PageRank, HITS and InDegree Benchmark

Ca' Foscari University of Venice
Department of Environmental Sciences, Informatics and Statistics

INFORMATION RETRIEVAL AND WEB SEARCH [CM0473]
Academic Year 2022 - 2023

Students

Zuliani Riccardo	875532
Aggio Nicola	880008

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The goal of this project is to produce a fair benchmark on the implementation of three algorithms for estimating the importance of a page in a given web graph. The algorithms we took into consideration were PageRank, HITS and In Degree, which were executed on the following datasets from the Stanford Large Network Dataset Collection:

Dataset	**Type**	**Nodes**	**Edges**	**Density**	**Web graph**
web-BerkStan	Directed	685,230	7,600,595	0.0000324	Of Berkeley and Stanford
web-Google	Directed	875,713	5,105,039	0.0000133	From Google
web-NotreDame	Directed	325,729	1,497,134	0.0000282	Of Notre Dame
web-Stanford	Directed	281,903	2,312,497	0.0000582	Of Stanford.edu

Table 1.1: Informations about the datasets

The structure of the project is the following: in Chapter 2 we provide an explanation of the theory behind the three algorithms we considered, with an extra focus on the mathematical formulae they use for measuring the importance of the pages in the graph; in Chapter 3 we describe the structure of our project and we explain the keys choices we made for implementing the algorithms; in Chapter 4 we first compare the algorithms in terms of elapsed time, number of steps to converge and, secondly, we compare their top-k results in terms of the Jaccard coefficient w.r.t. different values of k; finally, in Chapter 5, we draw the conclusions.

Chapter 2

Theoretical Background

2.1 PageRank

The year 1998 was an important year for Web link analysis and Web search, since both the PageRank and the HITS algorithms were reported in that year. The main ideas of PageRank and HITS are really quite similar. However, it is their dissimilarity that made a huge difference as we will see later. Since that year, **PageRank** has emerged as the **dominant link analysis model** for Web search, partly due to its **query-independent evaluation of Web pages** and its **ability** to **combat spamming**, and partly due to Google's business success.

PageRank relies on the **democratic nature of the Web** by using its vast link structure as an indicator of an individual page's quality. In essence, PageRank interprets a **hyperlink** from page x to page y as a **vote**, by page x, for page y. However, PageRank looks at **more than just the sheer number of votes** or links that a page receives. It also analyzes the **page** that **casts** the vote. **Votes** casted by pages that are themselves **"important"** weigh more heavily and help to make other pages more "important." This is exactly the idea of rank prestige in social networks.

2.1.1 PageRank algorithm

PageRank is a **static ranking of Web pages** in the sense that a **PageRank** value is **computed for each page off-line** and it does **not depend on search queries**. Since PageRank is based on the measure of prestige in social networks, the **PageRank value** of each page can be regarded as its **prestige**.

From the perspective of prestige, we use the following to derive the PageRank algorithm:

1. A **hyperlink** from a **page** pointing **to** another **page** is an implicit conveyance of authority to the target page. Thus, the **more in-links** that a page i receives, the **more prestige** the page i has;

2. Pages that point to page i also have their own prestige scores. A page with a **higher prestige** score **pointing** to i is **more important** than a page with a **lower prestige** score pointing to i. In other words, a **page** is **important** if it is **pointed** to by other **important pages**.

According to rank prestige in social networks, the importance of page i (i's **PageRank score**) is **determined** by **summing up the PageRank scores of all pages that point to** i. Since a page may point to many other pages, its **prestige score should be shared among all the pages** that it points to. Notice the difference from rank prestige, where the prestige score is not shared.

To formulate the above ideas, we treat the Web as a directed graph $G = (V, E)$, where V is the set of vertices or nodes, i.e., the set of all pages, and E is the set of directed edges in the graph, i.e., hyperlinks. Let the total number of pages on the Web be n (i.e., $n = |V|$). The PageRank score of the page i (denoted by $P(i)$) is defined by:

$$P(i) = \sum_{(j,i)\in E} \frac{P(j)}{O_j} \tag{2.1}$$

, where O_j is the number of out-links of page j, and it represents the "quantity" of prestige which is spread among the neighbors of i.

Mathematically, we have a system of n linear equations with n unknowns, so we can use a **matrix** to represent **all the equations**. Let P be a n-dimensional column vector of PageRank values, i.e.,

$$P = (P(1), P(2), ..., P(n))^T$$

Let A be the adjacency (usually sparse) matrix of our graph with

$$A_{ij} = \begin{cases} \frac{1}{O_i} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

We can write the system of n equations with as

$$P = A^T P \tag{2.2}$$

This is the **characteristic equation of the eigensystem**, where the **solution to P** is an **eigenvector** with the corresponding **eigenvalue of 1** ($\lambda = 1$). Since this is a **circular definition**, an **iterative algorithm is used to solve it**. It turns out that if **some conditions** are satisfied (which will be described shortly), **1 is the largest eigenvalue** and the **PageRank vector** P **is the principal eigenvector**. A well known mathematical technique called **power iteration** can be used to find P:

$$P_k = A^T P_{k-1}$$

Notice that the power iteration method stops when $P_k \approx P_{k-1}$.

However, the problem is that Equation 2.2 does not quite suffice because the Web graph does not meet the conditions. To introduce these conditions and the enhanced equation, let us derive the same Equation based on the **Markov chain**. In the Markov chain model, **each Web page** or node in the Web graph is **regarded as a state**. A **hyperlink** is a **transition**, which leads **from one state to another state with a probability**. Thus, this framework models **Web surfing as a stochastic process**. It models a Web surfer randomly surfing the Web as a state transition in the Markov chain. Recall that we used O_i to denote the number of out-links of a node i. Each **transition probability is** $1/O_i$ if we assume the Web surfer will click the hyperlinks in the page i **uniformly** at random, the "back" button on the browser is not used and the surfer does not type in an URL. Let A be the state transition probability matrix, a square matrix of the following format:

$$A = \begin{bmatrix} A_{11} & A_{12} & \dots & A_{1n} \\ A_{21} & A_{22} & \dots & A_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ A_{n1} & A_{n2} & \dots & A_{nn} \end{bmatrix}$$

, where A_{ij} represents the **transition probability** that the **surfer** in **state** i (page i) will move to **state** j (page j).

Given an **initial probability distribution vector** that a surfer is at each state (or page) $p_0 = (p_0(1), p_0(2), ..., p_0(n))^T$ (a column vector) and an $n \times n$ transition probability matrix A, we have

$$\sum_{i=1}^{n} p_0(i) = 1 \tag{2.3}$$

$$\sum_{j=1}^{n} A_{ij} = 1 \tag{2.4}$$

Equation 2.4 is not quite true for some Web pages because they have no out-links. If the matrix A satisfies Equation 2.4, we say that A is the **stochastic matrix of a Markov chain**. Let us assume A is a stochastic matrix for the time being and deal with it not being that later.

