Fully distributed PageRank computation with exponential convergence

Liang Dai and Nikolaos M. Freris[1]

***Abstract*— This work studies a fully distributed algorithm for computing the PageRank vector, which is inspired by the Matching Pursuit and features: 1) fully distributed 2) expected converges with exponential rate 3) low storage requirement (two scalar values per page). Illustrative experiments are conducted to verify the findings.**

I. Problem Statement

PageRank vector was proposed by the founders of the Google to quantify the importance rankings of the webpages of the Internet [1], [3]. Due to the generality of the idea, PageRank has been extended to application in Biology, Chemistry and some other domains, more details can be found in the review paper [4].

Suppose there are N pages in a network. The connectivity (i.e., the topology induced by the hyperlinks present in websites) can be characterized by the hyperlink matrix $A \in R^{N\times N}$ defined as follows: its (i,j)-th element is $\frac{1}{N_j}$, if there is a link from page-j to page-i, where N_j denotes the number of outgoing links of page j (the number of pages which the page j points to); otherwise $A_{i,j}=0$. By construction A is a non-negative, column stochastic matrix (i.e., a matrix with non-negative elements and each column summing up to one). In this work, we assume without any loss of generality that there are no dangling pages (i.e. pages with no outgoing pages), i.e., A has no zero columns.

One potential choice for the PageRank vector is the normalized principal eigenvector (with all the elements summing up to one) of matrix A. However, one drawback of such choice is that, when the network is not fully connected, the principal eigenvector of matrix A may not be unique.

To overcome this problem, a 'perturbed' version of A is adopted for defining the PageRank vector, given by

$$M = \alpha A + (1-\alpha)S,$$

where $S = \frac{1}{N}\mathbf{1}\mathbf{1}^T$, with $\mathbf{1} = [1,1,\cdots,1]^T \in R^{N\times 1}$ and $\alpha \in (0,1)$. The suggested value for α is 0.85 [1]. The original PageRank vector is defined as:

Definition 1 (PageRank): Given the perturbed hyperlink matrix M, the vector $\mathbf{x}^*$ is the unique vector $\mathbf{x}^*$ that satisfies:

1) $M\mathbf{x}^* = \mathbf{x}^*$,
2) $\sum_{i=1}^N \mathbf{x}_i^* = 1$ and $\mathbf{x}^* \geq 0$.

Note that since M is a positive, column stochastic, and irreducible matrix, the Perron-Frobenius Theorem [5] guarantees existence and uniqueness of a positive right-eigenvector with corresponding eigenvalue equal to 1, which is precisely $\mathbf{x}^*$. The second property is simply a normalization of its entries to sum up to one.

It is plain to see that the ranking will not be affected by a positive rescaling of the PageRank vector. In our work, we will adopt the following (positively rescaled by the network size N) scaled PageRank vector. The main advantage of this choice is that computations will not involve the network size N and will be made clear in the sequel.

Definition 2 (Scaled-PageRank): Given the perturbed hyperlink matrix M, the scaled PageRank vector $\mathbf{x}^*$ is the vector which satisfies:

1) $M\mathbf{x}^* = \mathbf{x}^*$,
2) $\sum_{i=1}^N \mathbf{x}_i^* = N$ and $\mathbf{x}^* \geq 0$.

As the internet is of huge scale, it becomes very difficult to save the entire matrix M and solve $M\mathbf{x} = \mathbf{x}$ in a single machine. This is performed by Google on a regular basis using the centralized power iteration [3] which requires large storage and computational power. Additionally, a change in matrix M (for example the creation or deletion of a website or changes in the hyperlinks present in a page) typically entails re-computation of the PageRank vector from scratch.

