The Page-Rank Algorithm and Google's Search Engine

By

YAEL KATZIR

MSci Mathematics Dissertation

School of Mathematics
Faculty of Science
University of Bristol

A 40 Credit-Point (MSci) dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Masters in Mathematics with Study Abroad in the Faculty of Science.

July 19, 2024

*Author's declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University's Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate's own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: ..... Yael Katzir ..... DATE: ..... 29/04/2024 .....

Abstract

Google and the internet have become an integral part to our daily lives, yet we often overlook the intricate processes and mechanisms involved in retrieving information.
This project explores how web adaptation has evolved, particularly following the introduction of the page-rank algorithm, focusing on Google's search engine. Various techniques, such as page click rates, loading times, key-word relevance, and other user metrics which are incorporated alongside the page-rank algorithm into Google's search engine will be uncovered.
Through this investigation, I aim to gain a deeper understanding of the sophisticated strategies employed by modern search engines, shedding light on the constantly changing structure of information retrieval in the digital age.

Contents

Listings

1 Introduction

In the year 2024, the internet contains approximately 1.09 billion websites engaged in various realms of content as mentioned in [NJ, 2024]. From news and weather updates to music, maps, games, social media, and educational materials. Growing up in the digital age, I entered a world where the internet was already a central part of each individual's daily life yet I rarely wondered about the mechanism behind how it all works.

Even today, the creation and deletion of websites continues, contributing to an ever-expanding pool of pages accessible to users through simple keyword searches on different search engines.

[OH, 2016] describes the evolution of web-engines, labels 'Archie', which emerged in 1990 by Alan Emtage, as the foundational milestone in search engine development. It was the first search engine which introduced the concept of indexing, enabling users to locate specific files based on their filenames within the indexed server.

The evolution of search engines took a significant step with the launch of 'web crawlers' where these automated programmes systematically crawled the web, constantly updating new content.

Later on 'JumpStation' marked another significant advancement in search engine technology. It was the first engine to implement a web crawler which catalogues the titles and headings of websites as well as filenames, so when users submitted queries via JumpStation's, they received a list of URLs relevant to their search terms.

Following the improvements of JumpStation, 'WebCrawler' took the algorithm even a step further by indexing the entire text (as well as headings and titles) of each web-page travelled by its web crawler.

1.1 The history of Google and the page-rank algorithm

Google, founded in 1998 by Sergey Brin and Larry Page, is an American search engine company. [Mark Hall, 2024] states that Google has been dominating more than 70% of global online search queries, it stands as a foundation of the internet experience for most users worldwide, solidifying its position as one of the most recognizable brands globally.

Google's influence extends far beyond its origins as a search engine, offering over 50 internet services and products ranging from email and online document creation to mobile phone and tablet software and have also expanded into hardware sales. This diverse product portfolio and expansive reach establish Google as one of the key players in the high-tech industry, alongside giants like Apple, IBM, and Microsoft.

Despite its diverse offerings, Google's original search tool remains fundamental to its success, generating the majority of its revenue through targeted advertising based on user search queries. Google's [Google,] current mission stands as "Our mission is to organise the world's information and make it universally accessible and useful"

When observing Google's timeline in Figure 1, we can notice a boosted growth and expansion of the company, which has led it to be how we know it today.

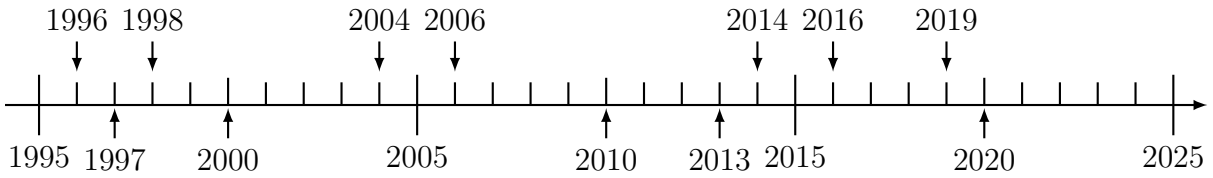


Figure 1: Google's timeline

Year - 1996 According to [Pannu, 2023], Larry Page and Sergey Brin created an algorithm with the idea of sorting and ranking the web-pages based on how many links there are between them as their PhD research project for Stanford University labelled under "BackRub", and by August 29th the first version of Google was created.

Year : 1997 The domain Google.com was registered, with the name Google derived from the misspelling of the word "Googol" - the number $1.0 * 10^{100}$ as outlined in [Mark Hall, 2024].

Year : 1998 Google rapidly indexed over 60 Million web-pages.

Year : 2000 Google introduced the sales of advertisements associated with search key-words.

Year : 2004 Google launched Gmail, Google-Docs and Google-Drive.

Year : 2005 Google launched Google Maps

Year : 2006 Google purchased Youtube for the price of $1.65 Billion.

Year : 2010 Google banned China from using the search engine, due to a Chinese sophisticated Phishing attack on their system.

Year : 2013 Chrome-Cast was launched.

Year : 2014 Google has delved into the AI world.

Year : 2015 Google's logo was changed, alongside the introduction of of smart home products such as the Google-Assistant.

Year : 2016 Google has developed their own smart devices and electronics, such as the Google Pixel - smart phone.

Year : 2019 Google introducted some changes to their search engine such as including shortcuts such as "People also ask" which pops up similar questions, and images are included on the search bar.

2020 onwards: Google is constantly and changing what features they offer, how their search engine works and their algorithms.

More information about the detailing of the expansion of Google in each year can be found in [Stumbles, 2017] and [Ray, 2024].

Before the introduction of Google, every query launched into the search bar resulted in a flood of relevant but unsorted pages based on key-words, where although the returned pages relate to the query, a web-crawler cannot be sure which one truly held the answers they are looking for.
Then came the page-rank algorithm. A game-changer in the digital world. The algorithm revolutionised the way search results were ranked, making the digital landscape more navigable and optimised.

As mentioned in the first year or the timeline, Larry Page and Sergey Brin created the page-rank algorithm with the idea of sorting and ranking the web-pages based on how many links there are between them. Each page published on google, could be a website, and image or even just an advert, may or may not choose to include links which lead to other pages.
For example, If we look at the Blog in Figure 2:

Figure 2: A blog's ranking to food websites in 2023 [Arellano, 2023]

We see that Kelly Arellano ranks her top 8 best recipe websites. The actual page-rank ranking of these websites in the blog does not directly affect the page-rank ranking of these websites, but just the fact that this blog has chosen to include outwards links towards these pages, "boosts" these websites rank up with the idea of them being more desirable or relevant.

I'll delve deeper into this during my research, but to put it simply, a page with numerous incoming links will be ranked higher than one with few or no links.

2 The Page-Rank algorithm

The page-rank algorithm begins with a directed graph, often referred to as the web graph, such that:

- Each node in the graph represents a unique web page on the internet and can be thought of as entities that hold information and content accessible via a URL.
- Each directed edges represent the connections between web pages as a URL, so that an edge from node A to node B indicates that there is a link from page A to page B with the direction of an edge indicating the flow of influence or connectivity between the pages.

2.1 One-Step Web Crawler

Figure 3 was created to illustrate a simpler version of the algorithm using a mock Web-Graph. Its aim is to make the algorithm easier to understand by showing it in action on a smaller example and make it easier for readers to grasp how the algorithm works without getting overwhelmed by real-world data.
Essentially, Figure 3 is a helpful tool for breaking down the algorithm's concepts into more manageable pieces, making learning about it less intimidating and more accessible.

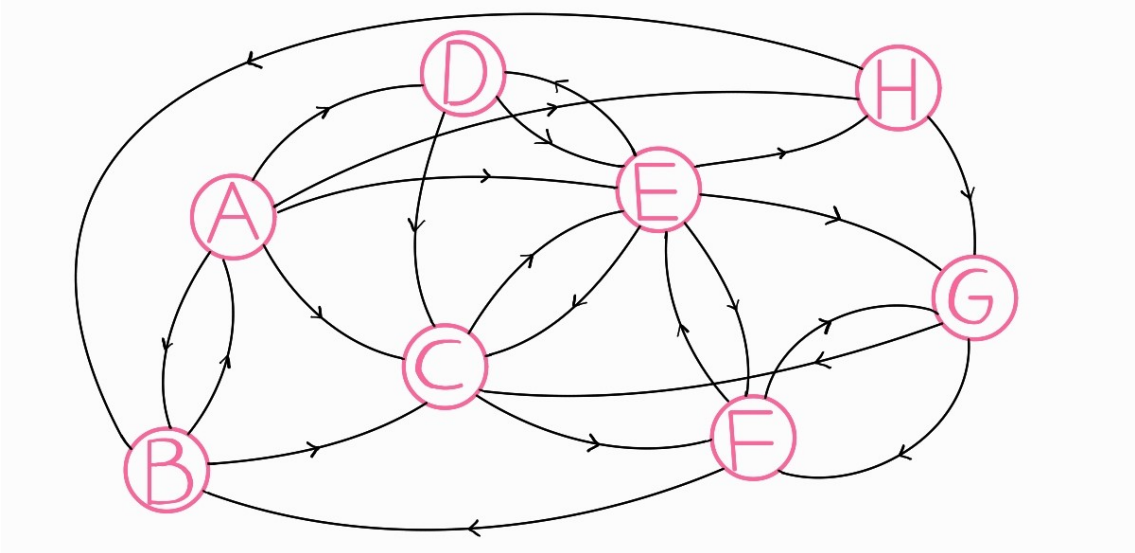


Figure 3: A sample of a web system

In Figure 3 we can see a system of $n = 8$ web-pages (nodes) connected to each other, with direction, through multiple links (edges). In this example I have labelled each web-page as an element from the set $\{A, B, C, D, E, F, G, H\}$ for simplicity. We can clearly see that:

- A has links to pages B, C, D, E, H.
- B has links to pages A, C
- C has links to pages E, F
- D has links to pages C, E
- E has links to pages C, D, F, G, H
- F has links to pages B, E, G
- G has links to pages C, F
- H has links to pages B, G

If we assume that reaching any page linked to a certain page has an equal probability, then moving from page A to page B has the same likelihood as moving from page A to page C, etc.

We can represent the probability of transitioning from one web-page to another in just one step as a list of probabilities.

$$\begin{gathered}
\mathcal{P}(B \mid A) = \mathcal{P}(C \mid A) = \mathcal{P}(D \mid A) = \mathcal{P}(E \mid A) = \mathcal{P}(H \mid A) = (\tfrac{1}{5}) \\
\mathcal{P}(A \mid B) = \mathcal{P}(C \mid B) = (\tfrac{1}{2}) \\
\mathcal{P}(E \mid C) = \mathcal{P}(F \mid C) = (\tfrac{1}{2}) \\
\mathcal{P}(C \mid D) = \mathcal{P}(E \mid D) = (\tfrac{1}{2}) \\
\mathcal{P}(C \mid E) = \mathcal{P}(D \mid E) = \mathcal{P}(F \mid E) = \mathcal{P}(G \mid E) = \mathcal{P}(H \mid E) = (\tfrac{1}{5}) \\
\mathcal{P}(B \mid F) = \mathcal{P}(E \mid F) = P\mathcal{P}G \mid F) = (\tfrac{1}{3}) \\
\mathcal{P}(C \mid G) = \mathcal{P}(F \mid G) = (\tfrac{1}{2}) \\
\mathcal{P}(B \mid H) = \mathcal{P}(G \mid H) = (\tfrac{1}{2})
\end{gathered}$$

We can clearly see that the following probabilities equate to 0 due to each row in the previous probability list adding up to 1.

$$\begin{gathered}
\mathcal{P}(A \mid A) = \mathcal{P}(F \mid A) = \mathcal{P}(G \mid A) = 0 \\
\mathcal{P}(B \mid B) = \mathcal{P}(D \mid B) = \mathcal{P}(E \mid B) = \mathcal{P}(F \mid B) = \mathcal{P}(G \mid B) = \mathcal{P}(H \mid B) = 0 \\
\mathcal{P}(A \mid C) = \mathcal{P}(B \mid C) = \mathcal{P}(C \mid C) = \mathcal{P}(D \mid C) = \mathcal{P}(G \mid C) = \mathcal{P}(H \mid C) = 0 \\
\mathcal{P}(A \mid D) = \mathcal{P}(B \mid D) = \mathcal{P}(D \mid D) = \mathcal{P}(F \mid D) = \mathcal{P}(G \mid D) = \mathcal{P}(H \mid D) = 0 \\
\mathcal{P}(A \mid E) = \mathcal{P}(B \mid E) = \mathcal{P}(E \mid E) = 0 \\
\mathcal{P}(A \mid F) = \mathcal{P}(C \mid F) = \mathcal{P}(D \mid F) = \mathcal{P}(F \mid F) = \mathcal{P}(H \mid F) = 0 \\
\mathcal{P}(A \mid G) = \mathcal{P}(B \mid G) = \mathcal{P}(D \mid G) = \mathcal{P}(E \mid G) = \mathcal{P}(G \mid G) = \mathcal{P}(H \mid G) = 0 \\
\mathcal{P}(A \mid H) = \mathcal{P}(C \mid H) = \mathcal{P}(D \mid H) = \mathcal{P}(E \mid H) = \mathcal{P}(F \mid H) = \mathcal{P}(H \mid H) = 0
\end{gathered}$$

Definition 2.1 (Transition matrix)**.** *The Transition matrix with entries $P = (p_{i,j})$, such that $p_{i,j} = \mathcal{P}(j \mid i), \quad \forall i, j \in \{A, B, C, D, E, F, G, H\}$ Where i represents the rows and j represents the columns.*

The transition matrix of the graph in Figure 3 is displayed below and is labelled as P:

$$P = \begin{bmatrix}
0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\
0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0
\end{bmatrix} \tag{1}$$

Example 2.1. *To find $\mathcal{P}(B \mid H)$ we look at row H (Eighth Row), and then observe column B (Second Column) which equals $\frac{1}{2}$.*

We can observe that if we choose any starting point, and multiply it by P, we get a vector that represents all the conditional probabilities in the row of that starting point.
For example, choosing F as a starting point results in all the conditional probabilities given F:

$$F^T \cdot P = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix}
0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} \\
\frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\
0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\
0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\
0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\
0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\
0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0
\end{bmatrix} = \begin{bmatrix} 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \end{bmatrix}$$

2.2 Infinite Web Crawler

As mentioned in Section 2.1 , the probabilities we provided only represent the weight of getting from one web-site to another through one edge only. We also have to consider the possibility of getting from one website to another through a series of 'stops'.
For example, we can get from E to F in a few possible ways:

- $E \to F$
- $E \to G \to F$
- $E \to H \to B \to C \to F$

We can also be in situations where we enter infinite loops such as where a web-traveller jumps from website C to E infinitely many times before reaching F:

- $E \to C \to E \to C \to E \to C \to E \to \cdots\cdots \to C \to F$

Definition 2.2 (**A random Process**, Definition 9.1 of [Rousseau et al., 2008])**.** *A random process* $\{X_k \,|k= 0, 1, 2, 3, \cdots\}$ *is a family of random variables parameterized by the integer* k*.*

Remark. *We let each of these random variables* X_k *takes its value from a finite set* $T = A, B, C, D, E, F, G, H$*. For each step* $k \in \{0, 1, 2, \cdots\}$*, the position of the web surfer is* X_k*.*

Now that we know what a Random Process is we can state that getting from $E \to F$ in two steps is the sum of probabilities of taking the routes $E \to C \to F$ and $E \to G \to F$ (the only possible 2-step routes) which gives us the probability for $\mathcal{P}(X_2 = F)$ given starting at$E$.

We find that the result for the sum of probabilities of all routines starting at position E and ending at column F if we multiply E^T (A vector of all 0s except for a signle 1 in the 5^{th} slot) by $P \cdot P$ or $E^T \cdot P^2$ and observe the result.

Example 2.2.

$$\begin{bmatrix}0 & 0 & 0 & 0 & 1 & 0 & 0 & 0\end{bmatrix} \cdot \begin{bmatrix} 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix} \cdot \begin{bmatrix} 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 \end{bmatrix}$$

Which equals

$$\begin{bmatrix}0 & \frac{1}{6} & \frac{1}{5} & 0 & \frac{4}{15} & \frac{1}{5} & \frac{1}{6} & 0\end{bmatrix}$$

We can now generalise what we have seen above.
For a route with k steps, we multiply the desired starting point by the matrix P, raised to the power k, to get a vector of probabilities of landing at each destination after k number of steps. After checking multiple choices for k large in the page below, we can notice that as $k \to \infty$ the matrix P^k converges to an equilibrium. See matrices in Figure 4.

$$
P^{10} = \begin{bmatrix}
0.0523399 & 0.1048478 & 0.1964591 & 0.0511703 & 0.2028590 & 0.2061143 & 0.1350393 & 0.0511703 \\
0.0525376 & 0.1046216 & 0.1968204 & 0.0508614 & 0.2030991 & 0.2066227 & 0.1345759 & 0.0508614 \\
0.0522816 & 0.1047771 & 0.1958544 & 0.0510844 & 0.2035863 & 0.2065216 & 0.1348102 & 0.0510844 \\
0.0525124 & 0.1046403 & 0.1968813 & 0.0507825 & 0.2032111 & 0.2067078 & 0.1344821 & 0.0507825 \\
0.0523586 & 0.1048156 & 0.1963398 & 0.0512050 & 0.2028877 & 0.2061262 & 0.1350620 & 0.0512050 \\
0.0524184 & 0.1048625 & 0.1966790 & 0.0511422 & 0.2026449 & 0.2059729 & 0.1351380 & 0.0511422 \\
0.0523974 & 0.1047001 & 0.1965917 & 0.0509716 & 0.2031690 & 0.2065634 & 0.1346352 & 0.0509716 \\
0.0524293 & 0.1049350 & 0.1969465 & 0.0512656 & 0.2021531 & 0.2056264 & 0.1353784 & 0.0512656
\end{bmatrix}
$$

$$
P^{100} = \begin{bmatrix}
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806
\end{bmatrix}
$$

$$
P^{1000} = \begin{bmatrix}
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806 \\
0.0523903 & 0.1047806 & 0.1964636 & 0.0510806 & 0.2030124 & 0.2062868 & 0.1349050 & 0.0510806
\end{bmatrix}
$$

Figure 4: Convergence of the Transition matrix

2.3 The Power Method and Convergence of the Transition matrix

Definition 2.3 (**Markov Chains**, Definition 9.1 of [Rousseau et al., 2008]). *Let* $\{X_k \mid k = 0, 1, 2, 3, \cdots\}$ *be a random process taking its values from the set* $T = A, B, C, \cdots$. *We say that* X_k *is a Markov chain if the probability* $\mathcal{P}(X_k = j), j \in T$, *depends only on the value of the process at the previous step,* X_{k-1}. *It does not depend on any of the preceding steps;* $X_{k-2}, X_{k-3}, \cdots$. *We define* $n < \infty$ *as the number of elements in* T.

We can recognise that in order to calculate the Probability of the position after 1 step, we used the data on the probabilities from position X_0, and when calculating the Probability of the position X_2, after 2 step, we used the data on the probabilities from position X_1. This exactly aligns with the properties of being able to calculate $\mathcal{P}(X_k)$ from $\mathcal{P}(X_{k-1})$

Since Markov chains exactly describe the process of our web traveling, the elements of the transition matrix P (as described previously) can now be treated in the following manner:

$$p_{i,j} = \mathcal{P}(X_k = j | X_{k-1} = i) \tag{2}$$

Definition 2.4 (**Stochastic matrices**, Definition 5.6.2 of [Dan Margalit, 2019]). *A square matrix* P *is right stochastic / stochastic if all of its entries are non-negative, and the entries of each row sum to* 1.

Definition 2.5 (**Eigen-values and Eigen-vectors**, As described in [Weisstein, a]). *Let* P *be a linear transformation represented by a matrix* P *ad described above. If there is a vector* $0 \neq x \in (\mathbb{R}^{\times})^{\mathbb{T}}$ *such that* $x^T \cdot P = \lambda \cdot x^T$ *for some scalar* λ, *then* λ *is called the eigenvalue of* P *with corresponding eigen-vector* x^T.

Lemma 2.1. *Let* P *be an* $n \times n$ *right stochastic matrix, similarly to the transition, then:*

1. *At least one eigenvalue* λ_i *is equal to* 1. *Property 9.2 of [Rousseau et al., 2008]*
2. *All Eigenvalues* $\lambda_i \; \forall i = \{1, 2, \cdots, n\}$ *satisfy* $|\lambda_i| \leq 1$. *Property 9.3 of [Rousseau et al., 2008]*

Proof. 1. In order to find the value of an Eigenvalue of a matrix, we have to check for which λ does $\det(P - \lambda I) = 0$, where det() is the determinant of the matrix.

By definition of determinant we have that $\det(A) = \det(A^T)$ therefore we have that $\det(P - \lambda I)^T = \det(P - \lambda I)$ and so $\det(P^T - \lambda I)$ =$\det(P - \lambda I)$, therefore P and P^T have the same eigen-values.
We can now replace matrix P with P^T when finding Eigenvalues.

If we let vector π be the vector of size $n \times 1$ with its elements beings all 1's, we get that

$$\pi^T \cdot P^T = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} \cdot \begin{bmatrix} p_{11} & p_{12} & p_{13} & \cdots & p_{1n} \\ p_{21} & P_{22} & \ddots & \ddots & \vdots \\ p_{31} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ p_{n1} & \cdots & \cdots & \cdots & p_{nn} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} p_{i1} & \sum_{i=1}^{n} p_{i2} & \cdots & \sum_{i=1}^{n} p_{in} \end{bmatrix}$$

$$= \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} = 1 \cdot \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix} = 1 \cdot \pi^T = \lambda \cdot \pi^T.$$

So $\pi^T \cdot P^T = 1 \cdot \pi^T$ therefore $\lambda = 1$ is an eigen-value for P^T.

