_0 = v_j] = \begin{cases} \infty, & \text{if } v_j \text{ is transient,} \\ \infty, & \text{if } v_j \text{ is recurrent,} \end{cases}$$

we have established the thesis. □

Proposition 1.2.2. *If the state v_i is recurrent and $f_{ij} > 0$, then the state v_j is also recurrent and $f_{ji} = 1$.*

Proposition 1.2.3. *If $\{X_t\}_{t\in\mathbb{N}}$ is an irreducible Markov chain, then all states are either transient or recurrent.*

Proposition 1.2.4. *An irreducible recurrent Markov chain visits every state infinitely often. This means that for each state v_j in the state space V, the chain will return to v_j an infinite number of times. Formally, this can be expressed as:*

$$\mathbb{P}\left(\bigcap_{m=1}^{\infty} \bigcup_{k\geq m} \{X_k = v_j\}\right) = 1 \quad \forall v_j \in V$$

This expression indicates that the probability of visiting state v_j infinitely often is 1 for every state v_j.

Proposition 1.2.5. *An irreducible transient Markov chain visits no state infinitely often. This means that for each state v_j in the state space V, the chain will only visit v_j a finite number of times. Formally, this can be expressed as:*

$$\mathbb{P}\left(\bigcap_{m=1}^{\infty} \bigcup_{k\geq m} \{X_k = v_j\}\right) = 0 \quad \forall v_j \in V$$

This expression indicates that the probability of visiting state v_j infinitely often is 0 for every state v_j.

1.3 Simple Random Walk on $\mathbb{Z}$

Suppose a player is betting on a roulette game where each spin can result in either winning or losing \$1. The probabilities are given by

$$P(\text{Winning the bet}) = p, \quad P(\text{Losing the bet}) = q\,.$$

Assume that

- the player starts with an initial balance $X_0 = 0$.
- with each spin, the player either wins or loses.
- X_t is the player's balance after the t-th spin.
- $\{X_t\}_{t\in\mathbb{N}}$ is a Markov chain on state space $\mathbb{Z}$ since it satisfies the Markov property. Knowing X_{t-1}, we can derive X_t as

$$X_t = \begin{cases} X_{t-1} + 1, & \text{with probability } p, \\ X_{t-1} - 1, & \text{with probability } q. \end{cases}$$

  For any $t \in \mathbb{N}$ and $j \in \mathbb{Z}$

$$\mathbb{P}(X_t = j \mid X_{t-1}, \ldots, X_1) = \mathbb{P}(X_t = j \mid X_{t-1})\,.$$

The state space is countable, the transition matrix P and the initial probability distribution has infinite dimensions. The transition matrix associated with $\{X_t\}_{t\in\mathbb{N}}$ is the matrix P such that for $i, j \in \mathbb{Z}$

$$P_{i,j} := \mathbb{P}(X_t = j \mid X_{t-1} = i) = \begin{cases} p, & \text{if } j = i + 1, \\ q, & \text{if } j = i - 1, \\ 0, & \text{if } j \neq \{i - 1, i + 1\}. \end{cases}$$

Hence, for example, we have $P(X_1 = 1) = p$, $P(X_1 = -1) = q$ and

$$P(X_2 = j) = P_{1,j} \cdot P(X_1 = 1) + P_{-1,j} \cdot P(X_1 = -1) = P(X_2 = j) =$$
$$= \begin{cases} p \cdot p + 0 \cdot q = p^2, & \text{if } j = 2, \\ q \cdot p + p \cdot q = 2pq, & \text{if } j = 0, \\ 0 \cdot p + q \cdot q = q^2, & \text{if } j = -2, \\ 0 \cdot p + 0 \cdot q = 0, & \text{if } j = \pm 1. \end{cases}$$

The graph G associated with the Markov chain $\{X_t\}_{t \in \mathbb{N}}$ has vertices representing all possible states, which are elements of $\mathbb{Z}$, the set of all integers. The edges of G are defined by pairs $(i, i+1)$ for each integer $i \in \mathbb{Z}$. This structure reflects the transitions where each state i can transition to $i+1$. The irreducibility of G implies that there is always a path between any two states in $\mathbb{Z}$.

Note that $P_{0,0}^m$ is the probability that, starting at 0, the chain visits 0 at time m. This probability can be different from 0 if and only if m is an even number. So,

$$P_{0,0}^m = \begin{cases} > 0, & \text{if } m \text{ is even;} \\ = 0, & \text{if } m \text{ is odd.} \end{cases}$$

For this reason, let us write $m = 2k$ for $k \in \mathbb{N}$. To return to $X_{2k} = 0$, the chain must take k steps to the right and k steps to the left. Combinatorially, this involves finding all possible rearrangements of k wins (W) and k losses (L), where each arrangement corresponds to a feasible chain:

$$P_{0,0}^{(2k)} = \binom{2k}{k} p^k q^k.$$

Here, $\binom{2k}{k}$ represents the binomial coefficient, which counts the number of ways to choose k steps (either W or L) from $2k$ total steps, and $p^k q^k$ gives the probability associated with each specific arrangement (where p and q are the probabilities of

winning and losing, respectively). When k is large, we can use Stirling's formula to asymptotically approximate $k!$, yielding:

$$k! \sim \sqrt{2\pi k}\left(\frac{k}{e}\right)^k.$$

Thus, by Stirling's formula for large k, we have:

$$\binom{2k}{k} = \frac{(2k)!}{(k!)^2} \sim \frac{\sqrt{4\pi k}\left(\frac{2k}{e}\right)^{2k}}{2\pi k\left(\frac{k}{e}\right)^{2k}} = \frac{2^{2k}}{\sqrt{\pi k}} \sim 2^{2k} = 4^k.$$

Recalling the geometric series:

$$\sum_{k=1}^{\infty} a_k = \begin{cases} \frac{1}{1-a}, & \text{if } a \in (0,1), \\ \infty, & \text{if } a \geq 1, \end{cases}$$

we have

$$\begin{aligned} \sum_{m=1}^{\infty} P_{0,0}^m &= \sum_{k=1}^{\infty} P_{0,0}^{2k} \\ &= \sum_{k=1}^{\infty} \binom{2k}{k}(pq)^k \\ &\approx \sum_{k=1}^{\infty} 4^k(pq)^k = \sum_{k=1}^{\infty}(4pq)^k \\ &= \begin{cases} \frac{1}{1-4pq}, & \text{if } 4pq < 1, \\ \infty, & \text{if } 4pq \geq 1. \end{cases} \end{aligned}$$

Let us define $g(p) = pq = p(1-p) = p - p^2$ and note that $g(p) < \frac{1}{4}$ if $p \in [0,1]\setminus\left\{\frac{1}{2}\right\}$, and $g\left(\frac{1}{2}\right) = \frac{1}{4}$. Thus, we conclude that the Markov chain $\{X_t\}_{t\in\mathbb{N}}$ is recurrent if $p = \frac{1}{2}$; otherwise, it is transient.

In probability theory, the process $\{X_t\}_{t\in\mathbb{N}}$ described here is known as a simple random walk on $\mathbb{Z}$. When $p = q$, it is specifically referred to as a symmetric simple random walk on $\mathbb{Z}$.

1.4 Stationary Distribution

Let V be the state space of a Markov chain $\{X_t\}_{t\in\mathbb{N}}$ with transition matrix P.

Definition 1.8. *A probability distribution ρ on the state space V is called a stationary distribution (or invariant distribution) for the matrix P (or equivalently, for the Markov chain $\{X_t\}_{t\in\mathbb{N}}$ with transition matrix P) if $\rho = \rho \cdot P$.*

Example 1.2. *Let us find the stationary distribution ρ for the Markov chain from Example 1.1:*

$$\begin{pmatrix}\rho_1 & \rho_2 & \rho_3 & \rho_4\end{pmatrix} \cdot \begin{pmatrix} 0 & p & 0 & q \\ q & 0 & p & 0 \\ 0 & q & 0 & p \\ p & 0 & q & 0 \end{pmatrix} = \begin{pmatrix}\rho_1 & \rho_2 & \rho_3 & \rho_4\end{pmatrix}$$

The solutions for the system are proportional to the vector $(1,1,1,1)$, so the final solution will be $\rho = (c,c,c,c)$ with $c \in \mathbb{R}$. Being a probability distribution, ρ has to satisfy the following conditions:

- $\rho_i \in [0,1]$ *for all i;*
- $\sum_i \rho_i = 1$.

Thus, we need $c = \frac{1}{4}$, and hence $\rho = \left(\frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}\right)$.

1.5 Period of a State and Aperiodic Markov Chains

Given a set of numbers $A = \{x_1, \ldots, x_r\}$, we denote by $\gcd(A)$ the greatest common divisor of the set A. For example, $\gcd(7, 42, 700) = 7$.

Definition 1.9. *Given a state $v_i \in S$, we define the* ***period of the state*** *v_i as*

$$d(i) := \gcd(D(i)),$$

where

$$D(i) := \{t \in \mathbb{N} \setminus \{0\} \mid P_{ii}^t > 0\}.$$

The set $D(i)$ consists of the indices corresponding to the rounds in which the chain returns to v_i, and $d(i)$ is the greatest common divisor of these indices.

Example 1.3. *Consider the problem in Section 1.3. For all v_i,*

$$D(i) = \{2, 4, 6, 8, \ldots\} = \{2k \mid k \in \mathbb{N}^+\} \Rightarrow d(i) = 2.$$

Proposition 1.5.1. *If $v_i, v_j \in S$ are such that $f_{i,j} > 0$ and $f_{j,i} > 0$, then $d(i) = d(j)$. In particular, in an irreducible Markov chain, all the states have the same period, denoted as $d(i)$, and hence we can speak about the period of the Markov chain.*

Definition 1.10. *If an irreducible Markov chain has a period of state 1, we say that the Markov chain is aperiodic.*

Proposition 1.5.2. *If an irreducible aperiodic Markov chain in S with transition matrix P has a stationary distribution ρ, then it is recurrent, and for any $v_i, v_j \in S$ we have*

$$\lim_{t\to\infty} P_{i,j}^t = \rho_j.$$

Furthermore, $\rho_j > 0$ for all $v_j \in S$.

Proposition 1.5.3. *If the Markov chain does not possess a stationary distribution, then for any $v_i, v_j \in S$, it holds that*

$$\lim_{t\to\infty} P_{i,j}^t = 0.$$

Proposition 1.5.4. *An irreducible Markov chain in a finite state space S has a stationary distribution.*

Proof. Let P be the transition matrix of the Markov chain, and suppose that a stationary distribution does not exist. Note that, being P a transition matrix, we have $\sum_{v_j \in S} P_{i,j}^t = 1$. According to Proposition 1.5.3, we have $\lim_{t\to\infty} P_{i,j}^t = 0$. Hence,

$$\sum_{v_j \in S} \lim_{t\to\infty} P_{i,j}^t = 0.$$

Being S finite, the summation over S is finite, and hence we can exchange the limit and summation symbols. So we have

$$0 = \sum_{v_j \in S} \lim_{t\to\infty} P_{i,j}^t = \lim_{t\to\infty} \sum_{v_j \in S} P_{i,j}^t = \lim_{t\to\infty} 1 = 1,$$

which contradicts the hypothesis. Thus, we conclude that the stationary distribution exists. □

Proposition 1.5.5. *Let $\{X_t\}_{t\in\mathbb{N}}$ be an irreducible, aperiodic, recurrent Markov chain in S with transition matrix P. Then one of the following conclusions holds:*

(a) $\mathbb{E}[T_i \mid X_0 = v_i] < \infty$ for all $v_i \in S$, and P has a unique stationary distribution ρ given by

$$\rho_i = \frac{1}{\mathbb{E}[T_i \mid X_0 = v_i]}$$

*for $v_i \in S$. In this case, the chain is said to be **positive recurrent**, and Proposition 1.5.2 holds.*

*(b) $\mathbb{E}[T_i \mid X_0 = v_i] = \infty$ for all $v_i \in S$, and P has no stationary distribution. In this case, the chain is said to be **null recurrent**, and Proposition 1.5.3 holds.*

Example 1.4. *Consider the Markov chain on the state space $S = \{v_1, v_2\}$ with fixed $\alpha, \beta \in (0, 1)$.*

$$P = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}.$$

- *The associated graph is* ***irreducible*** *because we can get to any index from any index.*

- *Recurrence of Irreducible Markov Chains: If the transition matrix P of a Markov chain is irreducible and satisfies $P_{i,j} > 0$ for all i, j, then the chain is* ***recurrent****.*

- *For the state v_1 of a Markov chain, the set $D(1)$ comprises all $t \in \mathbb{N}\backslash\{0\}$ such that $P^t_{1,1} > 0$. Given that $1 \in D(1)$, it follows that $d(1) = \gcd(D(1)) = 1$. Since the Markov chain is irreducible, Proposition 1.5.1 implies that $d(2) = d(1) = 1$, hence confirming that the Markov chain is* ***aperiodic****.*

Let us look for the stationary distribution of such a Markov chain. We have

$$\begin{pmatrix} \rho_1 & \rho_2 \end{pmatrix} \cdot \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix} = \begin{pmatrix} \rho_1 & \rho_2 \end{pmatrix}.$$

So we need to solve the system of equations:

$$\begin{cases} (1-\alpha)\rho_1 + \beta\rho_2 = \rho_1, \\ \alpha\rho_1 + (1-\beta)\rho_2 = \rho_2, \\ \rho_1 + \rho_2 = 1. \end{cases}$$

To solve this system, we have included the conditions $\rho_1 + \rho_2 = 1$ and $\rho_1, \rho_2 \in [0, 1]$ (since we want ρ to be a probability distribution). Hence, we obtain

$$\begin{cases} \rho_1 = \frac{\beta}{\alpha+\beta}, \\ \rho_2 = \frac{\alpha}{\alpha+\beta}. \end{cases}$$

Since the Markov chain is irreducible, aperiodic, and recurrent, by Proposition 1.5.5, the Markov chain is positive recurrent. According to Proposition 1.5.5, we have

$$E[T_1 \mid X_0 = v_1] = \frac{1}{\rho_1} = \frac{\alpha+\beta}{\beta}, \quad E[T_2 \mid X_0 = v_2] = \frac{1}{\rho_2} = \frac{\alpha+\beta}{\alpha}.$$

1.6 Time Reversal and Reversible Markov Chains

Definition 1.11. *A stochastic matrix P and a probability distribution λ are said to be in* ***detailed balance*** *if*

$$\lambda_i P_{i,j} = \lambda_j P_{j,i} \quad \textit{for all } i, j.$$

Proposition 1.6.1. *If the stochastic matrix P and the probability distribution λ are in detailed balance, then λ is a stationary distribution for P.*

Proof. To be proven: $\lambda P = \lambda$. Since λ and P are in detailed balance, we have

$$(\lambda P)_i = \sum_j \lambda_j P_{j,i} = \sum_j \lambda_i P_{i,j} = \lambda_i \sum_j P_{i,j} = \lambda_i.$$

Therefore, because $(\lambda P)_i = \lambda_i$, there is a full probability distribution λ

$$\lambda P = \lambda.$$

□

To determine a probability distribution λ that is in detailed balance with a transition matrix P, the following steps can be followed:

1. Check for the existence of a stationary distribution for P by solving the system $\rho \cdot P = \rho$.

2. If the system has no solutions, then no probability distribution λ exists that is in detailed balance with P.

3. If the system has a solution ρ, verify whether ρ satisfies the detailed balance condition with P.

Specifically, if P is symmetric (i.e., $P_{i,j} = P_{j,i}$), the unique probability distribution ρ that is in detailed balance with P is the uniform distribution over the state space (i.e., $\rho_i = \rho_j$ for all i, j). This is demonstrated as follows. For any i, j,

$$\lambda_i P_{i,j} = \lambda_j P_{j,i} \Rightarrow \lambda_i P_{i,j} = \lambda_j P_{i,j} \Rightarrow \lambda_i = \lambda_j.$$

Thus, the uniform distribution is the unique distribution in detailed balance with a symmetric transition matrix P.

