Fast PageRank Computation Via a Sparse Linear System (Extended Abstract)

Gianna M. Del Corso[1] Antonio Gullí[1,2⋆] Francesco Romani[1]

[1] Dipartimento di Informatica, University of Pisa, Italy
[2] IIT-CNR, Pisa

Keywords: Link Analysis, Search Engines, Web Matrix Reducibility and Permutation.

1 Introduction and Notation

The research community has devoted an increased attention to reduce the computation time needed by Web ranking algorithms. Many efforts have been devoted to improve PageRank [4, 23], the well known ranking algorithm used by Google. The core of PageRank exploits an iterative weight assignment of ranks to the Web pages, until a fixed point is reached. This fixed point turns out to be the (dominant) eigenpair of a matrix derived by the Web Graph itself. Brin and Page originally suggested to compute this pair using the well-known Power method [12] and they also gave a nice interpretation of PageRank in terms of Markov Chains. Recent studies about PageRank address at least two different needs. First, the desire to reduce the time spent weighting the nodes of the Web Graph which takes several days. Second, to assign *many* PageRank values to each Web page, as results of PageRank's personalization [14–16] that was recently presented by Google as beta-service (see http://labs.google.com/personalized/). The Web changes very rapidly and more than 25% of links are changed and 5% of "new content" is created in a week [8]. This result indicates that search engines need to update link based ranking metrics (such as PageRank) very often and that a week-old ranking may not reflect very well the current importance of the pages. This motivates the need of accelerating the PageRank computation. Previous approaches followed different directions such as the attempt to compress the Web Graph to fit it into main memory [3], or the implementation in external memory of the algorithms [13, 7]. The very interesting research track exploits efficient numerical methods to reduce the computation time. These kind of numerical techniques are the most promising and we have seen many intriguing results in the last few years to accelerate the convergence of Power iterations [18, 13, 21].

In the literature [1, 21, 23] are presented models which treat in a different way pages with no out-links. In this paper we consider the original PageRank model (see Section 2) and, by using numerical techniques, we show that this problem can be transformed in an equivalent linear system of equations, where the coefficient matrix is as sparse as the Web Graph itself. This new formulation of the

⋆ Work partially supported by the Italian Registry of ccTLD.it

problem makes it natural to investigate the structure of the sparse coefficient matrix in order to exploit its reducibility. Moreover, since many numerical iterative methods for linear system solution can benefit by a reordering of the coefficient matrix, we rearrange the matrix increasing the data locality and reducing the number of iterations needed by the solving methods (see Section 4). In particular, we evaluate the effect of many different permutations and we apply several methods such as Power, Jacobi, Gauss-Seidel and Reverse Gauss-Seidel [25], on each of the rearranged matrices. The disclosed structure of the permuted matrix, makes it possible to use block methods which turn out to be more powerful than the scalar ones. We tested our approaches on a Web Graph crawled from the net of about 24 million nodes and more than 100 million links. Our best result, achieved by a block method, is a reduction of 58% in Mflops and of 89% in time with the respect of the Power method taken as reference method to compute the PageRank vector.

We now give some notations and definitions that will be useful in the rest of the paper. Let M by an $n \times n$ matrix. A scalar λ and a non-zero vector $\mathbf{x}$, are an eigenvalue and a corresponding (right) eigenvector of M if they are such that $M\mathbf{x} = \lambda\mathbf{x}$. In the same way, if $\mathbf{x}^T M = \lambda\mathbf{x}$, $\mathbf{x}$ is called left eigenvector corresponding to the eigenvalue λ. Note that, a left eigenvector is a (right) eigenvector of the transpose matrix. A matrix is row-stochastic if its rows are non negative and the sum of each row is one. In this case, it is easy to show that there exists a dominant eigenvalue equal to 1 and a corresponding eigenvector $\mathbf{x} = (c, c, \ldots, c)^T$, for any constant c. A very simple method for the computation of the dominant eigenpair is the Power method [12] which, for stochastic irreducible matrices, is convergent for any choice of the starting vector with non negative entries. A stochastic matrix M can be viewed as a transition matrix associated to a family of Markov chains, where each entry M_{ij} represents the probability of a transition from state i to state j. By the Ergodic Theorem for Markov chains [24] an irreducible stochastic matrix M has a unique steady state distribution, that is a vector π such that $\pi^T M = \pi^T$. This means that the stationary distribution of a Markov chain can be determined by computing the left eigenvector of the stochastic matrix M.

Given a graph $G = (V, E)$ and its adjacency matrix A, we denote by $\mathrm{outdeg}(i)$ the out-degree of vertex i that is the number of non-zeros in the i-th row of A. A node with no out-links is called "dangling".