In a Markov chain, a question of common interest is: **given the initial probability distribution** p_0 at the beginning, what is the **probability** that m **steps**/transitions later that the **Markov chain will be at each state** j? We can determine the probability that the system (or the random surfer) is in state j after 1 step (1 state transition) by using the following reasoning:

$$p_1(j) = \sum_{i=1}^{n} A_{ij}(1) p_0(i)$$

where $A_{ij}(1)$ is the probability of going from i to j in 1 transition, and $A_{ij}(1) = A_{ij}$. We can write it with a matrix:

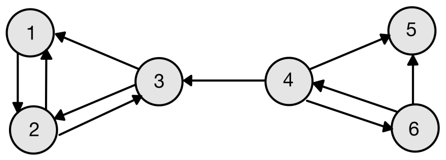
$$p_1 = A^T p_0$$

In general, the probability distribution after k steps is:

$$p_k = A^T p_{k-1} \tag{2.5}$$

By the **Ergodic Theorem of Markov chains**, a **finite Markov chain** defined by the stochastic transition matrix A has a **unique stationary probability distribution** if A **is irreducible and aperiodic**.

The **stationary probability distribution** means that **after a series of transitions** p_k **will converge to a steady-state probability vector** π **regardless of the choice of the initial probability vector** p_0, i.e.,



$$A = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

Figure 2.1: Example of a dangling node

$$\lim_{k \to \infty} p_k = \pi$$

When we reach the steady-state, we have $p_k = p_{k+1} = \pi$, and thus $\pi = A^T \pi$. π is the **principal eigenvector** of A^T with **eigenvalue of 1**. In PageRank, π is used as the PageRank vector P. Thus, we obtain:

$$P = A^T P \tag{2.6}$$

Using the **stationary probability distribution** π as the **PageRank vector** is **reasonable** and quite **intuitive** because it reflects the long-run probabilities that a random surfer will visit the pages. A page has a high prestige if the probability of visiting it is high.

Now let us come back to the real Web context and **see whether the above conditions are satisfied**, i.e., whether A is a stochastic matrix and whether it is irreducible and aperiodic. In fact, none of them is satisfied. Hence, we need to **extend** the ideal-case Equation 2.6 to produce the **"actual PageRank model"**. Let us look at each condition below.

First of all, A is **not a stochastic** (transition) **matrix**. A stochastic matrix is the transition matrix for a finite Markov chain whose entries in each row are non-negative real numbers and sum up to 1. This requires that **every Web page must have at least one out-link**. This is **not true on the Web because many pages have no out-links**, which are reflected in transition matrix A by **some rows of complete 0's**. Such pages are called the **dangling pages (nodes)**.

Example: Dangling node in a graph: as we can see, Picture 2.1 shows that node 5 is a dangling node, and the corresponding transition matrix is not a stochastic matrix.

$$A = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

We can fix this problem in several ways in order to convert A to a stochastic transition matrix. We describe only two ways here:

1. **Remove** those **pages** with **no out-links** from the system **during the PageRank computation** as these pages do not affect the ranking of any other page

directly. Out-links from other pages pointing to these pages are also removed. After PageRanks are computed, these pages and hyperlinks pointing to them can be added in. Note that the **transition probabilities** of those pages with removed links will be **slightly affected** but not significantly;

2. **Add** a **complete set of outgoing links from each such page** i **to all the pages on the Web**. Thus the **transition probability** of going from i to every page is $1/n$ assuming uniform probability distribution. That is, **we replace each row containing all 0's** with e/n, where e is n-dimensional vector of all 1's.

If we use the second method to make A a stochastic matrix by adding a link from page 5 to every page, we obtain a stochastic matrix:

$$\bar{A} = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

Figure 2.2: Transformation of matrix A into $\bar{A}$

Second, A is **not irreducible**. Irreducible means that the Web graph G is **strongly connected**, i.e. if and only if, for each pair of nodes $u, v \in V$, there is a path from u to v. A **general Web graph** represented by A is **not irreducible** because **for some pair of nodes** u and v, **there is no path** from u to v. For example, in Picture 2.1, there is no directed path from node 3 to node 4. The adjustment of adding a complete set of out-links is not enough to ensure irreducibility.

Finally, A is **not aperiodic**: a state i in a Markov chain being **periodic** means that **there exists a directed cycle that the chain has to traverse**. More formally, a state i is periodic with period $k > 1$ if k is the smallest number such that all paths leading from state i back to state i have a length that is a multiple of k. If a state is not periodic (i.e., k = 1), it is aperiodic. A **Markov chain** is **aperiodic** if **all states** are **aperiodic**.

Example: Periodic chain: Picture 2.3 shows a periodic Markov chain with $k = 3$. The transition matrix is given on the left. Each state in this chain has a period of 3. For example, if we start from state 1, to come back to state 1 the only path is 1-2-3-1 for some number of times, say h. Thus any return to state 1 will take $3h$ transitions.

It is easy to deal with the above **two problems** with a single strategy, called **teleportation**: we **add** a **link from each page to every page** and give **each link** a **small transition probability** controlled by a **parameter** d, with $0 < d < 1$.

In this way, the augmented transition matrix becomes **irreducible** because it is clearly **strongly connected**. It is also **aperiodic** because the situation in Picture 2.3 no longer exists as we now have **paths of all possible lengths from state** i **back to state** i. Finally, since the augmented transition matrix is both irreducible and aperiodic, we're ensured that it has a single stationary distribution.

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{bmatrix}$$

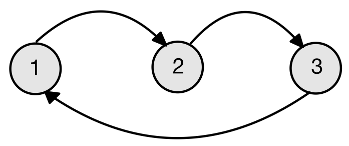


Figure 2.3: Example of periodic chain

After this augmentation, we obtain an **improved PageRank model**. In this model, at a page, the random surfer has two options:

1. With probability d, he randomly chooses an out-link to follow, i.e. it **clicks on a hyperlink**;
2. With probability $1 - d$, he jumps to a random page without a link.

Thus, the improved model is

$$P = ((1-d)\frac{E}{n} + dA^T)P \qquad (2.7)$$

where E is ee^T (e is a column vector of all 1's) and thus E is a $n \times n$ square matrix of all 1's. In this sense, E represents the adjacency matrix of a completely connected graph, and we multiply it by $(1-d)/n$, which is the probability of clicking on a teleportation link.