To overcome the difficulties, several distributed methods (where each page updates its PageRank value by exchanging the information only with neighbouring pages, i.e., pages that it links to or pages that link to it) have been suggested for this problem. Based on the idea of Monte Carlo simulations, [9] proposed the following approach: starting from each node, the algorithm performs multiple rounds of random walks via certain absorbing Markov chains, and PageRank vector is estimated by the frequency of visits to this node from all the random walks. The method features fast convergence as well as distributed implementation, however, the simultaneous runs of a large number of random walks may lead to the problem of congestion in the network. In the following, we will focus on reviewing the ideas based on linear algebraic techniques. In [6], a randomized distributed algorithm was proposed based on stochastic power iterations together with the Polyak averaging scheme. Recently, based on an application of the Stochastic Approximation (SA) framework [13], a randomized distributed algorithm was designed [12]. Nonetheless, in both [6] and [12], during each update, a webpage needs to request information from its incoming neighbours (i.e. the set of webpages that link to it), which might impose practical limitations in that: 1) it either requires additional storage of a list of incoming neighbours, which sometimes could be of huge size; 2) or it might incur delays (for example, wait till all the information has

[1]L. Dai and N. Freris are with the Department of Information Technology of Halmstad Unviersity, Halmstad, P.O. Box 823, Sweden and the Engineering Division of New York University Abu Dhabi, Saadiyat Island, P.O. Box 129188, UAE, respectively. E-mail: liadai@hh.se,nf47@nyu.edu

been transimitted) in obtaining the values from the incoming neighbours. Furthermore, the approaches in [6] and [12] are of (or can be reformulated as) SA-type algorithms, which feature sub-exponential convergence rate [14], [12]. In [15], a randomized incremental optimization based distributed algorithm was proposed: nonetheless, similarly to the work in [6] and [12], information from in-coming pages are required for the algorithm's updates.

In this work, seeking to overcome these issues, we propose a fully distributed algorithm (in which updating webpages only use the PageRank values of outgoing pages, while also no knowledge of the network size is required) with provable exponential convergence (in expectation). From a signal decomposition point of view the proposed method can be seen as randomized Matching Pursuit algorithm. The main attributes of the new algorithm are:

1) It uses only the knowledge of the out-going webpages and no knowledge of the network size is assumed;
2) It converges exponentially fast, in expectation;
3) It only requires storing two scalar values per webpage (the PageRank estimate along with a residual value, explicated below).

II. The proposed algorithm

In the following, $U[m,n]$ will be used to denote the uniform sampling of a natural number between m and n. Conventions in *Matlab* will be used to denote the rows and columns of a matrix. I, $\mathbf{1}$ and $\mathbf{0}$ denote the identity matrix, all-one vector and all-zero vector respectively, where dimension will be made clear from the context. $\mathbf{e}_k$ denotes the k-th unit vector (1 in k-th entry and 0 elsewhere), while $\|\cdot\|$ represents the l_2 norm of a vector.

A. Problem Reformulation

Substituting M to the definition of PageRank vector in Definition 2, and using the property of matrix S that $S\mathbf{x}=1$ for any $\mathbf{x}$ with $\sum_i x_i = 1$, we have the following equivalent characterization of the scaled PageRank vector

$$\begin{cases} (I-\alpha A)\mathbf{x}^* = (1-\alpha)\mathbf{1}, & \text{(1a)} \\ \sum_{i=1}^{N} \mathbf{x}_i^* = N \text{ and } \mathbf{x}^* \geq 0. & \text{(1b)} \end{cases}$$

From (1a), we get the vector

$$\hat{\mathbf{x}} = (1-\alpha)(I-\alpha A)^{-1}\mathbf{1}. \quad (2)$$

If we can further establish that all the elements of vector $\hat{\mathbf{x}}$ are nonnegative and summing up to one, then $\hat{\mathbf{x}}$ will be the PageRank vector as in the Definition 2. The following proposition confirms this.

