Project 3 NLA: PageRank implementations

Jaume Sánchez

December 2023

Contents

1 Introduction

This report corresponds to the study carried out solving the third Numerical Linear Algebra project. In this, you can find the explanation and analysis of the results of the practical part (code part).

The goal of this project is to code two different implementations of the PageRank algorithm, one of them storing the matrices that are being used and the other one without storing them.

2 A little explanation

Let us briefly introduce what we have delivered to the professor:

1. *Prac3_PageRank.py*: the only code file needed to answer the questions.
2. *p2p-Gnutella30.mtx*: the matrix in Sparse format that we will be using.

The matrix can also be downloaded from the following link: https://sparse.tamu.edu/SNAP/p2p-Gnutella30.

3 PageRank algorithm

In this exercise we will read a Sparse matrix and then apply the PageRank algorithm. Let us first explain the functions that we have created:

1. *D_matrix(G)*: given a (link) matrix G in a sparse format, we compute the out-degree of the page j and then we return a diagonal matrix D that satisfies $A = GD$.
2. *PR1(A, m,tol)*: given a matrix A it computes the PR vector of $(1-m)A + mS$ using the power method storing the matrices.
3. *PR2(G, m,tol)*: given a (link) matrix G it computes the PR vector of $(1-m)A + mS$ using the power method without storing the matrices.

3.1 Explanation

Recall that the PageRank problem consists on determining the importance of web pages within a hyperlinked set by assigning numerical scores based on the quantity and quality of links.

In our specific case, we are given a link matrix $G = (g_{ij})$, satisfying that g_{ij} is either 0 or 1 according to the existence or not of link between the pages i and j. Taking into account the the out-degree of page j, $(n_j = \sum_i g_{ij})$, and defining $D = diag(d_{11}, ..., d_{nn})$ where $d_{jj} = \frac{1}{n_j}$ if $n_j \neq 0$ and $d_{jj} = 0$ otherwise, we have that $A = GD$. This new matrix A, together with the equation $x = Ax$, summarizes the page rank score of a page k as $x_k = \sum_{j \in L_k} \frac{x_j}{n_j}$ where L_k is the number of webpages with link to page k.

In order to avoid several issues (such as disconnected networks or dangling nodes), we create the following matrix: $M_m = (1-m)A + ez^t$, where $m \in [0, 1]$, $e = (1, \ldots, 1)$, and z_j will be $\frac{m}{n}$ if the column j of the matrix A contains non-zero elements and $\frac{1}{n}$ otherwise. The parameter m, known as damping factor, represents the probability of teleportation or the likelihood that a user randomly jumps to any web page instead of following a link. This new matrix has the desirable properties, that is, is column stochastic and has a unique PR vector, therefore we will work from now on with this matrix.

With this being said, we apply the power method to the matrix M_m that reduces to iterate $x_{k+1} = (1-m)Ax_k + ez^t x_k$ until $||x_{k+1} - x_k||_\infty <$tol. This question is answered with the function $PR1(A, m, tol)$. Note that in the way that this function is written we are storing all the matrices, and that, for a large dataset, could lead to a problem of memory requirements exceeding the available resources. With that intention, we explode the information that we have in the link matrix G and answer the same question than before but without storing all the matrices. This question is answered with the function $PR2(G, m, tol)$.

3.2 Conclusions

Since the power method is an iterative method, it needs a first point from which to start. In both implementations we have started with the point $x_0 = (\frac{1}{n}, \ldots, \frac{1}{n})$, and we think that this is a reasonable starting point because, in the absence of any prior information, each page is assumed to be equally likely to be the starting point of a random user. Setting the tolerance to 10^{-12} we obtain the following results:

Method	Execution time (s)	Iterations
Storing matrices	0.06	60
Not storing matrices	13.82	60

Table 1: Results with $tol = 10^{-12}$.

The TOP 10 PR score correspond to the following pages:

[31803 31366 24973 9475 29641 12684 19063 31548 36465 33103]

with respectively scores:

[0.00144183 0.00132586 0.00126311 0.00111618 0.00110338 0.00110117 0.00096342 0.0009605 0.00094396 0.00093449]

Note that both methods, as expected, give us the same order. Also, the difference between the two output vectors is minimum, more concretely is approximately of the order of 10^{-11}.

Before commenting on the results, let us see what happens if we change the tolerance, for example, to 10^{-16}.

Method	Execution time (s)	Iterations
Storing matrices	0.09	88
Not storing matrices	18.51	88

Table 2: Results with $tol = 10^{-16}$.

Again we obtain the same TOP 10 PR scores and orders (not only the first 10) and now, as expected, the difference between output vectors is approximately of the order of 10^{-15}. So, changing the tolerance does not present drastic changes in the output (because we already new that it would take longer to execute and that more iterations were going to be necessary), as long as an appropriate tolerance is chosen.

Therefore, and as both methods need the same number of iterations, we can conclude that the first method, storing the matrices, although needing more memory it is faster than the second one (a big difference, around 200 times). For the second method, without storing matrices, it is slower than the first one but it doesn't use such memory as the first method. Hence, choosing one or the other method will depends in our preferences, that is, on using less memory (second option) or spending less time (first option).

4 Dificulties

I would say that at the beginning it was a little bit tricky to work with the Sparse matrices; but once the libraries and the functions needed to work with them were found the exercises were more easygoing. It is also worth mentioning that the second implementation was almost all explained (and the code was provided) in the wording of the project, and hence it facilitated the understanding of the algorithm.