2 Google's PageRank Model

In this section we review the original idea of Google's PageRank [4]. The Web is viewed as a directed graph (the Web Graph) $G = (V, E)$, where each of the N pages is a node and each hyperlink is an arc.

The intuition behind this model is that a page $i \in V$ is "important" if it is pointed by other pages which are in turn "important". This definition suggests an iterative fixed-point computation to assigning a rank of importance to each page in the Web. Formally, in the original model [23], a random surfer sitting

on the page i can jump with equal probability $p_{ij} = 1/\text{outdeg}(i)$ to each page j adjacent to i. The iterative equation for the computation of the PageRank vector $\mathbf{z}$ becomes

$$z_i = \sum_{j \in I_i} p_{ji} z_j,$$

where I_i is the set of nodes in-linking to the node i. The component z_i is the "ideal" PageRank of page i and it is then given by the sum of PageRank's assigned to the nodes pointing to i, weighted by the transition probability p_{ij}. The equilibrium distribution of each state represents the ratio between the number of times the random walks is in the state over the total number of transitions, assuming the random walks continues for infinite time. In matrix notation, the above equation is equivalent to the solution of the following system of equations $\mathbf{z}^T = \mathbf{z}^T P$, where $P_{ij} = p_{ij}$. This means that the PageRank vector $\mathbf{z}$ is the left eigenvector of P corresponding to the eigenvalue 1. In the rest of the paper, we assume that $||\mathbf{z}||_1 = \sum_{i=1}^{N} z_i = 1$, since the computation is not interested in assigning an exact value to each z_i, but rather in the relative rank between the nodes.

The "ideal" model has unfortunately two problems. The first problem is due to the presence of dangling nodes. They capture the surfer indefinitely. Formally, a dangling node corresponds to an all-zero row in P. As a consequence, P is not stochastic and the Ergodic Theorem cannot be applied. A convenient solution to the problem of dangling nodes is to define a matrix $\bar{P} = P + D$, where D is the rank one matrix defined as $D = \mathbf{d}\mathbf{v}^T$, and $d_i = 1$ iff $\text{outdeg}(i) = 0$. The vector $\mathbf{v}$ is a *personalization vector* which records a generic surfer's preference for each page in V [14, 16]. The matrix $\bar{P}$ imposes a random jump to every other page in V whenever a dangling node is reached. Note that the new matrix $\bar{P}$ is stochastic. In Section 3, we refer this model as the "natural" model and compare it with other approaches proposed in the literature.

The second problem, with the "ideal" model is that the surfer can "get trapped" by a cyclic path in the Web Graph. Brin and Page [4] suggested to enforce irreducibility by adding a new set of artificial transitions that with low probability jump to all nodes. Mathematically, this corresponds to defining a matrix $\widehat{P}$ as

$$\widehat{P} = \alpha \bar{P} + (1 - \alpha)\, \mathbf{e}\mathbf{v}^T, \tag{1}$$

where $\mathbf{e}$ is the vector with all entries equal to 1, and α is a constant, $0 < \alpha < 1$. At each step, with probability α a random surfer follows the transitions described by $\bar{P}$, while with probability $(1 - \alpha)$ she/he bothers to follows links and jumps to any other node in V accordingly to the personalization vector $\mathbf{v}$. We remark that the treatment for dangling nodes in the natural models is in agreement with the idea behind the random jump. That is, when we encounter a dangling node we jump, in a natural way, with probability one to any other node. The matrix $\widehat{P}$ is stochastic and irreducible and both these conditions imply that the PageRank vector $\mathbf{z}$ is the unique steady state distribution of the matrix $\widehat{P}$ such that

$$\mathbf{z}^T \widehat{P} = \mathbf{z}^T. \tag{2}$$

From (1) it turns out that the matrix $\widehat{P}$ is explicitly

$$\widehat{P} = \alpha(P + \mathbf{d}\mathbf{v}^T) + (1 - \alpha)\ \mathbf{e}\mathbf{v}^T. \tag{3}$$

The most common numerical method to solve the eigenproblem (2) is the Power method [12]. Since $\widehat{P}$ is a rank one modification of αP, it is possible to implement a power method which multiplies only the sparse matrix P by a vector and upgrades the intermediate result with a constant vector, at each step.

The eigenproblem (2) can be rewritten as a linear system. By substituting (3) into (2) we get $\mathbf{z}^T(\alpha P + \alpha\mathbf{d}\mathbf{v}^T) + (1 - \alpha)\mathbf{z}^T\mathbf{e}\mathbf{v}^T = \mathbf{z}^T$, which means that the problem is equivalent to the solution of the following linear system of equations

$$S\mathbf{z} = (1 - \alpha)\mathbf{v}, \tag{4}$$

where $S = I - \alpha P^T - \alpha\mathbf{v}\mathbf{d}^T$, and we make use of the fact that $\mathbf{z}^T\mathbf{e} = \sum_{i=1}^{N} z_i = 1$. The transformation of the eigenproblem (2) into the system (4) opens the route to a large variety of numerical methods not completely investigated in literature. In next section we present a lightweight solution to handle the problem of the non-sparsity of S.