Proposition 1.6.2. *Let P be irreducible and have a stationary distribution ρ. Fix $T \geq 1$. Suppose that $\{X_n\}_{0 \leq n \leq T}$ is a Markov chain with initial probability distribution ρ and transition matrix P. Define $Y_0 = X_T$ and $Y_n = X_{T-n}$ for $n \geq 1$. Then the process $\{Y_n\}_{0 \leq n \leq T}$ is a Markov chain with initial distribution ρ and transition matrix $\hat{P}$, where the entries of $\hat{P}$ satisfy the equations*

$$\rho_j \hat{P}_{j,i} = \rho_i P_{i,j} \quad \text{for all } i, j.$$

*Moreover, $\hat{P}$ is also irreducible with stationary distribution ρ. The chain $\{Y_n\}_{0 \leq n \leq T}$ is called the **time-reversal** of $\{X_n\}_{0 \leq n \leq T}$.*

Definition 1.12. *Let $\{X_n\}_{n \geq 0}$ be a Markov chain with initial distribution ρ and transition matrix P. If the Markov chain is irreducible, we say it is **reversible** if, for every $T \geq 1$, the sequence $\{X_{T-n}\}_{0 \leq n \leq T}$ also forms a Markov chain with initial distribution ρ and transition matrix P.*

Proposition 1.6.3. *Let P be an irreducible stochastic matrix and let ρ be a probability distribution. Suppose $\{X_n\}_{n \geq 0}$ is a Markov chain with initial distribution ρ and transition matrix P. The following statements are equivalent:*

- *$\{X_n\}_{n \geq 0}$ is reversible.*
- *P and ρ satisfy the detailed balance condition.*

Example 1.5. *Consider the Markov chain on the state space* $S = \{v_1, v_2, v_3\}$ *with transition matrix*

$$P = \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{pmatrix}.$$

We want to determine if the Markov chain is reversible. Since the associated graph is irreducible, it is sufficient to demonstrate the existence of a probability distribution $\lambda = (\lambda_1, \lambda_2, \lambda_3)$ *that satisfies the detailed balance condition with* P*. To find such a probability distribution, we need first to find the stationary distributions of* P *and then identify which are in detailed balance with* P*. Let us determine if* P *has a stationary distribution by finding the solution* $\rho = (\rho_1, \rho_2, \rho_3)$ *of*

$$\begin{pmatrix} \rho_1 & \rho_2 & \rho_3 \end{pmatrix} \begin{pmatrix} 0 & \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & 0 & \frac{2}{3} \\ \frac{2}{3} & \frac{1}{3} & 0 \end{pmatrix} = \begin{pmatrix} \rho_1 & \rho_2 & \rho_3 \end{pmatrix}.$$

This gives the system of equations:

$$\begin{cases} \rho_1 \cdot 0 + \rho_2 \cdot \frac{1}{3} + \rho_3 \cdot \frac{2}{3} = \rho_1 \\ \rho_1 \cdot \frac{2}{3} + \rho_2 \cdot 0 + \rho_3 \cdot \frac{1}{3} = \rho_2 \\ \rho_1 \cdot \frac{1}{3} + \rho_2 \cdot \frac{2}{3} + \rho_3 \cdot 0 = \rho_3 \\ \rho_1 + \rho_2 + \rho_3 = 1 \end{cases}$$

This system has the solution $\rho = (c, c, c)$ *for all* $c \in \mathbb{R}$*. We choose* $c \in \mathbb{R}$ *such that* $\sum_{i=1}^{3} \rho_i = 1$ *and* $\rho_i \in [0, 1]$ *for* $i = 1, 2, 3$*. Hence,*

$$1 = \sum_{i=1}^{3} \rho_i = \sum_{i=1}^{3} c = 3c \Rightarrow c = \frac{1}{3}.$$

So, $\rho = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ *is a stationary distribution for* P*, and it is unique since it is the only solution of the system that is also a probability distribution. To verify if*

$\rho = \left(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right)$ *is in detailed balance with* P*, we check if* $\lambda = \rho$ *satisfies the system of equations:*

$$\begin{cases} \lambda_1 P_{1,2} = P_{2,1}\lambda_2 \\ \lambda_1 P_{1,3} = P_{3,1}\lambda_3 \\ \lambda_2 P_{2,3} = P_{3,2}\lambda_3 \end{cases}$$

Examining each equation: For $\lambda_1 P_{1,2} = P_{2,1}\lambda_2$*:*

$$\frac{1}{3} \cdot \frac{2}{3} = \frac{1}{3} \cdot \frac{1}{3}$$

Simplifying gives $\frac{2}{9} = \frac{1}{9}$*, which is false. Hence, we conclude that* ρ *is not in detailed balance with* P*. Therefore, the Markov chain defined by the transition matrix* P *and the stationary distribution* ρ *is not reversible.*

1.7 Ergodic Theorem

Definition 1.13. *Denote by* $Y_i(n)$ *the number of visits to the state* v_i *before time* n*, that is,*

$$Y_i(n) = \sum_{k=0}^{n-1} 1_{\{X_k = v_i\}}.$$

Then $\frac{Y_i(n)}{n}$ *represents the proportion of time spent in state* v_i *before time* n*.*

Theorem 1.7.1 (Ergodic Theorem)**.** *Let* P *be irreducible and let* λ *be any distribution. Let* $\{X_n\}_{n\in\mathbb{N}}$ *be a Markov chain on the state space* S *with initial distribution* λ *and transition matrix* P*. Then*

$$\mathbb{P}\left(\frac{Y_i(n)}{n} \underset{n\to\infty}{\longrightarrow} \frac{1}{\mathbb{E}[T_i \mid X_0 = v_i]}\right) = 1.$$

Moreover, if the Markov chain is positive recurrent, for any bounded function $f : S \to \mathbb{R}$*, we have:*

$$\mathbb{P}\left(\frac{1}{n}\sum_{k=0}^{n-1} f(X_k) \underset{n\to\infty}{\longrightarrow} \sum_{v_i \in S} \rho_i f(v_i)\right) = 1,$$

where ρ *is the unique stationary distribution of* P*.*

In an irreducible and positive recurrent Markov chain, the average value of a function f over all states converges over time to a weighted sum. Each state's contribution to this average is weighted by how often the chain visits that state, as given by the stationary distribution ρ. This property highlights the long-term behaviour and stability of the chain's dynamics.

Example 1.6. *Consider Example 1.4 with the transition matrix*

$$P = \begin{pmatrix} 1-\alpha & \alpha \\ \beta & 1-\beta \end{pmatrix}.$$

We have established that this Markov chain is irreducible and positive recurrent. Furthermore, its stationary distribution is given by

$$\rho = \left(\frac{\beta}{\alpha+\beta}, \frac{\alpha}{\alpha+\beta} \right).$$

Let f be defined such that $f(v_1) = 1$ and $f(v_2) = -1$. According to the Ergodic Theorem (Theorem 1.7.1), we have

$$\frac{1}{n}\sum_{k=0}^{n-1} f(X_k) \underset{n\to\infty}{\longrightarrow} \rho_2 \cdot f(v_2) + \rho_1 \cdot f(v_1) = \frac{\beta}{\alpha+\beta}\cdot(-1) + \frac{\alpha}{\alpha+\beta}\cdot 1 = \frac{\alpha-\beta}{\alpha+\beta},$$

with probability 1.

1.8 A Practical Example

Example 1.7. *Consider the matrix*

$$P = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

Consider the Markov chain on the state space $S = \{v_1, v_2, v_3\}$ with transition matrix P, that is for $t \geq 1$

$$P(X_t = v_j \mid X_{t-1} = v_i) = P_{ij}.$$

(a) Compute the 2-step transition matrix.

(b) Is the Markov chain irreducible?

(c) Compute the period of the state v_1;

(d) Is the Markov chain aperiodic?

(e) Find a stationary distribution for the Markov chain.

(f) Is the Markov chain recurrent?

(g) Is the Markov chain positive recurrent? In the affirmative case compute $\mathbb{E}[T_1 \mid X_0 = v_1]$, where $T_1 = \min\{t \geq 1 \mid X_t = v_1\}$.

(h) Is the Markov chain reversible?

(i) Consider the function $f : S \to \mathbb{R}$ such that $f(v_1) = 3$, $f(v_3) = -3$, $f(v_2) = 0$. Use the Ergodic Theorem to compute the almost sure limit of $\frac{1}{n}\sum_{k=0}^{n-1} f(X_k)$ as $n \to +\infty$.

Solution:

The associated graph is drawn:

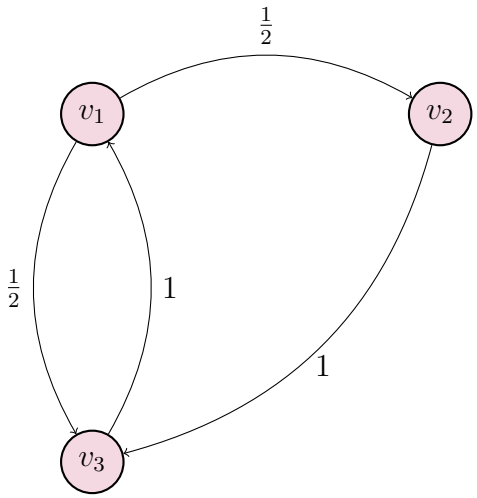


Fig. 1.2: Markov chain graph

(a)

$$P^2 = \begin{bmatrix} 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}^2 = \begin{bmatrix} 1/2 & 0 & 1/2 \\ 1 & 0 & 0 \\ 0 & 1/2 & 1/2 \end{bmatrix}$$

(b) The Markov chain is irreducible since for any $i, j \in \{1, 2, 3\}$, with $i \neq j$, there exists a path that starts at v_i and arrives at v_j.

(c) The period of v_1 is defined as $d(1) = gcd(D(1))$, where

$$D(1) = \{n \in \mathbb{N} \mid P_{11}^n > 0\}.$$

By exercise (3a) we know that $P_{1,1}^2 > 0$ and hence $2 \in D(1)$. Since

$P_{1,1}^3 = P(X_3 = v_1 \mid X_0 = v_1) \geq P(X_3 = v_1, X_2 = v_3, X_1 = v_2 \mid X_0 = v_1) = \frac{1}{2} \cdot 1 \cdot \frac{1}{2} > 0$,

we have also that $3 \in D(1)$. Equivalently, we can show that $P_{1,1}^3 > 0$ by computing P^3:

$$P^3 = P^2 \cdot P$$

$=$

$$\begin{bmatrix} 1/2 & 0 & 1/2 \\ 1 & 0 & \frac{1}{2} \\ 0 & 1/2 & 1/2 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 1/2 & 1/4 & 1/4 \\ 0 & 1/2 & 1/2 \\ 1/2 & 0 & 1/2 \end{bmatrix}$$

Since $P_{1,1}^3 > 0$, and $\{2, 3\} \subseteq D(1)$, we have $gcd(D(1)) = 1$ and hence $d(1) = 1$.

(d) Since the Markov chain is irreducible, all the states have the same period. So $1 = d(1) = d(2) = d(3)$ and hence the Markov chain is aperiodic.

(e) We have to find $\rho = (\rho_1, \rho_2, \rho_3)$ that solves the system

$$\begin{bmatrix} \rho_1 & \rho_2 & \rho_3 \end{bmatrix} \cdot \begin{bmatrix} 0 & 1/2 & 1/2 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix} = \begin{bmatrix} \rho_1 & \rho_2 & \rho_3 \end{bmatrix}$$

that is

$$\begin{cases} \rho_3 = \rho_1, \\ \frac{1}{2}\rho_1 + \rho_2 = \rho_3, \\ \frac{1}{2}\rho_1 + \rho_2 = \rho_3, \end{cases} \quad \Rightarrow \quad \begin{cases} \rho_1 = c \in \mathbb{R}, \\ \rho_3 = \rho_1 = c, \\ \rho_2 = \frac{1}{2}\rho_1 = \frac{1}{2}c. \end{cases}$$

That gives the solution

$$\rho = \begin{bmatrix} c & \frac{c}{2} & c \end{bmatrix} \quad \text{for } c \in \mathbb{R}.$$

To find the value of c, we have to impose the conditions $\rho_i \in [0,1]$ for $i = 1,2,3$ and $\sum_{i=1}^{3} \rho_i = 1$ that make ρ a probability distribution. So

$$\rho_i \in [0,1] \quad \Leftrightarrow \quad c \in [0,1],$$

$$1 = \sum_{i=1}^{3} \rho_i = \frac{5}{2}c \quad \Rightarrow \quad c = \frac{2}{5}.$$

So $\rho = \left(\frac{2}{5}, \frac{1}{5}, \frac{2}{5}\right)$ is a probability stationary distribution for the Markov chain. Since ρ is also the unique probability distribution that satisfies the system $\rho \cdot P = \rho$, we have also that ρ is the unique stationary distribution of the Markov chain.

(f) The Markov chain is irreducible, aperiodic and has a stationary distribution. It follows that the Markov chain is also recurrent.

(g) The Markov chain has a unique stationary distribution, it is irreducible, aperiodic and recurrent. It follows that the Markov chain is also positive recurrent and

$$\mathbb{E}[T_1 \mid X_0 = v_1] = \frac{1}{\rho_1} = \frac{1}{\frac{2}{5}} = \frac{5}{2}.$$

(h) The Markov chain is reversible if and only if the stationary distribution ρ is in detailed balance with P, that is $\rho_i P_{ij} = \rho_j P_{ji}$ for any $i,j = 1,2,3$. Since

$$\rho_1 P_{12} = \frac{2}{5} \cdot \frac{1}{2} = \frac{1}{5} \neq 0 = \rho_2 P_{21},$$

we have that P and ρ are not in detailed balance and hence the Markov chain is not reversible.

(i) Since the Markov chain is irreducible and has stationary distribution $\rho = \left(\frac{2}{5}, \frac{1}{5}, \frac{2}{5}\right)$, by the Ergodic Theorem we have that almost surely

$$\lim_{n\to\infty} \frac{1}{n} \sum_{k=0}^{n-1} f(X_k) = \sum_{i=1}^{3} \rho_i f(v_i) = 3 \cdot \rho_1 - 3 \cdot \rho_3 = 0.$$

Chapter 2

Markov Chains in PageRank Algorithm

2.1 Introduction

Link analysis algorithms are critical to the success of web search engines as they evaluate the importance and relevance of individual web pages. Notable examples include HITS (Hyperlink Induced Topic Search), PageRank, and SALSA (Stochastic Approach for Link Structure Analysis). These algorithms utilize the link structure of web pages to optimize search accuracy and relevance. HITS, developed by Jon Kleinberg, is a query-dependent (meaning its results can vary based on the specific search query) algorithm that calculates the authority (a measure of how trustworthy a page is) and hub values (a measure of how well a page links to other important pages) of a page. The SALSA algorithm combines the random walk feature of PageRank with the hub and authority concept from HITS. After Google was founded in 1999, it quickly became the leader in Internet search engines with the introduction of the PageRank algorithm, which changed how search results were determined.

2.2 The PageRank Algorithm

PageRank is the most popular link-based ranking algorithm. When Google entered the search engine market it quickly became highly efficient due to its query-independent and content-independent nature.

(a) Query-independent: This means the search engine does not need to analyze a specific query (word or phrase typed into a search engine to find information) to rank the pages. It ranks pages based on their importance, not just the search term used.

(b) Content-independent: This means the search engine does not look at the actual content of the pages to rank them. Instead, it focuses on the links between pages to decide their importance.

It operates faster by downloading, indexing, and ranking web pages offline. When a user submits a query, the PageRank algorithm identifies and ranks the relevant pages based on their PageRank, without analyzing the page content. PageRank assesses a page's importance based on the number of incoming links it receives, with higher value given to links from reputable pages than those from less reputable ones.