Example: Improved PageRank model: Picture 2.4 shows the resulting matrix after applying the improved model described above. On the left we have the original matrix made stochastic, while on the left the resulting matrix. Notice that the original matrix is quite sparse, while the resulting one is dense. In this case the parameter $d = 0.9$.

$$\bar{A} = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 1/2 & 0 & 1/2 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 1/3 \\ 1/6 & 1/6 & 1/6 & 1/6 & 1/6 & 1/6 \\ 0 & 0 & 0 & 1/2 & 1/2 & 0 \end{bmatrix}$$

$$(1-d)\frac{E}{n} + dA^T = \begin{bmatrix} 1/60 & 7/15 & 1/60 & 1/60 & 1/6 & 1/60 \\ 7/15 & 1/60 & 11/12 & 1/60 & 1/6 & 1/60 \\ 7/15 & 7/15 & 1/60 & 19/60 & 1/6 & 1/60 \\ 1/60 & 1/60 & 1/60 & 1/60 & 1/6 & 7/15 \\ 1/60 & 1/60 & 1/60 & 19/60 & 1/6 & 7/15 \\ 1/60 & 1/60 & 1/60 & 19/60 & 1/6 & 1/60 \end{bmatrix}$$

Figure 2.4: Resulting matrix

Notice that $(1-d)\frac{E}{n}+dA^T$ is a **stochastic matrix**, **irreducible and aperiodic**. Moreover:

$$P = ((1-d)\frac{E}{n} + dA^T)P = (1-d)\frac{1}{n}ee^TP + dA^TP$$

, since $E = ee^T$ and $e^TP = 1$, since P is a probability vector (i.e. it sums up to 1). For this reason, we obtain:

$$P = (1-d)\frac{1}{n}e + dA^TP$$

, and if we scale this equation by multiplying both sides by n, and considering that $e^TP = n$, then:

$$P = (1-d)e + dA^TP$$

This gives us the PageRank formula for each page i as follows:

$$\boxed{P(i) = (1-d) + d\sum_{j=1}^{n} A_{ji}P(j) = (1-d) + d\sum_{(j,i)\in E}^{n} \frac{P(j)}{O_j}}$$

The parameter d is called **damping factor**, $0 < d < 1$, and $d = 0.85$ was used in the paper.

The computation of PageRank values of the Web pages can be done using the well known **power iteration method**, which produces the principal eigenvector with the eigenvalue of 1. The algorithm is simple, and is given in the following pseudo-code 1.

Algorithm 1 PageRank-Iterate

```
Require: $G$
  $P_0 \leftarrow e/n$
  $k \leftarrow 1$
  repeat
      $P_k \leftarrow (1-d)e + dA^TP_{k-1}$
      $k \leftarrow k+1$
  until $||P_k - P_{k-1}||_1 < \epsilon$
  return $P_k$
```

The iteration ends when the PageRank values do not change much or converge: in the Picture above, the iteration ends after the 1-norm of the residual vector is less than a pre-specified threshold ϵ. Note that the 1-norm for a vector is simply the sum of all the components.

Example: PageRank computation: Picture 2.5 shows an example of PageRank computation: in this case the PageRank equation is not scaled, so we have $e^TP = 1$ and thus $P(i) = \frac{1-d}{n} + d\sum_{(j,i)\in E}\frac{P(j)}{O_j}$.

As we can see, the score of PageRank is written at the bottom of the page, e.g. $P(1) = 0.304$, and its score is distributed among all its 5 out-links $0.304 = 0.061 * 5$. In particular, the PageRank is obtained as $P(1) = \frac{1-d}{n} + d(0.023 + 0.166 + 0.071 + 0.045) = 0.304$, considering $d = 0.85$

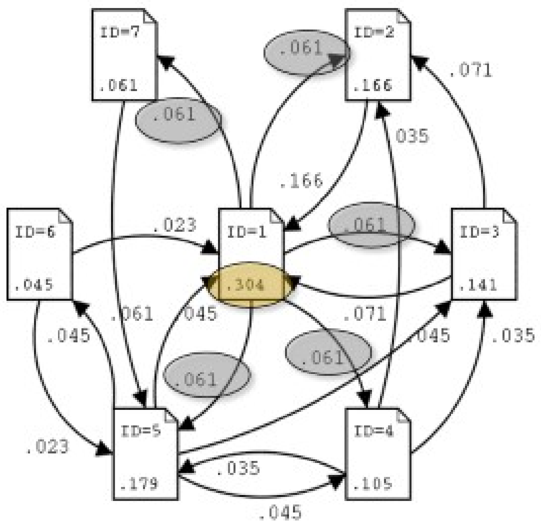


Figure 2.5: Example of PageRank

2.1.2 Strengths and weaknesses of PageRank

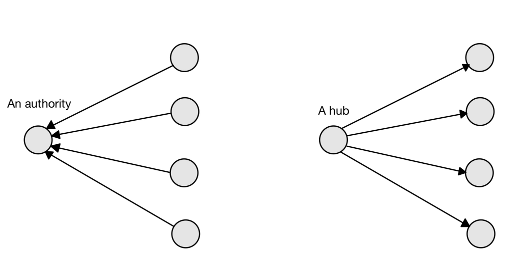
The main **strength** of PageRank is its **ability to fight spam**. A page is important if the pages pointing to it are important. Since it is **not easy** for Web page owner to **add in-links** into his/her page from other important pages, it is thus **not easy** to **influence PageRank**: nevertheless, there are reported ways to influence PageRank. Another major **advantage** of PageRank is that it is a **global measure** and is **query independent**. That is, the **PageRank values** of all the pages on the Web are **computed** and **saved off-line rather** than at the **query time**. At the **query time**, only a **lookup** is **needed** to find the value to be integrated with other strategies to rank the pages. It is thus **very efficient at the query time**. Both these two advantages contributed greatly to Google's success.

The main **weakness** is also the **query-independence** nature of PageRank. It **could not distinguish between pages that are authoritative in general** and pages that are **authoritative on the query topic**. Another **criticism** is that PageRank **does not consider time**, and we note again that the **link-based ranking** is **not** the **only strategy** used in a search engine. Many other information retrieval methods, heuristics, and empirical parameters are also employed.

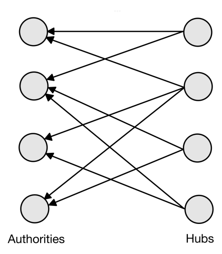
2.2 HITS

HITS stands for Hypertext Induced Topic Search. Unlike PageRank which is a static ranking algorithm, HITS is **search query dependent**. When the user issues a search query, HITS first expands the list of relevant pages returned by a search engine and then produces two rankings of the expanded set of pages, authority ranking and hub ranking.

An **authority** is a page with many **in-links**. The idea is that the page may have authoritative content on some topic and thus many people trust it and thus link to it. A **hub** is a page with many **out-links**. The page serves as an organizer of the information on a particular topic and points to many good authority pages on the topic. When a user comes to this hub page, he/she will find many useful



(a) An authority and a hub page



(b) A densely linked set of authorities and hubs

links which take him/her to good content pages on the topic. The following picture shows an authority page and a hub page.

The **key idea** of HITS is that a **good hub points** to **many good authorities** and a **good authority** is **pointed** to by **many good hubs**. Thus, authorities and hubs have a mutual reinforcement relationship.

2.2.1 HITS algorithm

Before describing the HITS algorithm, let us first **describe** how **HITS collects pages** to be ranked. Given a broad search query, q, HITS collects a set of pages as follows:

1. It sends the **query** q to a **search engine system**. It then **collects** t ($t = 200$ is used in the original paper) **highest ranked pages**, which assume to be highly relevant to the search query. This set is called the **root set** W;

2. It then **grows** W by **including** any **page pointed to by a page** in W and **any page that points to a page** in W. This gives a larger set called S. However, this set can be very large. The algorithm **restricts its size** by allowing each page in W to bring at most k pages ($k = 50$ is used in the HITS paper) pointing to it into S. The set S is called the **base set**.