Proposition 1: The scaled PageRank vector is given as

$$\mathbf{x}^* = (1-\alpha)(I-\alpha A)^{-1}\mathbf{1}. \quad (3)$$

Proof: From the Gershgorin circle theorem [5], the eigenvalues of A all have magnitudes in $[0,1]$, which implies that $I-\alpha A$ is invertible (since $0<\alpha<1$) and its inverse is given by

$$(I-\alpha A)^{-1} = \sum_{k=0}^{\infty} \alpha^k A^k. \quad (4)$$

Right-multiplying by $\mathbf{1}$ and using the non-negativity of A, we get that $\mathbf{x}^* > 0$. We are left to verify that

$$\mathbf{1}^T(1-\alpha)(I-\alpha A)^{-1}\mathbf{1} = N. \quad (5)$$

For any $k \geq 0$, since A is a column stochastic matrix, we have that $\mathbf{1}^T A^k \mathbf{1} = N$, thereby

$$\begin{aligned} \mathbf{1}^T(1-\alpha)(I-\alpha A)^{-1}\mathbf{1} &= (1-\alpha)\left(\sum_{k=0}^{\infty}\alpha^k \mathbf{1}^T A^k \mathbf{1}\right) \\ &= N(1-\alpha)\sum_{k=0}^{\infty}\alpha^k = N, \end{aligned}$$

which concludes the proof. ■

B. The algorithmic description

Following Proposition 1, we can find the scaled PageRank vector by solving the system of linear equations

$$(I-\alpha A)\mathbf{x}^* = (1-\alpha)\mathbf{1}. \quad (6)$$

Note that there is no more dependency on network size N, which means that if we design a distributed solver for (6) it will be fully distributed. We will take a signal processing point of view to solve this equation: we regard the columns of matrix $I-\alpha A$ as the atoms (or the basis) of a dictionary, and we are left to find the representation (or decomposition) of vector $(1-\alpha)\mathbf{1}$ using these atoms. In our case, this representation will be unique. To simplify notations, we let

$$B \triangleq I - \alpha A,$$

and

$$\mathbf{y} \triangleq (1-\alpha)\mathbf{1}$$

The Matching Pursuit (MP) algorithm in [2] has become a standard tool to find signal representations under a given set of atoms (termed as dictionary, which is often over-complete). In brief, the MP algorithm is an iterative algorithm: at each iteration, it identifies the "best matching" atom with the signal residual, and subsequently updates the residual by subtracting its projection to this "best matching" atom. In our case, even though the dictionary, i.e. the matrix B, is not over-complete, we can still apply the MP algorithm to find the unique representation of vector $(1-\alpha)\mathbf{1}$ using the columns (the atoms) of B.

However, the 'best matching' step in MP algorithm is not amendable to a distributed implementation, as it requires searching all columns, i.e. all webpages in our case of interest. To address this issue, we modify the original MP algorithm via randomization: instead of picking the 'best matching' atom at each iteration, the proposed algorithm picks a random atom from the dictionary and perform the projection step. By doing so, the derived algorithm can be implemented in a fully distributed fashion. In addition, we

shall show that the algorithm will converge exponentially fast (in expectation, since we introduce randomization) to the desired solution. The detailed algorithmic description is given in Algorithm 1.

Algorithm 1 Matching Pursuit based PageRank Computation

Initialization:
Initialize vectors $\mathbf{r}_0, \mathbf{x}_0 \in R^N$ as $\mathbf{y}$ and $\mathbf{0}$ respectively.
Iterations:
for $t = 0, 1, \cdots, T-1$ **do**
Generate $k = U[1, N]$, and update:

$$\mathbf{x}_{t+1} = \mathbf{x}_t + \frac{B(:,k)^T \mathbf{r}_t}{\|B(:,k)\|^2} \mathbf{e}_k \tag{7}$$

$$\mathbf{r}_{t+1} = \mathbf{r}_t - \frac{B(:,k)^T \mathbf{r}_t}{\|B(:,k)\|^2} B(:,k) \tag{8}$$

end for
Return:
Return $\mathbf{x}_T$.