Definition 2.1. *The PageRank algorithm views the Web as a directed labelled graph where the nodes represent pages and the edges represent hyperlinks between them. This directed graph structure is known as the* ***Web Graph****.*

Proposition 2.2.1. *A graph consists of two sets: V and E. The set V is a finite, nonempty set of* ***vertices****, and the set E is a collection of pairs of vertices called* ***edges****. The $V(G)$ and $E(G)$ represent the set of vertices and edges of the graph G, respectively. A graph is denoted as: $G = (V, E)$.*

2.2.1 Graph Theory Background

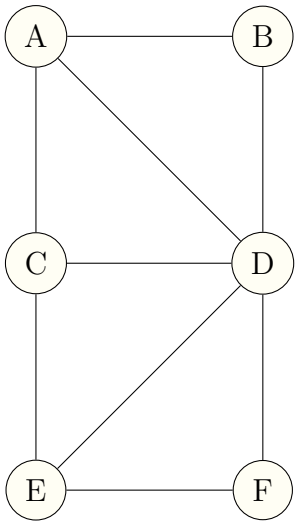


Fig. 2.1: Undirected Graph

Consider the undirected graph G in Figure 2.1. We have

$V(G) = \{A, B, C, D, E, F\}$

$E(G) = \{\{A, B\}, \{A, C\}, \{A, D\}, \{B, D\}, \{C, D\}, \{C, E\}, \{D, E\}, \{D, F\}, \{E, F\}\}$

This is not the Web Graph mentioned in Definition 2.1; being an undirected graph, it does not have directed links from one node to the other. For PageRank, the direction must be clearly defined to accurately represent the flow of influence or importance across the nodes.

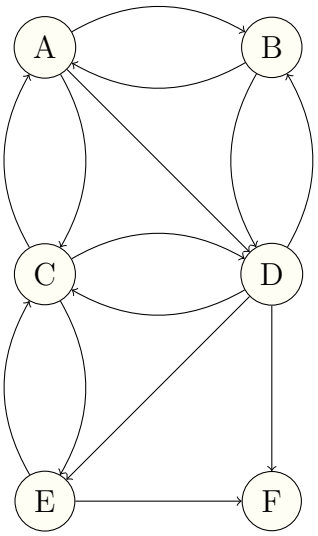


Fig. 2.2: A Directed Web Graph G

In a directed graph, edges such as $A \to B$ and $B \to A$ are distinct. Therefore, the edges (hyperlinks) in the web graph shown in Figure 2.2 can be written as:

$$\begin{aligned} E(G) =& \{(A,B),(B,A),(A,D),(D,A),(A,C),(C,A),(B,D),(D,B),(C,D), \\ & (D,C),(C,E),(E,C),(E,D),(D,E),(D,F),(F,D),(E,F),(F,D)\} \end{aligned}$$

Meanwhile, the set of vertices $V(G)$ (pages) remains the same as in the undirected graph shown in Figure 2.1. In a directed graph with n vertices, the maximum number of edges is $n \cdot (n-1)$. For our graph with $n = 6$ vertices, the maximum number of edges is $6 \cdot (6-1) = 6 \cdot 5 = 30$. In simpler terms, if you have a website with 6 pages, you could theoretically have up to 30 links between them if every page links to every other page. PageRank uses these links to calculate the importance of each page, helping Google decide which pages to show first in search results.

2.2.2 Calculating PageRank

The PageRank of a page p, that we denote by PR_p, is defined as

$$\mathrm{PR}_p = d \sum_{q \in pa(p)} \frac{PR_q}{O_q} + (1-d) \tag{2.1}$$

where $0 < d < 1$ is the damping factor, $pa(p)$ represents the set of pages pointing to p, and O_q is the number of out-going links of page q.

PageRank calculates a page's importance by combining the value from incoming links (adjusted by the number of links on those pages) and a baseline value, which is the damping factor. The damping factor accounts for the probability of randomly jumping to any page, ensuring that every page has some rank, even if it has few or no incoming links.

Example 2.1. *Let us consider the following figure: The set of vertices is:* $V(G) = \{1, 2\}$*, meaning the pages are* 1 *and* 2*. The set of edges is:* $E(G) = \{(1,2),(2,1)\}\}$*, indicating 2 links between these two pages.*

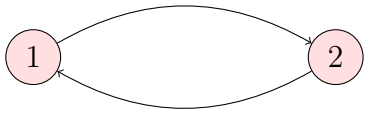


Fig. 2.3: A Directed Web Graph with Nodes 1 and 2

Using the PageRank (i.e., formula (2.1)*) we calculate the rank for pages* 1 *and* 2*. To start with, we assume the initial PageRank values as* 1 *and use the damping factor* d *set to* 0.85*.*

PageRank calculation of page 1*:*

$$PR_1 = d\left(\frac{PR_2}{O_2}\right) + (1-d) \Rightarrow PR_1 = 0.85\left(\frac{1}{1}\right) + (1-0.85) \Rightarrow PR_1 = 1\,. \tag{2.2}$$

Similarly, we calculate the PageRank of page 2*:*

$$PR_2 = d\left(\frac{PR_1}{O_1}\right) + (1-d) \Rightarrow PR_2 = 0.85\left(\frac{1}{1}\right) + (1-0.85) \Rightarrow PR_2 = 1\,. \tag{2.3}$$

The reason we consider the PageRank of page 1 *in the calculation of* PR_2 *and vice versa is that pages* 1 *and* 2 *point to each other (see Figure 2.3).*

Example 2.2. *Let us take for example the following figure: The set of vertices*

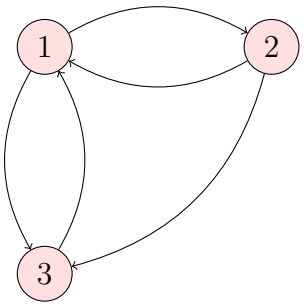


Fig. 2.4: A Directed Web Graph with Nodes 1, 2, and 3

is: $V(G) = \{1, 2, 3\}$*, meaning the pages are* 1, 2 *and* 3*. The set of edges is:* $E(G) = \{(1, 2), (2, 1), (1, 3), (3, 1), (2, 3)\}$*, there are 5 links between these three pages.*

Using the PageRank formula in (2.1) *we can now calculate the rank for pages* 1, 2 *and* 3*. To begin with, we assume the initial PageRank as* 1 *and do the calculation. The damping factor* d *is set to* 0.85*.*

PageRank calculation of page 1*:*

$$\begin{aligned} PR_1 = d\left(\frac{PR_2}{O_2} + \frac{PR_3}{O_3}\right) + (1-d) \Rightarrow PR_1 &= 0.85\left(\frac{1}{2} + \frac{1}{1}\right) + (1 - 0.85) \\ &\Rightarrow PR_1 = 1.425\,. \end{aligned} \tag{2.4}$$

To further explain what has been calculated above:

(a) The PageRank of page 2 *is initially assumed to be* 1*. The number of outgoing links,* O_2*, from page* 2 *is two, since this page points to pages* 1 *and* 3 *(see Figure 2.4).*

(b) The PageRank of page 3 *is initially assumed to be* 1*. The number of outgoing links,* O_3*, from page* 3 *is one, since this page points only to page* 1 *(see Figure 2.4).*

(c) *The damping factor of* 0.85 *in PageRank represents the probability that a user follows links (*85%*) versus randomly jumping to any page (*15%*). This balance ensures that all pages are reachable and prevents isolated pages from having zero rank.*

These factors: the initial PageRank values, the number of outgoing links, and the damping factor of 0.85 *are all used to calculate the PageRank of the first page. Similarly, we calculate the PageRank of pages* 2 *and* 3*:*

$$\begin{aligned} PR_2 = d\left(\frac{PR_1}{O_1}\right) + (1-d) \Rightarrow PR_2 &= 0.85\left(\frac{1.425}{2}\right) + (1-0.85) \\ \Rightarrow PR_2 &= 0.756 \end{aligned} \tag{2.5}$$

The reason we only consider the PageRank of page 1 *in the calculation of* PR_2 *in* (2.5) *is that only page* 1 *points to page* 2 *(see Figure 2.4).*
Lastly:

$$\begin{aligned} PR_3 = d\left(\frac{PR_1}{O_1} + \frac{PR_2}{O_2}\right) + (1-d) \Rightarrow PR_3 &= 0.85\left(\frac{1.425}{2} + \frac{0.756}{2}\right) + (1-0.85) \\ \Rightarrow PR_3 &= 1.077\,. \end{aligned} \tag{2.6}$$

Now, we consider the PageRank of pages 2 *and* 1 *in the calculation of* PR_3 *because both pages* 2 *and* 1 *point to page* 3 *(see Figure 2.4).*

In PageRank computation, convergence occurs when the iterative updates to PageRank values stabilize, meaning the values no longer change significantly between iterations. This ensures that the PageRank scores accurately reflect page importance. The details of this convergence process will be discussed in future sections.

2.3 Integrating Markov Chains with PageRank

Researchers Langville et al. and Bianchini et al. have explored the connection between the PageRank algorithm and Markov chains. This section discusses how the PageRank algorithm relates to the Markov chain framework.

Theorem 2.3.1. *A random surfer navigating the Web randomly selects an outgoing link from one page to move to the next. This process can result in dead ends (pages with no outgoing links) or cycles within a group of interconnected pages. To address this, the surfer occasionally selects a random page from the entire Web. This theoretical* ***random walk*** *is known as a* ***Markov chain or process****. The limiting probability of an infinitely dedicated surfer visiting a particular page represents its PageRank.*

A more intuitive explanation of Theorem 2.3.1:

1. Imagine a random surfer browsing the Web, clicking links to navigate from one page to another.
2. Occasionally, the surfer encounters pages with no links or gets caught in loops among a few pages.
3. To avoid this, the surfer sometimes jumps to a random page anywhere on the Web.
4. This behavior models a **Markov chain**.
5. The PageRank of a page is the long-term probability that the surfer will visit that page.

Proposition 2.3.2. *The number of links to and from a page indicates its importance. A page with* ***more backlinks*** *or* ***incoming*** *links is considered more*

important. Backlinks from highly reputable pages carry more weight than those from less significant pages. Additionally, if a reputable page links to several other pages, its weight is distributed equally among those linked pages.

2.3.1 Mechanism of Integration

Initial Setup

PageRank assigns an initial value of $PR_p^{(0)} = \frac{1}{n}$, where:

- $PR_p^{(0)}$ represents the initial PageRank value of a page p.
- $\frac{1}{n}$ indicates that each page starts with an equal rank, where n is the total number of pages on the Web.
- This initialization assumes that each page is equally likely to be the starting point of a random surfer.

Iteration Formula

The PageRank algorithm iterates according to the following formula:

$$PR_p^{(k+1)} = \sum_{q \in pa(p)} \frac{PR_q^{(k)}}{O_q} \tag{2.7}$$

where:

- p is a web page.
- q is a web page.
- $PR_p^{(k+1)}$ is the PageRank of page p at iteration $k+1$.
- $PR_q^{(k)}$ is the PageRank of page q at iteration k.
- $pa(p)$ is the set of pages linking to page p.

- O_q is the number of outgoing links from page q.

The above equation (2.7) is recursive, meaning the PageRank of p depends on the PageRanks of pages linking to it. This was seen in (2.1).

Matrix Representation

The iterative process of the PageRank algorithm can be represented in matrix notation. Let $\mathbf{q}^{(k)}$ be the PageRank vector at iteration k, and let $\mathbf{T}$ be the transition matrix for the Web. The update rule in matrix notation is:

$$\mathbf{q}^{(k+1)} = \mathbf{T}\mathbf{q}^{(k)}$$

where:

- $\mathbf{q}^{(k)}$ is a column vector with each element representing the PageRank of a page at iteration k.
- $\mathbf{T}$ is the transition matrix with the elements in it representing the probability of moving from page one to page another.

If there are n pages on the Web, let $\mathbf{T}$ be an $n \times n$ matrix such that t_{pq} is the probability of moving from page p to page q in a time interval. Unfortunately, the iterative process defined by

$$\mathbf{q}^{(k+1)} = \mathbf{T}\mathbf{q}^{(k)} \tag{2.8}$$

can have convergence problems, such as:

- **Cycles:** The algorithm might get stuck in cycles, rotating through a subset of pages without reaching a steady state.
- **Starting Vector Dependency:** The limit might depend on the initial PageRank vector $\mathbf{q}^0$, resulting in different limiting distributions for different

starting points. In other words: if different people start their web surfing from different pages, they might end up with different final rankings of pages if the transition matrix does not have certain properties.

To address these issues, Brin and Page developed an irreducible and aperiodic Markov chain characterized by a primitive transition probability matrix:

- **Irreducible:** A Markov chain is irreducible if every page can be reached from any other page, ensuring there are no isolated subsets of pages.
- **Aperiodic:** A Markov chain is aperiodic if it does not return to any state at fixed intervals, avoiding cycles.
- **Primitive Matrix:** A matrix is primitive if some power of the matrix has all positive entries, ensuring the Markov chain is both irreducible and aperiodic, thus guaranteeing a unique steady-state distribution.

Ensuring Irreducibility

Irreducibility guarantees the existence of a unique stationary distribution vector $\mathbf{q}$, which becomes the PageRank vector. This means that, regardless of where you start on the web, the importance scores of all pages will eventually stabilize to a single, unique distribution. The power method, when applied to a primitive stochastic matrix $\mathbf{T}$, will always converge to this unique stationary distribution $\mathbf{q}$. The key points are:

- **Irreducibility:** Ensures that every page can be reached from any other page, meaning the web is fully connected without isolated groups.
- **Unique Stationary Distribution:** Because the web is fully connected, there is one unique set of PageRank values where the importance scores of pages stabilize.

- **Power Method:** This iterative method starts with an initial guess and updates it using the transition matrix $\mathbf{T}$. Due to the properties of a primitive matrix, the method will converge to the unique stationary distribution $\mathbf{q}$, independent of the starting vector.

Proposition 2.3.3. *The PageRank algorithm models the hyperlink structure of the Web using a primitive stochastic matrix. Let* $\mathbf{T}$ *be an* $n \times n$ *matrix, where* n *is the total number of pages on the Web. Each element* t_{pq} *represents the probability of moving from page* p *to page* q *in one step.*
In the basic model, the transition probability is given by:

$$t_{pq} = \frac{1}{|O_p|} \tag{2.9}$$

where $|O_p|$ *is the number of outgoing links from page* p*. This means that if page* p *has a set of forward links* O_p*, the probability of moving to any one of these links, including* q*, is equally distributed. The specific formula can be expressed as:*

$$t_{pq} = \begin{cases} \frac{1}{|O_p|} & \text{if page } p \text{ has a link to page } q \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

This means that

- *if page* p *links to page* q*, the probability of transitioning from page* p *to page* q *is* $\frac{1}{|O_p|}$*, where* $|O_p|$ *is the total number of outgoing links from page* p*. Thus, each link is chosen with equal probability.*

- *if there is no link from page* p *to page* q*, the probability* t_{pq} *is 0, meaning there is no chance of moving from page* p *to page* q *in one step.*

2.4 Practical Examples

Example 2.3. *Let us consider the following example: The Web graph depicted in*

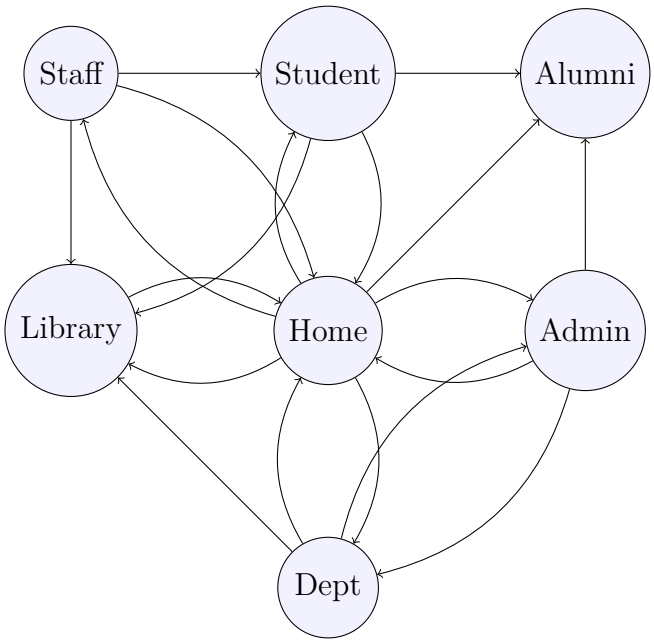


Fig. 2.5: A sample Web Graph W of a University

Figure 2.5 illustrates a sample extracted from a university website. It consists of 7 pages: Home, Admin, Staff, Student, Library, Dept, and Alumni. This sample Web graph is utilized for our Markov analysis and PageRank computations. By using the formula (2.10) *we can calculate the transition matrix,* $\mathbf{T}$ *for Figure 2.5:*

$$T = \begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 \end{bmatrix}$$

The transition matrix $\mathbf{T}$ *represents the probabilities of moving from one page to another.*