As we have shown before, P^T and P have the same eigenvalues so we can conclude that $\lambda = 1$ is also an eigenvalue for P.

2. To show that all Eigenvalues $\lambda_i \ \forall i = \{1, 2, \cdots, n\}$ of a right stochastic matrix satisfy $|\lambda_i| \leq 1$ we first observe the relations of eigenvalues and eigen-vectors under the 1-norm. We are given that for all $x \in \mathbb{R}^n$ we have that

$$x^T \cdot P = x^T \cdot \lambda, \text{ so: } ||x^T \cdot P||_1 = ||x^T \cdot \lambda||_1 = |\lambda| \cdot ||x^T||_1.$$

We can also observe that:

$$||x^T \cdot P||_1 = |x_1^T \cdot p_{11} + \cdots + x_n^T \cdot p_{n1}| + \cdots + |x_1^T \cdot p_{1n} + \cdots + x_n^T \cdot p_{nn}| \leq$$

$$\leq (|x_1^T| \cdot p_{11} + \cdots + |x_n^T| \cdot p_{n1}) + \cdots + (|x_1^T| \cdot p_{1n} + \cdots + |x_n^T| \cdot p_{nn}) =$$

$$= |x_1^T| \cdot (p_{11} + \cdots + p_{n1}) + \cdots + |x_n^T|(p_{1n} + \cdots + p_{nn}) =$$

$$= |x_1^T| + \cdots + |x_n^T| = ||x^T||_1 \text{ Since each } (p_{1j} + \cdots + p_{nj}) \text{ sums up to 1 by definition}$$

Therefore, using both results, we have that:

$$||x^T \cdot P||_1 \leq ||x^T||_1 \text{ and, } ||x^T \cdot P||_1 = |\lambda| \cdot ||x^T||_1 \text{ so, } |\lambda| \cdot ||x^T||_1 \leq ||x^T||_1$$

So we can conclude that the only way to achieve this result is by having $|\lambda| \leq 1$.

□

Definition 2.6 (**Spectral Radius**, from [Weisstein, b])**.** *For an $n \times n$ matrix P, with eigenvalues $\{\lambda_j\}$, we have that the Spectral radius $\rho(P)$ is defined by:*

$$\rho(P) = \max\{|\lambda_1|, \cdots\cdots, |\lambda_n|\}$$

Lemma 2.2. *[Chapter 16 from [Hefferon, 2011]] Let $v_1, v_2, \cdots$ be eigen-vectors of some linear transformation P with the same eigenvalue r. Then any linear combination of $v_1, v_2, \cdots$ is also an eigen-vector of P with the same eigenvalue r.*

Proof. Since P is a linear transformation, we have that $P(c_1v_1 + c_2v_2 + \cdots) =$
$c_1Pv_1 + c_2Pv_2 + \cdots$ by the linearity of P. And, $c_1rv_1 + c_2rv_2 + \cdots = r(c_1v_1 + c_2v_2 + \cdots)$ since r is an eigenvalue of P.
Therefore $(c_1v_1 + c_2v_2 + \cdots)$ is an eigen-vector of P with the same eigenvalue. □

Definition 2.7 (**Reducible** & **Irreducible Matrices**, From [Royle and Weisstein,])**.** *A square $n \times n$ matrix $P = p_{ij}$ is called reducible if the indices $1, 2, \cdots, n$ can be divided into two disjoint nonempty sets $i_1, i_2, ..., i_\mu$ and $j_1, j_2, ..., j_\nu$ (with $\mu + \nu = n$) such that $p_(i_\alpha, j_\beta) = 0$ for $\alpha = 1, 2, \cdots, \mu$ and $\beta = 1, 2, \cdots, \nu$.*
Further, a matrix is reducible if and only if it can be placed into block upper-triangular form by simultaneous row/column permutations.

*A square matrix that is not reducible is said to be **Irreducible**.*

Next we'll introduce an extraction of Perron-Frobenius Theorem without proof:

Theorem 2.3 (**First extraction of Perron-Frobenius Theorem**, pg.27 from [Wickerhauser, 2022])**.** *For any irreducible, positive, $n \times n$ matrix P with spectral radius $r = \rho(P)$, be an eigenvalue of P, then there exists a positive r-eigenvectorv of P, namely $v = (v_1, ..., v_n)$ with $v_i > 0 \quad \forall i$*

The original theorem proves a few more ideas, but for the sake of my research I will only focus on proving the second extraction in Theorem 2.4 which will allow me to show that every other eigenvalue λ of P, satisfies $|\lambda| < r$.
The uniqueness of r as a maximal value will be useful to the proof of the power method in Theorem 2.6 when we prove the convergence of π.

Theorem 2.4 (**Second extraction of Perron-Frobenius Theorem**, pg.27 [Wickerhauser, 2022])**.** *For any irreducible, positive, $n \times n$ matrix P with spectral radius $r = \rho(P)$, be an eigenvalue of P, we have that r is a simple eigenvalue. So the dimension of the eigenspace is* 1 *and that the eigen-vector corresponding to r is unique.*

Proof. We will show that r is a simple eigenvalue using contradiction.

Let v be a positive eigen-vector to P with the eigenvalue r.
Assume that r is not a simple eigenvalue, so we can let u be another eigen-vector such that v, u are two independent eigenvectors.

Since it is unknown whether u is positive or negative, we can assume two cases,
Case 1) u has some or all positive components.
Define w as the linear combination between v, u such that $w = v - \alpha u$ for α multiplicative scalar. Then we can maximise α to be such that all components of w are non-negative. This implies that at least one component of w will have the 0 value. However, since w is also an r-eigenvector using Lemma 2.2, and any non-negative r-eigenvector must in fact be positive using the Second extraction of Perron-Frobenius Theorem 2.3, therefore all components of w must be strictly bigger than 0, contradiction.
Case 2) u has no positive components,
Define w as the linear combination between v, u such that $w = v + \alpha u$ for α multiplicative scalar. Following the same ideas as in Case 1, we also reach a contradiction.

Since we get a contradiction in both cases, we can conclude that u is **not** an eigen-vector to the eigenvalue r and so r is a simple eigenvalue.

□

Theorem 2.5. *[Taboga, 2021] Let P be an $n \times n$ matrix with n distinct eigenvalues $\lambda_1, \lambda_2, \cdots \lambda_n$ then, the corresponding eigen-vectors $v_1^T, v_2^T, \cdots, v_n^T$ are linearly independent, and form a basis for $\mathbb{R}^n$.*

Proof. If we assume that P has n distinct eigenvalues $\lambda_1, \lambda_2, \cdots \lambda_n$ and the corresponding eigen-vectors $v_1, v_2, \cdots, v_n$, we can show that all eigen-vectors are linearly independent by induction:

1. We have that v_1 is linearly independent to itself
2. Assume that for $v_1, v_2, \cdots, v_{k-1}$ we have: $\alpha_1 v_1 + \alpha_2 v_2 + \cdots \alpha_k v_k = 0$
3. Then we observe that $P(\alpha_1 v_1 + \alpha_2 v_2 + \cdots \alpha_{k-1} v_{k-1} + \alpha_k v_k) = \alpha_1 \lambda_1 v_1 + \alpha_2 \lambda_2 v_2 + \cdots \alpha_{k-1} \lambda_{k-1} v_{k-1} + \alpha_k \lambda_K V_K = P(0) = 0$

   If we multiply both sides of $\alpha_1 v_1 + \alpha_2 v_2 + \cdots \alpha_k v_k = 0$ by λ_K we get $\alpha_1 \lambda_K v_1 + \alpha_2 \lambda_K v_2 + \cdots \alpha_k \lambda_K v_k = 0 \lambda_K = 0$

   We can subtract the top equation by bottom to get: $\alpha_1(\lambda_1 - \lambda_K)v_1 + \alpha_2(\lambda_2 - \lambda_K)v_2 + \cdots + \alpha_{k-1}(\lambda_{k-1} - \lambda_K)v_{k-1} + \alpha_k(\lambda_K - \lambda_K)v_k = 0$ and see that each $(\lambda_j - \lambda_K) \neq 0 \quad \forall j$ except $(\lambda_K - \lambda_K)$ so we must have that $\alpha_1, \alpha_2, \cdots, \alpha_{k-1} = 0$

   Therefore looking back at $\alpha_1 v_1 + \alpha_2 v_2 + \cdots \alpha_k v_k = 0$ since we have $\alpha_1, \alpha_2, \cdots, \alpha_{k-1} = 0$ we are left with only $\alpha_k v_k = 0$ and since $v_k \neq 0$ we must have that $\alpha_k = 0$
4. Therefore by mathematical induction we have that all eigen-vectors corresponding to distinct eigenvalues are linearly independent.

By definition of linear independence, a set of n linear independent vectors form a basis for $\mathbb{R}^n$. □

Theorem 2.6 (**The Power Method** From [Mathews and Fink, 2004]). *Assume that P is an $n \times n$ matrix, with n distinct eigenvalues $\lambda_1, \lambda_2, \cdots \lambda_n$ such that $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$. If X_0 is chosen appropriately then we can construct two sequences $\{X_k = \begin{bmatrix}(x_1^{(k)}) & (x_2^{(k)}) & \cdots & (x_K^{(k)})\end{bmatrix}\}$ and $\{c_k\}$ which are both generated by the following recurrence pattern:*

$$Y_k^T = X_k^T \cdot P, \quad X_{k+1}^T = \frac{1}{c_{k+1}} \cdot Y_k^T \quad \textit{such that} \quad c_{k+1} = \max_{1 \leqslant j \leqslant n} |y_j^{(k)}| \tag{3}$$

which will converge to the dominant eigen-vector v_1^T and eigenvalue λ_1, respectively. That is,

$$\lim_{k \to \infty} X_k = v_1 \quad \lim_{k \to \infty} c_k = \lambda_1$$

Proof. Since P has n distinct eigenvalues $\lambda_1, \lambda_2, \cdots \lambda_n$ with corresponding eigen-vectors $v_1, v_2, \cdots, v_n$ for $j = 1, 2, \cdots, n$, that are linearly independent, normalized, and form a basis for $\mathbb{R}^n$, we can let the starting vector X_0 be written as a linear combination

$$X_0 = \alpha_1 v_1 + \alpha_2 v_2 + \cdots + \alpha_n v_n \quad \text{with} \quad \alpha_j \neq 0 \quad \forall j = \{1, 2, \cdots, n\} \quad \text{while} \quad \max_{1 \leqslant j \leqslant n} |x_j^{(1)}| = 1.$$

Then we have $Y_0^T = X_0^T \cdot P = (\alpha_1 v_1^T + \alpha_2 v_2^T + \cdots + \alpha_n v_n^T) \cdot P = \alpha_1 \lambda_1 v_1^T + \alpha_2 \lambda_2 v_2^T + \cdots + \alpha_n \lambda_n v_n^T$
Using the facts that λ_j and v_j^T are eigenvalues and eigen-vectors respectively: $v_j^T \cdot P = \lambda_j \cdot v_j^T$.

$$\text{So we have that} \quad Y_0 = \lambda_1(\alpha_1 v_1 + \alpha_2(\frac{\lambda_2}{\lambda_1})v_2 + \cdots + \alpha_n(\frac{\lambda_n}{\lambda_1})v_n),$$
$$\text{and so} \quad X_0 = \frac{\lambda_1}{c_1}(\alpha_1 v_1 + \alpha_2(\frac{\lambda_2}{\lambda_1})v_2 + \cdots + \alpha_n(\frac{\lambda_n}{\lambda_1})v_n).$$

If we were to repeat this recurrence relation which can be written as:

$$\text{So we have that} \quad Y_{k-1} = \frac{\lambda_1^k}{c_1 c_2 \cdots c_{k-1}}(\alpha_1 v_1 + \alpha_2(\frac{\lambda_2}{\lambda_1})^k v_2 + \cdots + \alpha_n(\frac{\lambda_n}{\lambda_1})^k v_n),$$
$$\text{and so} \quad X_k = \frac{\lambda_1^k}{c_1 c_2 \cdots c_k}(\alpha_1 v_1 + \alpha_2(\frac{\lambda_2}{\lambda_1})^{k-1} v_2 + \cdots + \alpha_n(\frac{\lambda_n}{\lambda_1})^{k-1} v_n).$$

Since we were given that: $|\lambda_1| > |\lambda_2| \geq \cdots \geq |\lambda_n|$ we can say that each $(\frac{\lambda_j}{\lambda_1}) < 1$ such that $j = \{2, 3, \cdots, n\}$ and so $\lim_{k \to \infty}(\frac{\lambda_j}{\lambda_1})^{k-1} \longrightarrow 0, \quad \forall j = \{2, 3, \cdots, n\}$.

Therefore $\lim_{k \to \infty} X_K = \lim_{k \to \infty} \frac{\lambda_1^k}{c_1 c_2 \cdots c_k}(\alpha_1 v_1 + \alpha_2 \cdot 0 \cdot v_2 + \cdots + \alpha_n \cdot 0 \cdot v_n) = \lim_{k \to \infty} \frac{\lambda_1^k \cdot \alpha_1}{c_1 c_2 \cdots c_k} v_1$
We have mentioned before that X_k can be normalised, so v_1 can too and so we are able to remove the fraction of $\frac{\lambda_1^k \cdot \alpha_1}{c_1 c_2 \cdots c_k}$ by normalisation and letting $\lim_{k \to \infty} \frac{\lambda_1^k \cdot \alpha_1}{c_1 c_2 \cdots c_k} = 1$. So,

$$\lim_{k \to \infty} X_k = v_1$$

Using the fact that $\lim_{k \to \infty} \frac{\lambda_1^k \cdot \alpha_1}{c_1 c_2 \cdots c_k} = 1$ we can show that:

$$\lim_{k \to \infty} \frac{c_k}{\lambda_1} = \lim_{k \to \infty} \frac{\frac{\lambda_1^{k-1} \cdot \alpha_1}{c_1 c_2 \cdots c_{k-1}}}{\frac{\lambda_1^k \cdot \alpha_1}{c_1 c_2 \cdots c_k}} = \frac{1}{1} = 1,$$
$$\text{and so} \lim_{k \to \infty} c_k = \lambda_1.$$

□

Theorem 2.7. *[**Stationary Regime**, Property 9.4 of [Rousseau et al., 2008]] If an Irreducible transition matrix P of a Markov chain satisfies the following:*

1. *Suppose that there is exactly one eigenvalue such that $\lambda = 1$.*
2. *Suppose that this eigenvalue is not degenerate, which means that the associated eigensubspace has dimension 1.*
3. *Let the transition matrix P representing the web be diagonalizable, so its eigen-vectors form a basis.*

Then there exists a unique vector π such that the entries $\pi_j = P(X_k = j)$, $\forall j$, satisfy:

$$\pi_j \geq 0, \quad \pi_j^T = \sum_{i \in T} \pi_j^T \cdot p_{ij}, \quad \sum_{j \in T} \pi_j = 1.$$

We will call the vector π the stationary regime of the Markov chain.
Additionally, regardless of the initial point $p_j^0 = P(X_0 = j)$ (where $\sum_j p_j^0 = 1$), the distribution of probabilities $(X_k = j)$ will converge to the stationary regime π^T as $k \to \infty$.

Proof. Let P be the transition matrix of a map which satisfies the three conditions above, we can then let π be the Eigen-vector for the Eigen-value $\lambda = 1$, which is given to exist (although currently unknown). We can then see that using Definition 2.5 we have that $\pi^T \cdot P = \lambda \cdot \pi^T = 1 \cdot \pi^T = \pi^T$

Now that we know the following $\pi^T \cdot P = \pi^T$, we can then use the power method with X_0 to be a $1 \times n$ vector of all $\frac{1}{n}$'s (which represents the equal probability of starting the journey at each web-page - to make sure staring position is not relevant) with $v_1^T = \pi^T$ to conclude that $\lim_{k \to \infty} X_k = \pi$. We can see that this is true since $\lambda_1 = 1$ is maximal with respect to v_1 so the limit goes to $v_1 = \pi$ as we have shown in Lemma 2.1. Therefore $\pi_j = P(X_k = j)$ as $k \longrightarrow \infty$

We can also notice that all the criteria for π are true since,

1. $\pi_j \geq 0$ is true since each π_j represents a probability, so $\pi_j \geq 0$ always.
2. $\pi_j^T = \sum_{i \in T} \pi_j^T \cdot p_{ij}$ is true due to the nature of matrix multiplication $\pi^T \cdot P = \pi^T$
3. $\sum_j \pi_j = 1$ is also true since the given π^T vector is normalised, so all π_j sum up to 1.

□

2.4 Finding π using the Power Method

In the previous Subsection we have seen that for a transition matrix such as P, we have that at least one eigenvalue λ_i is equal to 1 with all other eigenvalues λ_i $\forall i = \{2, 3, \cdots, n\}$ satisfy $|\lambda_i| < 1$ using Lemma 2.1 and Theorem 2.4. Therefore we can implement the Power method onto the transition matrix P and receive the stationary regime π.

Definition 2.8 (**Ranking**, Definition 9.5 of [Rousseau et al., 2008])**.** *We have that the coefficients π_i from the stationary regime π contribute to the ranking, such that the score given to page i in our page-rank algorithm is equivalent to the numerical value of the placement of π_i in the sorted list of π_i values from largest to smallest.*

Lemma 2.8. *As we have seen in Subsection 2.1, Figure 3 creates a transition Matrix (1) P. As mentioned in Theorem 2.1, one of the Eigen-values of matrix P is $\lambda = 1$.*
Since $\lambda = 1$ is the Spectral radius, we have that $\lim_{k\to\infty} c_k = 1$ therefore, there is no use in dividing by c_{k+1} after each iteration, so we can implement an algorithm such that $X_{k+1}^T = X_k^T \cdot P$. Since we do not have to divide by c_{k+1} after each iteration, we have that:

- $X_1^T = X_0^T \cdot P^1$
- $X_2^T = X_1^T \cdot P = (X_0^T \cdot P) \cdot P = X_0^T \cdot P^2$

If we keep iterating for $k \in \mathbb{N}$ to get a sequence for X_k^T, we get:

- $X_k^T = X_0^T \cdot P^k$

Therefore we have $\lim_{k\to\infty} X_k^T = \lim_{k\to\infty} X_0^T \cdot P^k = \pi^T$.

In listing 1, I wrote a code which verifies whether the matrix provided is suitable for the page-rank algorithm by checking if it is square, positive with all rows summing up to 1. And in Listing 2, I wrote a code that implements the Power Method onto P being the matrix described in equation 1.

Example 2.3. *Using the Transition matrix P, raised to the power k=1000 with an initial starting vector being X_0 of fractions $\frac{1}{n}$ we get the result:*

$$X_0 = \begin{bmatrix} \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \\ \frac{1}{8} \end{bmatrix}, \quad \pi = \begin{bmatrix} 0.0523903 \\ 0.1047806 \\ 0.1964636 \\ 0.0510806 \\ 0.2030124 \\ 0.2062868 \\ 0.1349050 \\ 0.0510806 \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \end{bmatrix} \tag{4}$$

We can now sort the scoring of π in order of largest to smallest in order to find the ranking of the web-pages in our system.

1. $0.2062868 = \pi_6 = F$,
2. $0.2030124 = \pi_5 = E$,
3. $0.1964636 = \pi_3 = C$,
4. $0.1349050 = \pi_7 = G$,
5. $0.1047806 = \pi_2 = B$,
6. $0.0523903 = \pi_1 = A$,
7. $0.0510806 = \pi_4 = D$,
8. $0.0510806 = \pi_8 = H$.

So we can see that F would come at the top of the ranking, then E, C , $\cdots$,D, H.

3 Refining the Page-Rank Algorithm

In Figure 3 of the previous section, the graph portrays a 'perfect world' situation, in which the web-searcher can travel around the graph forever leading it to converge and result in finding the stationary regime π which ranks the web-pages.
We now have to consider a different case, where it is possible to wander to one of the nodes, with no way of returning to the rest of the graph. This can happen if a node contains no outwards links or if a group of nodes do not contain outwards link back into the main web, creating an infinite loop. Examples of pages which create this issue are image files, pdfs or even link-farms which will be introduced later on in the project. This is problematic to the algorithm since once the traveller lands on the troubling node, it can never return to the main graph and therefore will disturb convergence.

This creates a situation where the stationary regime π is weighed mostly on that one certain node, or of a few which are stuck in a loop, rather than considering the rest of the web as it may have more influence on the total distribution than the dangling nodes alone.

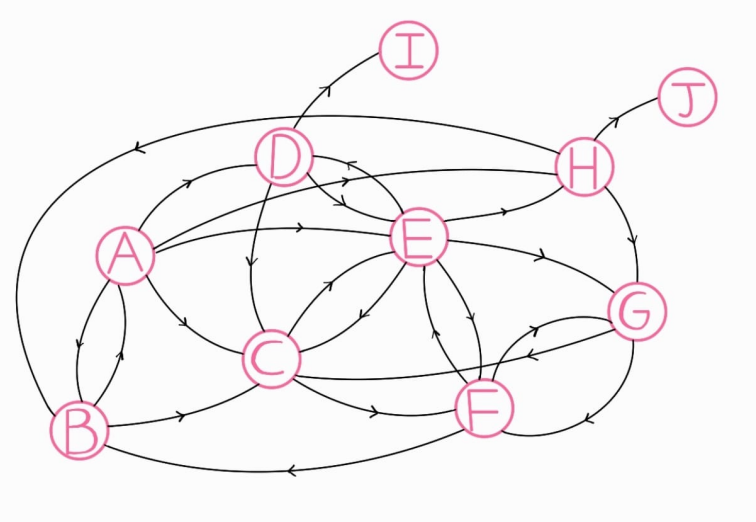


Figure 5: A simplified web system with dangling nodes

In Figure 9 we can see a new Web-graph similar to the one described in Figure 3 but with the addition of Two extra dangling nodes. We can also see that the addition of the dangling nodes will lead to an adapted Transition matrix S as below.

$$S = \begin{bmatrix} 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{5}$$

There are a few ways to combat this problem of landing on these problematic nodes such as completely removing them from all calculations, adding a teleportation matrix and adding artificial links.