Remark 1: A fully asynchronous scheme (i.e. the 'exponential clocks' approach) to implement the uniform (or more general) sampling in Algorithm 1 can be found in the 'Implementation Issues' section of [16], and the references therein.

Remark 2: The sequences of $\{\mathbf{x}_t\}_{t=0}^{\infty}$ and $\{\mathbf{r}_t\}_{t=0}^{\infty}$ in Algorithm 1 keep track of the approximations to $\mathbf{x}^*$ and the signal residuals, respectively. Note that the increment $\mathbf{x}_{t+1} - \mathbf{x}_t$ in equation (7) can be obtained by solving the following coordinate descent optimization problem:

$$\min_{\Delta} \|B\mathbf{x}_{t+1} - \mathbf{y}\|^2$$
$$\text{s.t. } \mathbf{x}_{t+1} = \mathbf{x}_t + \Delta \mathbf{e}_k.$$

C. Convergence analysis

Before proceeding further, we define some auxiliary quantities. For each $1 \leq k \leq N$, let

$$\mathbf{b}_k = \frac{B(:,k)}{\|B(:,k)\|},$$

and

$$\hat{B} = [\mathbf{b}_1, \cdots, \mathbf{b}_N], \; P_k = \mathbf{b}_k \mathbf{b}_k^T,$$

it follows that $\hat{B}\hat{B}^T = \sum_{k=1}^N P_k$.

Notice that the equation (8) can be rewritten as follows

$$\mathbf{r}_{t+1} = (I - P_k)\mathbf{r}_t,$$

which gives that

$$\|\mathbf{r}_{t+1}\|^2 = \mathbf{r}_{t+1}^T \mathbf{r}_{t+1} = \mathbf{r}_t^T (I - P_k)\mathbf{r}_t.$$

Conditioned on $\mathbf{r}_t$, we have that

$$\begin{aligned}\mathbb{E}[\|\mathbf{r}_{t+1}\|^2 | \mathbf{r}_t] =& \frac{1}{N} \sum_{k=1}^N \mathbf{r}_t^T (I - P_k)\mathbf{r}_t \\ =& \mathbf{r}_t^T \left(I - \frac{1}{N} \hat{B}\hat{B}^T \right) \mathbf{r}_t\end{aligned}$$

Since B is a full rank square matrix, so $\hat{B}$ as well. This implies that $\sigma(\hat{B})$, the smallest singular value of $\hat{B}$, is nonzero. Using

$$\frac{1}{N} \hat{B}\hat{B}^T \succeq \frac{\sigma^2(\hat{B})}{N} I,$$

we have

$$\mathbb{E}[\|\mathbf{r}_{t+1}\|^2 | \mathbf{r}_t] \leq \left(1 - \frac{\sigma^2(\hat{B})}{N} \right) \|\mathbf{r}_t\|^2.$$

Iterating this equation, we get

$$\mathbb{E}\|\mathbf{r}_t\|^2 \leq \left(1 - \frac{\sigma^2(\hat{B})}{N} \right)^t \|\mathbf{r}_0\|^2, \tag{9}$$

for any $t \geq 0$, which establishes the exponential decay of the squared norm of the signal residual.

A useful property of Algorithm 1, which will be useful for establishing Proposition 2, is that during the run of the algorithm the vector $B\mathbf{x}_t + \mathbf{r}_t$ is always kept constant. To see this, multiplying both sides of equation (7) by matrix B gives that

$$B\mathbf{x}_{t+1} = B\mathbf{x}_t + \frac{B(:,k)^T \mathbf{r}_t}{\|B(:,k)\|^2} B\mathbf{e}_k,$$

which simplifies to

$$B\mathbf{x}_{t+1} = B\mathbf{x}_t + \frac{B(:,k)^T \mathbf{r}_t}{\|B(:,k)\|^2} B(:,k). \tag{10}$$

Adding equation (10) and equation (8), we have that

$$B\mathbf{x}_{t+1} + \mathbf{r}_{t+1} = B\mathbf{x}_t + \mathbf{r}_t,$$

which gives that

$$B\mathbf{x}_t + \mathbf{r}_t = \mathbf{r}_0 = \mathbf{y} \tag{11}$$

for any $t \geq 0$.