HITS then works on the pages in S, and **assigns every page** in S an **authority score** and a **hub score**. Let the number of pages to be studied be n. We again use $G = (V, E)$ to denote the (directed) link graph of S. V is the set of pages (or nodes) and E is the set of directed edges (or links). We use L to denote the adjacency matrix of the graph:

$$L_{ij} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Let the **authority score** of the page i be $a(i)$, and the **hub score** of page i be $h(i)$. The mutual reinforcing relationship of the two scores is represented as follows:

$$a(i) = \sum_{(j,i)\in E} h(j)$$

, i.e. the authority score is given by the sum of the hub scores of the incoming links, and

$$h(i) = \sum_{(i,j)\in E} a(j)$$

, i.e. the hub score is given by the sum of the authority scores of the outgoing links.

Writing them in the **matrix form**, we use a to denote the column vector with all the authority scores, $a = (a(1), a(2), ..., a(n))^T$, and use h to denote the column vector with all the hub scores, $h = (h(1), h(2), ..., h(n))^T$:

$$a = L^T h \qquad h = La$$

The computation of authority scores and hub scores is basically the same as the computation of the PageRank scores using the **power iteration method**. If we use a_k and h_k to denote authority and hub scores at the k-th iteration, the iterative processes for generating the final solutions are:

$$\boxed{a_k = L^T L a_{k-1} \quad \text{and} \quad h_k = L L^T h_{k-1}}$$

starting with $a_0 = h_0 = (1, 1, ..., 1)$. After each iteration, the **values** are also **normalized** (to keep them small) so that:

$$\boxed{\sum_{i=1}^{n} a(i) = 1 \quad \text{and} \quad \sum_{i=1}^{n} h(i) = 1}$$

The algorithm is shown in the following pseudo-code.

Algorithm 2 HITS-Iterate

```
Require: $G$
  $a_0 \leftarrow h_0 \leftarrow (1, 1, ..., 1)$
  $k \leftarrow 1$
  repeat
      $a_k \leftarrow L^T L a_{k-1}$
      $h_k \leftarrow L L^T h_{k-1}$
      $a_k \leftarrow a_k / ||a_k||_1$                    ▷ normalization
      $h_k \leftarrow h_k / ||h_k||_1$                    ▷ normalization
      $k \leftarrow k + 1$
  until $||a_k - a_{k-1}||_1 < \epsilon_a$ and $||h_k - h_{k-1}||_1 < \epsilon_h$
  return $a_k$ and $h_k$
```

The iteration ends after the 1-norms of the residual vectors are less than some thresholds ϵ_a and ϵ_h. The pages with large authority and hub scores are better authorities and hubs respectively. HITS will select a few top ranked pages as authorities and hubs, and return them to the user.

Although **HITS** will **always converge**, there is a problem with **uniqueness of limiting (converged) authority and hub vectors**. It is shown that for certain types of graphs, **different initializations** to the power method **produce different final authority and hub vectors**.

2.2.2 Strengths and weaknesses of HITS algorithm

The main **strength** of HITS is its **ability to rank pages according to the query topic**, which may be able to provide more relevant authority and hub pages. The ranking may also be combined with information retrieval based rankings.

However, HITS has several **disadvantages**.

1. It **does not have the anti-spam capability of PageRank**. It is quite **easy to influence** HITS by **adding out-links from one's own page to point to many good authorities**. This **boosts** the **hub score** of the page. Because hub and authority scores are interdependent, it in turn **also increases the authority score** of the page;

2. Another problem of HITS is **topic drift**. In **expanding the root set**, it can easily **collect** many **pages** (including authority pages and hub pages) which have **nothing to do the search topic** because out-links of a page may not point to pages that are relevant to the topic and in-links to pages in the root set may be irrelevant as well because people put hyperlinks for all kinds of reasons, including **spamming**;

3. The **query time evaluation** is also a major drawback. Getting the root set, expanding it and then performing eigenvector computation are all **time consuming operations**.

2.3 InDegree

In Degree represents one of the measures that we can use to establish the degree of prominence of a node in a graph. As the previous two measures, it is associated to each node, and it is defined by the **number of incoming links to the given node**. This can be mathematically formalized by the following formula:

$$P_d(i) = \frac{d_I(i)}{n-1}$$

Where:

- $d_I(i)$ represents the **In Degree** of node i, i.e. the number of incoming links to node i;

- n represents the number of nodes.

Notice that the denominator $n-1$ is used to normalize the quantity $d_I(i)$, so that the final prestige of node i ranges from 0 to 1.

Chapter 3

Key Choices

In this chapter we describe the structure of the project and the key choices we made in the implementation of the algorithms.

3.1 Application structure

The structure and the content of the files that compose the application are the following one:

- *Main.cpp*: this is the only .cpp file, and it is used to compile and to generate the .exe file to run the application. In this sense, its objective is to run the benchmark over all the specified datasets and to store the obtained results in three .csv files;
- *Graph.hpp*: this is a library-style file that manages the storing operations of the web graphs and the functions for retrieving and printing the top-k nodes obtained by the algorithms;
- *Pagerank.hpp*: this is a library-style file that manages the implementation of the PageRank algorithm;
- *HITS.hpp*: this is a library-style file that manages the implementation of the HITS algorithm;
- *InDegree.hpp*: this is a library-style file that manages the implementation of the In Degree algorithm;
- *Utils.hpp*: this is a library-style file that manages all the useful functions to make the application run correctly.

3.2 Dataset storing strategy

During the process of studying the datasets we realized that *web-Google* and *web-NotreDame* had the first node set to 0, while *web-BerkStan* and *web-Stanford* had the first node set to 1. We decided to opt for a normalization regarding the numbering of nodes, this step was done by obtaining the smallest and largest node ID from each dataset.

Once we managed to implement this feature, we noticed that if we represented the **dense adjacency matrix** of the *web-NotreDame* dataset, for example, by using a **double** per entry, our running machine would have needed more or less **395 GBs** for storing the matrix. For this reason, as suggested by the professor, we directed our focus to the *Compressed Sparse Row* (CSR) or *Compressed Row Storage* (CRS) representation technique, in order to both manage and store the web graph in a smart way.

The main idea behind the CSR or CRS is that instead of storing a matrix by rows and columns, we memorize **two** separate **vectors**:

1. The first one stores pairs of elements, where the first represents the actual value to store and the second represents the column ID of that particular cell;

2. The second, again, stores pairs, composed by the row pointer and the non empty row pointer. In particular, this second element is used in order to skip rows filled with zeros.

These two vectors were in both PageRank and HITS. In particular, the first element of each pair in the first vector for the PageRank algorithm is referred to as the $1/O_i$ of a given node, whereas in the HITS algorithm since each value of the adjacency matrix and its transpose are respectively 0 or 1 we opt to use a normal vector without using any pairs.

But even using compressed techniques the amount of data to store exceeds the amount of RAM memory of our running machine, thus to overcome this problem again we decide to grab another suggestion by the professor to use the *mmap* function from the standard C++ library. The main advantage of using *mmap* instead of the classic *malloc* is that the former allows to store data directly on the main memory (hard disk), whereas the latter allocates the given data into the RAM, which of course has less memory capacity than the disk.