In the next subsection, we will delve into the advantages of addition of Artificial links.

3.1 Artificial Links

As we can see in (5), Matrix S is no longer stochastic, such that all the rows add up to 1, therefore S is no longer an irreducible matrix, so non of the previous theorems and methods apply to S, therefore, it does not converge to the stationary regime π.
One of the ways introduced in [Rousseau et al., 2008], to solve this issue is to add artificial links. The artificial links imitate links coming out of the dangling nodes to all other nodes in the graph. These artificial links are not real connections in the web but are introduced for computational purposes. Even though the links slightly change the structure of the map, they will not be too significant that the probabilities will change drastically.

We first start by observing four segments of the matrix S of the following:

$$S = \begin{bmatrix} S_{11} & S_{12} \\ 0 & 0 \end{bmatrix}$$

For S, $n \times n$ matrix, we have that:

$$S_{11} = \begin{bmatrix} 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 \end{bmatrix}$$

Figure 6: $k \times k$ matrix of all non-dangling nodes links to non-dangling nodes

$$S_{12} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ \frac{1}{3} & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & \frac{1}{3} \end{bmatrix}$$

Figure 7: $k \times (n-k)$ matrix of all non-dangling nodes links to dangling nodes

Where both matrices S_{11} and S_{12} are such that:

$$S_{11} \geq 0, \quad S_{12} \geq 0, \quad S_{11} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} + S_{12} \times \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix}, \quad \text{with} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ \vdots \\ 1 \end{bmatrix} \text{ being a } k \times 1 \text{ vector.}$$

So we see that each row of the first k rows sums up to 1.

We can now see that to make matrix S into a stochastic matrix, we need the last $(n-k)$ rows also to sum up to 1. In order to satisfy such request we add an artificial vector, called $w = \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$ which we will replace with each 0 row.
Ideally, we do not want to add the artificial vector w, which will severely change the initial graph, we want the ratios between all pages to still remain in proportion so that when we do manage to make S into a stochastic matrix, the values of the stationary regime, π still remain in the correct order. In other words, we do not want to "mess up" the initial graph too much.

The best way to make sure that such a case is true, is to assign equal probabilities to w_i, $\forall i$. Therefore, for n pages in a graph, we can let each value $w_i = \frac{1}{n}$ so that the following conditions are true:

1. $w_1 = k \times 1$ vector, $w_2 = (n-k) \times 1$ vector, such that, $w = n \times 1$ vector
2. $\sum_i w_i = 1$

When applied to our example in Figure 9 and Transition matrix S, we can create a new matrix with the artificial nodes which is indeed stochastic, and so we can apply the same method we had used in Section 2.4

Example 3.1. *To find π for matrix S using the power method, we have to add the artificial nodes to matrix S and then implement the exact same algorithm. We make modifications to the initial code in listing 3, which turns zero-rows into the vector w to get the new matrix H with the artificial nodes. Matrix H can be seem below:*

$$H = \begin{bmatrix} 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 \\ 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \\ \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} & \frac{1}{10} \end{bmatrix} \tag{6}$$

We can now apply the same method as described in Listing2 to find the new vector π with the addition of the artificial links as seen below .

$$\pi = \begin{bmatrix} 0.05376344 \\ 0.09865165 \\ 0.18330773 \\ 0.05325141 \\ 0.19030551 \\ 0.19713262 \\ 0.12596006 \\ 0.05325141 \\ 0.02218809 \\ 0.02218809 \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \end{bmatrix} \tag{7}$$

π can be sorted in order of largest to smallest so that we can find the ranking of the web-pages in our system.

1. $0.19713262 = \pi_6 = F$,
2. $0.19030551 = \pi_5 = E$,
3. $0.18330773 = \pi_3 = C$,
4. $0.12596006 = \pi_7 = G$,
5. $0.09865165 = \pi_2 = B$,
6. $0.05376344 = \pi_1 = A$,
7. $0.05325141 = \pi_4 = D$,
8. $0.05325141 = \pi_8 = H$,
9. $0.02218809 = \pi_9 = I$,
10. $0.02218809 = \pi_{10} = J$.

3.2 The Teleportation Matrix

As I have mentioned previously, it is possible for a surfer to land on a dangling node, which will affect how the matrix converges as we raise P to a higher power. But a dangling node is not the only way a convergence can be disturbed!
Instead of getting stuck on a single dangling node, we can get stuck in an infinite loop on a trapped section of the graph.

Example 3.2. *In Figure 8, we extended the previous graph by connecting node K to node J with two links in both directions. This creates an infinite loop such that once we land on node J as it would keep cycling between page J and K forever. This, again, does not represent the distribution of links among the web-pages and therefore affect to convergence and misrepresent the ranking of the graph.*

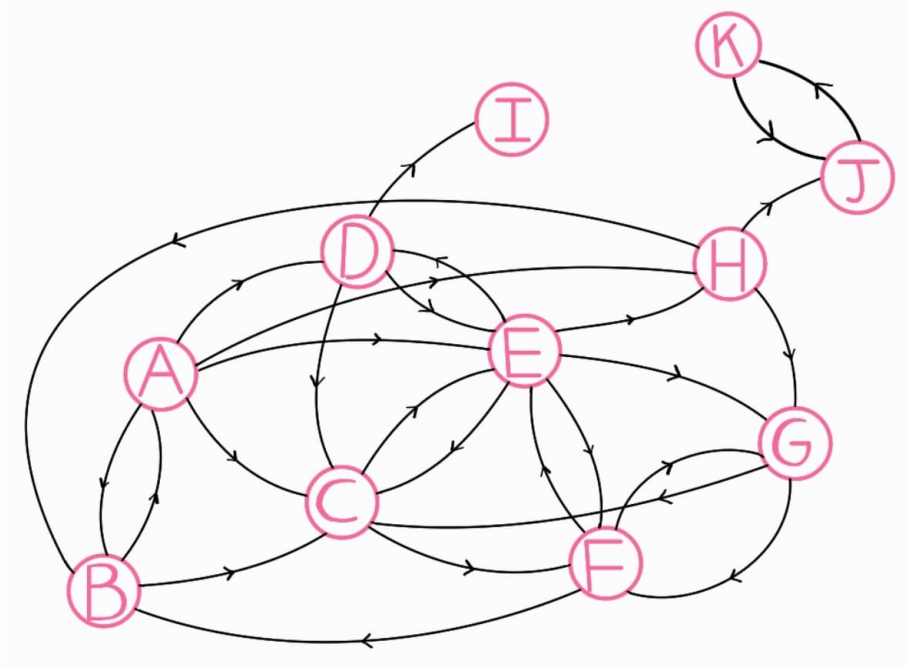


Figure 8: An Example of a loop in a Graph

We can represent the information from the new graph and the ideas we have previously

developed of adding the w vector on the zero-row in Matrix H^* below,

$$H^* = \begin{bmatrix} 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & \frac{1}{5} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & \frac{1}{5} & \frac{1}{5} & 0 & \frac{1}{5} & \frac{1}{5} & \frac{1}{5} & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 & \frac{1}{3} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 & \frac{1}{3} & 0 \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

which if we were to calculate the value of π for this matrix using the same method as before, the results would clearly not be representing the structure of the graph although it is stochastic and represents all the conditions required to have a stationary regime.

We can see that in the vector below, all values, except of the ones of J, K are significantly small which J, K splitting the majority of the weight of the graph between them.

$$\pi = \begin{bmatrix} \approx 1.9 \times 10^{-11} \\ \approx 3.6 \times 10^{-11} \\ \approx 6.8 \times 10^{-11} \\ \approx 1.9 \times 10^{-11} \\ \approx 7.1 \times 10^{-11} \\ \approx 7.4 \times 10^{-11} \\ \approx 4.7 \times 10^{-11} \\ \approx 1.9 \times 10^{-11} \\ \approx 7.1 \times 10^{-12} \\ 0.489004981 \\ 0.510995019 \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \\ K \end{bmatrix}$$

One of the ways we can overcome the issue of getting trapped inside a sub-graph of the whole graph, is by introducing the Teleportation matrix.

Definition 3.1 (Teleportation matrix)**.** *The Teleportation matrix, is a matrix which allows a surfer to land on any page in the web with equal probability.*

The implementation of this matrix is added in order to even further balance and combat the issue of a web-surfer converging into a dangling node or an infinite loop, by allowing the surfer to transport to any other page in the web with a small probability.

In order to achieve the randomness of the teleportation matrix we let the Teleportation matrix T be an $n \times n$ matrix made up of values t_{ij} which equal to $\frac{1}{n}$ for n being the number of nodes in the graph.

The code of the Teleportation Matrix is described in Listing 4 and visually is as follows :

$$T = \begin{bmatrix} \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \\ \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} & \frac{1}{11} \end{bmatrix}$$

3.3 The Final Google Matrix

Definition 3.2 (The Google Matrix)**.** *The complete computation of the Google Matrix is a balance between The Teleportation Matrix V and the structure of the web H after the addition of artificial links. The balance is controlled by a parameter $\alpha \in [0, 1]$ called - The Damping Factor, as displayed in Equation 8 and in the code of Listing 5:*

$$G = \alpha \cdot H + (1 - \alpha) \cdot T \tag{8}$$

The damping factor α, falling within the range $[0, 1]$, significantly influences the computation of the final rank score. [Paolo Boldi,] verifies that at $\alpha = 0$, the outcome reflects a basic uniform process. As α approaches 1, the relevance of web connectivity intensifies. The original value suggested by Brin and Page ($\alpha = 0.85$) is the most common choice which is believed to have been selected through experimentation and observation of user behaviour on the web and is possibly used till this day. It's common to wonder if there truly exists an optimal value for the damping factor.

In paper [Srivastava et al., 2017], Srivastava, Garg and Mishra conducted an experiment where they applied the page-rank algorithm using different values for alpha where they have "observed that for value of $\alpha = 0.7$, Page-Rank method takes fewer numbers of iterations to converge than $\alpha = 0.85$, and for these values of the top 25 web pages returned by page-rank method in the SERP are almost same, only some of them exchange their positions. From the experimental results it is observed that value of damping factor $\alpha = 0.7$ takes approximate 25-30% fewer numbers of iterations than $\alpha = 0.85$ to get closely identical web pages in top 25 result pages for personalized web search, selective crawling, intra-web search engine"

Hence, the optimal value for alpha remains uncertain. However, for future calculations, we will adopt alpha as 0.85, given its likelihood of being the actual value employed in practical applications.

Example 3.3. *We can implement our code in listing 5 with the input of Matrix H and the damping factor being 0.85, in order to find our final π for this algorithm.*

The result for π can be seen below in vector 9 which displays the final scores of the web-pages, after the addition of the artificial links and the combination of the teleportation matrix.

$$\pi = \begin{bmatrix} 0.04672534 \\ 0.07264535 \\ 0.12549575 \\ 0.04521116 \\ 0.12598105 \\ 0.12720404 \\ 0.08611882 \\ 0.04521116 \\ 0.0286609 \\ 0.15183534 \\ 0.1449111 \end{bmatrix} = \begin{bmatrix} A \\ B \\ C \\ D \\ E \\ F \\ G \\ H \\ I \\ J \\ K \end{bmatrix} \tag{9}$$

We can now sort the scoring of π in order of largest to smallest in order to find the ranking of the web-pages in our system.

1. $0.15183534 = \pi_{10} = J$
2. $0.1449111 = \pi_{11} = K$
3. $0.12720404 = \pi_6 = F$
4. $0.12598105 = \pi_5 = E$
5. $0.12549575 = \pi_3 = C$
6. $0.08611882 = \pi_7 = G$
7. $0.07264535 = \pi_2 = B$
8. $0.04672534 = \pi_1 = A$
9. $0.04521116 = \pi_4 = D$
10. $0.04521116 = \pi_8 = H$
11. $0.0286609 = \pi_9 = I$

It can be noticed that in this system, pages J, K are still leading the rank although they don't realistically represent the pages that should receive the higher scores, this could potentially be due to the fact that this system is pretty small and the loop isn't actually that small in relation to the main graph.

We can see that from the score received in Example 3.2 that pages J, K scored unproportionally higher than the rest of the pages, so it is possible that for such a small graph, the number 0.85 for α may not be strong enough to create a realistic representation of the web.

The value for π described in vector 9, represents the way Google initially calculated the scores of the web-pages on the web just when Brin and Page have began to develop Google's search engine.

Google has immensely evolved and grown since the initial structure of the algorithm, and has added many other functions to its core algorithm to provide a much more efficient and tailored user experience, some of which we'll explore in the following chapters.

4 Data Analysis and Algorithm Investigations

The preceding chapters have introduced the foundations of the page-rank algorithm, emphasizing the principles which Google initially constructed its algorithm with. However, it is crucial to recognize that Google is constantly and always evolving and fine-tuning its operations. Consequently, the page-rank algorithm has undergone modifications, introducing additional elements and procedures that has optimized the conventional approaches from the past.

In the subsequent section, we will explore specific features which are potentially used by Google which will have an influence on the page-rank scores assigned to individual web-pages. Some of these attributes have received official acknowledgment from Google, though the exact implementation details within the algorithm remain unpublished, whereas some of these features are speculative and theoretical, lacking confirmation from official sources. In an attempt to shed light on this ambiguity, I will propose potential algorithms and their implementations onto the original algorithm.

Before we jump into describing such algorithms, we introduce the three stages of the Google-search process as revealed by [Developers, 2023] :

1. **Crawling:** Google's automated crawlers download text, images, and videos from web pages they discover on the internet.

2. **Indexing:** Google's automated crawlers analyse the downloaded content and store it in the Google index, a large database.

3. **Serving search results:** When users provide search queries to Google, the platform matches them with relevant information to their queries.

Previously, we have used a very small sample of a potential web-graph, and although it was useful for understanding the basic algorithms and ideas on a smaller scale, I wanted to make the following algorithm a little more challenging and use a real large Data-base of many web-pages with some information about each them attached, and create a Google matrix for the large data set so I can implement the algorithms we'll explore onto that system.

For this research, I used a file of data named "Food.com Recipes with Search Terms and Tags" from the website 'Kaggle' [Li, 2021] as can be seen in Figure 9 Which includes around 500,000 recipes alongside their ID in the system, the name of the recipe, ingredients, serving sizes and more. I have treated each recipe as an individual web-page.
This will be the information that the search-engine then crawls and indexes in order to serve search results.

	A	B	C	D	E	F	G	H	I	J
1	id	name	descriptio	ingredient	ingredient	serving_si	servings	steps	tags	searc
2	96313	Grilled Gai	We love gr	['water', 'g	["4	cups	1 (155 g)	8	['I a sauce	['time-to-r	{'diab
3	232037	Simple Shr	Simple, ea	['onion', 'r	["1	mediu	1 (366 g)	4	['In a food	['60-minut	{'dinn
4	41090	black-and-white bea		['white be	["1	cup	c 1 (807 g)	1	['In a large	['15-minut	{'vege
5	60656	Crock Pot	This is a g	['zucchini'	["2	zucc	1 (244 g)	4	['Put all in	['weeknigh	{'side
6	232047	Beef Stew	This is a fa	['beef stev	["3	lbs	b 1 (358 g)	8	['Preheat o	['time-to-r	{'dinn
7	232050	Hot Sweet	This is one	['slivered a	["12	ounc	1 (832 g)	1	['Preheat o	['time-to-r	{'dess
8	232076	Retro Chic	From Cool	['chicken t	["4	cups	1 (85 g)	6	['In large b	['30-minut	{'cass
9	232083	Asparagus	These wra	['eggs', 'mi	["8	eggs	1 (499 g)	4	['Beat the	['60-minut	{'brea
10	79222	Potato-Cr	Soup for th	['butter', 'c	["2	tables	1 (362 g)	6	['Saute oni	['60-minut	{'heal
11	393638	Sweet and	Easy and k	['lean grou	["1	lb	lea 1 (103 g)	4	['Brown gr	['30-minut	{'low-
12	232086	Golden Ch	From King	['butter', 'g	["1/2	cup	1 (94 g)	12	['Preheat t	['60-minut	{'brea
13	232088	Potato Ch	There are	['all-purpo	["1 3/4	cu	1 (28 g)	54	['Position r	['60-minut	{'cool
14	232090	Steak Au F	An Americ	['unsalted	["4	cups	1 (610 g)	4	['Add beef	['60-minut	{'dinn
15	232095	Mushroom	From Cool	['button m	["","0.5 (8		1 (250 g)	2	['To prepar	['60-minut	{'vege
16	232096	Layered Ic	I just cam	['shortcak	["2 -3		in 1 (138 g)	1	['Put the n	['15-minut	{'cake
17	232097	Santa Fe-1	This is Rac	['vegetable	["	vegetab	1 (895 g)	4	['Heat a gr	['60-minut	{'soup
18	232099	Scarlett's	Ideal for	['butter', 's	["1	tables	1 (74 g)	6	['Preheat t	['time-to-r	{'appe

Figure 9: A sample of all the data given

In my following calculations and algorithms, I have filtered the search to only focus on the top 100 pages due to the lack of memory available on my computer and the length of processing and loading time of large files, but all algorithms should work on the full data set if calculated on a stronger computer.

As the data set does not include any information about links between each recipe, I had to create an algorithm that generated a transition matrix that could potentially represent the relationships between all these web-pages.
It is essential to ensure that the matrix created is a square matrix of the length of the number of recipes provided. It also needs to be stochastic, positive, and such that the elements for each row are of the same probability, or the 0 elements, with possibly some zero-rows. In Listing 6 we can see a code for the matrix generator that represents the links between our web-system recipes. In figure 10, we can see an example of a generated matrix by the code.

```
transition_matrix = generate_stochastic_transition_matrix(k)
transition_matrix
```

```
array([[0.05555556, 0.        , 0.        , ..., 0.        , 0.        ,
        0.05555556],
       [0.        , 0.02564103, 0.02564103, ..., 0.        , 0.02564103,
        0.        ],
       [0.02040816, 0.        , 0.        , ..., 0.        , 0.02040816,
        0.02040816],
       ...,
       [0.        , 0.        , 0.01886792, ..., 0.01886792, 0.01886792,
        0.        ],
       [0.03571429, 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.01428571, 0.01428571, 0.01428571, ..., 0.01428571, 0.01428571,
        0.01428571]])
```

Figure 10: Example of a Generated Transition Matrix

There are many different suggestions to what the optimal or suggested number of outward links a page should contain. Google has suggested in [Jay, 2024] that “More than 100 internal links on a single page are excessive.” and that “Google may opt not to follow or crawl all those links” although in 2022, Moz stated in [Batt, 2024] that ”particularly important pages can sometimes hover around 200 to 250 follow links”.

Therefore, as this is simply a generic test of the system, I have decided to let $k = 100$ as a cap for the number of links a page may contain

Now that we have our transition matrix to work with, we can proceed to exploring different algorithms that will revolve around changing the initial score for π while applying different variations of the page-rank algorithm. We start by studying of Word-Classification and how Google takes advantage of importance of key words in each document.

Before I proceed, I calculated the initial value for π in Listing 7 in order to compare the ranking after implementing future algorithms, and in Listing 8 I have created an algorithm that indexes the recipes and sorts them in order of highest to lowest scores. As can be seen in Figure 11, we have that based on the transition_matrix alone, recipe 54 would appear at the top of the search, then recipe 55, all the way to recipe 22 which will appear as the least relevant.

```
sorted_pi = score_indexing_sorted_in_order([i for i in range(100)],pi)
print(sorted_pi)
```

```
[(54, 0.015112875809868411), (55, 0.014623928658717162), (61, 0.01413143413119909), (21, 0.013687841049798168), (9, 0.013635961
20482545), (62, 0.013065709280510616), (36, 0.012500808515875449), (87, 0.012364467216074804), (97, 0.011984124812911823), (53,
0.011933795840899751), (37, 0.011916000842002315), (81, 0.011881658414481142), (20, 0.011862837110911512), (11, 0.0117952428729
38988), (3, 0.01177980058906475), (63, 0.01161396270562573), (92, 0.011558867173485799), (67, 0.011299415658063892), (84, 0.01
1145931274305182), (68, 0.011072455457228986), (90, 0.011002677018811732), (15, 0.010969360585183847), (47, 0.01094912355303439
2), (10, 0.01086811475106923), (93, 0.010841243733315584), (40, 0.010819555607889704), (43, 0.010776899567506526), (96, 0.01074
511514550477), (29, 0.010722140186618307), (45, 0.010697041823517283), (12, 0.01052963812272794), (69, 0.010440066827826623),
(26, 0.010432904370955388), (77, 0.010420888026654519), (98, 0.010412820971162001), (65, 0.010395565834685528), (16, 0.01036329
3622275276), (59, 0.010341135836859945), (28, 0.010313429196997754), (48, 0.010193509426229397), (25, 0.010186544477972856), (8
0, 0.010145878526532422), (82, 0.010120429052023304), (79, 0.010102056857715001), (86, 0.010053098168878494), (71, 0.0100195560
15610161), (32, 0.009872958295091156), (70, 0.009857175622187997), (46, 0.009834845597345885), (94, 0.009822067629731758), (2,
0.009811548971969364), (57, 0.009801411890377318), (8, 0.009791042996314026), (33, 0.009706705869131799), (6, 0.009675654945290
081), (38, 0.009673496859467055), (5, 0.009637143232910502), (13, 0.009591922067575314), (64, 0.009567668691044471), (89, 0.009
482703523455382), (49, 0.009390868921427197), (41, 0.009378031533206807), (78, 0.009277453874431813), (66, 0.00927275534099428
4), (58, 0.009271525232720165), (60, 0.009250888841161232), (34, 0.009189822596608607), (24, 0.009185738634516471), (52, 0.00914
0904647973777), (72, 0.00911174816677788), (50, 0.009102568881101336), (76, 0.009037683191260488), (42, 0.009033815413676233),
(19, 0.008984744686817443), (39, 0.008831865136320404), (7, 0.008729245919390765), (83, 0.008718638630941038), (0, 0.0086956225
40958537), (88, 0.00868205912531246), (91, 0.008570574535976167), (1, 0.008555690343898253), (4, 0.008540111778817677), (30, 0.
00852372005505559), (99, 0.008476353142302807), (27, 0.008466196755180667), (74, 0.008452501175412916), (31, 0.0082875552494515
76), (35, 0.008265904268586996), (14, 0.008256405243328758), (56, 0.008225906535527341), (23, 0.007995256161631803), (18, 0.007
948250974946171), (73, 0.007908789430829991), (51, 0.0078854777859131), (95, 0.00783625725952647), (44, 0.007758117583342307),
(17, 0.007714150569243282), (85, 0.0075886378721371816), (75, 0.007454523800733767), (22, 0.007048086908930695)]
```

Figure 11: Initial π ranked

4.1 Word Classification

As mentioned in the introduction, Before the introduction of Google, every query launched into the search bar resulted in a flood of relevant but unsorted pages based on key-words.
This is the equivalence of searching for information at a library, where it is easy to locate books and magazines based on their title and index, but is difficult to locate the specific book out of a selection which contains the answers to our questions.