Summarizing the conservation property in equation (11) and the exponential decreasing fact in equation (9) gives:

Proposition 2: The vector sequence $\{\mathbf{x}_t\}_{t=0}^{\infty}$ converges to the scaled PageRank vector $\mathbf{x}^*$ (in Definition 2) with the expected exponential rate. In specific, for all $t \geq 0$

$$\mathbb{E}\|\mathbf{x}_t - \mathbf{x}^*\|^2 \leq \sigma^{-2}(\hat{B}) \|\mathbf{r}_0\|^2 \left(1 - \frac{\sigma^2(\hat{B})}{N} \right)^t. \tag{12}$$

Proof: According to Algorithm 1, we have that $\mathbf{r}_0 = B\mathbf{x}^*$. Substituting it into equation (11) and rearranging the terms, we obtain

$$B(\mathbf{x}_t - \mathbf{x}^*) = \mathbf{r}_t.$$

From (9), it follows that

$$\|B(\mathbf{x}_t - \mathbf{x}^*)\|^2 \leq \|\mathbf{r}_0\|^2 \left(1 - \frac{\sigma^2(\hat{B})}{N} \right)^t.$$

Additionally, the definition of the smallest singular value gives that

$$\|B(\mathbf{x}_t - \mathbf{x}^*)\|^2 \geq \sigma^2(\hat{B}) \|\mathbf{x}_t - \mathbf{x}^*\|^2,$$

which concludes the proof. ■

D. Distributed implementation

In this section, for a vector $\mathbf{c}_i$, we use $\mathbf{c}_{i,j}$ to denote its j-th element. For web page k, the indices of its outgoing pages are denoted as

$$\mathcal{N}_k = \{n_1, \cdots, n_{N_k}\},$$

where N_k is the number of outgoing pages.

We first show the distributed implementation of equation (7). Since only the k-th element of $\mathbf{e}_k$ is nonzero, to update $\mathbf{x}_{t+1}$, we only need to update $\mathbf{x}_{t+1,k}$ (the value of the currently triggered page k) according to

$$\mathbf{x}_{t+1,k} = \mathbf{x}_{t,k} + \frac{B(:,k)^T \mathbf{r}_t}{\|B(:,k)\|^2}, \tag{13}$$

and $\mathbf{x}_{t+1,j} = \mathbf{x}_{t,j}$ for all $j \neq k$ and $1 \leq j \leq N$ (all other pages keep their previous estimates).

The numerator in (13) is

$$\begin{aligned} B(:,k)^T \mathbf{r}_t =& (\mathbf{e}_k - \alpha A(:,k))^T \mathbf{r}_t \\ =& \mathbf{r}_{t,k} - \alpha \frac{\sum_{j=1}^{N_k} \mathbf{r}_{t,n_j}}{N_k}, \end{aligned}$$

which can be computed by *reading* the residual values of $\{\mathbf{r}_{t,n_j}\}_{j=1}^{N_k}$ from all the outgoing neighbours of page k.

The denominator in (13) is given as

$$\begin{aligned} \|B(:,k)\|^2 =& (\mathbf{e}_k - \alpha A(:,k))^T (\mathbf{e}_k - \alpha A(:,k)) \\ =& 1 - 2\alpha A_{k,k} + \alpha^2 \|A(:,k)\|^2, \\ =& 1 - 2\alpha A_{k,k} + \frac{\alpha^2}{N_k}, \end{aligned}$$

which can be computed by knowing the local information N_k and $A_{k,k}$. Note that $A_{k,k} = 0$ if the page k does not link to itself and $A_{k,k} = \frac{1}{N_k}$ otherwise.