3.3 Implementations

Now we jump into a brief high-level discussion on how we implemented the three algorithms, focusing on the key choices and main data structures we used. For a deeper code analysis please refer to the GitHub Repository.

First of all, in order to store the graph we decided to identify each edge as a *std::pair<unsigned int, unsigned int>*.

3.3.1 In Degree

The first algorithm that we designed was the In Degree, because of its simplicity. Indeed, excluding the creation of the Graph object that stores the edges, the computation of the actual values are performed by a simple loop through the edges, increasing the value of each *ToNode_ID* normalized by the total number of nodes minus one, as described in 2.3.

3.3.2 PageRank

Moving forward, we developed the PageRank algorithm and here things became a little tricky.

The first step was to create the relative **cardinality map**, useful for computing the **transpose matrix**, and the list of **dangling nodes**, which are necessary for an accurate computation of the PageRank score. The cardinality map is designed as a *std::unordered_map<unsigned int, unsigned int>*, while the list of dandling nodes as a *std::vector<unsigned int>*. Here we sort the edges by the first element *(FromNode_ID)* and iterate through them by checking if the previous node ID is the same as the actual one, increasing its cardinality by 1 for a positive response. Moreover, once we detect a difference between nodes ID which is higher than 1, it means that we encountered some dangling nodes.

As previously mentioned, the following step was to compute the **transpose matrix**, and in order to do so we sorted the edges by the second element *(ToNode_ID)* and allocated the right amount of memory. Then, by iterating through them we populated the vector of row pointers and the transpose matrix. In the end, since the original graph now has no utility, we freed the sudden memory area.

The final **PageRank prestige** of each node is stored in *std::unordered_map<unsigned int, double>* and the corresponding computations can be resumed as a loop in which we update each value until we reach the convergence. In this loop we first compute the PageRank prestige for the dangling nodes, then by iterating through the edges we compute the $A^T \times P_K$ which is the effective matrix multiplication between a CSR and a vector. In conclusion, we update these values using the complete PageRank formula $P_{k+1} = (d_{P_k} + A^T \times P_k) \times d + (1-d)/n$, where:

- d_{P_k} is the PageRank prestige of the dangling nodes;
- A^T is the transpose matrix;
- P_k is the PageRank prestige at time k;
- d is the teleportation probability (in this project set to 0.85);
- n is the number of nodes.

3.3.3 HITS

The HITS algorithm uses the **adjacency matrix** and its **transpose** to compute the **authority** and **hub** score, so the computation of the two matrices represents the first operation to implement. The procedure is equivalent to the one of the transpose matrix in the PageRank case, with the only difference that for the computation of the normal adjacency matrix we sort the edges by the first element, whereas for the transpose by the second element.

The computation of the **authority** and **hub** score starts by initializing the scores at time $k+1$, and then it proceeds by computing $h_{k+1} = L \times a_k$ and $a_{k+1} = L^T \times h_k$ by iterating through the edges, where:

- a_{k+1} and a_k are the authority scores at time $k+1$ and k;
- h_{k+1} and h_k are the hub scores at time $k+1$ and k;
- L and L^T are the **adjacency matrix** and its **transpose**.

This iteration stops when we reach the convergence criteria for both scores.

3.3.4 Jaccard coefficient

The algorithm results for each top-k value are stored in a *std::unordered_map<unsigned int, std::vector<std::pair<unsigned int, double>>>*. This structure is useful since we pass each algorithm results to the Jaccard class that compute the sequence of top-k node ID for each k value. With that we can compute all the comparison pairs of Jaccard coefficients by firstly computing the length of the intersection of the two ordered vectors and then computing the effective Jaccard coefficient formula, that we recall to be the following.

Jaccard coefficient

The Jaccard coefficient between two sets A and B is computed as:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Chapter 4

Benchmarks

In this chapter we first compare the elapsed time and the number of steps taken by each algorithms w.r.t. the datasets we took into consideration. Then we measure the Jaccard coefficient between the top-k results obtained by the algorithms w.r.t. different values of k.

We would like to recall that for the PageRank algorithms, we set the teleportation probability value to 0.85.

4.1 Berkeley-Stanford dataset

Pictures 4.1 and 4.2 shows the comparison between the elapsed time and the number of steps of the three algorithms when executed on the Berkeley-Stanford dataset.

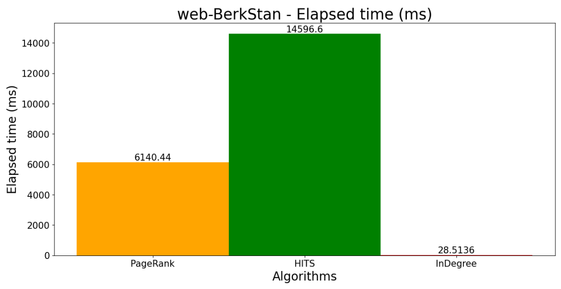


Figure 4.1: BS - Elapsed Time (ms)

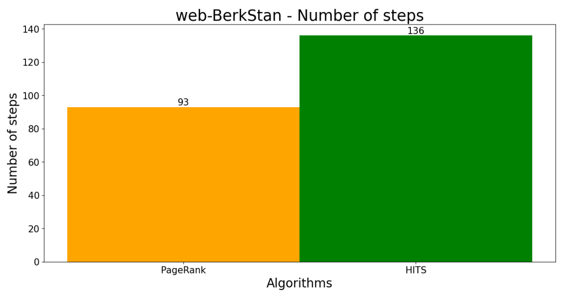


Figure 4.2: BS - Number of Steps

As we can see, in this case PageRank performs a little worse than HITS both in terms of elapsed time and in terms of number of steps to converge and, as expected, In Degree results to be the fastest algorithm. Clearly, this result comes from the fact that In Degree computation is only characterized by only a single analysis of the sequence of edges.

Picture 4.3 shows the values of the Jaccard coefficients for each algorithms pairs.