Google Search utilizes word classification algorithms to documents / pages in the web in order to enhance the precision and relevance of search results. These algorithms categorize words and phrases based on their semantic meanings, identifying the main themes and topics within the text where some algorithms may assess the importance of those words and phrases within the document, and establish relationships between them to determine the overall theme. By incorporating word classification into the search algorithm, Google optimizes the page-rank algorithm and enhances the efficiency of delivering a more satisfying and tailored search experience for users around the world.

In Figure 12 we can see that the text of the first recipe contains many special characters and letters from both lower and upper case. Both of these properties make it harder to analyse words in texts as its more difficult to compare between them and identify them, as words like 'Bread!' and 'bread' may seem as separate words.

```
'Grilled Garlic Cheese Grits We love grits, this is another good way to serve them. A great alternative to a baked potato when
served with grilled steak or chicken. I belive this recipe could be made with instant grits.The 2 1/2 hours for refrigeration i
s not include in time. The recipe comes from Tast of Home\'s Light and Tasty. [\'water\', \'grits\', \'salt\', \'cheddar cheese
\', \'garlic\', \'olive oil\'] ["4   cups    water","1   cup    uncooked old fashion grits","1   teaspoon    salt","4   ounces
shredded cheddar cheese","1 -2   clove    garlic, minced ","1   tablespoon    olive oil"] [\'I a sauce pan, bring water to a bo
il; slowly add grits and salt, stirring constantly; Reduce heat:simmer, uncovered, for 40-45 minutes or untill thickened, stirr
in occasionally.\', \'Add cheese and garlic; stir until cheese is melted, Spray 9-inch baking dish with nonstick cooking spray;
Cover and refrigerate for 2 to 2 1/2 hours or until frim.\', \'Before starting the grill, coat the grill rack with nonstick coo
king spray; Cut the grits into 3-inch squares; Brush both sides with olive oil.\', \'Grill, covered, over medium heat for 4 to
6 minutes on each side or until lightly browned.\'] [\'time-to-make\', \'course\', \'main-ingredient\', \'preparation\', \'occa
sion\', \'side-dishes\', \'eggs-dairy\', \'refrigerator\', \'diabetic\', \'vegetarian\', \'grains\', \'cheese\', \'stove-top\',
\'dietary\', \'low-cholesterol\', \'low-calorie\', \'comfort-food\', \'low-carb\', \'low-in-something\', \'pasta-rice-and-grain
s\', \'brunch\', \'taste-mood\', \'equipment\', \'presentation\', \'served-hot\', \'4-hours-or-less\'] {\'diabetic\', \'low-cal
orie\', \'vegetarian\', \'low-carb\', \'side\'}'
```

Figure 12: First recipe Un-simplified Text

In order to make analysing of words easier I have removed all special characters and have turned all text into lower-case letters in Listing 9, which outputs list_of_all_recipes_lowercase, a list containing all the recipes, without any special characters in lower-text. An example of the

first recipe simplified can be seen in Figure 13

```
grilled garlic cheese grits we love grits this is another good way to serve them a great alternative to a baked potato when ser
ved with grilled steak or chicken i belive this recipe could be made with instant grits the hours for refrigeration is not incl
ude in time the recipe comes from tast of home s light and tasty water grits salt cheddar cheese garlic olive oil cups water cu
p uncooked old fashion grits teaspoon salt ounces shredded cheddar cheese clove garlic minced tablespoon olive oil i a sauce pa
n bring water to a boil slowly add grits and salt stirring constantly reduce heat simmer uncovered for minutes or untill thicke
ned stirrin occasionally add cheese and garlic stir until cheese is melted spray inch baking dish with nonstick cooking spray c
over and refrigerate for to hours or until frim before starting the grill coat the grill rack with nonstick cooking spray cut t
he grits into inch squares brush both sides with olive oil grill covered over medium heat for to minutes on each side or until
lightly browned time to make course main ingredient preparation occasion side dishes eggs dairy refrigerator diabetic vegetaria
n grains cheese stove top dietary low cholesterol low calorie comfort food low carb low in something pasta rice and grains brun
ch taste mood equipment presentation served hot hours or less diabetic low calorie vegetarian low carb side
```

Figure 13: First recipe with simplified text (list_of_all_recipes_lowercase[0])

4.1.1 TF - IDF

The TF - IDF algorithm is a statistical algorithm that measures the importance of a word in a document or a collection of documents in a specific category or a web as described by [Liu et al., 2018].

The basic idea behind this algorithm is to sort words based on how many times they appear in a desired document compared to the rest of the documents. For example, there are several words that appear many times in all documents, such as "to", "and", "then" so we can figure, using that logic that they won't stand out as much as actual important words of a document since when compared to other documents the ratio will likely be similar. Unlike a case of reading an article or a blog about vegan dishes versus a blog on Japanese art, the word "Tofu" is more likely to be an important word for the vegan article blog and seen as 'rare' for the Japanese art one.
In order to proceed with our following calculations, I have created a dictionary in my code, which will be used to keep track of different scores throughout the TF-IDF algorithms. Initially, we start off with a dictionary such that for each word, it assigns a value of the count how many times each word appears in a given recipe and it will also include and also include the 'tf','count_of_pages_with_word','idf' and 'tf_idf' which I will go into more detail about what each of these are later on in this chapter. I have also created a list containing all the dictionaries for all the recipes called dictionaries_of_all_recipes, so all the data is contained in one place. This is portrayed in Listing 10.

[TFIDF,] states that the Algorithm TF-IDF is made out of two components; Term Frequency (TF) and Inverse Document Frequency (IDF). Term frequency, refers to the number of occurrences a given word has appeared in the file, and the Inverse document frequency measures how important a term is in reference to other documents.
In order to calculated the TF score we follow the following equation:

$$TF = tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

For $n_{i,j}$ be the number of occurrences of the word t_i in the file d_j, and the sum $\sum_k n_{k,j}$ is the sum of occurrences of all words in the file (or simply the sum of all words in the file). It is essential to calculate the ratio of occurrences per page rather than just record the number of times appeared as a way of normalisation due to documents having different lengths and sizes. The code for the addition of the TF score into the dictionary is displayed in Listing 11.
We can then find the IDF score using the following equation:

$$IDF = idf_i = \log(\frac{\mid D \mid}{\{j : t_i \in d_j\}})$$

For $\mid D \mid$ be the total number of files in the set we're comparing the values between, and $\{j : t_i \in d_j\}$ be the number of documents in the set which contain the word t_i.
Taking the log is essential for a smoothing effect, which can be important when dealing with terms that have very high document frequencies. Terms with high document frequencies might dominate the IDF values and lead to an imbalanced representation of term importance when we don't incorporate the log. The code for the addition of the TF score into the dictionary is displayed in Listing 12.

Therefore the final value of TF-IDF is as follows,

$$TF - IDF = tfidf_{i,j} = tf_{i,j} \times idf_i$$

The code for the addition of the TF score into the dictionary is displayed in Listing 13 and the final list of dictionaries containing the 'tf-idf' values and more, of each word in each recipe looks like the following,

```
dictionaries_of_all_recipes

Out[19]: [{'grilled': {'count_of_word_per_page': 2,
    'tf': 0.009708737864077669,
    'count_of_pages_with_word': 7,
    'idf': 2.659260036932778,
    'tf_idf': 0.02581805861099784},
   'garlic': {'count_of_word_per_page': 4,
    'tf': 0.019417475728155338,
    'count_of_pages_with_word': 37,
    'idf': 0.9942522733438669,
    'tf_idf': 0.019305869385317802},
   'cheese': {'count_of_word_per_page': 6,
    'tf': 0.02912621359223301,
    'count_of_pages_with_word': 34,
    'idf': 1.07880966137193,
    'tf_idf': 0.03142164062248 34},
   'grits': {'count_of_word_per_page': 6,
    'tf': 0.02912621359223301,
    'count_of_pages_with_word': 1,
    'idf': 4.605170185988092,
```

Figure 14: Filled in dictionary

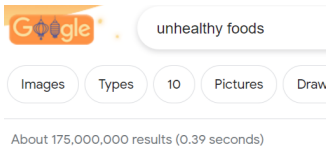
We can notice that the significance of a word in a document grows proportionally with its frequency within that document, but this impact is offset by the overall frequency of the word across the entire set of documents.

High Term frequency + low Inverse document frequency $\Longrightarrow$ high - weight TF-IDF

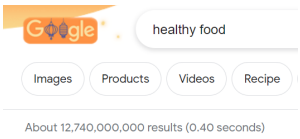
Remark. *The TF-IDF score varies for each word in each document. E.g., the word "house" in document A will have a different TF-IDF score than the word "house" in document B. This is a result of the TF score varying for each word in different documents although the IDF score remains the same per word all thorughout.*

As the information on the exact algorithms google currently uses are unavailable to the general public, we can only guess the implications of the TF-IDF algorithm.

We can see this in Figure 15 where we can see a display of 2 similar Google searches , the "unhealthy foods" search came back with 75,000,000 results while the "healthy foods" came back with 12,740,000,000 results, from this we can understand the the sorting process was only done on a specific collection of relevant web-page rather than the whole wide web which had allowed for time to be optimized.



(a) Google search for unhealthy foods



(b) Google search for unhealthy foods

Figure 15: Web results of two searches

As we know, usually Google searches are queries rather than specific words, so we can propose a method for the use of the TF-IDF algorithm in Google and how it linked to the page-rank ranking: [Ramos, 2003]

1. Given a query q composed of a set of words t_i, we calculate $tfidf_{i,j}$ for each word t_i in the collection of all document D.

2. Once $tfidf_{i,j}$ is found for all words for all files in D, we return a set $D^* \in D$ containing all the documents in D such that $\sum_i tfidf_{i,j} \in d_j > 0$ strictly, so we would like to return a set of all documents d_j that contains all the documents that have at least one positive $tfidf_{i,j}$ for a word t_i

3. Once the set D^* is found, we can allocate each page a score of the sum of the TF-IDF scores of all words in the query, and allocate a vector ϕ to these scores in order of pages.

4. In order to make sure the algorithm is balanced and coherent with the page-rank algorithm and the score of π , we then normalise the vector ϕ under the 1-norm in order to allow for all TF-IDF scores to be a ratio between all pages such that all these values sum up to 1

5. We can then pull out all the relevant pages D^* from π so that π and ϕ are of equal size and also normalise the new vector under the 1-norm so that all of of the elements of it sum up to one, and label the new vector as π^*

6. Finally we can control the balance between the page-rank scores and the TF-IDF scores using a parameter β in following way:

$$\sigma = \beta \cdot \pi^* + (1 - \beta) \cdot \phi$$

So in order to apply all these steps onto the code in listing 14, I first started off by making a function which inputs target words and all the dictionaries and then picks our the most relevant pages using their tfidf scores.

When I search for the words: 'bread' and 'shrimp' in the mock "search bar", I will get results such as in Figure 17, which portrays a list constraining many sub-lists with the info[recipe number, sum of tf_idf scores in the recipe], which is a list with the numeration of the recipes which are relevant, alongside their score.

```
relevant_pages_to_search = set_of_relevant_pages(['bread' , 'shrimp'], dictionaries_of_all_recipes)
relevant_pages_to_search
```

```
[[1, 0.07718168972628116],
 [10, 0.010183660011102732],
 [16, 0.026852291456997995],
 [19, 0.03543913683863751],
 [23, 0.0031032519123150182],
 [26, 0.018267596308576033],
 [29, 0.08157239377094411],
 [34, 0.1162518048932362],
 [38, 0.013174400311761154],
 [39, 0.005258032171904675],
 [44, 0.12009561457115771],
 [50, 0.013457900065305383],
 [64, 0.024497099197214867],
 [65, 0.015774096515714025],
 [71, 0.008859784209659377],
 [75, 0.01136435913219136],
 [79, 0.008578258183654121],
 [80, 0.03880197464084399],
 [87, 0.016457803485435376],
 [91, 0.017719568419318755],
 [94, 0.006089198769525345],
 [95, 0.0079105216157673],
 [96, 0.03236450852843608]]
```

Figure 16: The relevant pages to search alongside their TF-IDF scores

I have then separated the sub-lists into individual vectors in order to use them for future calculations, as portrayed below and in Listings 15 and 16.

Which returns:

```
TF_IDF_relevant_pages_vector = vector_of_pages(relevant_pages_to_search)
print(TF_IDF_relevant_pages_vector)
```

```
[1, 10, 16, 19, 23, 26, 29, 34, 38, 39, 44, 50, 64, 65, 71, 75, 79, 80, 87, 91, 94, 95, 96]
```

Figure 17: Relevant pages to search

```
TF_IDF_scores_of_relevant_pages = vector_of_tf_idf_scores(relevant_pages_to_search)
print(TF_IDF_scores_of_relevant_pages)
```

```
[0.07718168972628116, 0.010183660011102732, 0.026852291456997995, 0.03543913683863751, 0.0031032519123150182, 0.018267596308576
033, 0.08157239377094411, 0.1162518048932362, 0.013174400311761154, 0.005258032171904675, 0.12009561457115771, 0.01345790006530
5383, 0.024497099197214867, 0.015774096515714025, 0.008859784209659377, 0.01136435913219136, 0.008578258183654121, 0.0388019746
4084399, 0.016457803485435376, 0.017719568419318755, 0.006089198769525345, 0.0079105216157673, 0.03236450852843608]
```

Figure 18: Scores of relevant Pages

As mentioned in (4), the vector TF_IDF_scores_of_relevant_pages needs to be normalised in order to create a ratio with it and the value of π. So we introduce a function that will be used throughout the next few algorithms that normalises a vector under the 1-norm in Listing 17.

In the figure below, we can see the new normalised vector which I call Φ, which is created after applying the normalising function onto TF_IDF_scores_of_relevant_pages, and sorting it from highest to lowest.

```
sorted_phi
```

```
[(44, 0.16932643961454963),
 (34, 0.16390693608278029),
 (29, 0.11501138536483434),
 (1, 0.10882079892303323),
 (80, 0.05470807772131642),
 (19, 0.049966710985468996),
 (96, 0.04563169953010867),
 (16, 0.03785985794853181),
 (64, 0.034539201142029255),
 (26, 0.02575603659044793),
 (91, 0.024983355492734498),
 (87, 0.023204354946812226),
 (65, 0.022240375809555932),
 (50, 0.018974700374228732),
 (38, 0.01857498549645687),
 (75, 0.016022953687579486),
 (10, 0.014358250283180744),
 (71, 0.012491677746367249),
 (79, 0.012094745686753545),
 (95, 0.011153283702113614),
 (94, 0.008585345530149311),
 (39, 0.00741345860318531),
 (23, 0.004375368737781873)]
```

Figure 19: Sorted ϕ

As mentioned in (5), we pull out the relevant recipes out of vector π based on the indices in the vector in Figure 17, which will then need to be normalised and results in $\pi*$ as described in Listing 18 and shown in Figure 20

```
sorted_pi_star
```

```
[(87, 0.05632969852105303),
 (10, 0.049512657255788),
 (96, 0.04895229904717495),
 (29, 0.04884763036351946),
 (26, 0.047529937816557025),
 (65, 0.047359831943449594),
 (16, 0.04721280708876653),
 (80, 0.04622231339558105),
 (79, 0.04602267184612831),
 (71, 0.04564681678644107),
 (94, 0.04474710464813906),
 (38, 0.04407025003307153),
 (64, 0.043588120983909656),
 (34, 0.04186674017428943),
 (50, 0.041469232105120345),
 (19, 0.040932451892393584),
 (39, 0.0402359674552529),
 (91, 0.03904558694903581),
 (1, 0.03897777795752598),
 (23, 0.036424567376246775),
 (95, 0.03570020456592465),
 (44, 0.03534421798558383),
 (75, 0.033961113809047046)]
```

Figure 20: Sorted π^*

Sigma can then be found using the equation in (6), and is described in Listing 19 which the results in a sorted order shown in Figure 21.

```
sorted_sigma
```

```
[(34, 0.0662747793559876),
 (44, 0.06214066231137698),
 (29, 0.06208038136378243),
 (1, 0.05294638215062743),
 (87, 0.049704629806204874),
 (96, 0.0482881791437617),
 (80, 0.04791946626072812),
 (16, 0.04534221726071959),
 (26, 0.04317515757133521),
 (19, 0.04273930371100866),
 (10, 0.04248177586126655),
 (65, 0.042335940716670864),
 (64, 0.04177833701553357),
 (79, 0.03923708661425336),
 (71, 0.03901578897842631),
 (38, 0.0389711971257486),
 (94, 0.03751475282454111),
 (50, 0.03697032575894202),
 (91, 0.036233140657775546),
 (39, 0.03367146568483939),
 (95, 0.030790820393162444),
 (75, 0.030373481784753534),
 (23, 0.030014727648553795)]
```

Figure 21: Sorted σ

I have selected $\beta = 0.8$, for the calculation of $\sigma = \beta \cdot \pi^* + (1 - \beta) \cdot \phi$ based on the ideology mentioned in Subsection 1.1 which specifies that the page-rank 'algorithm revolutionised the way search results were ranked, making the digital landscape more navigable and optimised.' therefore we can let the page-rank score have higher importance rate than of the TF-IDF score (or any other word classification algorithm) which was implemented previously to the page-rank algorithm.

4.2 Analyzing the impact of clicks on a page's ranking

Among the algorithms that Google is likely to integrate into its complete web-ranking algorithm, we will now shift our focuses onto ranking pages according to the number of clicks they receive.

The number of clicks per page can significantly impact a page's ranking within search engine results pages. When a page receives a high number of clicks, it signals to Google that the content is likely relevant and useful to users searching for specific queries. Consequently, Google tends to prioritize pages with higher click-through rates in its search results, as they are perceived to offer valuable information.

Moreover, utilizing click-through rates as a ranking factor aligns with Google's overarching goal of providing the best possible user experience. By prioritizing pages with higher click-through rates, Google ensures that users are more likely to find what they're looking for quickly and easily. This approach encourages website owners to create high-quality, engaging content that resonates with their target audience,

It is clear to see the the number of clicks per page is a number that constantly updates and throughout each day the number of clicks increases. We can additionally recognise that the Google matrix also constantly updates, It is estimated that "Google updates and modifies its search algorithm about 500 or 600 times every year." as cited by [Hall, 2023], which implies that the matrix probably updates once or twice a day.
If we allow the number of clicks on each link to reset at every google algorithm update we may be able to create a suitable ratio between the pages based on the number of clicks in the time span of a day / half a day.

Using data from [Fitzgerald, 2023], it was found that for the websites participating in survey, 46% of websites have between 1001-15k clicks, 19.3% of websites have between 15,001-50k clicks, 23.2% of websites have between 50,001-250k clicks, 11% of websites have between 250,001-10M clicks and finally 0.5% of websites have between 10M clicks, when counting the number of clicks per month.
I calculated the average number of clicks per website using the upper bound of each sector, and with 20M clicks for the highest sector for the span of 12 hours as it is estimated to be refreshed around twice a day and received an average of about 21,243 clicks per website per 12 hours.

In order to incorporate the number of clicks inside the page-rank algorithm, we propose a method suggested by Gyanendra Kumar, Neelam Duhan and A. K. Sharma in their research [Kumar et al., 2011] that provide weight to each link in the graph based on their number of clicks

4.2.1 Adapted Page-Rank Algorithm based on Visits Of Links (VOL)

In our original algorithm for finding the rank of all pages, we have assumed that getting from a certain page, to any of the other pages linked to that certain page has equal probability, so the probabilities were uniformly distributed. For example,

$$\mathcal{P}(B \mid A) = \mathcal{P}(C \mid A) = \mathcal{P}(D \mid A) = \mathcal{P}(E \mid A) = \mathcal{P}(H \mid A) = (\frac{1}{5})$$

To our data collected, we add a variable containing the information of how many clicks each website receives in a time span of about half a day, although this statistic gets updated

constantly, we let these numbers stay fixed in our algorithm for the sake of simplicity. We label as the number of clicks per page as C_j for all pages $j \in \{1, 2, 3, \cdots, n\}$ we then create a probability for the weight of each link in the following way:

$$\mathcal{P}_{C_j} = \frac{C_j}{|C|} \tag{10}$$

Where $|C|$ represents the total number of clicks across all pages which page i links to.
All these probabilities will sum up to 1 and therefore can be used to create a new transition matrix which will be used for the page-rank algorithm we have constructed previously.
Using the same data sets as before, we implement the new algorithm onto the original graph.

