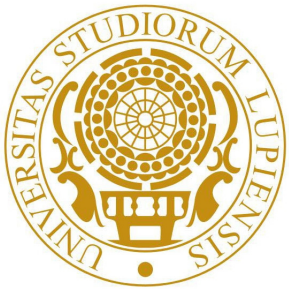
UNIVERSITÀ DEL SALENTO



Facoltà di Ingegneria

Corso di Laurea Magistrale in Computer Engineering

Advanced Control Techniques Project

Matlab implementation of Distributed Randomized Algorithms for the PageRank Computation

Professor: **Giuseppe Notarstefano**

Students: **Francesco Margiotta**
Marco Cervellera
Matteo Conte

Academic year 2016-2017

Abstract

In the search engine of Google, the PageRank algorithm plays a crucial role in ranking the search results. The algorithm quantifies the importance of each web page based on the link structure of the web. We first provide an overview of the original problem setup. Then, we propose several distributed randomized schemes for the computation of the PageRank, where the pages can locally update their values by communicating to those connected by links. The main objective of the paper is to show that these schemes asymptotically converge in the meansquare sense to the true PageRank values. A detailed discussion on the close relations to the multi-agent consensus problems is also given.

Contents

List of Figures

Introduction

Motivations

The PageRank algorithm quantifies the importance of each web page based on the link's structure of the whole network. In this way, an user can find most relevant information sources visiting the first results of a search. The main idea that allows PageRank to work is that links that point to a page from other pages considered important, make it more important. Practically each page "vote" the pages linked to it. This problem, is mathematically solved by calculating the eigenvector corresponding to the greatest eigenvalue of a stochastic matrix,a method known as Power method [3].This process is completed by a centralized algorithm which given the big dimension of the network (more than 8 billion of pages) takes a week to finish. In order to solve the problem,In the paper we consulted is proposed an approach based on randomized distributed algorithms. This algorithm is characterized by 3 important aspects:

1. Each page can calculate its PageRank locally communicating with the pages directly linked to it
2. Each page decides to start the communication with the others at a random time.
3. The workload requested to each page is very mild

This algorithm is related to the consensus theory in such way that some nodes exchange their values with their neighbours in order to reach the consensus i.e. all nodes reach the same value.

Contributions

Our work concerned the development of a Matlab simulation both of the centralized and distributed approach based on the mathematical formulation of the paper on which we have done our preliminary study [1].Our Matlab implementation computes the PageRank vector for the centralized case first, and then we compare it with the PageRank vector computed through a

distributed approach, noticing that this one converges to the PageRank vector of the centralized approach. The implementation is structured in a modular way,this allows to modify just few lines of codes of the main file in order to get a full functionally environment fitting a specific request based on the dimension of the network. //boh aggiungere altra roba nel caso

Chapter 1

The PageRank problem

1.1 The Centralized approach

Let's consider a network of n web pages indexed from 1 to n.The network is represented by a graph G = (V,E) where V is the set of nodes (web pages) and E is the set of edges (links between pages) which belongs to VxV. (i,j) is an edge if exists a link that lead from page i to page j. The PageRank of a page i is is a real number in [0,1] denoted by

$$\sum_{j \in L_i} \frac{x_j^*}{n_j} \tag{1.1}$$

where L_i is the set of the pages which has an entering link in i, is the number of links which are going out from the node j. In practice a page i has greater importance of a page j if $x_i^* > x_j^*$. It's a good practice to normalize the values such that $\sum_{i=1}^{N} x_i^* = 1$
Let's rewrite values in a vectorial form: $x^* \in [0,1]^n$. Using this notation we can rewrite the problem in the following form:

$$x^* = Ax^* \tag{1.2}$$

with $x^* \in [0,1]^n$ and $\sum_{i=1}^{N} x_i^* = 1$, where the matrix$A = (a_{ij})$ is called link matrix and their element are defined as:

$$a_{ij} = \begin{cases} \frac{1}{n_j} \; if \; j \in L_i \\ 0 \; otherwise \end{cases} \tag{1.3}$$

Notice that the vector x* is the eigenvector corresponding to the eigenvalue 1 of the matrix A. In general this eigenvector exists and is unique if the graph is strongly connected, i.e there exists a direct path that connects each node i, to another node j of the graph. Since the actual network is not strongly connected in the following we will assume that every page are

somehow connected for simplicity, in particular we assumed that each page has a link back for each page linked to it. So, the matrix A becomes column stochastic and will always exist the eigenvalue 1. To guarantee uniqueness of the eigenvalue 1 is introduced a modified version of the matrix A as stated in [1], defined as follows:

$$M := (1-m)A + \frac{m}{n}S \tag{1.4}$$

With m parameter with typical value of 0.15 and S matrix with all entries being 1. M is positive and stochastic and, for Perron Theorem, primitive; in particular the eigenvalue 1 is unique Using the M matrix defined in (1.4), we can redefine the update of the x^* vector, as follows:

$$x^* = Mx^* \; with \; x^* \in [0,1]^n \; and \; \sum_{i=1}^{n} x_i^* = 1 \tag{1.5}$$

As we told, compute the eigenvector for large dimension matrixes is complex, for this reason we will use the recursion based on the power method:

$$x(k+1) = Mx(k) = (1-m)Ax(k) + \frac{m}{n}\mathbf{1} \tag{1.6}$$

Where x(k) Is a probability vector and notice that $Sx(k) = \mathbf{1}$. The equation (1.6) is very important because allows us to use directly the sparse matrix A (with several zeros) and not the dense matrix M. Based on this method, it's ensured as stated in Lemma 2.3 of [1] that the vector x(k) converges always to the consensus value for k $\to \infty$.