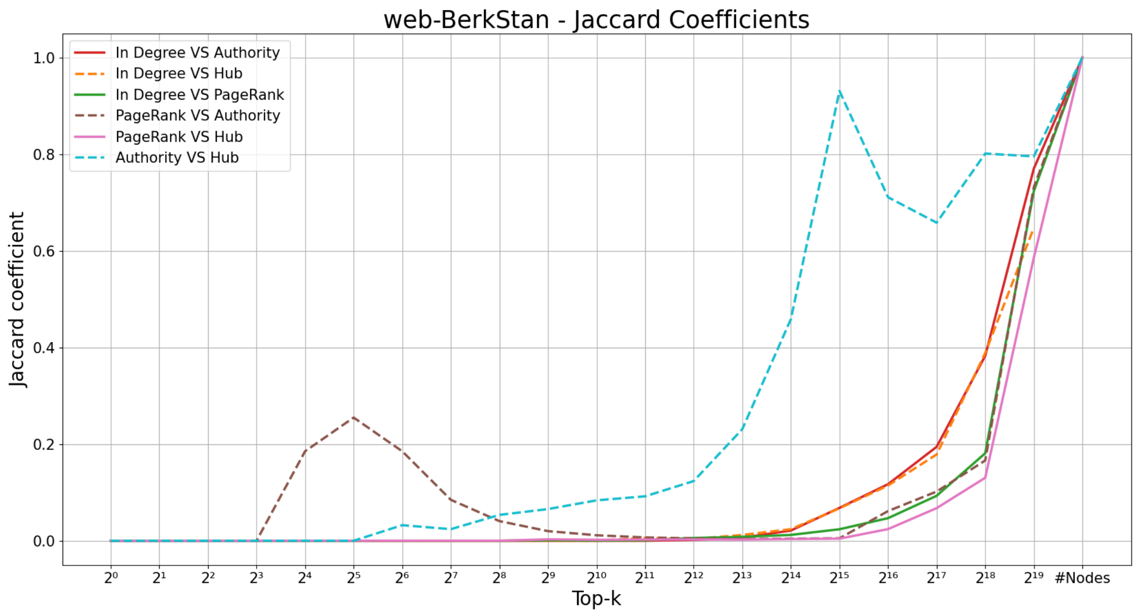


Figure 4.3: BS - Jaccard coefficients vs Top-K

As we can see, for almost all the pairs the Jaccard coefficient is close to zero until the value $k = 2^{13}$, with the exceptions of *PageRank VS Authority*, whose Jaccard score starts to increase from $k = 2^3$ until $k = 2^5$, and *Authority VS Hub*, whose score begin to increase at $k = 2^5$.

4.2 Google dataset

Pictures 4.4 and 4.5 shows the comparison between the elapsed time and the number of steps of the three algorithms when executed on the Berkeley-Stanford dataset.

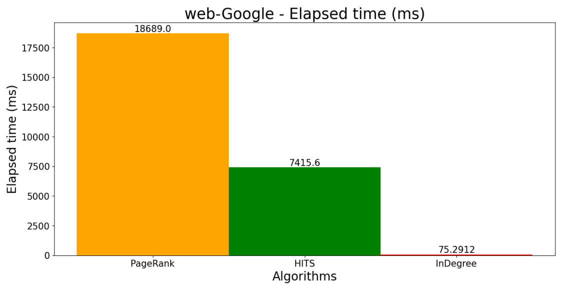


Figure 4.4: G - Elapsed Time (ms)

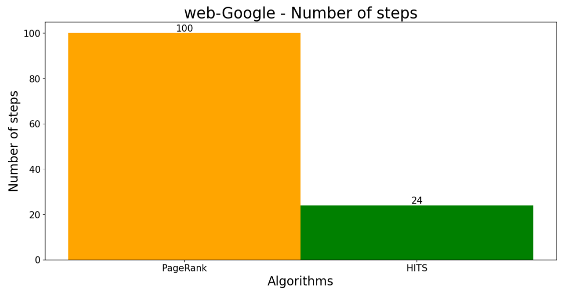


Figure 4.5: G - Number of Steps

Differently from the previous dataset, in this case PageRank performs much better than HITS both in terms of elapsed time and number of steps to converge. Again, In Degree represents the fastest algorithm.

Picture 4.6 shows the values of the Jaccard coefficients for each algorithms pairs.

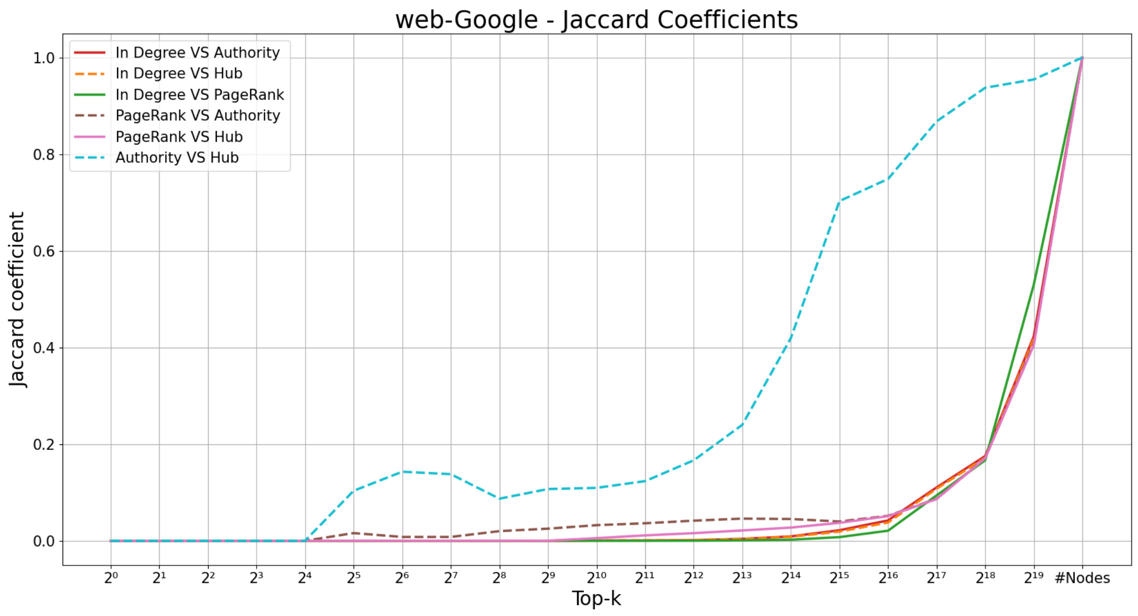


Figure 4.6: Google - Jaccard coefficients vs Top-K

As we can see, already at $k = 2^4$ *Authority VS Hub* and *PageRank VS Authority* start to increase, and in particular the former faster than the latter. Whereas the other coefficients increase much later, with a heavy boost in the last values of k, reaching 1 for a value of k corresponding to the total number of nodes of the graph.

4.3 NotreDame dataset

Pictures 4.7 and 4.8 shows the comparison between the elapsed time and the number of steps of the three algorithms when executed on the Berkeley-Stanford dataset.

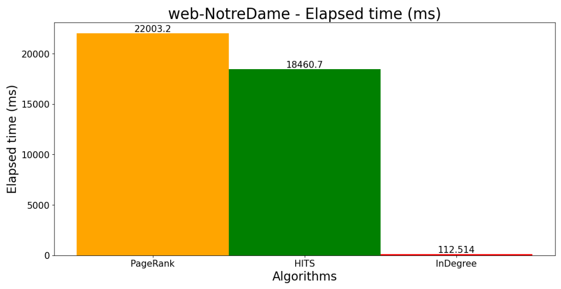


Figure 4.7: ND - Elapsed Time (ms)

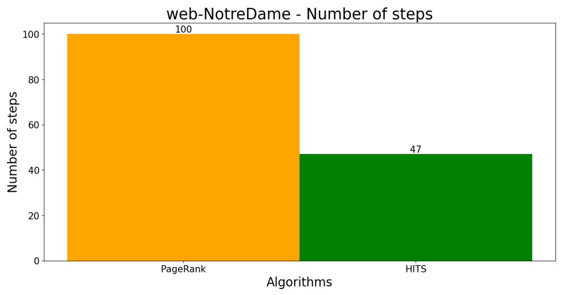


Figure 4.8: ND - Number of Steps

As in the previous case, PageRank outperformed HIST both in terms of elapsed time and number of steps to converge.