In the code in Listing 20, I assigned a number of clicks per page using a Normal distribution. This was essential as the ratio $\mathcal{P}_{C_j} = \frac{C_j}{|C|}$ would be extreme and unrealistic if we had just used a random distribution of numbers, which could severally affect the final score for π. A normal distribution is much more realistic as more numbers are extracted from the region revolving the mean, depending on the standard deviation, which represents a much more realistic counts of clicks per page and a less extreme ratio for $\mathcal{P}_{C_j}$.

I let the mean be 20,000 for the rounding down of 21,243 clicks, with a standard deviation of 5,000 which I extracted using trial and error and seeing the range of results and deciding which list of clicks per page represents the most realistic scenario. For example, if we try s_d to be 20,000, 10,000 and 5,000, we see that in Figure 23 the most reasonable and realistic results that won't lead to an extreme convergence would be at about 5,000 clicks.

```
print(assign_score_per_page(0,20000,10000,20000,k))
print(assign_score_per_page(0,20000,10000,10000,k))
#Below is the selected number for standard deviation

clicks_per_recipe = assign_score_per_page(0,20000,10000,5000,k)
print(clicks_per_recipe)
```

```
[14734, 7273, 17770, 19672, 10034, 10353, 16610, 15219, 752, 15095, 18691, 17048, 1290, 13440, 16581, 3083, 9192, 15936, 18207,
11019, 4538, 4743, 7532, 17463, 10346, 6273, 10799, 19268, 3800, 11870, 6548, 18582, 7619, 7655, 4250, 4249, 16833, 8902, 1018,
3173, 4165, 3327, 17844, 5112, 3871, 1428, 15278, 14911, 14146, 1570, 3602, 5564, 4347, 4442, 5253, 10587, 8452, 2802, 578, 800
4, 13601, 18767, 16157, 11157, 201, 18412, 16000, 9307, 7340, 15396, 14102, 12117, 8897, 9247, 19773, 9408, 17521, 7193, 10981,
2394, 4860, 19456, 16100, 16985, 6981, 11329, 12556, 10801, 16360, 12945, 11981, 6644, 4292, 8696, 15331, 12610, 13809, 5519, 7
795, 11315]
[17477, 13622, 9444, 18258, 17362, 19568, 18200, 2051, 7305, 11961, 6367, 17154, 14664, 4182, 2132, 10926, 12586, 14795, 6623,
14703, 3193, 6710, 10312, 1673, 11099, 6714, 14339, 13801, 19476, 956, 17560, 11528, 12568, 15465, 12617, 585, 9168, 12159, 550
6, 9226, 5453, 5250, 9389, 7485, 8773, 12347, 746, 9749, 6377, 16490, 2729, 9480, 12242, 9224, 3266, 6142, 12945, 15084, 4491,
8956, 984, 8676, 8516, 13625, 13090, 280, 14328, 14800, 19508, 18684, 10349, 5859, 19869, 9328, 3918, 19946, 16901, 10706, 902
8, 12233, 10879, 748, 13687, 9147, 13663, 7166, 2283, 2505, 3074, 1776, 13843, 19641, 19496, 8716, 19880, 8250, 9689, 5609, 964
9, 14282]
[16383, 7323, 13024, 10966, 8645, 10197, 5534, 3216, 15948, 8350, 18058, 10238, 16035, 13207, 7528, 16223, 9900, 12771, 9744, 1
4308, 10809, 14340, 14993, 7685, 12809, 10702, 16134, 6424, 2108, 1262, 5628, 10038, 12272, 1371, 14481, 9917, 16034, 3483, 120
17, 3182, 13637, 3240, 14009, 6921, 13168, 10733, 8698, 9255, 5741, 13907, 7316, 11675, 16136, 12486, 8828, 8734, 10565, 10099,
18324, 13091, 6111, 15863, 14464, 9461, 14719, 15595, 14989, 6873, 6383, 4057, 13785, 16594, 12513, 8114, 9764, 11540, 3730, 94
91, 11745, 5459, 1220, 10864, 7058, 18020, 2356, 13671, 7487, 8121, 6512, 16165, 3767, 12637, 8131, 17184, 10781, 7855, 10273,
16998, 5871, 6731]
```

Figure 22: Different versions of Standard Deviation

Now that I have a list for the number of clicks per recipe, I created a new transition matrix that represents the probabilities described in equation, 10 as seen in Figure 23 and in Listing 21.

```
print(clicks_matrix)
```

```
[[0.08047253 0.         0.         ... 0.         0.         0.03306236]
 [0.         0.01831776 0.03257824 ... 0.         0.01468572 0.        ]
 [0.03484825 0.         0.         ... 0.         0.01248819 0.0143175 ]
 ...
 [0.         0.         0.02305573 ... 0.03009071 0.01039314 0.        ]
 [0.05593491 0.         0.         ... 0.         0.         0.        ]
 [0.02268312 0.01013908 0.01803241 ... 0.02353462 0.00812871 0.00931942]]
```

Figure 23: Different versions of Standard Deviation

In Listing 22 I applied the page-rank and sorting algorithms onto the new matrix to find sorted_clicks_pi as seen in Figure 24.

```
sorted_clicks_pi = score_indexing_sorted_in_order([i for i in range(100)],clicks_pi)
print(sorted_clicks_pi)
```

```
[(61, 0.022338989039585148), (97, 0.02143692349711708), (62, 0.019173705827445563), (10, 0.0177055191988588), (93, 0.0163137457
0274989), (36, 0.016255811015161718), (71, 0.01622088938755609), (26, 0.015562972121153356), (15, 0.015466809826267343), (21,
0.015459073487592616), (58, 0.015424319908882548), (12, 0.01455619382279902), (83, 0.014545835219024444), (53, 0.01436873857393
5318), (8, 0.014295865852497314), (55, 0.013980532118648607), (89, 0.013848637136567839), (11, 0.013559851666657085), (65, 0.01
3536712186598068), (52, 0.013460116818209607), (40, 0.013420207479571688), (64, 0.013311604624603866), (66, 0.01328140877189191
5), (3, 0.013252156293559475), (0, 0.012603407559458789), (59, 0.012525486888479168), (19, 0.012437771866001974), (34, 0.012328
081143287338), (20, 0.012218303250515362), (70, 0.012118847982456662), (13, 0.012069756887304982), (81, 0.012057514332388827),
(49, 0.011897465500171811), (42, 0.01174817281252054), (2, 0.011667936658681446), (54, 0.011661572611811756), (32, 0.0115545284
72319685), (72, 0.011385030568603117), (87, 0.011164970785202162), (38, 0.01087144682825413), (24, 0.010847342819773322), (63,
0.010833249962094954), (96, 0.010812994300642764), (25, 0.010581083456580277), (57, 0.010543985745677631), (47, 0.0105096806485
83751), (94, 0.010340314394776408), (44, 0.010309340245655836), (5, 0.010249870400940508), (78, 0.01008398628116845), (16, 0.01
0064450154375997), (91, 0.009646253641630518), (9, 0.009553965065029174), (77, 0.0095304041770882), (22, 0.0094746468462265 17),
(85, 0.00909566669800099), (17, 0.009002422481071625), (45, 0.00892455150935155), (51, 0.008732383167725906), (92, 0.0087231639
36247716), (56, 0.008677255383672144), (31, 0.008566603609160076), (75, 0.008437232370192888), (46, 0.008412510161169296), (35,
0.008109501169248681), (67, 0.007834657053908905), (43, 0.007779943112870302), (86, 0.007559751899302881), (18, 0.0075238374596
93095), (74, 0.007439431861352036), (4, 0.0071548421781575415), (68, 0.007014926507231694), (50, 0.007012810950757679), (98, 0.
006995050082972041), (48, 0.006915778946259492), (82, 0.006774223486602675), (95, 0.006420263558009552), (1, 0.0064120467936126
9), (23, 0.006324202539476963), (79, 0.006323667608453530 4), (99, 0.006251365967431338), (88, 0.006192869640574659), (27, 0.006
094090183174383), (14, 0.006076846973074297), (73, 0.006071682866887775), (6, 0.005974751811254049), (60, 0.00566197610378683
6), (90, 0.005418279028678481), (30, 0.005317976936645754), (37, 0.005057895311265613), (69, 0.0048996090559566), (76, 0.004144
125216215163), (41, 0.0040703826011973376), (39, 0.0038134389817242945), (84, 0.0036979666971009347), (7, 0.00358430054788910
2), (28, 0.0032371347088394845), (33, 0.002608751884032152), (29, 0.0026031722654920957), (80, 0.00258817682959955)]
```

Figure 24: Clicks π

Just as we did to π before, we now do the same to to clicks_pi, we pull out the values of the elements of the relevant pages based on the words we put in the "search bar" and then the vector gets normalised in order to get $\widetilde{\pi}$, as done in Listing 23 and shown in Figure 25.

```
sorted_pi_tilde
```

```
[(10, 0.07895496470224062),
 (71, 0.07233449268835836),
 (26, 0.06940061461551593),
 (65, 0.06036482866574899),
 (64, 0.0593609970688419),
 (19, 0.05546427798163654),
 (34, 0.05497512953911837),
 (87, 0.04978842271420106),
 (38, 0.04847949901646574),
 (96, 0.04821883920737019),
 (94, 0.04611099786908614),
 (44, 0.045972873546210914),
 (16, 0.04488082488636848),
 (91, 0.04301594362919826),
 (75, 0.0376245043419 0704),
 (50, 0.031272522136275294),
 (95, 0.028630150683996807),
 (1, 0.028593509321739402),
 (23, 0.02820178175325172),
 (79, 0.028199396312892762),
 (39, 0.017005428466373165),
 (29, 0.011608435314849603),
 (80, 0.011541565538352623)]
```

Figure 25: Sorted $\widetilde{\pi}$

Now that I have the result of $\widetilde{\pi}$, it can be used in order to find a new sorted $\widetilde{\sigma}$ generated using the new clicks Transition matrix Which can be seen in Figure 26 and explained in Listing 24

```
sorted_sigma_tilde = score_indexing_sorted_in_order(TF_IDF_relevant_pages_vector,sigma_tilde)
sorted_sigma_tilde
```

```
[(34, 0.07676149084785075),
 (44, 0.07064358675987864),
 (10, 0.06603562181842863),
 (26, 0.06067169901050233),
 (71, 0.06036592969996014),
 (64, 0.05439663788347937),
 (19, 0.05436476458240303),
 (65, 0.052739938094510386),
 (96, 0.04770141127191789),
 (1, 0.044638967241998166),
 (87, 0.04447160916072329),
 (16, 0.04347663149880114),
 (38, 0.042498596312463964),
 (91, 0.03940942600190551),
 (94, 0.038605867401298774),
 (75, 0.033304194211041534),
 (29, 0.032289025324846544),
 (50, 0.02881295778386598),
 (95, 0.025134777287620168),
 (79, 0.02497846618766492),
 (23, 0.023436499150157754),
 (80, 0.02017486797494538),
 (39, 0.015087034493735595)]
```

Figure 26: Sorted $\widetilde{\sigma}$

This score can be compared with the values of σ we have found at the end of Subsection 4.1.1. When comparing the two scores, we can notice that recipes 34, 44 both all remain as top scores for both σ and $\widetilde{\sigma}$, but recipe 29 which was the 3^{rd} most relevant recipe in σ was bumped all down to be in the bottom ten for $\widetilde{\sigma}$, which shows that that recipe 29, may have been very well connected to other recipes, but in terms of interest and relevance to the web-surfer, it wasn't a popular choice at all. This may indicate that page 29 is potentially a spam-page, as pages with low click-through rates may be less useful, spammy or are a page in a Link-Farm.

Additionally it is important to keep in mind that pages that receive an unusually high number of clicks without providing significant value to the website's content may raise red flags also as a spam page taking advantage of a Link-Farm. More on Spam Pages and Link-Spamming will be introduced in Subsection 5.3.

4.3 Exploring website loading times and their effects on Page-Rank scores

Now that the idea of the number of clicks per page affecting a page's rank was introduced, we move onto expressing a potential method for how a page's loading time / speed may affect the final ranking of the websites too.

As we know, over time, pages have become more and more complex in order to personalise the design and product they are interested in portraying to the web-traveller. web-pages may include videos, different graphics, advertisements, long Java scripts and many more. The more features a page is interested in including, the heavier the code to produce the website is, the more information there is to process and the more steps required to loads the full website.

As mentioned in [Bhan, 2022], especially for mobile users, the loading time is the most important feature relation whether they will enter a page or not. As we can see in Figure 27 below, loading time of a page is a much more relevant criteria than availability of information or page display aesthetics.

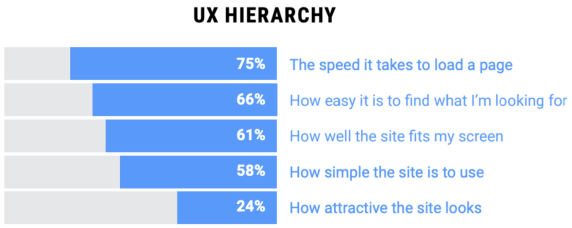


Figure 27: A Chart Comparing Interest rates on different features of a site, from [Bhan, 2022]

In a previous research that was conducted by [Google/SOASTA, 2017], it was found that a 1-3 seconds delay will increase the likelihood of a user to dis-engage and exit the website by 32% and that "53% of visits are likely to be abandoned if pages take longer than 3 seconds to load" as described in a [Shellhammer, 2016] Google Blog.

Therefore we can conclude that the slower a website is, the unhappier the Web traveller will be so the more likely they are to be disengaged or leave the website.
Google has picked up on these characteristics of behaviours, so Google's team have chosen to include Page-Speed into the page-rank calculations. It is clear that the lower the performance of a page with respect to the loading time is, the more of a negative effect it will have on the page-rank score.

In 2010, two Google team members wrote: "Like us, our users place a lot of value in speed, that's why we've decided to take site speed into account in our search rankings." as mentioned in [Oziemblo, 2019]
Then later in July 2018 Google released the following statement in a blog by [Camarena, 2023], "Users want to find answers to their questions quickly and data shows that people really care about how quickly their pages load. The Search team announced speed would be a ranking signal for desktop searches in 2010, and as of this month (July 2018), page speed will be a ranking factor for mobile searches too." So all the speculations that were taking place were then confirmed.

As mentioned in Section 4, there is no clear indication on how Google calculates page-speed and how it is incorporated into the algorithm, so in the rest of the chapter I will propose an algorithm for calculating a vector which represents the ratios between the speeds of the websites:

1. We start with a list of websites, or in our case - recipes and a list of loading times for each website.

2. We then assign the following time ranges (in seconds) a probability between 0 and 1 of how likely a user is to stay on the website and wait for it to load, given its loading time. The probabilities and time ranges will vary between a desktop and a mobile device, so in the following example, I will provide data for a desktop's standard loading time.

   (a) if a website loading time is between: (0 , 0.5] seconds, its assigned probability is: 1

   (b) if a website loading time is between: (0.5 , 1] seconds, its assigned probability is: 0.9

   (c) if a website loading time is between: (1 , 2] seconds, its assigned probability is: 0.8

   (d) if a website loading time is between: (2 , 3] seconds, its assigned probability is: 0.7

   (e) if a website loading time is between: (3 , 5] seconds, its assigned probability is: 0.5

   (f) if a website loading time is between: (5 , 10] seconds, its assigned probability is: 0.2

   (g) if a website loading time is above: 10 seconds, its assigned probability is: 0.05

3. Create a new vector called ρ containing the probabilities assigned to each website using the ranges above.

4. We can then pull out all the relevant pages D^* from ρ so that we can create a suitable ratio between all the other vectors and let ρ^* be the normalised vector under the 1-norm of ρ so that all of of the elements of it sum up to one.

5. Finally we can control the balance between the page-rank scores with the weighted links $\widetilde{\pi}$, the TF-IDF scores ϕ and the loading time vector ρ^* using the parameters β, γ in following way:
$$\theta = \beta \cdot \widetilde{\pi} + \gamma \cdot \rho^* + (1 - \beta - \gamma)\phi$$

So if we apply all of this into code, we have to start off with a function that assigns a loading time for each recipe, it is very similar to listing 23 but I removed the conversion of decimals into integers, as we are able to have non-rounded seconds for the loading times as we can see in the example in Figure 28 and in Listing 25.

I then made a function that assigns the probabilities in bullet (2) to each recipe based on which range the loading time falls in Listing 26 and labels it as ρ with the time frames [0,0.5], [0.5,1], [1,2], [2,3], [3,5], [5,10], 10+ , and the probabilities 1, 0.9, 0.8, 0.7, 0.5, 0.2, 0.05, displayed in Figure 29

I then applied the same process I had before on π where I extracted the relevant pages and then normalised the vector in order to find ρ^* as shown in Figure 30 and in Listing 27

Now that we have the value of ρ^* we can find the balance between ρ^* and the two other vectors called θ, using the equation in (5) and Listing 28.

In the Figure 31 below I let $\beta = 0.5$ and $\gamma = 0.3$ so that $1 - \beta - \gamma = 0.2$ in order to find the value of θ and so the relevance of the scoring is page-rank $\rightarrow$ Loading-time $\rightarrow$ TF-IDF.

```
print(loading_time_per_recipe)
```

```
[8.380722793020436, 7.181734462565376, 10.951532927328461, 0.9398860369020343, 8.112262122319748, 2.1121423818014033, 1.8010792
498455435, 6.252077015906475, 3.740481663072782, 2.584016017755131, 8.332631828546802, 1.2790492987920536, 6.852522471539485,
5.430493820364008, 1.7696433863664804, 2.6117119814950533, 3.2798652056025652, 8.0005387450204, 12.151228348090056, 2.148741923
162116, 2.7793217007503954, 2.0445836788394613, 5.525667806463451, 1.1090125907256831, 3.678140534487058, 8.015329174097609, 4.
8831153495254815, 3.225393078162789, 4.0110123884147875, 4.9224028369134025, 2.2540648998673958, 1.4955546096290135, 4.53540972
4792123, 11.483081610171054, 5.694671249229552, 3.6706426034582487, 1.8764289119243258, 3.794540452498775, 5.45601554938667, 6.
306639798073286, 3.3656743808583496, 4.073738303152181, 1.659143956179903, 16.719451219482835, 5.479996282943932, 1.14097101415
2775, 6.872064340770228, 5.344069756138288, 7.019189891607078, 1.6418518798530561, 7.8767147868691545, 1.86516600057327257, 2.12
2052502533471, 4.385318838148734, 3.5931324996600615, 6.528760504063865, 7.620451060082319, 9.68576212769332, 7.19045240610187
1, 5.8355350802052275, 9.393570938829683, 0.053329966240682314, 4.840966647584898, 1.4125241633888572, 5.838174329570325, 17.22
458622921898, 3.1386716862611777, 4.293834802882345, 2.3214269462794865, 2.7185223399271563, 5.8992601278321795, 5.900432144688
605, 3.6979121931544956, 5.352851271524945, 5.19394542725689, 3.8230417147099987, 5.439137697543687, 3.59302366174889, 2.434140
7916180455, 2.019904906777436, 4.234793305032712, 3.278416328746114, 3.620194306815234, 2.6562372370665495, 3.360385675521021,
7.09378561258211, 0.6253594447136077, 7.473186284664072, 8.666489527046801, 2.4306173432050575, 2.8726819587124033, 0.613556298
8418473, 0.9085435228261791, 0.3425849842151325, 8.068800315045277, 4.364872441587938, 4.665842842839286, 6.619240053457631, 7.
687719543749349, 3.4675590545978943]
```

Figure 28: A set of 100 loading times

```
print(sorted_rho)
```

```
[(3, 0.9), (61, 0.9), (86, 0.9), (91, 0.9), (92, 0.9), (93, 0.9), (6, 0.8), (11, 0.8), (14, 0.8), (23, 0.8), (31, 0.8), (36, 0.
8), (42, 0.8), (45, 0.8), (49, 0.8), (51, 0.8), (63, 0.8), (5, 0.7), (9, 0.7), (15, 0.7), (19, 0.7), (20, 0.7), (21, 0.7), (30,
0.7), (52, 0.7), (68, 0.7), (69, 0.7), (78, 0.7), (79, 0.7), (83, 0.7), (89, 0.7), (90, 0.7), (8, 0.5), (16, 0.5), (24, 0.5),
(26, 0.5), (27, 0.5), (28, 0.5), (29, 0.5), (32, 0.5), (35, 0.5), (37, 0.5), (40, 0.5), (41, 0.5), (53, 0.5), (54, 0.5), (62,
0.5), (66, 0.5), (67, 0.5), (72, 0.5), (75, 0.5), (77, 0.5), (80, 0.5), (81, 0.5), (82, 0.5), (84, 0.5), (95, 0.5), (96, 0.5),
(99, 0.5), (0, 0.2), (1, 0.2), (4, 0.2), (7, 0.2), (10, 0.2), (12, 0.2), (13, 0.2), (17, 0.2), (22, 0.2), (25, 0.2), (34, 0.2),
(38, 0.2), (39, 0.2), (44, 0.2), (46, 0.2), (47, 0.2), (48, 0.2), (50, 0.2), (55, 0.2), (56, 0.2), (57, 0.2), (58, 0.2), (59,
0.2), (60, 0.2), (64, 0.2), (70, 0.2), (71, 0.2), (73, 0.2), (74, 0.2), (76, 0.2), (85, 0.2), (87, 0.2), (88, 0.2), (94, 0.2),
(97, 0.2), (98, 0.2), (2, 0.05), (18, 0.05), (33, 0.05), (43, 0.05), (65, 0.05)]
```

Figure 29: ρ, Assigned scored for each page based on loading time

```
sorted_rho_star
```

```
[(91, 0.10169491525423728),
 (23, 0.0903954802259887),
 (19, 0.07909604519774009),
 (79, 0.07909604519774009),
 (16, 0.05649717514124293),
 (26, 0.05649717514124293),
 (29, 0.05649717514124293),
 (75, 0.05649717514124293),
 (80, 0.05649717514124293),
 (95, 0.05649717514124293),
 (96, 0.05649717514124293),
 (1, 0.022598870056497175),
 (10, 0.022598870056497175),
 (34, 0.022598870056497175),
 (38, 0.022598870056497175),
 (39, 0.022598870056497175),
 (44, 0.022598870056497175),
 (50, 0.022598870056497175),
 (64, 0.022598870056497175),
 (71, 0.022598870056497175),
 (87, 0.022598870056497175),
 (94, 0.022598870056497175),
 (65, 0.005649717514124294)]
```

Figure 30: Sorted ρ^*

```
theta = finding_theta(pi_tilde,TF_IDF_scores_of_relevant_pages,rho_star,0.5,0.3)
sorted_theta = score_indexing_sorted_in_order(TF_IDF_relevant_pages_vector,theta)
sorted_theta
```

```
[(34, 0.06688288024882948),
 (44, 0.06353489515575221),
 (19, 0.05418308908195754),
 (26, 0.051480021228579365),
 (91, 0.04716282539124221),
 (10, 0.047052354365750566),
 (96, 0.045118207190464506),
 (71, 0.04334495561837643),
 (1, 0.04197103559005348),
 (16, 0.041795534908532224),
 (29, 0.04157537081695662),
 (64, 0.04154940230488485),
 (65, 0.036044586790413564),
 (87, 0.034351326414030575),
 (75, 0.033520994938859516),
 (23, 0.033110962495518106),
 (38, 0.032711843613060655),
 (79, 0.03249238465109064),
 (94, 0.029402032576700106),
 (80, 0.028710810189678096),
 (95, 0.02798766685497718),
 (50, 0.024193405099028696),
 (39, 0.0145998978960574 2)]
```

Figure 31: Sorted θ

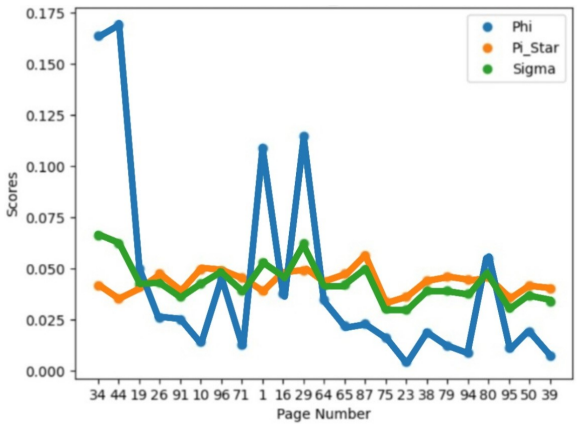
Below, in Figure 32 we can see four graphs, where each graph compares the scores of each vector found in the previous algorithms, ordered from biggest to smallest using the final ranking of sorted θ

(**graph a**) As ϕ represents the vector of the relevant pages based on TF-IDF, it would have no real correlation to the score of π^* and we can see that both π^* and σ lie pretty flat on the graph and have a pretty strong correlation due to π^* weighing 80% of the score of σ.