- ***Row*** q ***in*** $\mathbf{T}$*: Non-zero elements in row* q *indicate which pages are linked from page* q*. This means that if page* q *links to other pages, those links will be shown by non-zero values in row* q*.*

- ***Column*** p ***in*** $\mathbf{T}$*: Non-zero elements in column* p *indicate which pages have a link pointing to page* p*.*

- ***Sum of Row*** q*: If page* q *has links to other pages, the sum of the values in row* q *of the matrix* $\mathbf{T}$ *will be 1. This ensures that all possible transitions from page* q *(through its links) add up to 100%.*

- ***Rows in*** $\mathbf{T}$*: Show where a page links to. The sum of values in a row is 1 if there are links (outgoing links).*

- ***Columns in*** $\mathbf{T}$*: Show where links to a page come from (incoming links).*

Theorem 2.4.1. *In the transition matrix, if the sum of any row is zero, it indicates that there is a page with no forward links (outgoing links). This type of page is called a* ***dangling node*** *or* ***hanging node****. Dangling nodes cannot exist in the Web graph if it is to be represented using a Markov model.*

The transition matrix for Figure 2.5 contains dangling nodes. Specifically, the third row has a sum of 0*.*

Proposition 2.4.2. *Langville et al. suggested addressing dangling nodes by replacing each row with* $\frac{e}{n}$*, where* e *is a row vector of all ones and* n *is the number of pages. In our example,* n *is 7.*

We now apply the method in Proposition 2.4.2 to the graph in Figure 2.5:

$$\bar{T} = \begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 \end{bmatrix}$$

Row 3 of the transition matrix $\bar{T}$ (for the Alumni page) connects to all nodes,

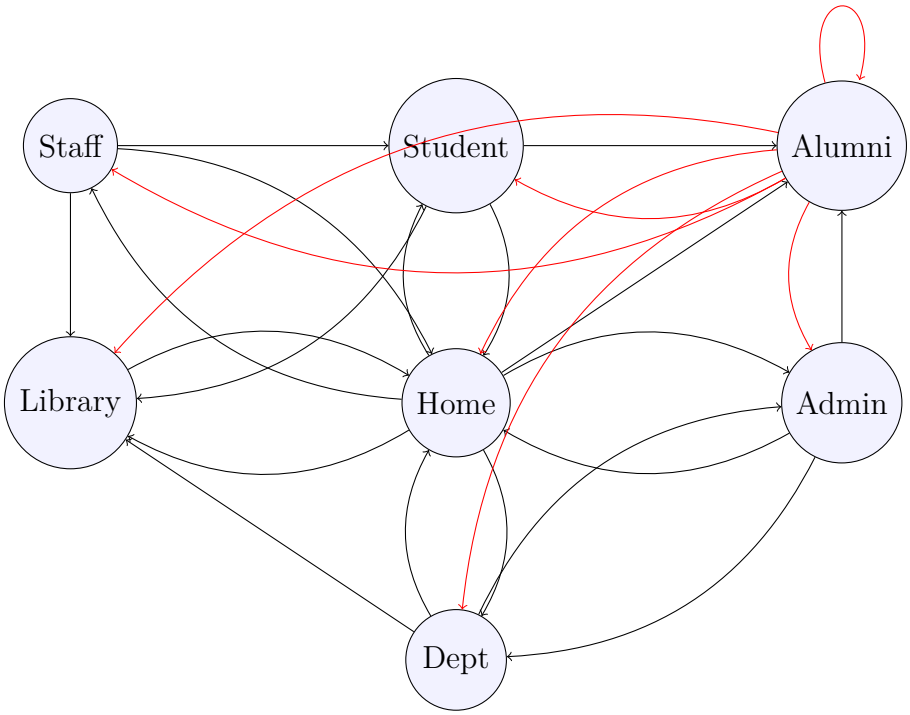


Fig. 2.6: Modified Web Graph W using Proposition 2.4.2

including itself, illustrating a key aspect of stochastic matrices. Each entry represents the probability of transitioning from the Alumni page to other nodes. The

presence of a self-loop indicates a probability of remaining on the Alumni page during transitions.

Proposition 2.4.3. *Bianchini et al. and Singh et al. propose connecting a hypothetical node, h_i, with a self-loop and linking all dangling nodes to this hypothetical node. This approach also ensures that the transition matrix becomes a stochastic matrix.*

We now apply the method from Proposition 2.4.3 to Figure 2.5:

$$\bar{T} = \begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

The last row and column in the transition matrix $\bar{T}$ correspond to the hypothetical node h_i, ensuring the Alumni page has a transition probability of 1, resolving its status as a dangling page. The node h_i has a self-loop with a transition probability of 1, making the graph stochastic. Figure 2.7 below illustrates h_i with a self-loop, connecting to the Alumni page.

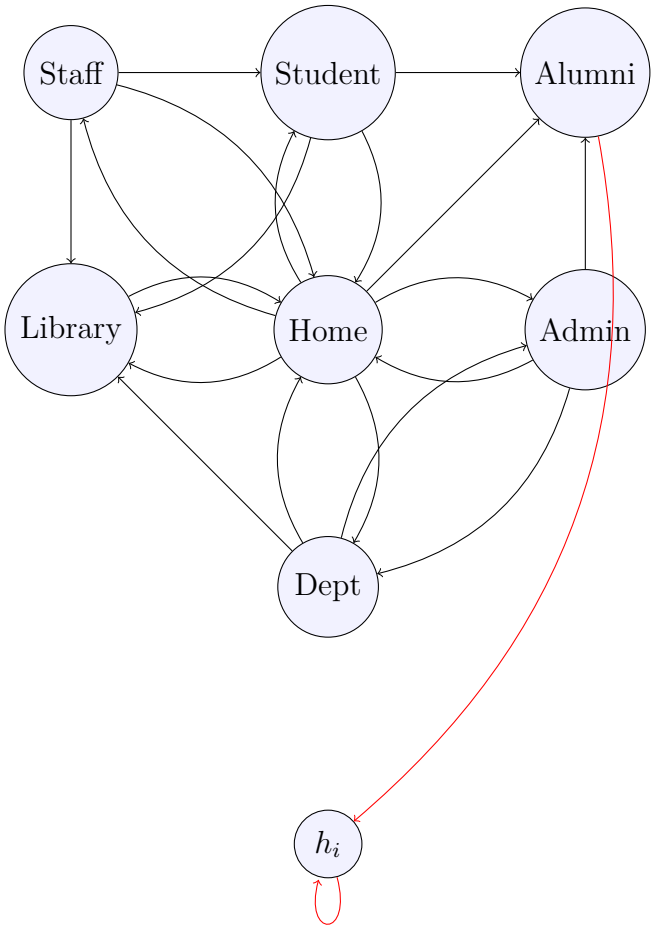


Fig. 2.7: Modified Web Graph W using Proposition 2.4.3

Example 2.4. *Let us consider the following: By using the formula* (2.10) *we can*

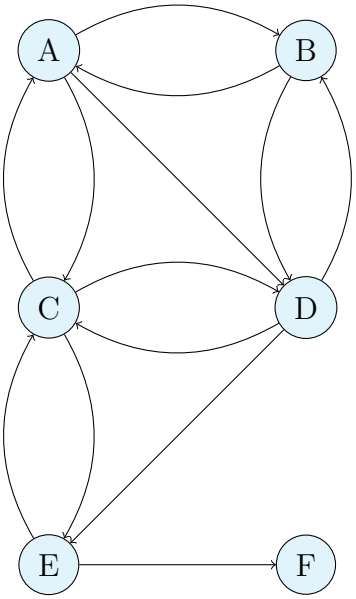


Fig. 2.8: A Directed Web Graph G

calculate the transition matrix, $\mathbf{T}$ *for the graph* G *in Figure 2.8*

$$T = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The transition matrix for Figure 2.8 has dangling nodes. Namely, the sixth row has a sum of 0.

We now apply the method in Proposition 2.4.2 to the graph in Figure 2.8:

$$\bar{T} = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} \\ \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} & \frac{1}{6} \end{bmatrix}$$

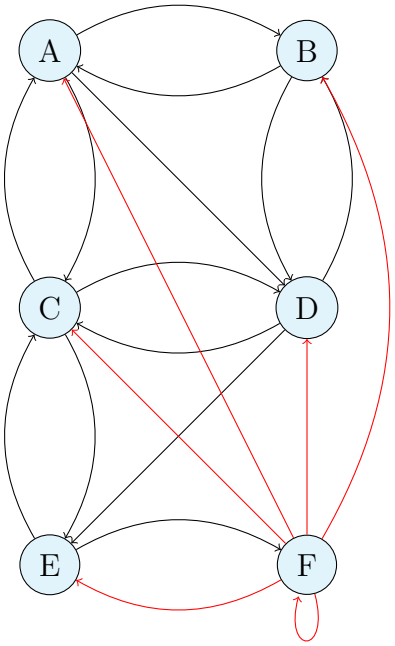


Fig. 2.9: Modified Web Graph G using Proposition 2.4.2

We now apply the method from 2.4.3 to Figure 2.8:

$$T = \begin{bmatrix} 0 & \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ 0 & \frac{1}{3} & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

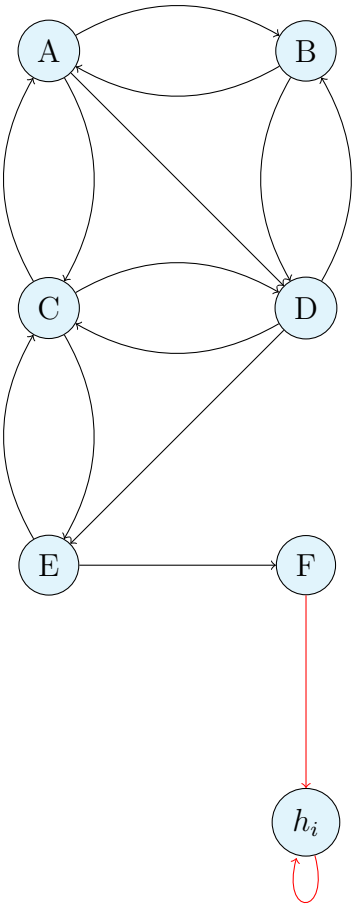


Fig. 2.10: Modified Web Graph G using 2.4.3

While ensuring that the transition matrix $\bar{T}$ is stochastic is necessary, it does not guarantee the convergence of the Markov model or the existence of a steady-state vector.

A significant issue with $\bar{T}$ is that it may not be regular. The general nature of the Web often means that $\bar{T}$ is not regular, as every node in the graph needs to be connected to every other node (i.e., the matrix should be irreducible).
In reality, however, the Web is not fully connected; not every page links to every other page, meaning the graph is not strongly connected. To address this, Brin et al. modified the transition matrix so that all entries satisfy $0 < t_{pq} < 1$. This adjustment ensures that $\bar{T}$ is regular and guarantees that the sequence q_n converges to a unique, positive steady-state vector.

2.5 The Google Matrix

Proposition 2.5.1. *The Google matrix, $\bar{\bar{T}}$, is constructed by adding a perturbation matrix E to the original transition matrix $\bar{T}$, weighted by a damping factor α. This adjustment ensures that the resulting matrix $\bar{\bar{T}}$ is both stochastic (all rows sum to 1) and irreducible (every page can be reached from any other page, either directly or indirectly). The formula for the Google matrix is given by:*

$$\bar{\bar{T}} = \alpha\bar{T} + (1 - \alpha)E$$

where:f

- $\bar{\bar{T}}$ *is the Google matrix, ensuring both stochasticity and irreducibility.*
- α *is the damping factor, with* $0 < \alpha < 1$. *It represents the probability that a web surfer will continue clicking on links on the current page, typically set to 0.85 according to researchers.*

- *$\bar{T}$ is the original transition matrix, representing the probabilities of moving from one page to another based on the link structure of the web.*
- *E is the perturbation matrix, which ensures that every page has a chance of being visited, even if it is not directly linked and is defined as $E = \frac{ee^t}{n}$, where:*
  - *e is a column vector of all ones.*
  - *e^t is the transpose of e, resulting in a row vector of all ones.*
  - *n is the number of web pages (or the order of the matrix).*
- *$1 - \alpha$ is the probability that a web surfer will jump to a random page rather than following a link, accounting for random web surfing behaviour such as typing a URL directly.*

This construction of the Google matrix ensures that the PageRank algorithm can find a unique, steady-state solution that accurately reflects the behaviour of real-life web surfers.

2.6 Further Examples

We compute the Google Matrix as defined in the equation in Proposition 2.5.1 using the sample Web Graph W shown in Figure 2.6 and Web Graph G shown in Figure 2.9, with a damping factor value of $\alpha = 0.85$.
The resulting Google Matrix is represented by $\bar{\bar{T}}$. This matrix can be normalized to a stationary vector by calculating its powers until the matrix values become stationary.

Example 2.5. *Keeping in mind the formula in Proposition 2.5.1:*

(a) $\alpha = 0.85$*;*

(b)

$$\bar{T} = \begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 \end{bmatrix}$$

(c) $1 - \alpha = 0.15$*;*

(d)

$$E = \frac{1}{7} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}.$$

We calculate the Google Matrix for Figure 2.6:

$$\bar{\bar{T}} = 0.85 \begin{bmatrix} 0 & 1/3 & 0 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1/6 & 1/6 & 1/6 & 1/6 & 0 & 1/6 & 1/6 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 & 1/3 \\ 0 & 0 & 0 & 1/3 & 1/3 & 1/3 & 0 \end{bmatrix} + 0.15 \begin{bmatrix} 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \\ 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 & 1/7 \end{bmatrix}$$

$$\bar{\bar{T}} = \begin{bmatrix} 0.0214 & 0.3048 & 0.0214 & 0.3048 & 0.3048 & 0.0214 & 0.0214 \\ 0.0214 & 0.0214 & 0.3048 & 0.3048 & 0.3048 & 0.0214 & 0.0214 \\ 0.1429 & 0.1429 & 0.1429 & 0.1429 & 0.1429 & 0.1429 & 0.1429 \\ 0.0214 & 0.0214 & 0.0214 & 0.0214 & 0.8714 & 0.0214 & 0.0214 \\ 0.1631 & 0.1631 & 0.1631 & 0.1631 & 0.0214 & 0.1631 & 0.1631 \\ 0.0214 & 0.0214 & 0.3048 & 0.0214 & 0.3048 & 0.0214 & 0.3048 \\ 0.0214 & 0.0214 & 0.0214 & 0.3048 & 0.3048 & 0.3048 & 0.0214 \end{bmatrix}$$

The values presented correspond to the PageRank scores for the 7 pages in the sample Web graph W. After performing enough iterations of the PageRank algorithm, the resulting scores have reached a stable state, known as the stationary vector. This indicates that the PageRank scores have converged and are no longer significantly changing with additional iterations. Therefore, the stationary vector represents the final ranking of each page in the web graph.

Assume that after a certain number of iterations, the stationary vector for our sample 7-page Web graph W is:

$$\mathbf{s} = \begin{bmatrix} 0.0798 & 0.1024 & 0.1404 & 0.16298 & 0.2917 & 0.1114 & 0.1114 \end{bmatrix}$$

The values in $\mathbf{s}$ *indicate the relative importance of each page. Higher values mean more important pages. For example, "Home" (with a score of 0.2917) is the most important, while "Staff" (with a score of 0.0798) is the least important. This process helps search engines rank web pages based on their importance and how often they will likely be visited. This calculation will also be shown in Chapter 3 using Python.*

Example 2.6. *We do the same procedure as in Example 2.5 with the Web Graph*

associated with Figure 2.9. We calculate the Google Matrix for Figure 2.9:

$$\bar{\bar{T}} = 0.85 \begin{bmatrix} 0 & 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 & 0 \\ 1/3 & 0 & 0 & 1/3 & 1/3 & 0 \\ 0 & 1/3 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 1/2 & 0 & 0 & 1/2 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{bmatrix} + 0.15 \begin{bmatrix} 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \end{bmatrix}$$

$$\bar{\bar{T}} = \begin{bmatrix} 0.025 & 0.30833333 & 0.30833333 & 0.30833333 & 0.025 & 0.025 \\ 0.45 & 0.025 & 0.025 & 0.45 & 0.025 & 0.025 \\ 0.30833333 & 0.025 & 0.025 & 0.30833333 & 0.30833333 & 0.025 \\ 0.025 & 0.30833333 & 0.30833333 & 0.025 & 0.30833333 & 0.025 \\ 0.025 & 0.025 & 0.45 & 0.025 & 0.025 & 0.45 \\ 0.16666667 & 0.16666667 & 0.16666667 & 0.16666667 & 0.16666667 & 0.16666667 \end{bmatrix}$$

After a certain number of iterations we get to a stationary vector. This calculation will be shown in Chapter 3 using Python.