Picture 4.9 shows the values of the Jaccard coefficients for each algorithms pairs.

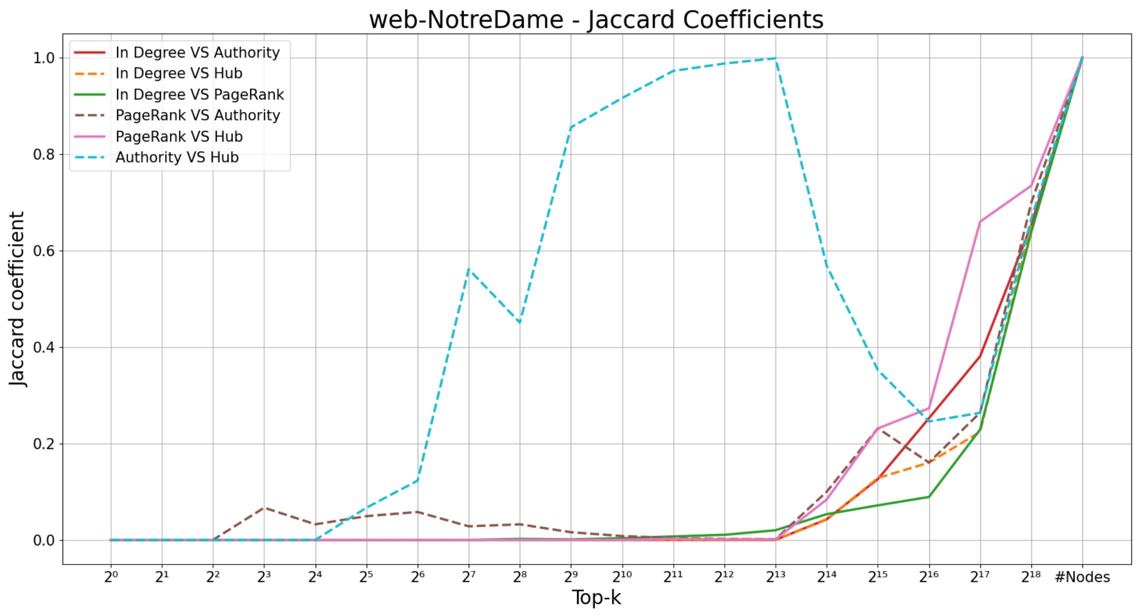


Figure 4.9: ND - Jaccard Coefficient Top-K

As we can see, the Jaccard score of *Authority VS Hub* reaches 1 at $k = 2^{13}$, but after that value it decreases. The score of *PageRank VS Authority* starts to increase early at $k = 2^2$, but soon it returns more or less to the zero value at $k = 2^{12}$. Regarding the other coefficients, they all start to increase up to 1 from $k = 2^{13}$, with the exception of *In Degree VS PageRank*.

4.4 Stanford dataset

Pictures 4.11 and 4.11 shows the comparison between the elapsed time and the number of steps of the three algorithms when executed on the Berkeley-Stanford dataset.

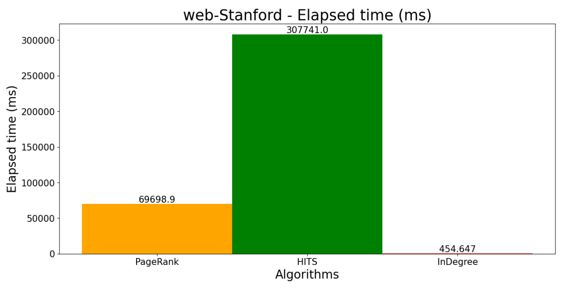


Figure 4.10: S - Elapsed Time (ms)

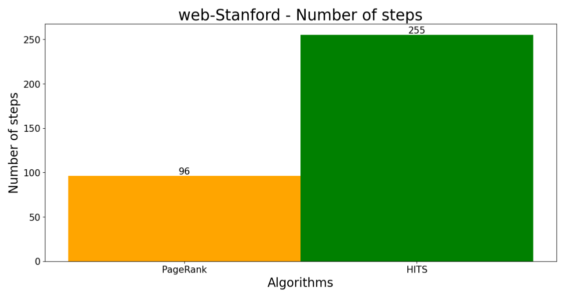


Figure 4.11: S - Number of Steps

Like in the Berkeley-Stanford dataset 4.1, HITS performs better than PageRank in both measures.

Picture 4.12 shows the values of the Jaccard coefficients for each algorithms pairs.

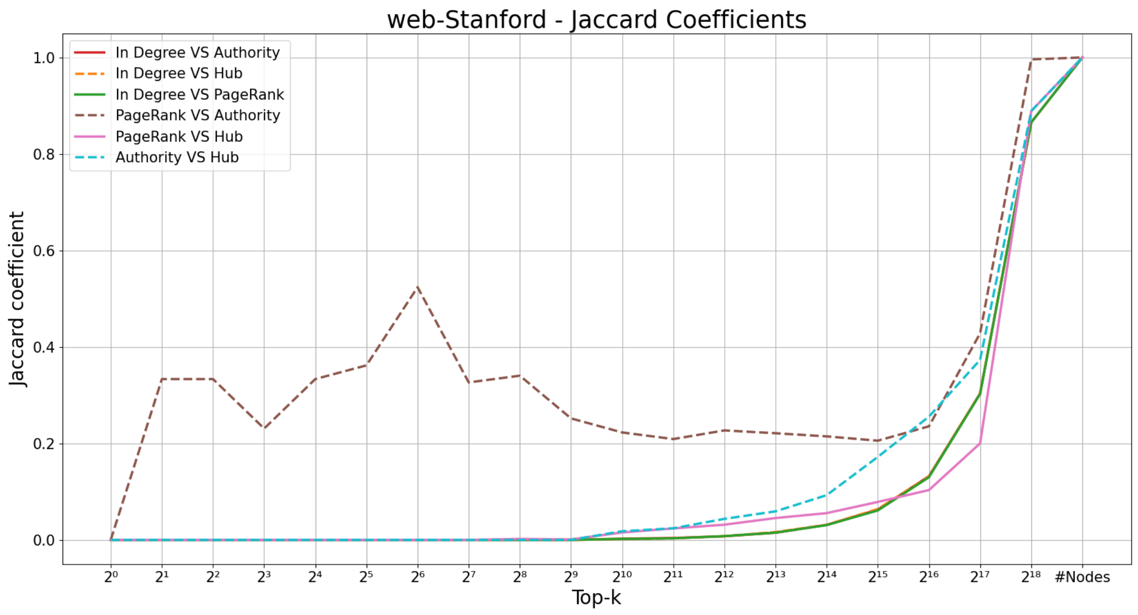


Figure 4.12: S - Jaccard Coefficient Top-K

Finally, we see that for almost all the pairs the Jaccard value increases from $k = 2^9$, while for *PageRank VS Authority* it increases at early stages from $k = 2^1$, and then it remains more or less stable until heavily increasing at $k = 2^{16}$, where in later stages it reaches 1 like the others comparison pairs.