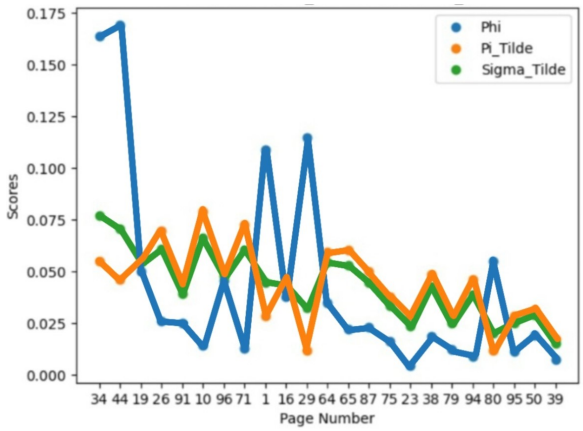
(**graph b**) We can see that it is also true for $\widetilde{\pi}$, although $\widetilde{\pi}$ and π^* do seem to have quite a more drastic change between the values of the pages and so $\widetilde{\pi}$ and $\widetilde{\sigma}$ are seen as less flat.

(**graph c**) $\widetilde{\pi}$ and ρ^* also have no correlation due to the fact the the loading times are randomly generated but we can see that for the final score of θ the graph is now in a decreasing order as desired.

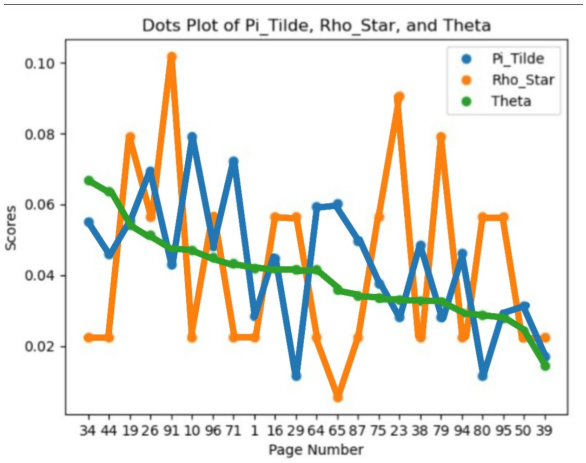
(**graph d**) Finally I have chosen to present the two final scores after creating the balancing using α and β compared to the original θ^* just to show how the scores ares seeming to align closer toegther as we add more variables.



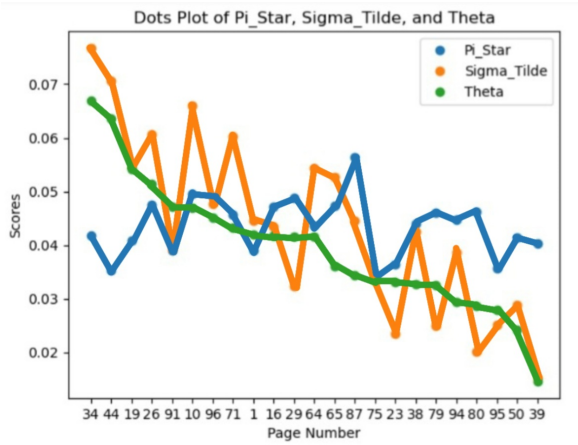
(a) Graph of π^*, ϕ and σ



(b) Graph of $\widetilde{\pi}$, ϕ and $\widetilde{\sigma}$



(c) Graph of $\widetilde{\pi}$, ρ^* and θ



(d) Graph of π^*, $\widetilde{\sigma}$ and θ

Figure 32: Graphs aiding in visualisation of the patterns of the vector values found previously

Now that we have found these scores, we'll explore other ideas that may affect Google's search ranking, without implementation.

5 Additional factors influencing Page-Rank scores: Insights and Considerations

In the previous chapter, I have introduced several concepts regarding how I assume Google enhances its page-rank algorithm. These include considerations such as clicks per page, page loading time, and the incorporation of the TF-IDF algorithm to prioritise relevant pages in search results. In this chapter, I further research additional ideas, aiming to comprehend them without necessarily implementating them.

Google most likely employs individual algorithms for each proposed idea suggested by their team, so it becomes impractical for me to test all of these algorithms myself. Especially given Google's nature of continuously evolving and updating. Although I will not be attempting to implement the following algorithms, I am still interested in exploring these concepts in-depth to gain a better understanding of the complex ways Google works.

I intend to investigate the impact of location-based searches, the significance of mobile-friendly websites, and the potential benefits of incorporating images and videos to enhance page rankings, and more.

5.1 Mobile-First indexing

Information about the evolution of Smartphones was presented in an article by [CHANTEL, 2023], some of which I have listed below:
In 1992, IBM created the first smartphone, called "The Simon Personal Communicator." Its claim to fame was the introduction of a touchscreen, a revolutionary feature at the time.
The device became available for the public to buy in 1994, including an array of features such as a calendar, address book, calculator, and email capabilities.

The evolution of smartphones continued with great advancements.
By 1997, mobile phones began incorporating entertainment features, such as Nokia's 'Snake' game and a few other classic games.
The year 2000 marked a significant turn point to the industry with the introduction of the first phone camera by Sharp, a Japanese electronics company. Though the phone was initially limited to Japan only,, the inclusion of a camera in mobile devices laid the groundwork for the possibilities that future smartphones are capable of.

By 2001, smartphones had evolved to support internet connection to 3G networks, providing users with access to a wide range of online services and content.
Since then it was noticed that tech giants like Google have expanded their innovation strategies to continually push the boundaries of smartphone technology, introducing more sophisticated features and functionalities.

At around 2016, [Schubert, 2016] defined three main types of Webs:

1. **Standard Web:** Websites which are able to function on all electronic devices, but are not optimized / friendly for mobile devices, and display the desktop version of a computer or a laptop.

2. **Mobile Web:** Optimized on mobile display. A serve will analyse which device a search was done from in order to customise the page's display to fit. A user may be able to switch between the mobile and desktop modes if desired

3. **Responsive web:** A responsive website isn't customized for each individual device out of the growing array them, it sends the same content information into all devices, but modify the screen-size depending the device. This version is recommended by Google, as it's easiest to work with alongside Google's constant updates

But today, there are many more which reply and more things besides what device a User is using , such as: Voice-enabled websites, Virtual Reality (VR) websites etc

Smartphones and other smart-devices such as writing pads, watches and more, come in many different sizes and shapes with 96% of 16 to 24 year-olds and about 69% of over-65s owning a smartphone in the UK as listed in [Barber, 2023].

In 2014, one in two users would access the Web through a mobile device as mentioned in [Schubert, 2016] whereas in April 2024, 10 years later, there are over 4.32 billion active mobile internet users. [Howarth, 2024].
The immerse growth of mobile phone users in ratio to the users accessing the web through a PC or a laptop has lead Google to run an Update to its search-engine algorithm which aims to 'boost' pages which cater the user experience if they are Mobile-friendly, and 'penalize' others if the website's display is not suitable for mobile devices.

"Mobile-First Indexing" was another update that was later introduced in [G. Southern, 2018], which stated that when Google indexes and crawls websites that use a responsive web design and follow good web-practices, it would only crawl and index the version of the mobile-device rather that the website version. This was put in place due to the fact that in the modern world most users have shifted into using a mobile device more regularly so Google has shifted into prioritising web-pages who suit a mobile design.

5.2 Local-Search Algorithms

Google's search has evolved to incorporate the context of geographical relevance. It can be noticed that certain search queries show correlation with specific locations, whether it's seeking weather updates, local news, nearby dining options, or upcoming events, Google tailors its results to reflect the geographic context of the user.

This integration of location-based information into Google's search algorithm shows its commitment to delivering personalised results for each individual user. Consider, for instance, a resident of Bristol, UK, making a search query for "weather today", in such a scenario, it's not unreasonable to expect Google returning weather updates relevant to Bristol rather than those of Liverpool, UK. By prioritising geographical context, Google improves user experience by ensuring that search results align with the user's location and intentions of search.

[Baker, 2024] describes how Google uses several method to recognise a user's location, some are of the following:

1. Device location. For example, through WiFi or GPS
2. Labeled locations on Google Maps
3. Home address or saved locations on behalf of the user.
4. Previous activity in terms of Location
5. IP address

But it extends to even less obvious strategies such as the area code of phone number and even languages used.

When a website wants to be classified with a certain location, some SEO strategies can be implemented, such as including key-words relevant to locations, ensuring the NAP (Name, Address, Phone) is accurate, optimizing location-based reviews, images and other forms of content and apply Schema Markup (as displayed in Figure 33)

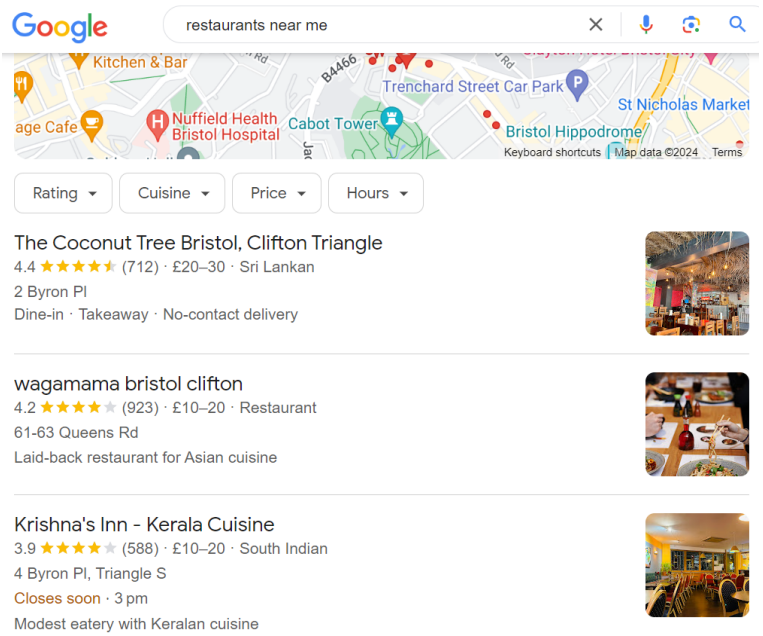


Figure 33:
Schema Markup: as described in [Umbraco,] defines Schema Markup, as a code in the form of structured data that communicates the meaning of your page, elements, and how users should see it to the searching tool, and in a language, the search engine can understand.

Some websites may not require to necessarily be classified with a location if they are considered to be 'Global'. These are companies that mostly operate online or offer digital products/services, such as E-commerce stores, online education platforms or blogs and freelancer services that could be done in a remote location.

5.2.1 GeoSearcher

Definition 5.1 (**Geo-Tags**). *[Watters and Amoudi, 2003] defines Geo-tags as the geographic coordinates which get added to many types of media, such as; photos, videos, websites, text messages, QR codes and more.*
"Geo-Tags use longitude and latitude coordinates to rank results by distance from some reference point and to position results on a digital map."

Whether the algorithm works or not depends on the ability of Google to determine a 'reference point' and map that location to geographic coordinates. One process suggested in [Watters and Amoudi, 2003] is of the following;

1. A user searches for a query made of key-words with an additional reference point (i.e weather in Bristol)

2. The search engine receives the search query and accepts the results (which are found using methods such as page-rank, tf-idf and etc)

3. The algorithm attempts to identify the geographic location of the first 200 URLs returned based on information presented in features like Geo-Tags and then determines the distance between the location identified for the website and the reference point mentioned in the search query.
   For P_1 be the coordinates of the reference position and P_2 be the coordinates of the resulted website, with both coordinates found by assigning them latitudes and longitudes. Then, the distance can be calculated as follows:

$$\text{Distance} = \text{acos}(P_1 + P_2) \times \text{Earth-Radius}$$

4. URLs are then rank by distance with the results incorporated into Google's core algorithm.

The section above emphasises the importance of factoring in an individual's geographical location when crafting a customized user experience for a Google search. The method which was introduced earlier offers a systematic approach to achieve this goal, allowing for the integration of location-based considerations into search outcomes.

Incorporating this approach enhances user satisfaction by providing users with relevant and contextually appropriate information tailored to their specific geographic circumstances.

5.3 Spam-Prevention updates

In the modern world, search engines such as Google, play a significant role in how we discover and learn new information on the Web. The more popular and relevant Google gets, the more likely it is to be manipulated by malicious actors or be taken advantage of negatively.
There are several Spamming techniques used to unfairly boost the visibility of certain content. Understanding these practices is crucial for ensuring the reliability of search results.

In the section below, we'll explore some algorithms used by Google which attempt to prevent these malicious manipulations from accessing the general public and Web-users.

5.3.1 Web-Spam

As the web improves and increases its platform, more and more ventures which seek to make a profit such as businesses, charities and individuals are interested in using the Internet as a resource to reach an abundance of audience to promote themselves at a low cost. This creates an economic incentive for immorally manipulating the search-engine's mechanism in order to increase a page's page-rank regardless of the page's content and popularity.

Search Engine Optimisers spend their time figuring out what is the best way to optimise a website to portray it as more relevant and relatable to the public. But there is a very fine line between ethical SEO practices and unethical spam.

Any harmful methods made to boost a web-page's search engine ranking are commonly known as 'web-spam' or 'spam-dexing,' a term formed by blending 'spam' and 'indexing.'
A spam-page or host refers to a web-page that is either utilized for spamming purposes or generates a significant portion of its ranking from other spam pages.
There exist numerous techniques for Web spam, which can generally be categorised into two main groups: content spam, and link spam.

Content spam refers to changes in the page's content, for example, as mentioned in [Becchetti et al., 2008]

: Key-Word Stuffing and using hidden text in pages. Often a web-page would use a content-mill to improve a page's ranking.

Definition 5.2 (**Content-Mill**, as defined in [Team, 2023a])**.** *"Content mills, also known as content farms, are companies that use freelance writers to produce high volumes of content for clients. Mills offer a wide range of written content to help clients looking to build awareness, generate leads, increase sales, or rank well in search engines."*

Very often, the writers charge low-prices to satisfy the companies seeking a content-mill as can be seen in the example in Figure 34, but as a direct result they then sacrifice the quality of the website for the quantity of the Key-Words it's containing.
So although a website may contain the right words to be relevant for a certain google search, the website itself is of low-quality.

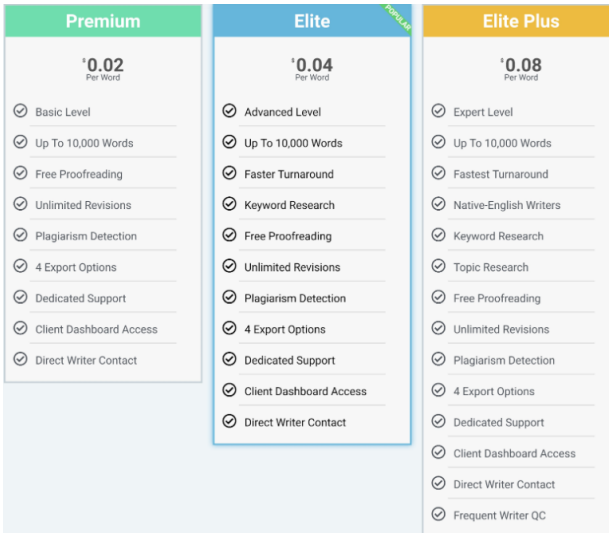


Figure 34: An Example of the Pricing and offers from iwriter

Link spam involves alterations to the linking structure of websites, often through the creation of link farms.

Definition 5.3 (**Link-Farm**, defined by [Becchetti et al., 2008])**.** *A link farm is a network of interconnected pages intentionally designed to manipulate link-based ranking algorithms. Link farms can receive links in many ways, such as using automated software or scripts to generate large numbers of links quickly, acquiring expired domains that were previously used for legitimate purposes or even by posting links in comments sections of blogs, forums, or social media platforms.*

Figure 35 below was taken from an Example from [Becchetti et al., 2008] which specifies the difference between a page which contains inwards links from a Link-Farm while disconnected to the main Web, versus a page connected to the rest of the Web in a normal environment.

The growth of link and content spamming as tactics to manipulate the web led to a response from search engines, such as Google, in the form of sophisticated algorithms which combat these manipulative practices. These algorithms, often referred to collectively as the 'Google Zoo'.

Definition 5.4 (**The Google Zoo**)**.** *The Google Zoo is a term used to describe the new set of algorithms developed by Google to ensure the quality and relevance of its search results. These algorithms contain a wide range of functions, from determining search result rankings to detecting and penalizing manipulative practices such as keyword stuffing / content spamming and link spamming.*

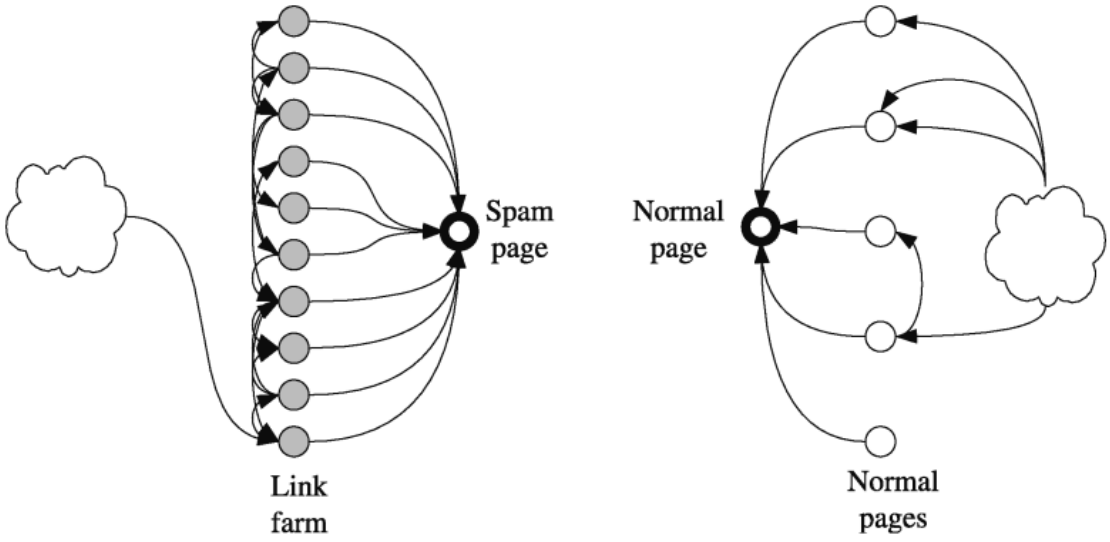


Figure 35: A comparison between a Link-Farm and a normal Web

With names like Panda, Penguin, and Hummingbird, Pigeon and Possum, Google shows its commitment to creating an 'ecosystem' where high-quality content is ranked higher and rewarded, and spammy pages are penalized.