Chapter 3

Conceptual Approach and Python Application

3.1 Network Components

Definition 3.1. *A network G is composed of two key components:*

- *A set N of elements known as **nodes** or **vertices**.*
- *A set of node pairs, referred to as **links** or **edges**, where each link (i, j) connects nodes i and j.*

Definition 3.2. *Networks can be classified as **directed** or **undirected**:*

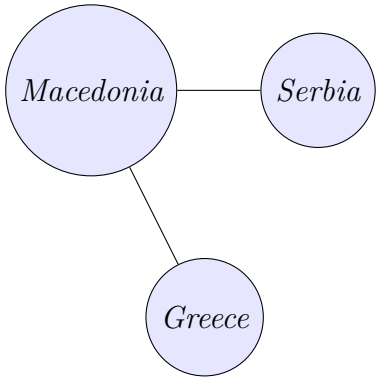
- *A **directed network**, also known as a **digraph**, includes links where the sequence of nodes indicates direction.*
- *An **undirected network** contains bi-directional links, making the node order in a link irrelevant.*

Definition 3.3. *Networks can also be **weighted** or **unweighted**:*

- *In a **weighted network**, each link has an associated weight. A weighted link (i, j, w) indicates a connection between nodes i and j with a weight w.*

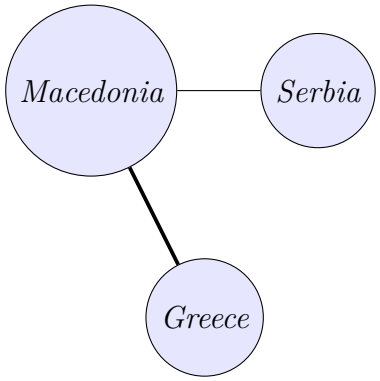
- *In an **unweighted network**, all links carry the same weight, typically 1 for the presence of a link or 0 for its absence.*

Example 3.1. *The following is an example of an undirected and unweighted network:*



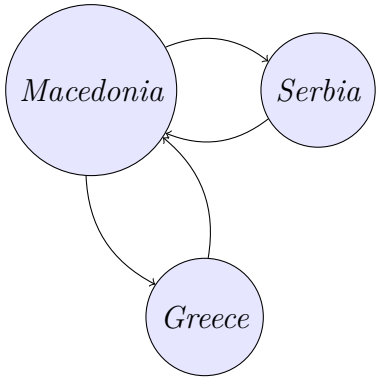
Explanation: *In this network, imagine a traveller going from Serbia to Greece via Macedonia. All roads are bidirectional and equal in quality. The traveller can travel freely between these cities without concern for road direction or quality.*

Example 3.2. *The following is an example of an undirected and weighted network where thicker lines have weight twice as much as the thinner ones:*



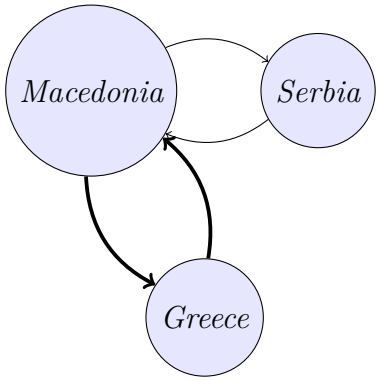
Explanation: *In this network, roads between cities are still bidirectional, but some roads are more efficient. The traveller prefers the thicker lines, representing better-maintained roads, making the journey smoother or faster.*

Example 3.3. *The following is an example of a directed and unweighted network with bent edges:*



Explanation: *In this network, roads have specific directions. The traveller must follow the prescribed path from Serbia to Macedonia, then to Greece, and back. The directionality of the roads dictates a route on how to get to Greece but does not specify which roads are considered the most efficient.*

Example 3.4. *The following is an example of a directed and weighted network where thicker lines have weight twice as much as the thinner ones:*



Explanation: *In this network, roads are directed and vary in quality. The traveller must follow the directional paths and may prefer thicker, more efficient roads for a smoother or quicker trip. The network enforces directionality and rewards using better roads.*

3.2 Directed Networks

Similarly, PageRank can be understood through the lens of weighted networks. In PageRank, the importance or "weight" of a page is determined by its PageRank score. Pages with higher PageRank scores are considered more important and thus more likely to be preferred or prioritized. Pages with a higher PageRank score are analogous to roads with greater weights in a network. Just as in a weighted network where thicker lines indicate more efficient or preferred routes, a higher PageRank signifies a more influential or significant page within the web.

Explanation: Much like how better-maintained roads are preferred for a smoother journey, pages with higher PageRank scores are more likely to appear at the top of search results.

3.2.1 Network Degree

Definition 3.4. *The average degree $\langle k \rangle$ of a network is its number of links (or neighbours) and is defined as:*

$$\langle k \rangle = \frac{2L}{N}$$

where:

(i) L is the number of edges.

(ii) N is the number of nodes.

Each edge in an undirected network contributes to the degree of two nodes, thus $2L$ represents the total degree contributions.

In a directed network, the degree of a node is split into incoming and outgoing links. The in-degree (k_i^{in}) is the number of incoming links (predecessors) to node i, while the out-degree (k_i^{out}) is the number of outgoing links (successors) from node i.

Recalling Formula (2.1)**,** *O_q in the PageRank formula is equivalent to the out-degree k_i^{out} of a node. In other words, O_q represents the number of outgoing links from a node, which directly relates to the out-degree k_i^{out}in the network. Node i coincides with node q.*

3.2.2 Network Weight and Strength

Definition 3.5. *We define the following terms:*

- *The weight w_{ij} of an edge connecting node i to node j indicates the strength of their connection. We assume $w_{ij} = 0$ if there is no connection between nodes i and j. In a network, weight is analogous to the PageRank score, representing nodes' importance or influence.*

- *In an undirected graph, the weighted degree or strength of a node i is calculated as:*

$$s_i = \sum_j w_{ij}$$

- *In a directed graph, the weighted in-degree or strength of a node i is given by:*

$$s_i^{in} = \sum_j w_{ji}$$

- *In a directed graph, the weighted out-degree of a node i is determined by:*

$$s_i^{out} = \sum_j w_{ij}$$

Example 3.5. *Let us consider the data from Example 1.7. The graph is weighted and directed: The weights of the edges are as follows:*

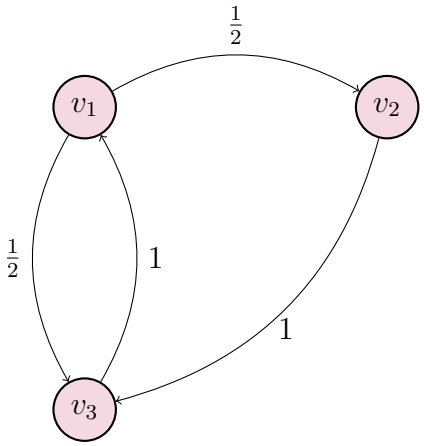


Fig. 3.1: Markov chain graph

$$w_{1,2} = \frac{1}{2}, \quad w_{1,3} = \frac{1}{2}, \quad w_{3,1} = 1, \quad w_{2,3} = 1.$$

The weighted degrees for each node are:

$$\begin{aligned} s_{1,out} &= w_{1,2} + w_{1,3} = 1, \\ s_{1,in} &= w_{3,1} = 1, \\ \Rightarrow s_1 &= s_{1,out} + s_{1,in} = 1 + 1 = 2. \end{aligned}$$

$$\begin{aligned} s_{2,out} &= w_{2,3} = 1, \\ s_{2,in} &= w_{1,2} = \frac{1}{2}, \\ \Rightarrow s_2 &= s_{2,out} + s_{2,in} = 1 + \frac{1}{2} = \frac{3}{2}. \end{aligned}$$

$$\begin{aligned} s_{3,out} &= w_{3,1} = 1, \\ s_{3,in} &= w_{1,3} + w_{2,3} = \frac{1}{2} + 1 = \frac{3}{2}, \\ \Rightarrow s_3 &= s_{3,out} + s_{3,in} = 1 + \frac{3}{2} = \frac{5}{2}. \end{aligned}$$

The average degree:

$$\langle k \rangle = \frac{2L}{N} \quad \Rightarrow \quad \langle k \rangle = \frac{2 \times 4}{3} = \frac{8}{3} \approx 2.67.$$

3.3 Visualizing Networks with Python

Importing the relevant libraries:

```
import matplotlib.pyplot as plt #used for creating
    visualizations and plots
import networkx as nx #creating, manipulating and analyzing
    graphs and networks
```

3.3.1 Example Code for Undirected and Directed Graphs

Example: Undirected Graph of Former Yugoslav Countries

To illustrate an undirected graph representing the countries of former Yugoslavia and their neighbouring connections, use the following Python code:

```
G = nx.Graph() #this is how you create an undirected and
    empty graph
G.add_node('Macedonia') # we add a single node 'Macedonia'
    to the graph; this is one way to add nodes
nodes_to_add = ['Serbia', 'Slovenia', 'Croatia', 'Montenegro
    ', 'B&H'] #we create a list of nodes
G.add_nodes_from(nodes_to_add) #we add all nodes from the
    list to the graph; this is another way
edges_to_add = [("Slovenia", "Croatia"), ("Croatia", "B&H"),
     ("Croatia", "Serbia"), ("B&H", "Serbia"), ("Serbia", "
    Montenegro"), ("Montenegro", "B&H"),
("Serbia", "Macedonia")] #list of edges (connections) to add
     to the graph
G.add_edges_from(edges_to_add) #we add all edges from the
    list to the graph; another way to add edges is G.add_edge
```

```
    ('Macedonia', 'Serbia'), one by one
nx.draw(G, with_labels=True, node_size=2500, font_size=9) #
    draw the graph with labels, set node size to 2500, and
    font size to 9:
#G: the graph object to be drawn.
#with_labels: boolean and if True, node labels are drawn
```

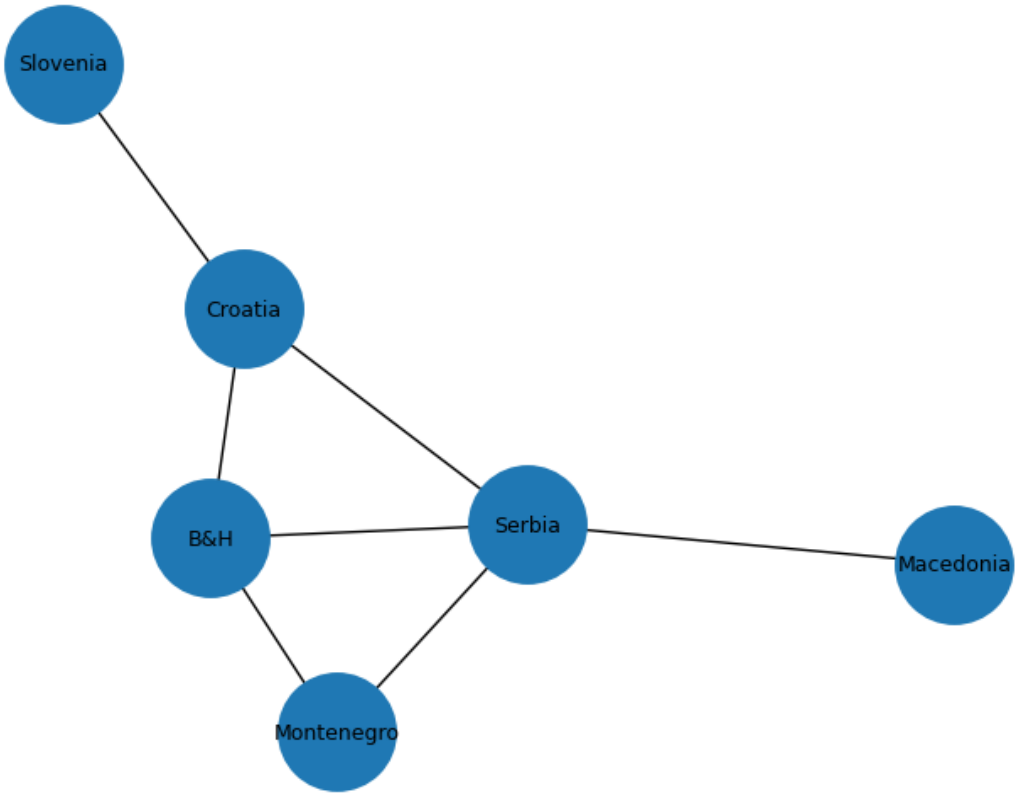


Fig. 3.2: Plot generated from the code above.

Example: Directed Graph of a LUISS Student Taking a Shuttle

The following Python code illustrates the creation of a directed graph using NetworkX, representing a LUISS student taking the shuttle between different locations:

```
D = nx.DiGraph() #create an empty directed graph
nodes_to_add=('Pola', 'Parenzo', 'Romania') #we make a list
   of nodes to be added to the empty graph
D.add_nodes_from(nodes_to_add) #add the nodes from the list
   above
edges_to_add = [("Pola", "Parenzo"), ("Parenzo", "Romania"),
    ("Romania", "Parenzo"), ("Parenzo", "Pola")] #define
   edges (directed connections) to be added to the graph
D.add_edges_from(edges_to_add) #adding the edges
nx.draw(D, with_labels=True, node_color='red', node_size
   =2500, font_size=11)
#draw the directed graph with labels, set node color to red,
    node size to 2500, and font size to 11
#D: the directed graph object to be drawn
#with_labels: boolean; if True, node labels are drawn
#node_color: color of the nodes
#node_size: size of the nodes
#font_size: size of the labels' font
```

In this example, nodes represent locations, and directed edges represent the shuttle's direction between these locations.

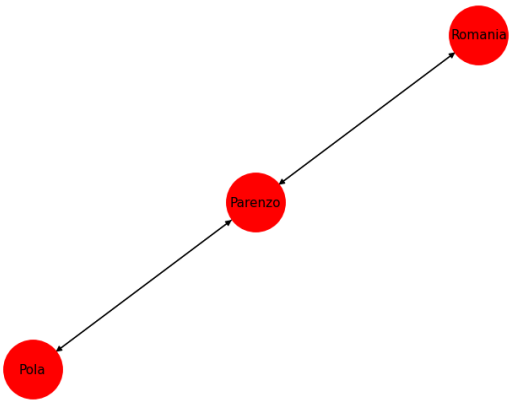


Fig. 3.3: Plot generated from the code above.

3.3.2 Graphing Techniques

We will run these codes and show the output for the Yugolsav countries example (3.3.1).

Listing nodes:

```
G.nodes()
```

Output: `NodeView(('Macedonia', 'Serbia', 'Slovenia', 'Croatia', 'Montenegro', 'B&H'))`.