Remark 3: $\{\|B(:,k)\|^2\}_{k=1}^N$ can be calculated in a pre-processing step to avoid recalculation at every iteration.

Next, we show distributed computation of equation (8). Note that, we have just shown distributed computation for the values $B(:,k)^T \mathbf{r}_t$ and $\|B(:,k)\|^2$. Therefore, we have: for each $n_j \in \mathcal{N}_k$ and $n_j \neq k$

$$\begin{aligned} \mathbf{r}_{t+1,n_j} =& \mathbf{r}_{t,n_j} - \frac{B(:,k)^T \mathbf{r}_t}{\|B(:,k)\|^2} (-\frac{\alpha}{N_k}) \\ =& \mathbf{r}_{t,n_j} + \frac{\alpha}{N_k} \frac{N_k \mathbf{r}_{t,k} - \alpha \sum_{j=1}^{N_k} \mathbf{r}_{t,n_j}}{N_k + \alpha^2 - 2\alpha N_k A_{k,k}}, \end{aligned}$$

which can also be computed solely using information from page k's outgoing links and its local information.

If $k \in \mathcal{N}_k$, i.e., page k has a link to itself, the update of $\mathbf{r}_{t+1,k}$ is given as follows

$$\begin{aligned} \mathbf{r}_{t+1,k} =& \mathbf{r}_{t,k} - \frac{B(:,k)^T \mathbf{r}_t}{\|B(:,k)\|^2} (1 - \frac{\alpha}{N_k}) \\ =& \mathbf{r}_{t,k} - (1 - \frac{\alpha}{N_k}) \frac{N_k \mathbf{r}_{t,k} - \alpha \sum_{j=1}^{N_k} \mathbf{r}_{t,n_j}}{N_k + \alpha^2 - 2\alpha}, \end{aligned}$$

otherwise

$$\mathbf{r}_{t+1,k} = \mathbf{r}_{t,k} \text{ for } j \notin \mathcal{N}_k,$$

which are all distributed implementable as well.

To summary, our distributed implementation picks a single page at each iteration, reads the residuals from its outgoing pages, and updates selected page's PageRank estimate, as well as the residuals of the outgoing pages. Therefore, at each iteration, the number of 'reads' and 'writes' is exactly equal to the number of outgoing webpages of the selected webpage.

III. Illustrative Experiments

In this part, we will conduct one synthesised example to verify the findings in the previous sections. The hyperlink matrix A is generated as follows: We first generate a $N \times N (N = 100)$ random matrix, with entries i.i.d. generated following a uniform distribution in $[0, 1]$. Each element is then thresholded with a given constant, which is set to be 0.5 in this experiment. The α is chosen as 0.85. To illustrate the exponential decreasing result in Proposition 2, we run 100 rounds of simulations and then average them. The method is compared with the method in [6] (initialized with an all one vector) and the method in [15] (initialized with a zero vector). The results are reported in Figure 1, and the analysis is given in its caption.

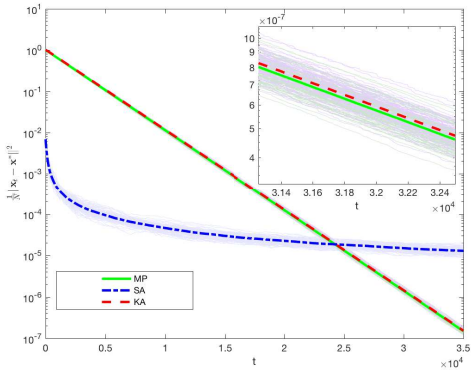

Fig. 1. This figure illustrates the trajectories of $\frac{1}{N}\|\mathbf{x}_t - \mathbf{x}^*\|^2$ as well as their averaged trajectories. The solid green and dot red lines represent the averaged trajectory for the proposed Matching-Pursuit based method and the method in [15], which decreases exponentially with a similar rate; the dash-dot blue line shows the averaged trajectory for the work in [6], which decreases sub-exponentially. Also note that the variance of the trajectories of method in [6] is large than the trajectories obtained by the other two approaches.