Without this consideration of Spam-Pages in the search algorithm, Google would be a space full of very unfiltered content that may not be as relevant in reality to the users as they seem, although they may be scored as so with the page-rank algorithm or others introduced in the previous chapter. This can lead to a very unsatisfying User-Experience and an un-optimized Search-Engine

5.3.2 Panda

A described in [Patil et al., 2021], Panda was the first algorithm introduced to the Google Zoo. It was launched in February 24, 2011 with the purpose of penalizing web-pages with very thin-content which can be seen as useless to users while boosting web-pages which provide meaningful original information related to search.

Google launched Panda in order to combat the issue of content-mills as described in section 5.3.1, which are generally used to boost a web-pages rank in order to create higher website user traffic, which then increases revenue made from advertisements.
Initially when the algorithm was launched, the sorting of web-pages based on their content was done monthly and was considered as a 'filter algorithm' but as mentioned in [Goodwin, 2021], in Jan 11, 2016, the algorithm was embedded into Google's Core search algorithm.
Before the Panda update was created, [Team, 2023b] emphasises that there was an abundance of repetitive, untruthful and even plagiarised content on the web which received high rankings and appear at the top of the search
So once the Panda algorithm was implemented it paid special attention to the following criteria portrayed by [Patil et al., 2021], in order to Penalized pages which satisfied them:

1. **Thin Content:** web-pages or content that lack depth, relevance, or quality. This could include pages with very little text, or content that's been generated automatically without much human input.

2. **Duplicate Content and Plagiarism:** Content that is substantially similar or identical to content found elsewhere on the web and can be distinguishes from others on the web. content that is not properly cited can lead to plagiarism.

3. **User Experience:** Websites with confusing and unfriendly navigation, intrusive ads, slow loading time or other design flaws.

4. **Key Word Stuffing:** Excessively repeating keywords or phrases to manipulate search engine rankings. For example, inserting keywords into their content, meta tags, and other areas of their web pages.

The changes in content filtering after the algorithm was implemented were drastic. A quote form [Kumar and Arpana, 2016] is as such: "CNET reported a surge in the rankings of news websites and social networking sites, and a drop in rankings for sites containing large amounts of advertising. This change reportedly affected the rankings of almost 12% of all search results."

The introduction of the Panda algorithm reshaped the way Google returns results, marking a pivotal moment in the search of quality and relevance online. It brought search results into closer alignment with user expectations and preferences, creating a new standard for content integrity and user experience in the digital domain.

5.3.3 Penguin

Penguin was the second algorithm to be introduced to the Google Zoo. It was launched in April 24, 2012 directly after Panda in an effort to continue combating low-quality content which still circulated the media
At a conference mentioned in [Taylor, 2022] , Matt Cutts, the head of the web-spam team said: "We look at it something designed to tackle low-quality content. It started out with Panda, and then we noticed that there was still a lot of spam and Penguin was designed to tackle that."

Penguin specifically ensured that natural and relevant links award the sites they're linked to, at the same time as identifying low-quality pages through link-spamming and other manipulative Link-Building practices to penalize those that do not, [Patil et al., 2021] describes.

Manipulative Link-Building practices include:

1. **Link-Farming:** As mentioned in Subsection 5.3.1
2. **Page Cloaking:** Content presented to search engine crawlers is different from what is presented to human users. Cloaking involves showing one version of a Web-Page to search engines, which is optimized for specific keywords or phrases (can be done through word stuffing), while displaying a different version to actual users.
3. **Site Mirroring:** The practice of creating multiple websites with almost identical content and design, with all URLs pointing to the same page, usually with the intention of manipulating search engine rankings.
4. **URL Redirection:** When users are redirected into web-pages that are not the ones represented by the URL which was clicked on.

Once the Penguin algorithm was implemented, more than 3% of web-pages were affected according to Google [Taylor, 2022], which is slightly less dramatic than the Panda algorithm, but 3% still represents a very large number of web-pages when considering all pages in the Web.

Similarly to the Panda Algorithm, Penguin also used to be an algorithm which gets refreshed periodically, around every few months. But then, later in September 23, 2016 it also became a part of Google's Core algorithm.
Since then, many updates were implemented onto the Penguin algorithms constantly improving the penalization of link schemes and different sorts of Spam techniques which had then even improved the User-Experience in the web even further and created a better representation of which pages seem to be relevant and authentic and which are not.

6 The future of Google

In Chapter 4 and Chapter 5, I have introduced several functions which when combined with the page-rank algorithm, Google's search engine is formed. From key-word sorting algorithms to seeing whether the number of clicks, speed of page loading affect the algorithm or even location of device affect the algorithm.
Google is constantly updating, creating new features and algorithms and evolving to fit the user. The algorithms portrayed in previous chapters are only a small fragment of the mega-mind behind the Google search core algorithm, many more such as the inclusion of videos, images and other graphics, the incorporation of other social-media links and even providing voice-search options are all features that can improve a site ranking.

If we wonder what Google would look like in one year, we can probably assume that nothing much would change. We can expect modifications of existing algorithms for optimization, improvements in cyber-security, spam control and providing a smoother and faster user experience as possibilities for Google.
Although when considering short term we may not expect extreme advancements, wondering what Google would look like five or even ten years from now is something that would be hard to predict.

6.1 AI and ChatGPT

Definition 6.1 (**Artificial intelligence** as defined by [IBM, a])**.** *"Artificial intelligence, or AI, is technology that enables computers and machines to simulate human intelligence and problem-solving capabilities"*

The introduction of AI to the technology world has lead to several advancements, some are:

1. **Natural Language Processing (NLP)** as noted by [IBM, c] , A model which is able to understand and generate human-like text by translating texts between languages, recognising a users voice and behaviours, summarise a large text into a condensed version or even generate unique graphics based on text.

2. **Healthcare**, by helping analyse medical diagnoses and even creating treatment plans tailored to an individual.

3. **Finance**, "AI can enable predictive modeling, which can help financial organizations anticipate market trends, potential risks and customer behavior", quoted from [Matthew Finio, 2023].

4. **Deep Learning**, [IBM, b] identifies it as a model which instructs computers to process data similarly to the way the human brain's functions. Deep learning models recognise patterns like images, text, and sounds, in order to generate precise predictions.

Following the introduction and integration of AI in our daily lives and by several companies looking to grow, ChatGBT was created in November, 2022 by an AI company called OpenAI, [WRIGHT, 2024] specifies that the company was founded by individuals such as Elon Musk and Sam Altman who dominate the technology world and is currently the most popular AI tool by far.

Definition 6.2 (**ChatGPT**)**.** *[Hetler, 2023] defines ChatGPT as an AI chatbot, which uses NLP to generate natural and human-like conversational interactions. This language model is able to answer questions provided by a user and craft unique written content such as essays, code snippets, and emails.*

As we have seen in Figure 33, Google can return personalised results based on your location and what you're seeking. But we can notice that ChatGPT is actually able to do the same in Figure 36

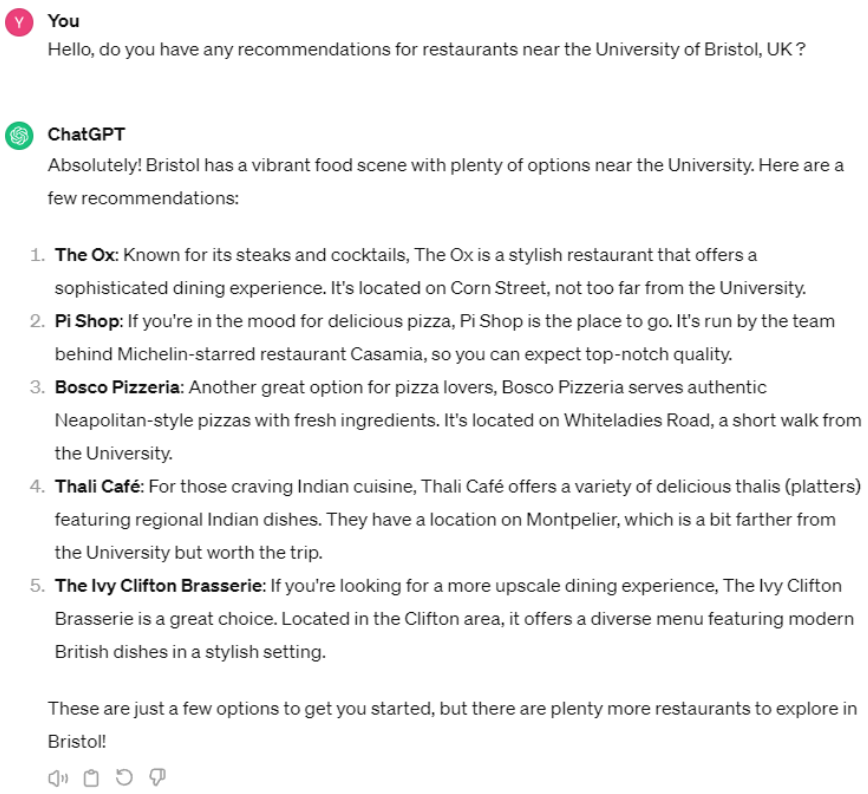


Figure 36: A conversation with the ChatGPT bot regarding restaurants near me

Which to some individuals may be preferable, as it feels more engaging and personalised to a user's requests.

In order to keep training and teaching the Bot what human-like behaviours are like, users are able to down-vote responses which are seen to be irrelevant or incorrect. ChatGPT also provides a user with the opportunity to regenerate an answer until the solution is provided.
Considering the fact that ChatGPT is able to go trough a large collection of data in order to return the most human like response, we may wonder whether users will turn to ChatGPT for quick answers rather than use the web to find their answers.

Sofia Blum, a writer for CNBC, conducted an experiment [Blum, 2022] where she asked ChatGPT to "write an article about whether or not ChatGPT is a good alternative to Google as a search engine." which resulted in a few paragraphs written by the bot which concluded the advantages and disadvantages of ChatGPT while linking it back to Google's search engine.
It started off by stating that as a bot, it is unable to conduct 'proper' research therefore it has only provided data based on what is already present on the web.

"One potential advantage of using ChatGPT as a search engine is that it's able to provide more personalized and conversational results. Because ChatGPT is a language model, it is able to understand and respond to natural language queries, allowing users to ask questions and receive answers in a more intuitive and human-like way." which was then followed by a disadvantage.
"ChatGPT may not have access to the same breadth and depth of information as other search

engines. This means that it may not be able to provide as comprehensive or up-to-date results, and may not be as effective for certain types of searches"
The article was concluded by specifying that ChatGPT may not be a very good alternative to Google's search engine, but generally it depends on the preference of the user and what they're seeking to explore.

Article [Strzelecki, 2024] concluded that for now, AI language models such as ChatGPT and even Gemini (Google's own Chat Bot, a less popular choice) cannot replace Google's search engine for the additional reasons:

1. Instead of ChatGPT replacing Google, it is likely that they will integrate into one, especially for users' informational queries.

2. Users have been searching for information on the web for about 25 years now, and it is possible that ChatGPT may not be replaced strictly out of comfort of users to stick to services they are used for.

3. Profit may be lost from advertisements on personalised web-searches. For example, when searching for "Shoes", I will get a pop-up section which boosts some of the advertised products as well as some websites to buy shoes from, unlike ChatGPT which simply responds with a human-like conversation on what is the purpose of shoes.

4. Access to ChatGPT services is limited by user capacity due to the high consumption of computing power to produce answers.

As seen in some other aspects of life, AI can negatively affect our regular day to day lives. For example, as mentioned previously, AI can be used for disease-diagnosis, although it is mostly reliable, the lack of human interaction can lead to misdiagnosis or bias towards specific individuals.

ChatGPT specifically could present several potential disadvantages such as
Job Displacement particularly in customer service and support roles (which can be seen in several online website already),
Privacy Risks as technology improves, more data is being collected about each user, which can cause a strong sense of discomfort to many users and even
Discrimination As stated in many articles produced recently which show that ChatGPT has used societal biases to make conclusion about specific users. For example, a study in Stanford School of Medicine [GARANCE BURKE and PRESS, 2023] has shown that AI tools such as ChatGPT can perpetuating racist, debunked medical ideas and will return different responses depending on a patients skin colour.
These are all things that will need to be worked on before ChatGPT and other bot tools can be taken seriously on the same level as a Google engine.

I strongly believe that as we move forward into the future, the use of AI will become more and more popular, especially within the younger generations who are already starting to use it in their daily lives. It is possible that in the future Google will operate and work completely differently, especially in terms of how pages are ranked and are tailored to the user, if not done so already.

To conclude this section, the future of Google in terms of AI developments and the potential replacement of its search engine by ChatGPT presents many possibilities and challenges. The emergence of AI, and specifically a human-like bot such as ChatGPT, creates a significant

shift in how users interact with search engines. ChatGPT offers a more engaging and conversational alternative to traditional search engines whereas Google's search engine provides a more reliable set of data with more features.

As we explore the evolution of AI and search technology, it is shown that the future of Google will be shaped by the integration of AI capabilities and the evolving needs and expectations of users.
While the specifics of what it entails are still unknown, the growth in popularity of AI among the younger generations as a quick method for search and the ongoing improvements in technology suggest that Google's operations and user experiences will continue to evolve how we expect.

7 Conclusion

In this project, we have explored the evolution of Google, from a PhD project to its current status as a central necessary part to our digital lives. We've travelled through a journey of learning about the foundational algorithms of Google's search engine and the factors that shape the user experience.

At the heart of Google's core lies the page-rank algorithm, a revolutionary idea that transformed the way we navigate the web using the analysis of the link structure of the web as a graph.
We explored how artificial links and the Teleportation matrix were integrated into the creation of the Google-matrix as methods which address challenges such as dangling nodes and loops in the graph's structure.

We have also introduced ideas which provide refinements of the initial Google algorithm to allow for the most accurate and useful search results to users. Incorporating signals such as TF-IDF, clicks per page, and page loading times into the page-rank framework enhances the search experience, ensuring that results are not only relevant but also accessible and user-friendly.

Moreover, Google's adaptation to the mobile era and its emphasis on location-based search optimization reflect its commitment to meeting the evolving needs of users in an increasingly connected world.
We have looked at how we expect Google to continue with this trend in the future. As AI technologies continue to evolve, we may witness innovations in search algorithms containing some AI features such as, personalised recommendations and natural language processing. Yet, despite these advancements, one question is always kept in mind: Can AI, embodied in functions like ChatGPT, replace Google?

In conclusion, my journey of understanding the details of Google's algorithms has only deepened my curiosity and appreciation for the ever-evolving landscape of information retrieval. I am interested in further expanding my learning journey.
I remain committed to exploring how advancements in technology, such as artificial intelligence and machine learning, will further enhance the search experience. With each update, Google presents new challenges and opportunities for exploration, and I look forward to delving deeper about each and everyone of those updates in the future.

8 Python codes

Listings

Code 8.1. *The function below verifies that the matrix provided is suitble for the page-rank algorithm by checking if it is square, positive with all rows summing up to 1:*

```
def matrix_verifier(P):
    epsilon = 0.00000000000001
#I have chosen to include espsilon due to the fact that the python turns fractions into decimals and rounds them up so when we add fractions later on in the functions it seems as if they don't add up to 1, although they are epsilon close to being 1.

    num_rows = len(P)

#For each row index in the matrix we check if the sum of all elemets in that matrix equals 1 - ensuring stochasticity
    for row_index in range(num_rows):

        row = P[row_index]
        sum_row = sum(row)
        if abs(sum_row-1) > epsilon:
        return None
            print("Invalid Transition Matrix, row sum illegal", sum_row, row_index)
            return None

#Checking if the matrix is a square matrix by comparing the number of columns to the number of rows
        num_columns = len(row)
        if num_rows != num_columns:
            print("Invalid Transition Matrix, not square matrix", num_columns, row_index, num_rows)
            return None

#If any of the elements of the matrix are negative it is impossible as the elements represent probabilities
        for column_index in range(num_columns):
            if P[row_index][column_index] < 0:
                print("Invalid Transition Matrix, negative matrix", P[row_index][column_index], row_index, column_index)
                return None
```

Listing 1: Verifying the Transition matrix is sutible for calculations

Code 8.2. *In the code below I described how I implemented the Power-Method, using the starting vector X_0 and for the raised power to be 1000, as it was noticed in Figure 4*

```
#We raise matrix P once its been verified to a given power - the higher the better for this algorithm. We then multiply the vector X_0 found above by our results to recieve X_k.

def power_method(P,X_0,power):
    result_matrix = np.linalg.matrix_power(P, power)
    X_k = np.dot(X_0, result_matrix)
    return X_k

k = len(P)

```

```
#X_0 is our starting vector of the algorithm of equal probabilities of
    landing on either vertex.
X_0 = [1/k for _ in range(k)]

#For finding \pi we use power = 1000 although higher power can help find a
    more percise result.
pi = power_method(P,X_0,1000)
```

Listing 2: Implementing the power method on P

Code 8.3. *In the code below I explained which lines should be added to Listing 1 in order to modify the original Transition matrix*

```
# If the sum of rows is 0 - so ALL elements in the row are 0, then replaces
    each of those elements by 1/num_rows
            if sum_row == 0:
                P[row_index][column_index] = 1/num_rows

#So now we print pi again to see the changeses that were made after we apply
    matrix_verifier on S:
matrix_verifier(S)
pi = power_method(S,X_0,1000)
```

Listing 3: Dangling nodes

Code 8.4. *This code creates the Teleportation Matrix.*

```
def Transportation_Mat(H):
    matrix_verifier(H)
#We need to make sure that it is of equal size to the transition matrix
    k = len(H)

# Creating a k x k matrix filled with fractions 1/k
    fraction_matrix = np.full((k, k), 1/k)

    return fraction_matrix
```

Listing 4: The Teleportation Matrix

Code 8.5. *The code below describes the combination of the modified matrix with the dangling nodes alongside the Teleportation Matrix, using the damping factor.*

```
def google_matrix(H,a):
    matrix_verifier(H)
    G= a* H + (1 - a) * Transportation_Mat(H)
    return np.array(G)
-------------------------------------------------------------------------------

#Implementation of the Google matrix with the damping factor being 0.85
G = google_matrix(H,0.85)
X_0 = [1/len(H) for _ in range(len(H))]
pi = power_method(G,X_0,1000)
```

Listing 5: Google Matrix

Code 8.6. *Below we can see a code for the Matrix-Generator that represents the links between our web-system recipes.*

```
#We set k = 100 since we are only interested in the top 100 pages and the
    links between them
k=100
```

```
def generate_stochastic_transition_matrix(l):
#Start off with a l x l zero matrix (all elements are 0)
    matrix = np.zeros((l, l))

    for page in range(l):
#for each row (page) in the matrix, we randomly determine the number of outgoing links from it (so how mant elements in the row are non - zero)
        if l>= 100:
            num_outgoing_links = np.random.randint(0, 100) #(how many links in each row)
        else:
            num_outgoing_links = np.random.randint(0, l) #(how many links in each row)

        if num_outgoing_links > 0:
#Randomly select which columns (pages) j, page i links to, can only be choices in the range of l, the set of columns selected is of the size num_outgoing_links, and replace=False indicated that a certain column cannot be assigned a probability several times.
            outgoing_links = np.random.choice(range(l), size=num_outgoing_links, replace=False)

# Assign equal probabilities for each outgoing link in its slot in the matrix
            matrix[page][outgoing_links] = 1 / num_outgoing_links

    return matrix
```

Listing 6: Transition matrix generator

It may be noticed that between row [9-12] in the code above, I considered two cases for "num_outgoing_links":

1. *If $l \leq 100$, I let the number of links going out of recipe i to be any choice of a random number from [0,l]*

2. *If $l \geq 100$, I let the maximum number of links going out of recipe i to be capped at 100.*

Of course it is possible that for a search between billions of pages, there will be a page linking to all billion pages, but it is highly unlikely. It is also highly unlikely that the number of links will even be in the millions, hundred-thousands or even thousands.