Listing edges:

```
G.edges()
```

Output: `EdgeView([('Macedonia', 'Serbia'), ('Serbia', 'Croatia'), ('Serbia', 'B&H'), ('Serbia', 'Montenegro'), ('Slovenia', 'Croatia'), ('Croatia', 'B&H'), ('Montenegro', 'B&H')])`

Iteration over edges:

```
for edge in G.edges:
    print(edge)
```

Output:

```
('Macedonia', 'Serbia')
('Serbia', 'Croatia')
('Serbia', 'B&H')
('Serbia', 'Montenegro')
('Slovenia', 'Croatia')
('Croatia', 'B&H')
('Montenegro', 'B&H'))
```

Iteration over nodes:

```
for node in G.nodes:
    print(node)
```

Output:

```
Macedonia
Serbia
Slovenia
Croatia
Montenegro
B&H
```

Number of nodes:

```
G.number_of_nodes()
```

Output: 6

Number of edges:

```
G.number_of_edges()
```

Output: 7

Creating a list of neighbours:

```
neighbors_of_Macedonia = list(G.neighbors('Macedonia'))
print(neighbors_of_Macedonia)
```

Output: `['Serbia']`

3.3.3 Node and Edge Existence

To check if a node is present in a graph:

```
G.has_node('Macedonia')
# or
'Serbia' in G.nodes
```

Output: `True`

To check if two nodes are connected by an edge:

```
G.has_edge('Macedonia', 'Serbia')
# or
('B&H', 'Croatia') in Graph.edges
```

Output: `True`

3.3.4 Calculating Node and Edge Degree

One of the key aspects to investigate for a node in a graph is the number of connections it has with other nodes.

```
len(list(G.neighbors('Macedonia')))
```

This will return `1`. However, since this is a common operation, NetworkX offers a more straightforward method to obtain this information:

```
G.degree('Macedonia')
```

Remark 1. *Up until this point, we have run the code for the Yugoslav countries. The same procedure can be applied to the LUISS Shuttle case, except for the node degree case, where incoming and outgoing links are distinguished. In the following section, we will run the corresponding code for this case.*

Instead of the symmetric relationship "neighbors", nodes in directed graphs have predecessors ("in-neighbors") and successors ("out-neighbors"):

- **Predecessor**: If there is a directed edge (i, j), then i is a predecessor of j.
- **Successors**: If there is a directed edge (i, j), then j is a successor of i.
- **Neighbors**: If there is an edge (i, j), then j is a neighbor of i, and i is a neighbor of j.

```
print('Successors of Parenzo:', list(D.successors('Parenzo'))
#get the successors of the node 'Parenzo' (nodes that 'Parenzo' points to);
#D.successors('Parenzo') returns an iterator over these successor nodes
#we use the list() function to convert this iterator into a list for easy handling and printing
print('Predecessors of Parenzo:', list(D.predecessors('Parenzo')))
#get the predecessors of the node 'Parenzo' (nodes that point to 'Parenzo');
#D.predecessors('Parenzo') returns an iterator over these predecessor nodes
#we use the list() function to convert this iterator into a list for easy handling and printing
```

Output:

```
Successors of Parenzo:  ['Romania', 'Pola']
Predecessors of Parenzo:  ['Pola', 'Romania']
```

To check the number of predecessors of a node:

```
D.in_degree('Parenzo')
```

Output: 2

To check the number of successors of a node:

```
D.out_degree('Parenzo')
```

Output: 2

In a directed graph, relationships between nodes are represented by edges that have a specific direction, like arrows pointing from one node to another. To understand the connectivity of a particular node, we often examine its incoming and outgoing edges.

- **Incoming Edges:** These are edges pointing toward a node, indicating which other nodes have a direct connection leading to it.
- **Outgoing Edges:** These are edges pointing away from a node, showing which other nodes it directly connects to.

To check the the incoming edges:

```
D.in_edges('Parenzo')
```

Output: `InEdgeDataView([('Pola', 'Parenzo'), ('Romania', 'Parenzo')])`

To check the the outgoing edges:

```
D.out_edges('Parenzo')
```

Output: `OutEdgeDataView([('Parenzo', 'Romania'), ('Parenzo', 'Pola')])`

3.3.5 Implementing Weight

To illustrate the concept of edge weights, we will use the graph from Example 1.7. The graph is weighted and directed. The following Python code demonstrates how to create and visualize this graph:

```
W = nx.DiGraph() #create a directed and weighted empty graph
W.add_edge('v1', 'v2', weight=1/2)
W.add_edge('v1', 'v3', weight=1/2)
W.add_edge('v2', 'v3', weight=1)
W.add_edge('v3', 'v1', weight=1)
#W.add_edge(node1, node2, weight=value):
#function: adds a directed edge from node1 to node2 with a
   specified weight
#usage: adds edges with weights between nodes in the graph
pos = nx.spring_layout(W, seed=42)
#generate positions for nodes pos function is used
#nx.spring_layout(G, seed=42):
#function: computes positions for nodes using the spring
   layout algorithm
#usage: positions nodes in a visually appealing way, with
   the seed parameter for reproducibility
nx.draw(W, pos, with_labels=True, node_color='lightblue',
   node_size=2000, font_size=15, font_weight='bold',
   edge_color='gray')
edge_labels = nx.get_edge_attributes(W, 'weight')
#function: retrieves edge attributes for labeling
#usage: returns a dictionary with edge labels for drawing
nx.draw_networkx_edge_labels(W, pos, edge_labels=edge_labels
   , font_color='red')
```

```
#function: draws labels on the edges
#usage: displays the weight of each edge in red color
plt.show() #to display the plot
```

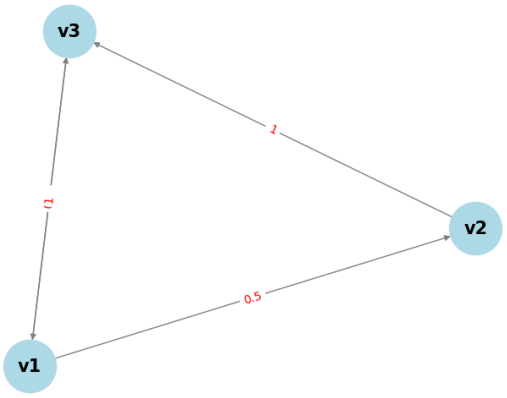


Fig. 3.4: Plot generated from the code above.

In order to print the weights of each edge:

```
for (i, j, w) in W.edges(data='weight'):
    if w > 0:
        print(f"Edge from {i} to {j} has weight {w}")
#iterate over all edges in the graph W, including their
   weights
#W.edges(data='weight') returns an iterator of tuples (node1
   , node2, weight)
```

Output:

```
Edge from v1 to v2 has weight 0.5
Edge from v1 to v3 has weight 0.5
Edge from v2 to v3 has weight 1
Edge from v3 to v1 has weight 1
```

To get the strength of a given node:

```
W.degree('v1', weight='weight')
#this method calculates the weighted degree of the specified
    node ('v1')
```

Output: 2.0

To print the adjacency matrix (transition matrix):

```
import scipy #import the SciPy library, which provides
   support for sparse matrices (in which most elements are
   0) among other things
W = nx.DiGraph()
W.add_edge('v1', 'v2', weight=1/2)
W.add_edge('v1', 'v3', weight=1/2)
W.add_edge('v2', 'v3', weight=1)
W.add_edge('v3', 'v1', weight=1)
adj_matrix = nx.adjacency_matrix(W, weight='weight')
#generate the adjacency matrix of the graph, including edge
   weights
#the 'weight' parameter specifies that weights should be
   used
print(adj_matrix.todense())
#print the adjacency matrix as a dense matrix
#dense matrix: provides a complete view of the matrix,
   showing all elements including zeros, which can be
   helpful for visualization or understanding the full
   structure of the matrix
#sparse: only non-zero elements and their indices are stored
```

Output:

```
[[0.  0.5 0.5]
```

```
[0.  0.  1. ]
[1.  0.  0. ]]
```

As illustrated, this is the transition matrix used in PageRank calculations. In the following section, we will compute transition matrices and perform PageRank calculations using Python.

3.4 Network Graphs and PageRank Computation with Python

3.4.1 Example 1

Revisiting Example 2.3, we will now visualize the network using Python.

```
import networkx as nx
import matplotlib.pyplot as plt
W = nx.DiGraph()
nodes = ["Staff", "Student", "Alumni", "Admin", "Dept", "
   Library", "Home"]
W.add_nodes_from(nodes)
edges = [("Staff", "Home", 1/3), ("Staff", "Student", 1/3),
    ("Staff", "Library", 1/3),
         ("Student", "Home", 1/3), ("Student", "Library",
            1/3), ("Student", "Alumni", 1/3),
         ("Library", "Home", 1),
         ("Home", "Staff", 1/6),("Home", "Student", 1/6),("
            Home", "Library", 1/6),("Home", "Alumni", 1/6),(
            "Home", "Dept", 1/6),("Home", "Admin", 1/6),
         ("Admin", "Alumni", 1/3), ("Admin", "Home", 1/3),("
            Admin", "Dept", 1/3),
```

```
11          ("Dept", "Library", 1/3),("Dept", "Home", 1/3),("
               Dept", "Admin", 1/3)]
12 W.add_weighted_edges_from(edges)
13 pos = nx.spring_layout(W)
14 nx.draw(W, pos, with_labels=True, node_color='lightblue',
      node_size=2000, font_size=10, font_weight='bold',
      edge_color='black')
15 plt.show()
```

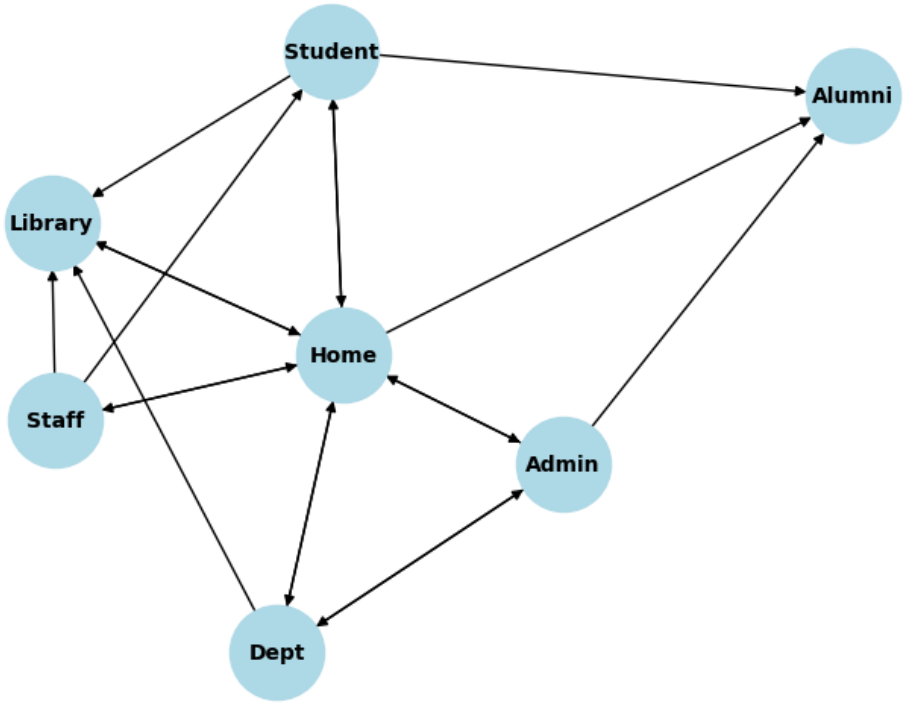


Fig. 3.5: Plot generated from the code above.

The code gives the transition matrix for the graph:

```
adj_matrix = nx.adjacency_matrix(W, weight='weight')
print(adj_matrix.todense())
```

Output:

```
[[0.         0.33333333 0.         0.         0.         0.33333333
  0.33333333]
 [0.         0.         0.33333333 0.         0.         0.33333333
  0.33333333]
 [0.         0.         0.         0.         0.         0.
  0.        ]
 [0.         0.         0.33333333 0.         0.33333333 0.
  0.33333333]
 [0.         0.         0.         0.33333333 0.         0.33333333
  0.33333333]
 [0.         0.         0.         0.         0.         0.
  1.        ]
 [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667
  0.        ]]
```

Checking if there are any dangling nodes:

```
W.out_degree('Staff')
W.out_degree('Home')
W.out_degree('Library')
W.out_degree('Admin')
W.out_degree('Alumni')
W.out_degree('Dept')
W.out_degree('Student')
```

```
# check the out-degree (number of outgoing links) for each
    specified node in the graph W
#a node is considered dangling if its out-degree is 0 (i.e.,
     it has no outgoing links)
```

Output:

```
3
6
1
3
0
3
3
```

We can see that the Alumni page is a dangling node. This cannot happen in PageRank if it is to be a Markov Process.

Solution: As described in Proposition 2.4.2, we address the issue of dangling nodes using the same approach, implemented in Python.

```
import networkx as nx
import matplotlib.pyplot as plt
W = nx.DiGraph()
nodes = ["Staff", "Student", "Alumni", "Admin", "Dept", "
    Library", "Home"]
W.add_nodes_from(nodes)
edges = [("Staff", "Home", 1/3), ("Staff", "Student", 1/3),
     ("Staff", "Library", 1/3),
     ("Student", "Home", 1/3), ("Student", "Library", 1/3), (
        "Student", "Alumni", 1/3),
     ("Library", "Home", 1),
```

```
    ("Home", "Staff", 1/6),("Home", "Student", 1/6),("Home", "Library", 1/6),
    ("Home", "Alumni", 1/6),("Home", "Dept", 1/6),("Home", "Admin", 1/6),
    ("Admin", "Alumni", 1/3), ("Admin", "Home", 1/3),("Admin", "Dept", 1/3),
    ("Dept", "Library", 1/3),("Dept", "Home", 1/3),("Dept", "Admin", 1/3)]
W.add_weighted_edges_from(edges)
for node in W.nodes():
#iterate through each node in the graph
    if W.out_degree(node) == 0:
#check if the node is dangling (i.e., has no outgoing links)
        W.add_edge(node, "Staff", weight=1/len(W.nodes))
        W.add_edge(node, "Student", weight=1/len(W.nodes))
        W.add_edge(node, "Alumni", weight=1/len(W.nodes))
        W.add_edge(node, "Admin", weight=1/len(W.nodes))
        W.add_edge(node, "Library", weight=1/len(W.nodes))
        W.add_edge(node, "Dept", weight=1/len(W.nodes))
        W.add_edge(node, "Home", weight=1/len(W.nodes))
#add outgoing links from this node to all the other nodes: so we implement a technique where we assign 1/n weight and we make that node that pointed to nothing now point to every node
        print(f"Out-degree of {node} after fixing: {W.out_degree(node)}")
pos = nx.spring_layout(W)
nx.draw(W, pos, with_labels=True, node_color='lightblue', node_size=2000, font_size=10, font_weight='bold',
```

```
    edge_color='black')
plt.title("Directed Graph W after Fixing Dangling Nodes")
plt.show()
```

Output: `Out-degree of Alumni after fixing:  7`

Double-checking:

```
W.out_degree('Alumni')
```

Output: 7

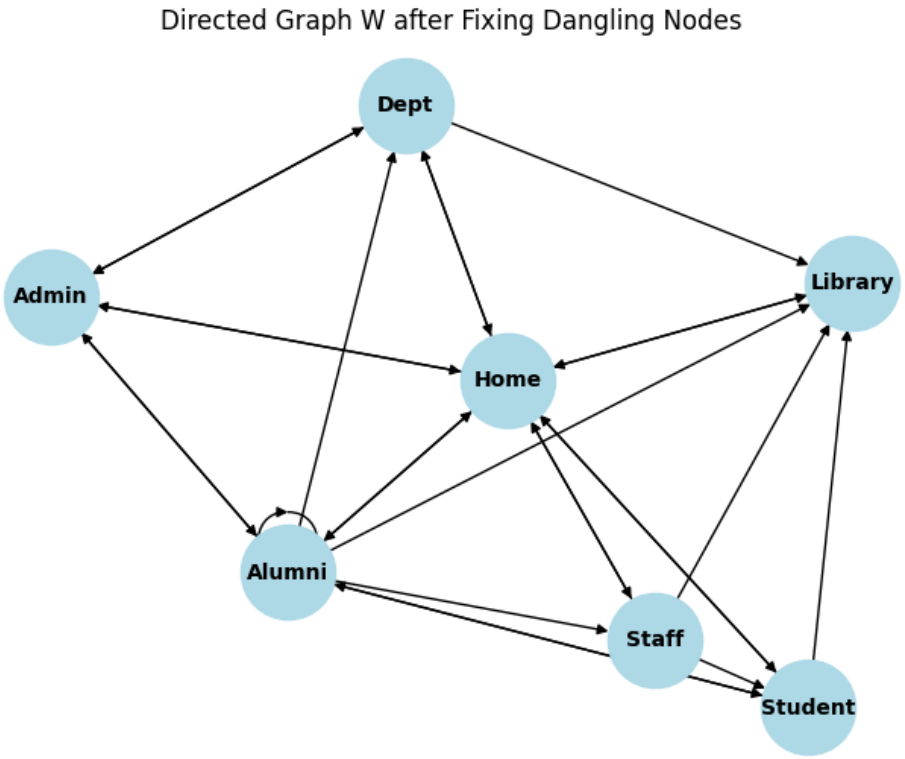


Fig. 3.6: Plot generated from the code above.