1.2 A Distributed randomized approach

In this section, we recap the mathematical formulation of the PageRank problem, using a distributed approach to compute the value of the vector x^* as proposed in [1].The basic protocol of the scheme is the following: At time instant k, the page i starts the update of the PageRank value by sending its value to the other pages connected to it and by requesting the values from the pages that are linked to page i. In each time instant, only a single page can start the update process. In this case, the update scheme for the distributed approach, the update scheme is the following:

$$x(k+1) = Mx(k) = (1-\hat{m})A_{\theta(k)}x(k) + \frac{\hat{m}}{n}\mathbf{1} \tag{1.7}$$

With A_i distributed link matrices computed as described in the next section, while $\hat{m}$ parameter computed using the formula:

$$\hat{m} = \frac{2m}{n - m(n-2)} \tag{1.8}$$

with m = 0.15 as usual. Defining the time average of the state x as

$$y(k) := \frac{1}{k+1} \sum_{l=0}^{k} x(l) \tag{1.9}$$

we say that for the distributed update scheme,the time average y(k) converge to the consensus value x^* in the mean square sense i.e. $E[||y(k) - x^*||^2] \to 0$ for k $\to \infty$

1.2.1 Distributed link matrices

We define the distributed link matrices A_i i in such way that:

- The i-th row and column is equal to the same row and column of A
- The elements on the diagonal are $1 - a_{il}$ if l=1...n and $l \neq i$
- All the other elements are 0.

Formally

$$(A_i)_{jl} = \begin{cases} a_{jl}, & if\ j = i\ or\ l = i \\ 1 - a_{il},\ if j = l \neq i \\ 0, & otherwise \end{cases} \tag{1.10}$$

These matrices are stochastic because the original matrix is stochastic. So, as we did in the centralized case,we avoid the possible non-unicity of the eigenvalue 1 from matrix A by writing the distributed update scheme as:

$$x(k+1) = M_{\theta(k)} x(k) \tag{1.11}$$

Where M_i is computed as

$$M_i := (1 - \hat{m}) A_i + \frac{\hat{m}}{n} S \tag{1.12}$$

We conclude this section enunciating the main result for the distributed approach, stated in [1] that guarantee the convergence of the algorithm to the true PageRank value, in the mean square sense.

Theorem In the distributed update scheme (1.11), the PageRank value x^* is obtained through the time average y in (1.9) as $E[||y(k) - x^*||^2] \to 0$ for k $\to \infty$.

Chapter 2

Matlab Implementation and tests

2.1 Project structure

The work has been organized in a modular way such that each component is a function that accomplish a specific job. We developed the following function:

1. edgeToCentralized: A function that receive as a parameter a vector of edges and use it to compute a weighted column stochastic adjency matrix A by using the method the definitions used in the paper
2. computeM: this is a function that apply some calculations on A in order to get an output matrix M with the same behaviour of A but with the guardantee of having the eigenvector associated to the eigenvalue 1 unique.
3. pageRankCentralized: take the matrix M as a parameter and use the power method to compute the pagerank value associated to the network described by M
4. in the main function we operate some operations on the edge vector E in order to simulate the exchange of informations between agents, each agent receive a reduced set of informations composed by the in-neightbours, the out-neightbours and the out-neightbours grade of it's whole neightbourhood
5. edgeToDistributed: use the reduced the of information to build the adjency matrix for each agent
6. PageRankDistributed: compute the PageRank value for the distributed scheme simlating the random activation of each agent using a simulated inner timer

In the following sections, we describe in depth how these functions work both in centralized and distributed case.

2.1.1 Generation of a random digraph

The first task we accomplished, was to build a function that allows us to generate at run-time a random strongly connected network defined through a vector of edges of the dimension $\mathbb{R}^{2*N}$ in which N is the desired number of nodes. randomEdgesGenerator is the function that does this job. This function takes as argument a number of nodes and a seed necessary to setup the random functions used in the script. The file is composed by two principal parts

- The first while cycle is used to generate a random Adjacency matrix in which each entry a_{ij} is 1 with a probability of 0.2. This is done thank's to the binornd function, as shown in the following piece of code: *Adj = binornd(1,0.2,NN,NN).* After that, it's necessary to check if the random matrix generated corresponds to a connected graph, and once we have this condition we have a break from this cycle.

- The second cycle is used to build the edge vector from the Adjacency matrix computed in the previous cycle. It simply reads the matrix element by element adding the couple of indexes in the edge vector if the correspondent element is 1. After each row traversal is added a return link from a random node to the node whose index correspond to the current iteration, as shown in the following code:

```
randomReturn = randi(NN, 1);
    while ( randomReturn == i)
        randomReturn = randi(NN, 1);
    end
    edgesProgress = [edgesProgress ; [ randomReturn , i ]];
end
```

2.1.2 Implementation of centralized Pagerank

2.1.3 Implementation of distributed Pagerank

Citation [2]

Conclusions

Bibliography

[1] Hideaki Hishi and Roberto Tempo. Distributed randomized algorithms for the pagerank computation.

[2] Mario Rossi and Giulio Bianchi. Just an example. *La Repubblica*, 2011.

[3] Wikipedia. Power iteration.