Code 8.7. *In the code below, I have calculated the value of π for the Transition matrix portrayed in Figure 10*

```
#Using the algorithms I described in the sections above for finding pi and the transition_matrix created above:

matrix_verifier(transition_matrix)
transitionG = google_matrix(transition_matrix,0.85)
pi = power_method(transitionG,X_0,100)
```

Listing 7: Finding initial pi for large Data set

Code 8.8. *We will now sort the values of π from highest to lowest as we did previously to get the initial ranking of the pages.*

```
#We now sort the scores of pi with their indexes
def score_indexing_sorted_in_order(pages_index,scores):
    joined_list = []

```

```
#Go through the scores in the list of scores with indexes of len(scores) and put a tuple into a new list of the posititon that socre is in with the score
    for index,score in enumerate(scores):
        joined_list.append((pages_index[index],score))

#Sort the list of tuples in order of highest to lowest
    sorted_scores = sorted(joined_list, key=lambda x: x[1], reverse=True)
    return sorted_scores
```

Listing 8: Sorting Ranking

Code 8.9. *In the next code, I have removed all special characters and have turned all Upper-case text into Lower-case letters.*

```
#The line below extracts row K, which is a row I have combined all text words from the other columns into ones, and only returns 100 recipes instead of all 500,000 recipes for simplicity.

only_combined_col = pd.read_excel('data of recipes.xlsx', usecols='K', nrows=100)
-------------------------------------------------------------------------------

def remove_extra_char(text):
# Make all letters lowercase
    lower_text = text.lower()
    character_list = []

# Make a list of ASCII characters besides lowercase letters
    for i in range(33, 97):
        character_list.append(chr(i))
    for i in range(123, 256):
        character_list.append(chr(i))

#Replace every character other than letters and spaces with a space
    text_removed = lower_text.translate({ord(i): ' ' for i in character_list})

# Split so that multiple spaces are removed and then joined together by a space
    text_fullremoved = ' '.join(text_removed.split())

    return text_fullremoved
-------------------------------------------------------------------------------

#I then wanted to apply this to all recipes rather than one, so I created a code that when inserting a column (such as column K) it creates a list of all the recipes in there in lowercase only.

def new_list_of_modified_text(col):
    list_of_modified_recipes = []

#For every recipe in the range 100 as I have defined before
    for recipe in range(k):

#I removed the extra characters for each recipe and then appended it as an item to the empty list above.
        list_of_modified_recipes.append(remove_extra_char(only_combined_col.loc[recipe][0]))

    return list_of_modified_recipes
```

```
36 ------------------------------------------------------------------------------

37 #Here I applied it onto column K and lebelled it as
       list_of_all_recipes_lowercase, this will be used many times in future
       code.
38 list_of_all_recipes_lowercase = new_list_of_modified_text(only_combined_col)
```

Listing 9: Text to lower-case

Code 8.10. *Here I have created a dictionary with an input of a single recipe which will count how many times each word appears in each recipe.*

```
1  def string_dictionary(str):
2
3  # Create an empty dictionary named 'page_dictionary' to store word
       frequencies.
4      page_dictionary = dict()
5
6  # Split the input string 'str' into a list of words using spaces as
       separators and store it in the 'words' list.
7      words = str.split()
8
9  # Iterate through each word in the 'words' list.
10     for word in words:
11
12 # Check if the word is already in the 'counts' dictionary.
13         if word in page_dictionary:
14
15 # If the word is already in the dictionary, increment its frequency by 1.
16             page_dictionary[word]['count_of_word_per_page'] += 1
17
18         else:
19 # If the word is not in the dictionary, add it to the dictionary with a
       frequency of 1. We set 'tf','count_of_pages_with_word','idf' and 'tf_idf'
       as -1, as we are able to have the scores be 0, which is different to
       empty, so for the time being we set them as -1.
20
21             page_dictionary[word] = {'count_of_word_per_page':1,
22                 'tf':-1,
23                 'count_of_pages_with_word':-1,
24                 'idf':-1,
25                 'tf_idf':-1}
26
27 # Return the 'page_dictionary', which now contains the values of word
       frequencies.
28     return page_dictionary
29 ------------------------------------------------------------------------------

30 #Here I created a list that will contain the dictionaries of all recipes
       which we will use in the following few codes:
31 dictionaries_of_all_recipes = []
32 for recipe in list_of_all_recipes_lowercase:
33     dictionaries_of_all_recipes.append(string_dictionary(recipe))
```

Listing 10: Creating a Dictionary

Code 8.11. *In the code below I added the TF score calculated into the dictionary created in 10*

```
1
2  def TF(list_of_recipes):
3      for recipe in range(k):
```

```
#For each recipe, split it into individual words and then check the length of it to find the total number of words in a recipe.
        total_words = len(list_of_recipes[recipe].split())

#Go through all the keys in the dictionary
        for word in list(dictionaries_of_all_recipes[recipe].keys()):

#for each key in the dictionary, divide it's 'count_of_word_per_page' value by the total number of words in the page
            TF_word = dictionaries_of_all_recipes[recipe][word]['count_of_word_per_page']/total_words

#replace the -1 value that was in 'tf' when we set the dictionary with the new value we had just found
            dictionaries_of_all_recipes[recipe][word]['tf'] = TF_word
-------------------------------------------------------------------------------

#apply the function onto the dictionary:
TF(list_of_all_recipes_lowercase)
```

Listing 11: TF Score

Code 8.12. *In the code below I added the IDF score calculated into the dictionary created*

```
#I then created a function which counts in how many documents a specific word if found in:

def count_strings_with_word(list_of_recipes, target_word):
    count = 0

    for recipe in list_of_recipes:
        words = recipe.split()

#If a target word is found at a recipe then add +1 to the total count of that word
        if target_word in words:
            count += 1

    return count
-------------------------------------------------------------------------------

#Put all string into one in order to get a a list of all possible target words available to choose from with with count_strings_with_word > 0

single_string = " ".join(list_of_all_recipes_lowercase)
new_list = list(string_dictionary(single_string).keys())

-------------------------------------------------------------------------------

#Here I made sure to add the count of each word from new_list to the dictionary, if a certain word appears in more than one recipe then add the same count to that word in all recipes its in.

def adding_count_to_dictionary(list_of_recipes):

    for word in new_list:

#Apply the function on all possible target words
        count = count_strings_with_word(list_of_recipes, word)

#For each recipe, we want the score to be the same for the same word
```

```
        for recipe in range(k):

#Only works if the word is in the recipe so check:
            if word in list(dictionaries_of_all_recipes[recipe].keys()):

#replace the -1 value that was in 'count_of_pages_with_word' when we set the dictionary with the new value we had just found
                dictionaries_of_all_recipes[recipe][word]['count_of_pages_with_word'] = count
-------------------------------------------------------------------------------

#apply the function onto the dictionary:
adding_count_to_dictionary(list_of_all_recipes_lowercase)
-------------------------------------------------------------------------------

#So finally we can write the function for the IDF score:

def IDF(list_of_all_target_words):
    for recipe in range(k):

        for word in list_of_all_target_words:

            if word in list(dictionaries_of_all_recipes[recipe].keys()):

#Implement the function we have described above to each word in each recipe of dictionaries_of_all_recipes and replace the -1 value that was in 'idf' when we set the dictionary with the new value we had just found
                idf = log(k/dictionaries_of_all_recipes[recipe][word]['count_of_pages_with_word'])
                dictionaries_of_all_recipes[recipe][word]['idf'] = idf
-------------------------------------------------------------------------------

#apply the function onto new_list of all possible target words:
IDF(new_list)
```

Listing 12: IDF Score

Code 8.13. *In the code below I added the TF-IDF score calculated into the dictionary created*

```
def TF_IDF():
    for recipe in range(k):
        for word in list(dictionaries_of_all_recipes[recipe].keys()):

#The TF-IDF score if the multiplication of the values 'tf' and 'idf' and we replace the -1 value that was in 'idf' when we set the dictionary with the new value we had just found
            tf_idf = (dictionaries_of_all_recipes[recipe][word]['tf'])*(dictionaries_of_all_recipes[recipe][word]['idf'])

            dictionaries_of_all_recipes[recipe][word]['tf_idf'] = tf_idf


-------------------------------------------------------------------------------

#apply the function all together:
TF_IDF()
```

Listing 13: TF-IDF Score

Code 8.14. *Function that selects relevant pages*

```
#We input a few target words to imitate a google search query, and we also input all the dictionaries of the recipes we're comparing between.
```

```
def set_of_relevant_pages(target_Words_as_string, list_of_dictionaries):

    list_of_relevant_recipes_and_their_score = []

    for recipe in range(k):

#Start a count of the sum of all tf_idf's in each recipe of the words in query, can be 0.
        total_tf_idf = 0

        for word in target_Words_as_string:

#If the target word is in the recipe's dictionary
            if word in list(dictionaries_of_all_recipes[recipe].keys()):

#Add's the tf-idf score of that word in recipe to the sum:
                total_tf_idf += dictionaries_of_all_recipes[recipe][word]['tf_idf']

#We only want pages such that their tf_idf score is bigger than 0 (as they are correlated to the search)
        if total_tf_idf > 0:

#Adds to a list, a tuple of a page with its score
            list_of_relevant_recipes_and_their_score.append([recipe, total_tf_idf])

    return list_of_relevant_recipes_and_their_score
```

Listing 14: Relevant pages to search

Code 8.15. *Separating the list of pages from the list of sub-lists*

```
#creates a vector of the relevant pages
def vector_of_pages(list_of_tuples):
    vector = []
    for atuple in list_of_tuples:
        vector.append(atuple[0])
    return vector
```

Listing 15: Vector of Relevant Pages

Code 8.16. *Separating the list of scores from the list of sub-lists*

```
#creates a vector of the relevant Scores
def vector_of_tf_idf_scores(list_of_tuples):
    vector = []
    for atuple in list_of_tuples:
        vector.append(atuple[1])
    return vector
```

Listing 16: Vector of Relevant Scores

Code 8.17.

```
def normalise(vector):
    normalised_vector = []

#Finding the sum of all elements in the vector, then dividing each element by the sum, so that all elements add up to 1
    m = 1/sum(vector)
    for element in vector:
        normalised_vector.append(element * m)
```

```
    return normalised_vector
```

Listing 17: Normalise under 1-Norm

Code 8.18. *Finding the value of π^* of the selected pages*

```
pi_of_selected_pages = [pi[i] for i in TF_IDF_relevant_pages_vector]
--------------------------------------------------------------------------------

pi_star = normalise(pi_of_selected_pages)
```

Listing 18: Normalisation of extracted π

Code 8.19. *Now I'll create a balance between the two vectors using the equation provided in (6).*

```
def finding_sigma(page_rank_vector,tf_idf_vector,b):

#As b-->1, the imporatnace of the page rank score increases
    list1 = [element0 * b for element0 in page_rank_vector]
    list2 = [element1 * (1-b) for element1 in tf_idf_vector]

    sigma = [x + y for x, y in zip(list1, list2)]
    return sigma
```

Listing 19: Balancing between the page-rank and TF-IDF scores

Code 8.20. *Here I assigned a number of clicks per page on the web*

```
def assign_score_per_page(lower_bound,upper_bound,mean,s_d,size):

# Generate random numbers with truncated normal distribution
    X = truncnorm((lower_bound - mean) / s_d, (upper_bound - mean) / s_d,
    loc=mean, scale=s_d)

    random_numbers = X.rvs(size=size)

# Convert the array to a list and turns the random numbers into integers.
    random_numbers_list = list(map(int, random_numbers))

    return random_numbers_list
```

Listing 20: Assign clicks Per Page

Code 8.21. *Code which created a matrix based on the weight of number of clicks per website.*

```
#Input a transition matrix and a list of clicks per page
def transition_based_on_clicks(t_matrix, list_of_clicks):
    N = len(t_matrix)

# Create a new zero matrix of the size of t_matrix
    t_matrix_adjusted = np.zeros((N, N))

    for row in range(N):

#Find all the indices of the columns in each row which arent 0 (which
    represent a probability)
        positive_indices = [column for column in range(N) if t_matrix[row][
    column] > 0]  # Get indices of positive values

# Calculate the sum of clicks with the indicies of columns which have a
    positive value
```

```
        empty_list =[]
        for column in positive_indices:
            empty_list.append(list_of_clicks[column])

        click_sum = sum(empty_list)

#If the sum of the clicks in each row is not 0 (which could happen for dangling nodes)
        if click_sum>0:
            for column in positive_indices:

#change the value of t_matrix_adjusted[row][column] to the ratio described in the notes.
                t_matrix_adjusted[row][column] = list_of_clicks[column]/click_sum

     return t_matrix_adjusted
```

Listing 21: Transition Matrix Based on the number of clicks

Code 8.22. *Finding clicks π*

```
#Using the algorithms I described in the sections above for finding pi and the clicks_matrix created above:

matrix_verifier(clicks_matrix) #checks if the new matrix is valid
clicks_G = google_matrix(clicks_matrix,0.85) #creating G out of new transition
clicks_pi = power_method(clicks_G,X_0,10000) #power method on it
```

Listing 22: page-rank on Clicks Matrix

Code 8.23. *Finding $\widetilde{\pi}$*

```
clicks_pi_of_selected_pages = [clicks_pi[i] for i in TF_IDF_relevant_pages_vector]
pi_tilde = normalise(clicks_pi_of_selected_pages)
```

Listing 23: Normalisation of extracted π

Sigma tilde

```
sigma_tilde = finding_sigma(pi_tilde,phi,0.8)
```

Listing 24: Balancing between the page-rank of clicks matrix and TF-IDF scores

Code 8.24.

Code 8.25. *Assign loading time*

```
def assign_score_per_page_decimal(lower_bound,upper_bound,mean,s_d,size):

# Generate random numbers with truncated normal distribution
     X = truncnorm((lower_bound - mean) / s_d, (upper_bound - mean) / s_d, loc=mean, scale=s_d)

     random_numbers = X.rvs(size=size)

# Convert the array to a list.
     random_numbers_list = list(random_numbers)

     return random_numbers_list
```

```
-------------------------------------------------------------------------

#Eseential as its impossible for a website to load in 0 seconds
epsilon = 0.00000000000000000001

#I capped it at 20 seconds for the loading page as it seems like most pages load in less than that
#I let the mean be 3 as its the highest acceptable time for a website to load in, and the standard deviation be 4 after doing some trial and error and seeing that for s_d = 4, we get the most realistic case scenario.
# we do this for k being 100 (pages)

loading_time_per_recipe = assign_score_per_page_decimal(0+epsilon,20,3,4,k)
```

Listing 25: Assigned Loading Time

Code 8.26. *Finding ρ*

```
#Each Bi is a time range, Each bi is the probability of users staying on the website given the loading time, b7 is for anything above range B6

def loading_vector(list_of_times,B1,B2,B3,B4,B5,B6,b1,b2,b3,b4,b5,b6,b7):

#Making a copy so it doesnt modify the original
    list_of_times_copy = list_of_times.copy()

    for index_of_time in range(len(list_of_times_copy)):

#checking if a number falls in each range
        if B1[0] <= list_of_times_copy[index_of_time] <= B1[1]:
            list_of_times_copy[index_of_time] = b1

        if B2[0] <= list_of_times_copy[index_of_time] <= B2[1]:
            list_of_times_copy[index_of_time] = b2

        if B3[0] <= list_of_times_copy[index_of_time] <= B3[1]:
            list_of_times_copy[index_of_time] = b3

        if B4[0] <= list_of_times_copy[index_of_time] <= B4[1]:
            list_of_times_copy[index_of_time] = b4

        if B5[0] <= list_of_times_copy[index_of_time] <= B5[1]:
            list_of_times_copy[index_of_time] = b5

        if B6[0] <= list_of_times_copy[index_of_time] <= B6[1]:
            list_of_times_copy[index_of_time] = b6

#If it's outside the upper range, give it a very small probability of users waiting for site to load.
        if B6[1] <= list_of_times_copy[index_of_time]:
            list_of_times_copy[index_of_time] = b7

    return list_of_times_copy
-------------------------------------------------------------------------

#We now input the info in (2) and label it as rho:

rho = loading_vector(loading_time_per_recipe, [0,0.5], [0.5,1], [1,2], [2,3], [3,5], [5,10], 1,0.9,0.8,0.7,0.5,0.2,0.05)
```

Listing 26: Assigned probabilities

Code 8.27. *finding ρ^**

```
#Only located the relevant pages based on search query out of rho
rho_of_selected_pages = [rho[i] for i in TF_IDF_relevant_pages_vector]
-------------------------------------------------------------------------------

#Use function normalise created in listing: Normalisation of extracted pi
rho_star = normalise(rho_of_selected_pages)
```

Listing 27: Finding ρ^*

Code 8.28. *Finding θ*

```
def finding_theta(page_rank_vector,tf_idf_vector,loading_vector,b,g):

    if b+c <= 1:

        list1 = [element0 * b for element0 in page_rank_vector]
        list2 = [element1 * g for element1 in tf_idf_vector]
        list3 = [element1 * (1-b-g) for element1 in loading_vector]

        theta = [x + y + z for x, y, z in zip(list1, list2, list3)]
        return theta

    else:
        print("Ratio does not sum to 1")
        return None
```

Listing 28: Finding θ

References

[Arellano, 2023] Arellano, K. (2023). Top 8 best recipe websites [for 2023] - rapid blog. *Rapid Food*.

[Baker, 2024] Baker, L. (2024). How to see google search results and rankings for different locations.

[Barber, 2023] Barber, M. B. . S. (2023). Mobile phone and internet usage statistics.

[Batt, 2024] Batt, J. (2024). How many internal links per page.

[Becchetti et al., 2008] Becchetti, L., Castillo, C., Donato, D., Baeza-YATES, R., and Leonardi, S. (2008). Link analysis for web spam detection. *ACM Trans. Web*, 2(1).

[Bhan, 2022] Bhan, M. (2022). Google pagespeed insights reports: A technical guide.

[Blum, 2022] Blum, S. (2022). Google vs. chatgpt: Here's what happened when i swapped services for a day.

[Camarena, 2023] Camarena, A. (2023). What is page speed how to improve it.

[CHANTEL, 2023] CHANTEL, J. (2023). Smartphone history: The timeline of a modern marvel.

[Dan Margalit, 2019] Dan Margalit, J. R. (2019). *Interactive Linear Algebra*.

[Developers, 2023] Developers, G. (2023). In-depth guide to how google search works.

[Fitzgerald, 2023] Fitzgerald, A. (2023). How many visitors should your website get? [data from 400+ web traffic analysts].

[G. Southern, 2018] G. Southern, M. (2018). Google officially announces rollout of mobile-first indexing.

[GARANCE BURKE and PRESS, 2023] GARANCE BURKE, M. O. and PRESS, T. A. (2023). Bombshell stanford study finds chatgpt and google's bard answer medical questions with racist, debunked theories that harm black patients.

[Goodwin, 2021] Goodwin, D. (2021). A complete guide to the google panda update: 2011-21.

[Google,] Google. Google's mission.

[Google/SOASTA, 2017] Google/SOASTA (2017). Page load time in relation to bounce rate.

[Hall, 2023] Hall, J. (2023). Google ranking algorithm update: How has it changed?: Five channels.

[Hefferon, 2011] Hefferon, J. (2011). Eigenvalues, eigenvectors ii.

[Hetler, 2023] Hetler, A. (2023). What is chatgpt? everything you need to know.

[Howarth, 2024] Howarth, J. (2024). Internet traffic from mobile devices (apr 2024).

[IBM, a] IBM. What is artificial intelligence ?

[IBM, b] IBM. What is deep learning?

[IBM, c] IBM. What is natural language processing ?

[Jay, 2024] Jay (2024). Too many internal links? what is too much according to google?

[Kumar et al., 2011] Kumar, G., Duhan, N., and Sharma, A. K. (2011). Page ranking based on number of visits of links of web page. In *2011 2nd International Conference on Computer and Communication Technology (ICCCT-2011)*, pages 11–14.

[Kumar and Arpana, 2016] Kumar, M. and Arpana (2016). Survey on search engine optimization techniques to achieve high page rank. *Imperial journal of interdisciplinary research*, 2.

[Li, 2021] Li, S. (2021). Food.com recipes with search terms and tags.

[Liu et al., 2018] Liu, C.-z., Sheng, Y.-x., Wei, Z.-q., and Yang, Y.-Q. (2018). Research of text classification based on improved tf-idf algorithm. In *2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE)*, pages 218–222.

[Mark Hall, 2024] Mark Hall, W. L. H. (2024). Google, american company.

[Mathews and Fink, 2004] Mathews, J. H. and Fink, K. K. (2004). *, 4th Edition.*

[Matthew Finio, 2023] Matthew Finio, A. D. (2023). What is ai in finance?

[NJ, 2024] NJ (2024). How many websites are there in the world?

[OH, 2016] OH, D. (2016). Before google: A brief history of search engines.

[Oziemblo, 2019] Oziemblo, A. (2019). Council post: Why site speed should be an optimization priority.

[Pannu, 2023] Pannu, J. (2023). Google's pagerank algorithm.

[Paolo Boldi,] Paolo Boldi, Massimo Santini, S. V.

[Patil et al., 2021] Patil, A., Pamnani, J., and Pawade, D. (2021). Comparative study of google search engine optimization algorithms: Panda, penguin and hummingbird. In *2021 6th International Conference for Convergence in Technology (I2CT)*, pages 1–5.

[Ramos, 2003] Ramos, J. E. (2003). Using tf-idf to determine word relevance in document queries.

[Ray, 2024] Ray, K. (2024). The history of google search - 1998 to 2024.

[Rousseau et al., 2008] Rousseau, C., Saint-Aubin, Y., Rousseau, C., and Saint-Aubin, Y. (2008). Google and the page rank algorithm. *Mathematics and Technology*, pages 1–24.

[Royle and Weisstein,] Royle, G. and Weisstein, E. W. Reducible matrix. *MathWorld–A Wolfram Web Resource.*

[Schubert, 2016] Schubert, D. (2016). Influence of mobile-friendly design to search results on google search. *Procedia - Social and Behavioral Sciences*, 220:424–433. 19th International Conference Enterprise and Competitive Environment 2016.

[Shellhammer, 2016] Shellhammer, A. (2016). The need for mobile speed.

[Srivastava et al., 2017] Srivastava, A., Garg, R., and Mishra, K. (2017). Discussion on damping factor value in pagerank computation. *International Journal of Intelligent Systems and Applications*, 9:19–28.

[Strzelecki, 2024] Strzelecki, A. (2024). Is chatgpt-like technology going to replace commercial search engines? *Library Hi Tech News*.

[Stumbles, 2017] Stumbles, T. (2017). History of google timeline.

[Taboga, 2021] Taboga, M. (2021). Linear independence of eigenvectors.

[Taylor, 2022] Taylor, D. (2022). A complete guide to the google penguin algorithm update.

[Team, 2023a] Team, S. (2023a). What are content mills? everything new writers should know.

[Team, 2023b] Team, S. (2023b). What is google panda? a complete guide.

[TFIDF,] TFIDF. Tf-idf: Term frequency-inverse document frequency. Tf-idf stands for term frequency-inverse document frequency, often used in information retrieval and text mining.

[Umbraco,] Umbraco. What is schema markup and how do you implement it?

[Watters and Amoudi, 2003] Watters, C. and Amoudi, G. (2003). Geosearcher: Location-based ranking of search engine results. *Journal of the American Society for Information Science and Technology*, 54(2):140–151.

[Weisstein, a] Weisstein, E. W. Eigenvalue. *MathWorld–A Wolfram Web Resource.*

[Weisstein, b] Weisstein, E. W. Spectral radius. *MathWorld–A Wolfram Web Resource.*

[Wickerhauser, 2022] Wickerhauser, M. V. (2022). Perron-frobenius theorem. LaTeX with Beamer class. Washington University in St. Louis, Missouri victor@wustl.edu http://www.math.wustl.edu/˜victor.

[WRIGHT, 2024] WRIGHT, A. (2024). Inside elon musk's feud with openai and sam altman as chatgpt makers reveal private emails in dramatic escalation.