IV. Discussions

In this note, we proposed a fully distributed scheme to compute the PageRank vector. The method activates one random webpage at each step, then communicates and updates information with its outgoing webpages. It also converges fast with an exponential speed in expectation. There are several questions for future work: 1) parallelization of the algorithm; 2) generalization to a dynamic network setting; 3) improvement by a non-uniform sampling; 4) stopping criteria: when the iterations can be terminated to certify a correct ranking.

V. Appendix – Network size estimation

We make the additional assumption that network is fully connected. Note that $\mathbf{s} = \frac{1}{N}\mathbf{1} \in R^N$ is the eigenvector of matrix A^T corresponding to its principal eigenvalue (which is 1), hence $A^T\mathbf{s} = \mathbf{s}$, i.e.

$$(I - A)^T\mathbf{s} = 0.$$

Let $C \triangleq (I - A)^T$, it is evident that $C\mathbf{s} = 0$ and its nullspace is of dimension 1 (under the assumption of network strong connectivity). This implies that, to find $\mathbf{s}$, we can find a vector which lies in the nullspace of C with its entries summing up to 1. Algorithm 2 will return back such vector. The intuition behind Algorithm 2 is that, it starts with a vector with its entries summing up to 1, and then iteratively subtracts out its projections on the row vectors of C, which eventually will give a vector in the nullspace of C. Note that during the run of iterations, the summation of the entries of $\{\mathbf{s}_t\}$ will remain unchanged, which can be easily verified by multiplying both sides of equation (14) by $\mathbf{1}^T$. Once the estimated vector $\hat{\mathbf{s}}$ is of $\mathbf{s}$ is obtained, each webpage, say page i, can estimate the web size as $\frac{1}{\hat{\mathbf{s}}_i}$.

It is important to note that the computation in (14) also only requires communications with the webpage's out-going links, hence (14) can be distributed implemented in the same way as in Algorithm 1. An exponential convergence in mean can be established for the sequence $\|\mathbf{s}_t - \mathbf{s}\|^2$, and the key steps are given as follows. From equation (14), it follows that

$$\mathbf{s}_{t+1} - \mathbf{s} = (I - C_k)(\mathbf{s}_t - \mathbf{s}),$$

where $C_k \triangleq \frac{C(k,:)^T C(k,:)}{\|C(k,:)\|^2}$, which implies

$$\begin{aligned}\mathbb{E}[\|\mathbf{s}_{t+1} - \mathbf{s}\|^2|\mathbf{s}_t] =& (\mathbf{s}_t - \mathbf{s})^T \left(I - \frac{1}{N}\sum_{k=1}^{N} C_k\right)(\mathbf{s}_t - \mathbf{s}).\\ \leq& \left(I - \frac{\sigma_2(\hat{C})}{N}\right)\|\mathbf{s}_t - \mathbf{s}\|^2,\end{aligned}$$

where $\sigma_2(\hat{C})$ denotes the second smallest singular value of matrix $\hat{C} \triangleq \sum_{k=1}^{N} C_k$. Note that the smallest singular value of $\hat{C}$ will be zero. The inequality follows directly from decomposing $\mathbf{s}_t - \mathbf{s}$ into two parts: its projection on $\mathbf{s}$ and the residual.

Algorithm 2 Network Size Estimation

Initialization:
Initialize vectors $\mathbf{s}_0 \in R^N$ as $[1, 0, \cdots, 0]$.
Iterations:
for $t = 0, 1, \cdots, T-1$ **do**
Generate $k = U[1, N]$, and update:

$$\mathbf{s}_{t+1} = \mathbf{s}_t - \frac{C(k,:)^T\mathbf{s}_t}{\|C(k,:)\|^2}C(k,:) \tag{14}$$

end for
Return:
Return $\mathbf{s}_T$.

