Report: Page Rank Algorithm with MPI

Contents

[Explanation of files included in project 1](#_Toc448066075)

[The Page Rank Algorithm 1](#_Toc448066076)

[Introduction 1](#_Toc448066077)

[The equation 1](#_Toc448066078)

[Sequential algorithm 2](#_Toc448066079)

[Parallel page rank algorithm explanation 3](#_Toc448066080)

[Introduction 3](#_Toc448066081)

[Matrix vector product parallelization 3](#_Toc448066082)

[Problem initialization and matrix representation 3](#_Toc448066083)

[Master/slave structure. 3](#_Toc448066084)

[Scattering the initial matrix 4](#_Toc448066085)

[The loop 4](#_Toc448066086)

[Loop synchronization 4](#_Toc448066087)

[The slave processers 4](#_Toc448066088)

[Assembling the global result 4](#_Toc448066089)

[Memory management 4](#_Toc448066090)

[Application and interpretation 4](#_Toc448066091)

[Building M 4](#_Toc448066092)

# Explanation of files included in project

pageRankMpi.c: contains the source code of the page rage algorithm adapted to parallel computing. Takes as input a matrix in csv format and outputs the result vector in csv.

Csvparser.c, csvwriter.c: libraries (found on sourceforge) for reading and writing csv files

Example\_test\_matrix.csv /.xlsx: Represents the input matrix mapping the linking structure if [www.financeutile.com](http://www.financeutile.com). Generated using python code (not included) with the scrapy library to crawl the website and parse the pages.

Functions1d.c: contains auxiliary functions, mainly for matrix calculations (generating matrices, calculating norms, matrix addition/multiplication, copying matrices…)

Pagerank.csv: the results generated by the algorithm for [www.financeutile.com](http://www.financeutile.com) (i.e. using example\_test\_matrix.csv as input)

# The Page Rank Algorithm

## Introduction

Page rank is an algorithm used by Google Search to rank websites in their search results. The algorithm is based on the links between webpages. It attributes a score to every website based on the number of incoming links from other pages.

For more detail: see <https://en.wikipedia.org/wiki/PageRank>

## The equation

The calculation is as follows:

R is the vector containing the page ranks of the webpages (p1, p2, …pN)  
d is a damping factor  
N is the total number of pages  
l(pi,pj) is a normalized coefficient that represents a link between pj and pi

Solving this equation for R gives us the page rank of all webpages.

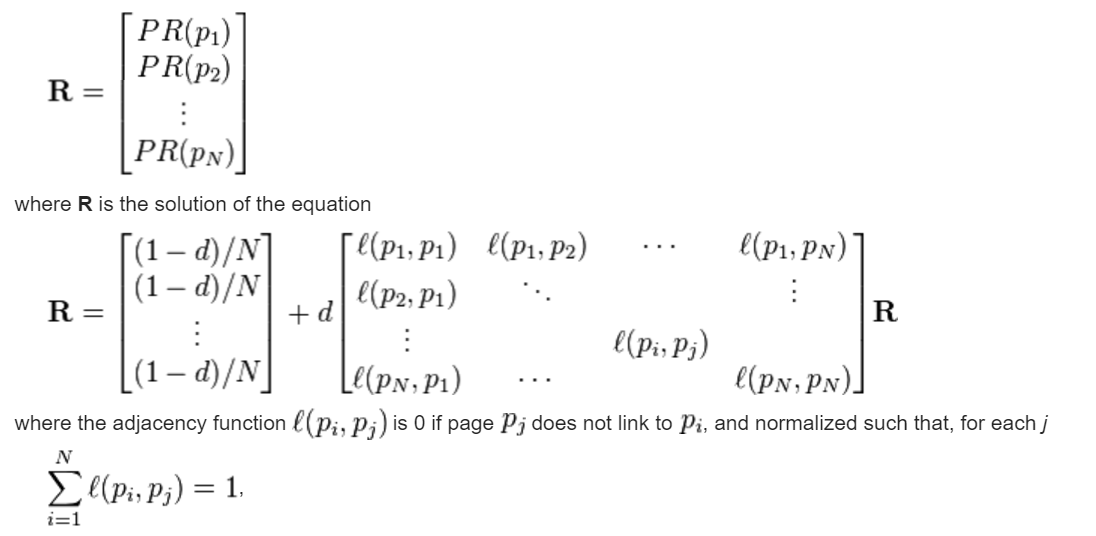
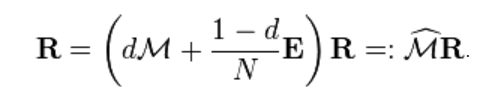


Figure : The Equation to Solve (source: wikipedia)