After addressing the issue of dangling nodes, the updated transition matrix is as follows:

```
adj_matrix = nx.adjacency_matrix(W, weight='weight')
print(adj_matrix.todense())
```

Output:

```
[[0.         0.33333333 0.         0.         0.         0.33333333
  0.33333333]
 [0.         0.         0.33333333 0.         0.         0.33333333
  0.33333333]
 [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]
 [0.         0.         0.33333333 0.         0.33333333 0.
  0.33333333]
 [0.         0.         0.         0.33333333 0.         0.33333333
  0.33333333]
 [0.         0.         0.         0.         0.         0.
  1.        ]
 [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667
  0.        ]]
```

Let us proceed with calculating the Google matrix:

```
import numpy as np #fundamental library for numerical and
   scientific computing in Python
def google_matrix(T, alpha): #define the function to create
   the Google Matrix
    n = T.shape[0] #get the number of nodes in the
       transition matrix T; T.shape[0]: accesses the number
       of rows in the matrix T
```

```
    E = np.ones((n, n)) / n
#create a matrix E where each element is 1/n (uniform distribution)
#this represents the probability of randomly jumping to any node and is equally likely
    G = alpha * T + (1 - alpha) * E
#calculate the Google Matrix G
#G = alpha * T + (1 - alpha) * E
#alpha * T scales the original transition matrix T by the damping factor alpha
#(1 - alpha) * E adds a uniform distribution to account for random jumps
#this ensures that there is a probability of jumping to any node, not just following the usual transition probabilities
    return G
T = np.array([[0.0, 0.33333333, 0.0, 0.0, 0.0, 0.33333333, 0.33333333],
    [0.0, 0.0, 0.33333333, 0.0, 0.0, 0.33333333, 0.33333333],
    [0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714, 0.14285714],
    [0.0, 0.0, 0.33333333, 0.0, 0.33333333, 0.0, 0.33333333],
    [0.0, 0.0, 0.0, 0.33333333, 0.0, 0.33333333, 0.33333333],
    [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 1.0],
    [0.16666667, 0.16666667, 0.16666667, 0.16666667, 0.16666667, 0.16666667, 0.0]])
```

```
alpha = 0.85
G = google_matrix(T, alpha)
#creating the Google Matrix using the defined function
print("Google Matrix:\n", G)
```

Output:

```
Google Matrix:
 [[0.02142857 0.3047619  0.02142857 0.02142857 0.02142857 0.3047619
  0.3047619 ]
 [0.02142857 0.02142857 0.3047619  0.02142857 0.02142857 0.3047619
  0.3047619 ]
 [0.14285714 0.14285714 0.14285714 0.14285714 0.14285714 0.14285714
  0.14285714]
 [0.02142857 0.02142857 0.3047619  0.02142857 0.3047619  0.02142857
  0.3047619 ]
 [0.02142857 0.02142857 0.02142857 0.3047619  0.02142857 0.3047619
  0.3047619 ]
 [0.02142857 0.02142857 0.02142857 0.02142857 0.02142857 0.02142857
  0.87142857]
 [0.16309524 0.16309524 0.16309524 0.16309524 0.16309524 0.16309524
  0.02142857]]
```

We can compute the PageRank using Python:

```
pagerank = nx.pagerank(W, alpha=0.85, weight='weight')
#calculate PageRank for each node in the graph W
#nx.pagerank(G) is a general form
#alpha=0.85 is the damping factor, which balances the
   following links and random jumps
```

```
#weight='weight' means it uses edge weights in the
    calculation
print("PageRank:")
for node, rank in pagerank.items():
#loop through each node and its PageRank score
#print the node and its corresponding PageRank score
#the nx.pagerank() function returns a dictionary where keys
    are node labels and values are the corresponding PageRank
     scores
#call the .items() method on the dictionary to get a view of
     the key-value pairs (that is how we get pagerank.items()
    )
     print(f"{node}: {rank}")
```

Output:

```
PageRank:
Staff:  0.0798020288250451
Student:  0.1024128935466225
Alumni:  0.14036885245403036
Admin:  0.11135184916867426
Dept:  0.11135184916867426
Library:  0.16297971717560775
Home:  0.29173280966134585
```

3.4.2 Example 2

We will analyze Example 2.4 and perform the same analysis using Python in this subsection.

```
import networkx as nx
import matplotlib.pyplot as plt
W=nx.DiGraph()
nodes=['A', 'B', 'C', 'D', 'E', 'F']
W.add_nodes_from(nodes)
edges=[('A','B', 1/3), ('A','C', 1/3),('A','D', 1/3),('B','A
   ',1/2),('B','D',1/2), ('C','A',1/3),('C','D',1/3),('C','E
   ',1/3),('D','B',1/3),('D','E',1/3),
       ('D','C',1/3),('E','C',1/2),('E','F',1/2)]
W.add_weighted_edges_from(edges)
pos=nx.spring_layout(W)
nx.draw(W, pos, with_labels=True, node_color='lightblue',
   node_size=2000, font_size=10, font_weight='bold',
   edge_color='black')
plt.show()
```

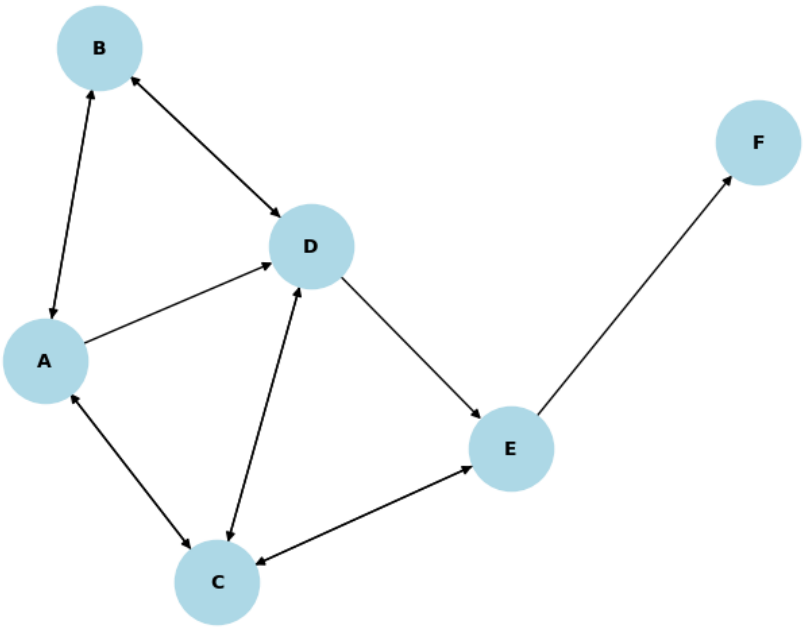


Fig. 3.7: Plot generated from the code above.

The code gives the transition matrix for the graph:

```
adj_matrix = nx.adjacency_matrix(W, weight='weight')
print(adj_matrix.todense())
```

Output:

```
[[0.         0.33333333 0.33333333 0.33333333 0.         0.        ]
 [0.5        0.         0.         0.5        0.         0.        ]
 [0.33333333 0.         0.         0.33333333 0.33333333 0.        ]
 [0.         0.33333333 0.33333333 0.         0.33333333 0.        ]
 [0.         0.         0.5        0.         0.         0.5       ]
 [0.         0.         0.         0.         0.         0.        ]]
```

Checking if there are any dangling nodes:

```
W.out_degree('A')
W.out_degree('B')
W.out_degree('C')
W.out_degree('D')
W.out_degree('E')
W.out_degree('F')
```

Output:

```
3
2
3
3
2
0
```

We can see that the F page is a dangling node.

Solution: As described in Proposition 2.4.2, we address the issue of dangling nodes using the same approach, implemented in Python.

```
import networkx as nx
import matplotlib.pyplot as plt
W=nx.DiGraph()
nodes=['A', 'B', 'C', 'D', 'E', 'F']
W.add_nodes_from(nodes)
edges=[('A','B', 1/3), ('A','C', 1/3),('A','D', 1/3),('B','A
   ',1/2),('B','D',1/2), ('C','A',1/3),('C','D',1/3),('C','E
   ',1/3),('D','B',1/3),('D','E',1/3),
       ('D','C',1/3),('E','C',1/2),('E','F',1/2)]
W.add_weighted_edges_from(edges)
for node in W.nodes():
    if W.out_degree(node)==0:
        W.add_edge(node, "A", weight=1/len(W.nodes))
        W.add_edge(node, "B", weight=1/len(W.nodes))
        W.add_edge(node, "C", weight=1/len(W.nodes))
        W.add_edge(node, "D", weight=1/len(W.nodes))
        W.add_edge(node, "E", weight=1/len(W.nodes))
        W.add_edge(node, "F", weight=1/len(W.nodes))
        print(f"Out-degree of {node} after fixing: {W.
           out_degree(node)}")
pos = nx.spring_layout(W)
nx.draw(W, pos, with_labels=True, node_color='lightblue',
   node_size=2000, font_size=10, font_weight='bold',
   edge_color='black')
plt.title("Directed Graph W after Fixing Dangling Nodes")
plt.show()
```

Output: `Out-degree of F after fixing:  6`

Double-checking:

```
W.out_degree('F')
```

Output: 6

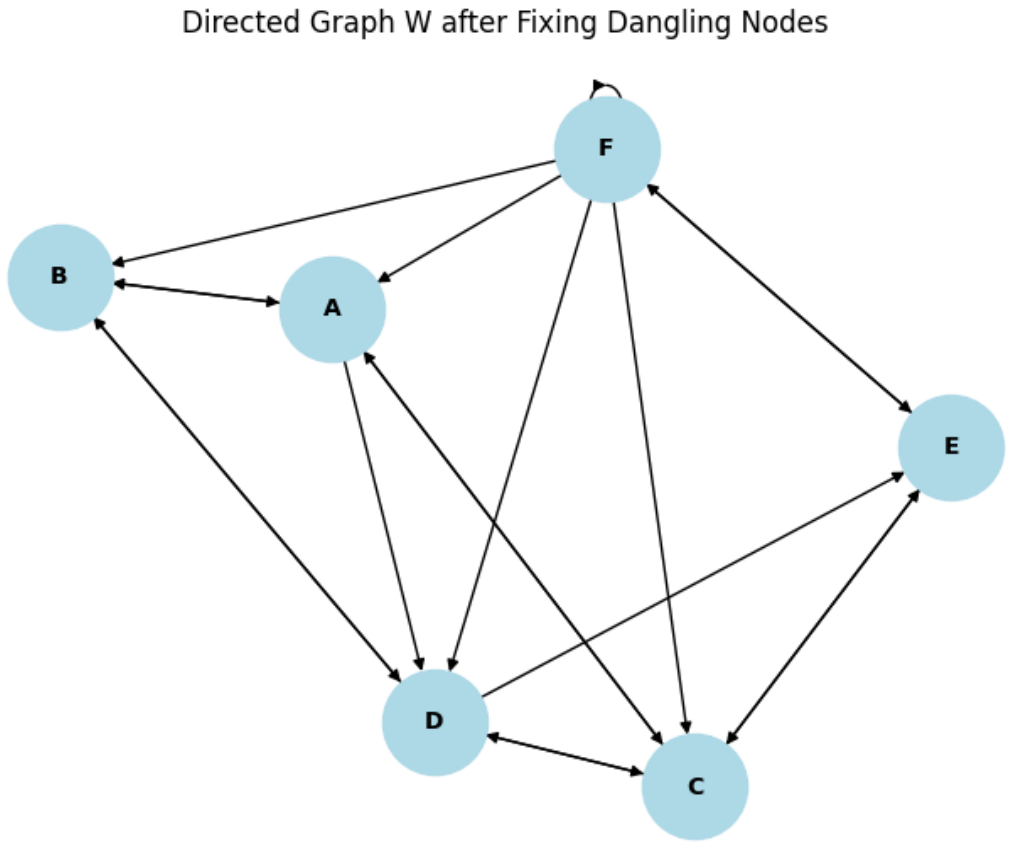


Fig. 3.8: Plot generated from the code above.

After addressing the issue of dangling nodes, the updated transition matrix is as follows:

```
adj_matrix = nx.adjacency_matrix(W, weight='weight')
print(adj_matrix.todense())
```

Output:

```
[[0.         0.33333333 0.33333333 0.33333333 0.         0.        ]
 [0.5        0.         0.         0.5        0.         0.        ]
 [0.33333333 0.         0.         0.33333333 0.33333333 0.        ]
 [0.         0.33333333 0.33333333 0.         0.33333333 0.        ]
 [0.         0.         0.5        0.         0.         0.5       ]
 [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667]]
```

Next, we will proceed with the computation of the Google matrix:

```
import numpy as np
def google_matrix(T, alpha):
    n = T.shape[0]
    E = np.ones((n, n)) / n
    G = alpha * T + (1 - alpha) * E
    return G
T = np.array([[0.0, 0.33333333, 0.33333333, 0.33333333, 0.0,
    0.0],
    [0.5, 0.0, 0.0, 0.5, 0.0, 0.0],
    [0.33333333, 0.0, 0.0, 0.33333333, 0.33333333, 0.0],
    [0.0, 0.33333333, 0.33333333, 0.0, 0.33333333, 0.0],
    [0.0, 0.0, 0.5, 0.0, 0.0, 0.5],
    [0.16666667, 0.16666667, 0.16666667, 0.16666667,
       0.16666667, 0.16666667]])
alpha = 0.85
```

```
G = google_matrix(T, alpha)
print("Google Matrix:\n", G)
```

```
Google Matrix:
 [[0.025      0.30833333 0.30833333 0.30833333 0.025      0.025     ]
 [0.45       0.025      0.025      0.45       0.025      0.025     ]
 [0.30833333 0.025      0.025      0.30833333 0.30833333 0.025     ]
 [0.025      0.30833333 0.30833333 0.025      0.30833333 0.025     ]
 [0.025      0.025      0.45       0.025      0.025      0.45      ]
 [0.16666667 0.16666667 0.16666667 0.16666667 0.16666667 0.16666667]]
```

With the updated transition matrix, we compute the PageRank using Python:

```
pagerank = nx.pagerank(W, alpha=0.85, weight='weight')
print("PageRank:")
for node, rank in pagerank.items():
    print(f"{node}: {rank}")
```

Output:

```
PageRank:
A: 0.16287151524282845
B: 0.14572882210740518
C: 0.2137958726952636
D: 0.20901810495555176
E: 0.16015802747251817
F: 0.10842765752643249
```

Node C has the highest PageRank of 0.2138, indicating it is the most important node, while node F has the lowest PageRank of 0.1084, suggesting it is the least important node.

Chapter 4

Real-Life Application

4.1 Introduction

In the modern era of connectivity, social media has improved how individuals interact, share information, and influence one another. Among these platforms, Twitter stands out as a vital tool for public discourse, enabling millions of users to connect, engage, and exchange ideas instantaneously. Understanding the dynamics of influence within such a vast network is a key focus in network science and social media analysis.

This investigation focuses on network analysis, which examines the relationships between individual entities, specifically Twitter users, and how information flows through these connections. PageRank, a robust technique originally designed for ranking web pages, has since become a popular method for evaluating influence in social networks.

The objective of this study is to apply these concepts to deepen our understanding of the influence structure on Twitter. By examining a dataset of anonymized Twitter interactions, the aim is to uncover patterns of connectivity and identify influential users within the network. These insights have far-reaching implications,

from understanding the spread of information online to identifying key figures in various social and cultural movements.

4.2 Twitter Dataset

The **twitter_combined.txt**[1] file contains a list of edges representing a Twitter network. To protect user privacy, the data has been anonymized, with each number serving as an ID for a user. The users in the first column represent Twitter users who follow others.