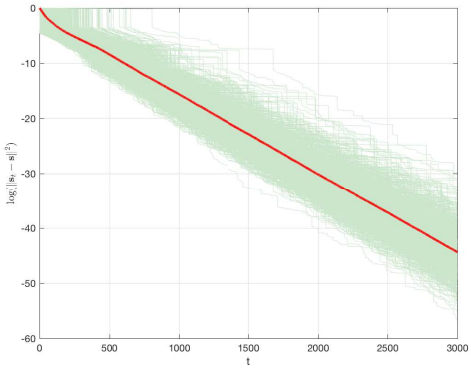

Fig. 2. This figure illustrates the behaviour of $\|\mathbf{s}_t - \mathbf{s}\|^2$. The network is generated in the same way as in section III. We run the experiment 1000 times, and the thick red line illustrate the average trajectory which decreases with an exponential speed.

References

[1] S. Brin, L. Page, The Anatomy of a Large-Scale Hypertextual Web Search Engine, *Computer Networks and ISDN System*, 30:107-117, 1998.

[2] Mallat, Stephane G., and Zhifeng Zhang. "Matching pursuits with time-frequency dictionaries." IEEE Transactions on signal processing 41.12 (1993): 3397-3415.

[3] A.N. Langville, C.D. Meyer, Google's PageRank and Beyond: The Science of Search Engine Ranking, Princeton University, 2006.

[4] Gleich, David F. "PageRank beyond the Web." SIAM Review 57.3 (2015): 321-363.

[5] R.A. Horn, C.R. Johnson, Matrix Analysis, Cambridge, U.K. Cambridge University Press, 1985.

[6] H. Ishii, R. Tempo, Distributed Randomized Algorithms for the PageRank Computation, *IEEE Transactions on Automatic Control*, 55:1987-2002, 2010.

[7] H. Ishii, R. Tempo and E.W. Bai, A Web Aggregation Approach for Distributed Randomized PageRank Algorithms, *IEEE Transactions on Automatic Control*, 57:2703-2717, 2012.

[8] B.T. Polyak, A. Timonina, PageRank: New Regularizations and Simulation Models, *18th World Congress of the International Federation of Automatic Control*, 2011.

[9] A. D. Sarma, A. R. Molla, G. Pandurangan, and E. Upfal, E., Fast distributed pagerank computation, *International Conference on Distributed Computing and Networking* , Springer Berlin Heidelberg, 2013.

[10] R. Tempo, H. Ishii, Monte Carlo and Las Vegas Randomized Algorithms for Systems and Control: An Introduction, *European Journal of Control*, 13:189-203, 2007.

[11] Borkar, Vivek S., and Adwaitvedant S. Mathkar. "Reinforcement Learning for Matrix Computations: PageRank as an Example." International Conference on Distributed Computing and Internet Technology. Springer International Publishing, 2014.

[12] Lei, Jinlong, and Han-Fu Chen. "Distributed Randomized PageRank Algorithm Based on Stochastic Approximation." IEEE Transactions on Automatic Control 60.6 (2015): 1641-1646.

[13] Robbins, Herbert, and Sutton Monro. "A stochastic approximation method." The annals of mathematical statistics (1951): 400-407.

[14] Zhao, Wenxiao, Han-Fu Chen, and Hai-Tao Fang. "Convergence of distributed randomized PageRank algorithms." IEEE Transactions on automatic control 58.12 (2013): 3255-3259.

[15] You, Keyou, Roberto Tempo, and Li Qiu. "Randomized incremental algorithms for the PageRank computation." 2015 54th IEEE Conference on Decision and Control (CDC). IEEE, 2015.

[16] Freris, Nikolaos M., and Anastasios Zouzias. "Fast distributed smoothing of relative measurements." Decision and Control (CDC), 2012 IEEE 51st Annual Conference on. IEEE, 2012.