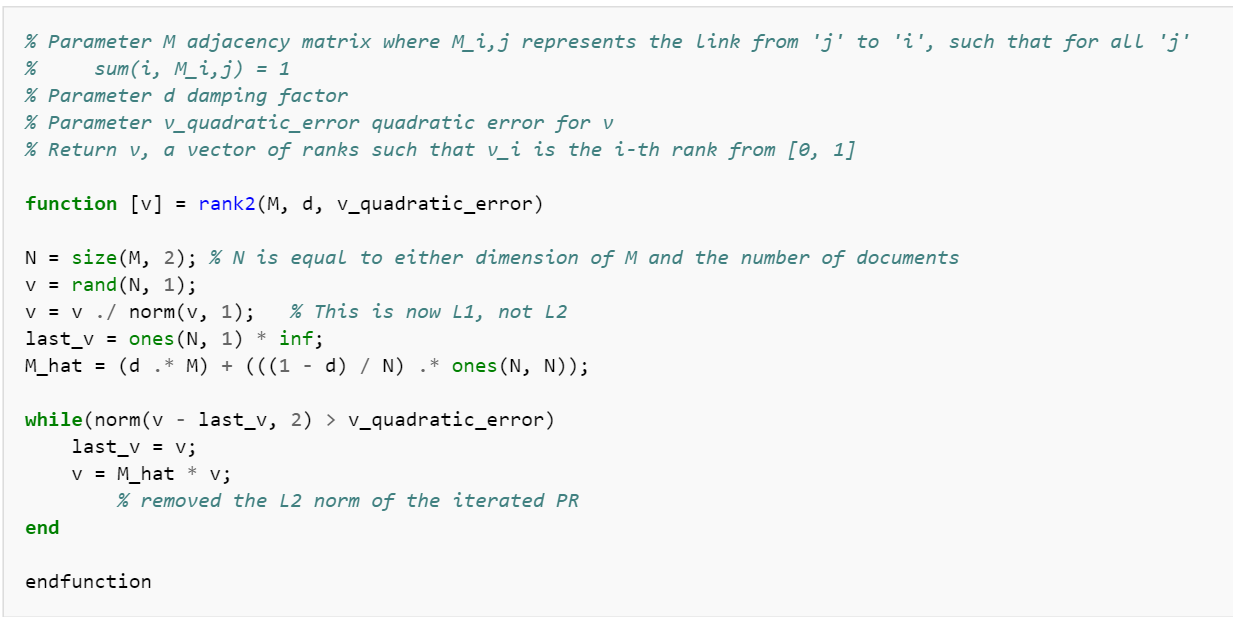
This equation can be written as:



Where M is column stochastic, R a probability distribution, and E the matrix consisting of ones. The page rank is thus the principal eigenvector of M\_hat. We will use **the power iteration** method to solve this eigenvector problem

## Sequential algorithm

The parallel algorithm used is based on the following sequential algorithm (taken from Wikipedia). This code uses the power iteration method to solve v = M\_hat \* v for v. The basic idea is that by taking a random v and multiplying it successively by M\_hat, the result will converge towards the principal eigenvector of M\_hat. For more information see: <https://en.wikipedia.org/wiki/Power_iteration>



# Parallel page rank algorithm explanation

## Introduction

The power iteration method involves successively multiplying v by M\_hat until convergence (as defined by v\_quadratic\_error). It is this matrix vector multiplication that we will parallise using MPI. As this operation is done many times during the algorithm, the time gained on large calculations will be substantial.

## Matrix vector product parallelization

The idea is to cut the matrix up into row bands. Each row band is then sent to a separate processor, which executes the local multiplication of the row band and the vector v to give a local result. These calculations are carried out in parallel. The local results are then reassembled to get the global result of the multiplication.

## Problem initialization and matrix representation

Matrices are represented by 1 dimensional arrays. This facilitates the sending of rows with MPI. Memory is allocated dynamically with malloc, and the coefficient types are doubles.

Functions contains basic matrix operations used for initializing the problem. These functions are all sequential. The initialization involves importing the matrix M, calculating M\_hat, and generating v.

### Master/slave structure.

The master (rank 0) is responsible for initialising the problem, cutting it up into different parts and distributing the work to the slave processers. It is also responsible for reassembling the different local results once they have been received from the slave processers. The slave processors simply take care of the local matrix vector products with the sub matrices received from the master. As such the code if divided into two parts: (rank == 0) and (rank !=0) i.e. master and slaves.

## Scattering the initial matrix

As the matrices are stored in one dimensional arrays, it is easy to send row bands to the slave processors. An Mpi\_send command starting at the first coefficient to send with the number of coefficients to send does the job nicely.

## The loop

The loop is controlled by the master process. It continues until the difference between v and last\_v (L2 norm) is smaller than v\_quadratic\_error. The loop consists of updating last\_v to v and of multiplying v by M\_hat. This multiplication is done in parallel. The new v is distributed to the slave processes in every iteration.

## Loop synchronization

The master and slaves each have their own while loop. The loop Boolean is used to synchronise the two. The master does the check of when to stop the loop (norm(v-last\_v)<v\_quadratic\_error), and then sends to message to the slaves to tell them to stop when needed.

## The slave processers

The processers are responsible for executing the local matrix vector multiplications. They received data from the master, execute the calculation and send the result back. These commands are executed in a loop until the master exits the loop.

## Assembling the global result

On every iteration, the master processes receives the local results from the slaves and assembles them into a global result. The local results are received in the order of the processers, making the reassembling relatively easy.

## Memory management

Variables are freed as soon as they are no longer needed. For example, M and E are freed after M\_hat has been calculated. M\_hat is freed once it has been divided into row bands and sent out to the individual processers. Note also that M\_hat, M and E are only allocated (and freed) in the master process.

## Limits

Currently p must divide N. This case is accounted for in matrixVectorProductmpi.c but has not yet been transposed to pagerankmpi.c

The vector v is stored completely in each process. The memory management could be optimized by using mpi message passing.

# Application and interpretation

Sebastien Baur and I have applied this algorithm to a real life application. We used the website [www.financeutile.com](http://www.financeutile.com) to demonstrate the algorithm and calculate the page rank of all of its pages (70 in total).

To do this, we crawled the website using the scrapy library for python and parsed the pages to extract the external links. This allowed us to build the adjacency matrix for the website. From this we were able to generate the matrix M used for the input to the algorithm.

The calculation and results can be found in example\_test\_matrix.xlsx

## Building M

M = (K-1A)T

Where K is the diagonal matrix with the outgoing degrees of each page as diagonal coefficients and A is the adjacency matrix of the website (1 if pi links to pj and 0 otherwise).

Results interpretation