Chapter 5

Conclusion

As expected, from 4.3, 4.6, 4.9 and 4.12 we can see that all the comparisons converge to 1 w.r.t. increasing the number of top-k retrieved nodes. This happens since taking a higher value of k we are increasing the number of nodes that are retrieved, so the cardinality of the intersection improves until it reaches the cardinality of the union.

Now, to draw the final conclusions, we discuss the bar and box plots of the mean elapsed time and number of steps taken by the algorithms.

5.1 Elapsed time (ms)

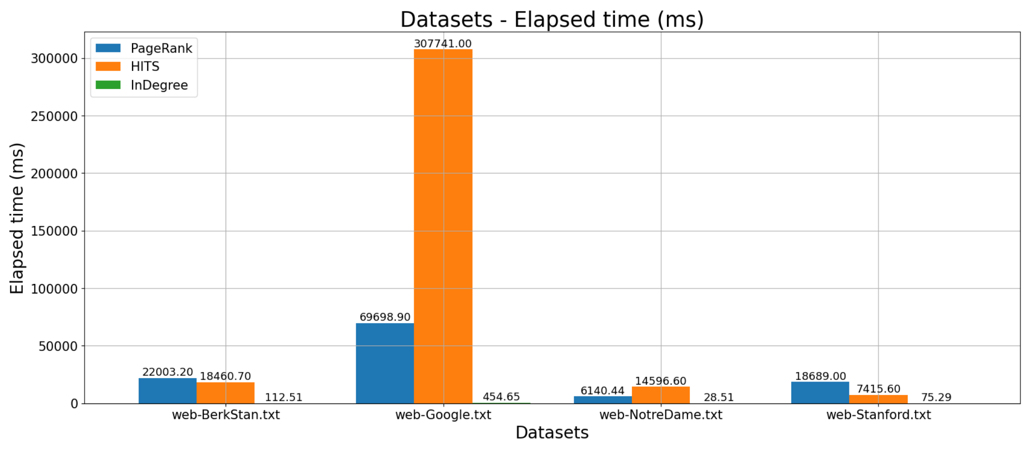


Figure 5.1: Bar Plot Dataset - Elapsed time (ms)

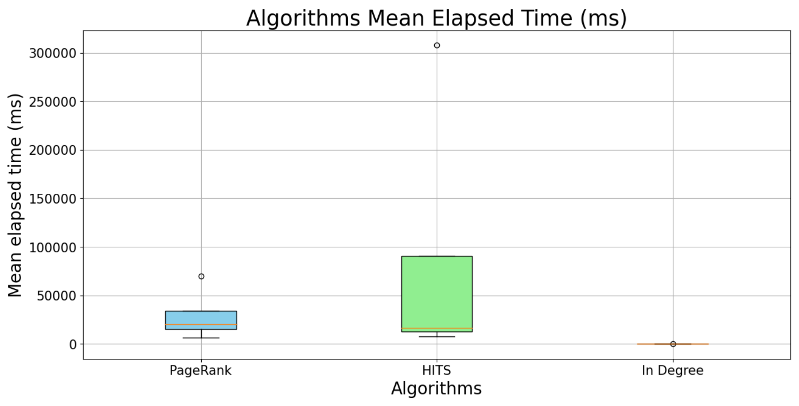


Figure 5.2: Box Plot Dataset - Elapsed time (ms)

As we can see, the best performing algorithm, as expected, is of course In Degree for its simplicity. Analyzing the comparison between PageRank and HITS, we can see that PageRank is more constant than HITS: indeed, the worst performance is held by HITS in the *web-Google* dataset. However, the latter has a mean execution time slightly lower respect the former.

5.2 Number of steps

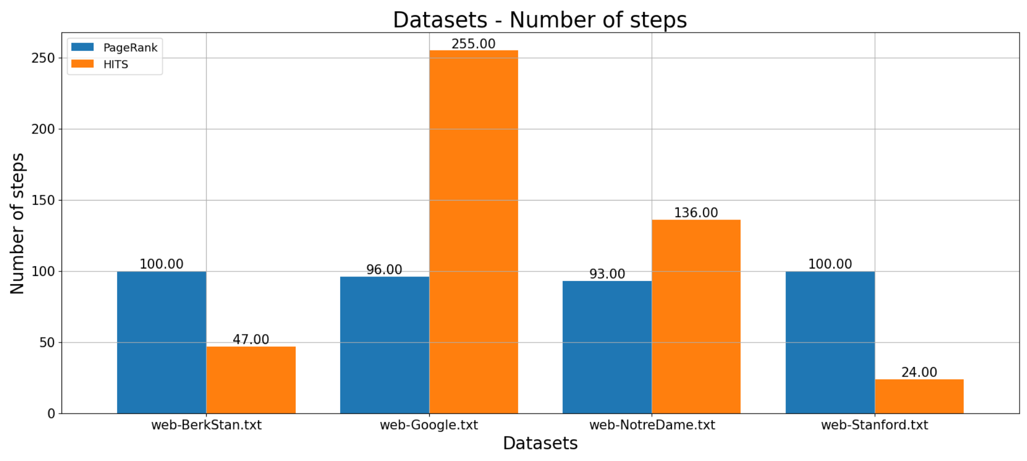


Figure 5.3: Bar Plot Dataset - Number of steps

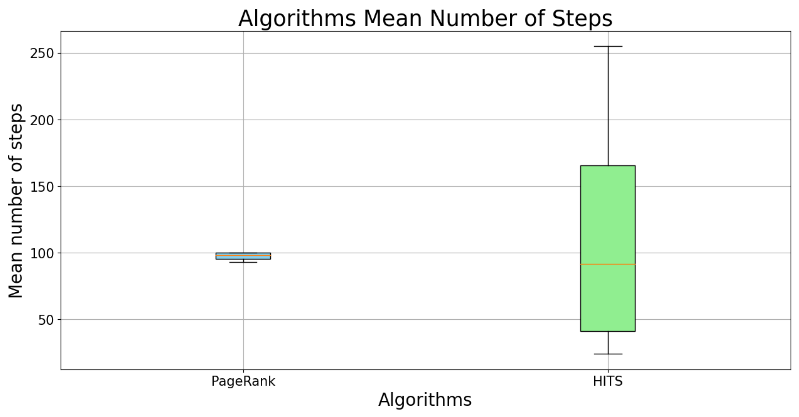


Figure 5.4: Box Plot Dataset - Number of steps

During the analysis of the number of steps we excludes the In Degree algorithm, since it iterates only once the whole graph and this does not grab some useful information regarding the benchmark. As before, PageRank seems to be much more stable than HITS in the number of steps, which again holds the worst number of steps in the same dataset of the previous comparison, the *web-Google*. This phenomenon can be seen much clearer on the upper box plot 5.4 where the PageRank box plot is thin as a leaf. Finally, speaking about the mean number of steps, like the previous comparison, HITS has a slightly lower value than PageRank.

5.3 Final verdict

In conclusion, we personally prefer the PageRank algorithm as it is more consistent in both considered measures, although it is slightly performs worse than HITS, which instead has variable performance depending on the dataset on which it is run.