4.2.1 Setting Up the Environment

To begin our analysis in Jupyter, we first need to install the necessary libraries to ensure that our code runs properly.

```
!pip install networkx
!pip install matplotlib
```

Next, we import these libraries:

```
import networkx as nx
import matplotlib.pyplot as plt
```

4.2.2 Creating the Directed Graph

We create a DiGraph object using all the edges from the `twitter_combined.txt` file:

```
import networkx as nx
```

[1]This dataset is sourced from the following link, and all the information provided is extracted from there: `https://snap.stanford.edu/data/ego-Twitter.html`.

```
#general form: import library_name as alias
D=nx.DiGraph()
with open('twitter_combined.txt', 'r') as file:
#open the file 'twitter_combined.txt' for reading; r stands for read mode
#general form: with open('file_path', 'mode') as file_variable:
    for line in file:
#to iterate through every line
#general form: for item in iterable:
        user, follower=map(int, line.split())
#line.split() is to split each line into 2 strings; map function is to convert them to integers so they can be added to the graph
#general form: variable1, variable2 = map(conversion_function, iterable.split())
#purpose: split a string into parts, convert each part to a specific type, and unpack the result
        D.add_edge(user, follower) #add a directed edge from user to follower
print(f"Number of nodes: {D.number_of_nodes()}")
print(f"Number of edges: {D.number_of_edges()}")
```

Output:

```
Number of nodes:  81306
Number of edges:  1768149
```

The nodes, representing users in this context, can also be viewed as webpages, while the edges that represent the following links can be seen as hyperlinks. This showcases the diverse applications of the PageRank algorithm.

Remark 2. *Generating a visualization of the entire Twitter network, with over one million edges, would be too large and computationally intensive to handle within the Jupyter Notebook environment. Consequently, only a subset of the network was visualized for practical reasons.*

To create a smaller subset of this rather large network:

```
import matplotlib.pyplot as plt
import networkx as nx
%matplotlib inline
D=nx.DiGraph()
with open('twitter_combined.txt', 'r') as file:
    for line in file:
        user, follower=map(int, line.split())
        D.add_edge(user, follower)
subgraph_nodes = list(D.nodes())[:100] #selects the first
   100 nodes from the full graph D to create a smaller
   subgraph
subgraph = D.subgraph(subgraph_nodes) #creates the subgraph
pos = nx.spring_layout(subgraph, seed=42) #generates the
   layout positions for the nodes in the subgraph, arranging
    them for a clear visualization
nx.draw_networkx(subgraph, pos, node_size=50, node_color='
   blue')
plt.figure(figsize=(12, 12)) #sets the size of the figure to
    12x12 inches for better visibility
plt.title("Subset of Large Network")
plt.show()
```

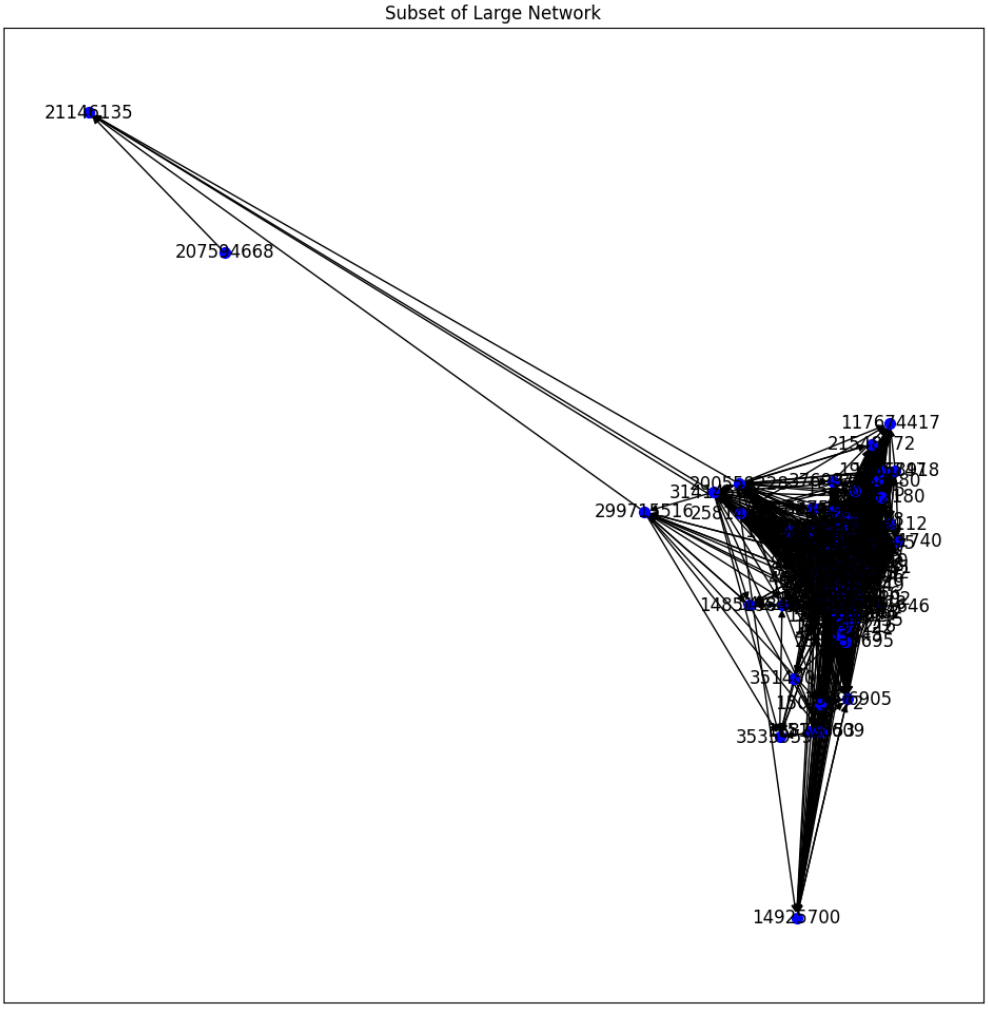


Fig. 4.1: Plot generated from the code above.

4.2.3 PageRank Calculation

Subsequently, we calculate the PageRank for this large network:

```
import networkx as nx
D = nx.DiGraph()
with open('twitter_combined.txt', 'r') as file:
    for line in file:
        user, follower = map(int, line.split())
        D.add_edge(user, follower)
page_rank = nx.pagerank(D)
print("PageRank:")
for node, rank in page_rank.items(): # print each user ID
   and their PageRank score from the dictionary(page_rank.
   items())
    print(f"User {node}: {rank:.6f}")
```

Remark 3. *The output of the PageRank computation is extensive and includes scores for all nodes in the graph. Only the PageRank scores for the 20 generated users by Python are shown below for brevity. The complete PageRank results can be found in Appendix A.5.*

Output:

```
PageRank:
User 214328887: 0.000031
User 34428380: 0.000668
User 17116707: 0.000056
User 28465635: 0.000132
User 380580781: 0.000066
User 18996905: 0.000287
```

```
User 221036078: 0.000016
User 153460275: 0.000029
User 107830991: 0.000048
User 17868918: 0.000270
User 151338729: 0.000122
User 222261763: 0.000041
User 19705747: 0.000111
User 88323281: 0.000147
User 19933035: 0.000008
User 149538028: 0.000076
User 158419434: 0.000080
User 17434613: 0.000052
User 153226312: 0.000075
User 364971269: 0.000009
```

4.2.4 Overview of the Top 10 Ranked Users

We then write the code to generate the output for the top 10 users:

```
import networkx as nx
import heapq # this is a Python module that provides
   functions for working with heaps (a type of priority
   queue) and it is used to efficiently retrieve the
   smallest or largest elements from a collection
D = nx.DiGraph()
with open('twitter_combined.txt', 'r') as file:
    for line in file:
        user, follower = map(int, line.split())
        D.add_edge(user, follower)
```

```
page_rank = nx.pagerank(D)
top_10_users=heapq.nlargest(10, page_rank.items(), key=
    lambda x: x[1])
# heapq.nlargest(n, iterable, key=None):
# finds the n largest elements from iterable
# n: number of largest elements to retrieve
# iterable: the collection to search through
# key: optional function to specify which part of each item
    to use for comparison
# lambda x: x[1] extracts the second element of each tuple
    for sorting, 0 if we want the first element
print("Top 10 Most Influential Users and Their PageRank
    Scores:")
for user, rank in top_10_users:
    print(f"User {user}: {rank:.6f}")
```

Output:

```
Top 10 Most Influential Users and Their PageRank Scores:
User 115485051: 0.004331
User 116485573: 0.004133
User 813286: 0.002339
User 40981798: 0.001372
User 7861312: 0.001254
User 11348282: 0.001222
User 17093617: 0.001076
User 15439395: 0.001031
User 18396070: 0.001029
User 14230524: 0.001014
```

4.2.5 Analysis of In-Degree and Out-Degree for Top 10 Ranked Users

In this subsection, we examine the in-degree and out-degree of the top 10 users based on their PageRank scores. The in-degree represents the number of followers each user has, while the out-degree indicates how many users they follow. Special attention is given to the second-highest ranked user to understand why they might have such a high PageRank, exploring their network position and interaction patterns.

In-degree and out-degree of the top 10 users:

```
print("In-degree and Out-degree of Top 10 Users:")
for user, _ in top_10_users: # _ is used when we want to
   omit the second variable and it is not important in the
   context
#we iterate through the 10 users
    in_degree = D.in_degree(user)
    out_degree = D.out_degree(user)
#we calculate the in and out degrees and then we print them
    print(f"User {user} - In-degree: {in_degree}, Out-degree
       : {out_degree}")
```

Output:

```
In-degree and Out-degree of Top 10 Users:
User 115485051 - In-degree: 3383, Out-degree: 1
User 116485573 - In-degree: 4, Out-degree: 1
User 813286 - In-degree: 2647, Out-degree: 1111
User 40981798 - In-degree: 3216, Out-degree: 119
User 7861312 - In-degree: 2074, Out-degree: 224
```

```
User 11348282 - In-degree: 1707, Out-degree: 172
User 17093617 - In-degree: 1186, Out-degree: 687
User 15439395 - In-degree: 1108, Out-degree: 334
User 18396070 - In-degree: 265, Out-degree: 45
User 14230524 - In-degree: 1214, Out-degree: 62
```

Analyzing the second-highest-ranked user:

```
second_highest_user = top_10_users[1][0] #from a list with
    tuples [(user1, rank1), (user2, rank2),...] [1] means we
    take the second tuple and [0] indicates that we are
    interested in the first item of that tuple, i.e: user2
second_highest_in_degree = D.in_degree(second_highest_user)
second_highest_out_degree = D.out_degree(second_highest_user
    )
print(f"Second highest ranked user: {second_highest_user}")
print(f"In-degree: {second_highest_in_degree}")
print(f"Out-degree: {second_highest_out_degree}")
```

Output:

```
Second highest ranked user: 116485573
In-degree: 4
Out-degree: 1
```

We observe that the user has relatively few incoming and outgoing links. To understand why this user is ranked so highly, we examine the **'in_edges'** and **'out_edges'** functions in NetworkX.

```
D.out_edges(second_highest_user)
D.in_edges(second_highest_user)
```

Output:

```
OutEdgeDataView([(116485573, 115485051)])
InEdgeDataView([(114636253, 116485573), (115485051, 116485573),
(11625912, 116485573), (12771872, 116485573)])
```

The second-highest-ranked user is significant because the top-ranked user, 115485051, follows them. This is shown in the code above. This connection highlights the central role of the second-highest-ranked user in the network. In the PageRank algorithm, when a more important page links to a less important one, it boosts the latter's rank. This is what occurs in this case: the second-highest-ranked user gains significance because they are followed by the top-ranked user, 115485051.

4.3 Final Thoughts

This study applied PageRank to a portion of the Twitter network to identify key users and understand their roles within the network. The findings revealed that the second-highest-ranked user, despite having only a few direct connections, is highly influential due to being followed by the top-ranked user.

This highlights an important aspect of PageRank: influence is determined not just by the number of connections, but also by the significance of the users or pages making those connections. Endorsements from high-status individuals or pages can significantly elevate a user's or page's ranking, illustrating how influence and importance are distributed in complex ways within a network.

Pages that are linked to by other significant pages are considered more important themselves, creating a hierarchy of relevance. A link from a highly respected page (like a major news site or a renowned academic journal) carries more weight than a link from a blog.

Conclusion

Throughout the dissertation the PageRank algorithm is explored, starting with examining the foundational concepts of Markov chains through practical examples. After explaining the basis, practical examples were also used to illustrate how Markov chains are integrated within the PageRank algorithm. It was demonstrated that pages, or nodes in the Markov chain model, can be represented as a directed graph, where links denote hyperlinks with positive probabilities, captured in the transition matrix, consistent with the Markov process nature of PageRank.

Furthermore, the PageRank algorithm was implemented using Python and the NetworkX library. This approach enabled the representation of graphs and data using a programming language, rather than manually, allowing for the analysis of larger datasets, including a Twitter dataset with over 80,000 PageRank calculations.

PageRank is extended beyond web analysis to various domains. Consider a political analysis, where nodes could represent political parties or politicians, and edges could signify alliances or collaborations. A high PageRank score in this context might indicate significant influence or centrality within the network.

On a broader level, viewing life events through the lens of a Markov chain offers a unique perspective. Calculating PageRank in this context could help identify which life events are most central in shaping personal journeys. Major life milestones–such as going to school, graduating, or getting married–are modelled

as nodes in the chain. Transitions between these events occur with certain positive probabilities, creating links or edges that represent new achievements. The concepts of irreducibility and aperiodicity in life's transitions are also evident: one can graduate and get married, then later return to school and graduate again, illustrating the absence of a fixed timeline or predetermined path. Mathematically, each individual's PageRank scores are tailored and optimal to their unique life path.

Appendix

A.1 Dataset Overview

Dataset Name: Twitter Combined Dataset

Number of Nodes: 81,306

Number of Edges: 1,768,149

A.2 Sample Data

Below is a sample of the dataset:

```
214328887 34428380
17116707 28465635
380580781 18996905
221036078 153460275
107830991 17868918
...
```

A.3 External Resources

The full dataset is available at `https://snap.stanford.edu/data/ego-Twitter.html`.

A.4 Sample PageRank Results

Below is a sample of the PageRank results for the dataset. This sample includes a subset of user rankings with their corresponding PageRank scores:

```
PageRank:
User 214328887: 0.000031
User 34428380: 0.000668
User 17116707: 0.000056
User 28465635: 0.000132
User 380580781: 0.000066
...
```

A.5 PageRank Results

The PageRank results for all nodes in the dataset were calculated using the PageRank algorithm in NetworkX. The full results can be accessed online at the following link:

PageRank Results.

Bibliography

[1] C. A. Davis, S. Fortunato, and F. Menczer, *A first course in network science*, ch. Network Elements, Cambridge University Press, 2020.

[2] Stewart N. Ethier, *The doctrine of chances: Probabilistic aspects of gambling*, Springer, 2010.

[3] J. Humpherys, T. J. Jarvis, and E. Evans (eds.), *Foundations of applied mathematics, lab manual for volume 1*, ch. The PageRank Algorithm, Brigham Young University, Department of Mathematics, 2024, Available at `http://foundations-of-applied-mathematics.github.io/`.

[4] R. Kumar, A. G. K. Leng, and A. K. Singh, *Application of markov chain in the pagerank algorithm*, Transition **1** (1912).

[5] J. Leskovec and J. Mcauley, *Ego-twitter dataset (twitter combined)*, `https://snap.stanford.edu/data/ego-Twitter.html`, 2012, Stanford Network Analysis Project (SNAP).

[6] David A. Levin and Yuval Peres, *Markov chains and mixing times*, vol. 107, American Mathematical Soc., 2017.

[7] H. A. Mimun, *Gambling: Probability and decision - exercises 9*, 2021.

[8] H. A. Mimun, *Notes and slides of the course "gambling: Probability and decision"*, https://sites.google.com/view/hlafoalfie-mimun/teaching/gambling-probability-and-decision-20202021, 2023.

[9] B. Moor, *Mathematics behind google's pagerank algorithm*, Ph.D. thesis, 2018.

[10] James R. Norris, *Markov chains*, no. 2, Cambridge University Press, 1998.

[11] S. M. Ross, *Introductory statistics*, Academic Press, 2017.