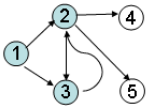
3 A Sparse Linear System Formulation

In this section we show how we can compute the PageRank vector as the solution of a sparse linear system. We remark that the way one handles the dangling node is crucial, since they can be a huge number. According to [19], a 2001 sample of the Web containing 290 million pages had only 70 million non-dangling nodes.

Page et al. [23], adopted the drastic solution of removing completely the dangling nodes. Doing that, the size of the problem is sensibly reduced but a large amount of information present in the Web is ignored. This has an impact on both the dangling nodes - which are simply not ranked - and on the remaining nodes - which don't take into account the contribution induced by the random jump from the set of dangling nodes. Moreover, removing this set of nodes could potentially create new dangling nodes, which must be removed in turn.

Arasu et al. [1] handled dangling nodes in a different way respect to the natural model presented in Section 2. They modify the Web Graph by imposing that every dangling node has a self loop. In terms of matrices, $\bar{P} = P + F$ where $F_{ij} = 1$ iff $i = j$ and outdeg$(i) = 0$. The matrix $\bar{P}$ is row stochastic and the computation of PageRank is solved using a random jump similar to the equation (3), where the matrix F replaces D. This model is different from the natural model, as it is evident from the following example.

Example 1. Consider the graph in Figure 1 and the associated transition matrix. The PageRank obtained, by using the natural model, orders the node as $(2, 3, 5, 4, 1)$. Arasu's model orders the node as $(5, 4, 2, 3, 1)$. Note that in the latter case node 5 ranks better than node 2, which is not what one expects.



$$P = \begin{pmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 0 & 0 & 1/3 & 1/3 & 1/3 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Fig. 1. An example of graph whose rank assignment differs if the dangling nodes are treated as in the model presented in [1].

From the above observations we believe that it is important to take into account the dangling nodes and that the natural model is the one which better capture the behavior of a random surfer. The dense structure of the matrix S poses serious problems to the solution of the linear system (4). On the contrary, the sparsity of the matrix P is easily exploited computing the PageRank vector with the Power method. In fact, it is common to implement the Power method in a way where the matrix-vector multiplications involve only the sparse matrix P while the rank-one modifications are handled separately [13].

In the following, we show how to manage dangling nodes in a direct and lightweight manner which makes it possible to use iterative methods for linear systems. In particular, we prove formally how the solution of (4) is equivalent to the solution of a system involving only the sparse matrix $R = I - \alpha P^T$. The following theorem holds.

Theorem 1. *The PageRank vector* $\mathbf{z}$ *solution of* (4) *is obtained by solving the system* $R\mathbf{y} = \mathbf{v}$ *and taking* $\mathbf{z} = \mathbf{y}/||\mathbf{y}||_1$.

Proof. Since $S = R - \alpha\ \mathbf{v}\mathbf{d}^T$ equation (4) becomes $(R - \alpha\ \mathbf{v}\mathbf{d}^T)\mathbf{z} = (1 - \alpha)\ \mathbf{v}$, that is, a system of equations where the coefficient matrix is the sum of a matrix R and a rank-one matrix. Note that R is non singular since $\alpha < 1$ and therefore all the eigenvalues of R are different from zero. We can use the Sherman-Morrison formula [12], paragraph 2.1.3, for computing the inverse of the rank-one modification of R. As a consequence, we have

$$(R - \alpha\mathbf{v}\mathbf{d}^T)^{-1} = R^{-1} + \frac{R^{-1}\mathbf{v}\mathbf{d}^T R^{-1}}{1/\alpha + \mathbf{d}^T R^{-1}\mathbf{v}}. \tag{5}$$

From (5), denoting by $\mathbf{y}$ the solution of the system $R\mathbf{y} = \mathbf{v}$, we have

$$\mathbf{z} = (1 - \alpha)\left(1 + \frac{\mathbf{d}^T\mathbf{y}}{1/\alpha + \mathbf{d}^T\mathbf{y}}\right)\mathbf{y},$$

that means that $\mathbf{z} = \gamma\mathbf{y}$, and the constant $\gamma = (1 - \alpha)\left(1 + \frac{\mathbf{d}^T\mathbf{y}}{1/\alpha + \mathbf{d}^T\mathbf{y}}\right)$ can be computed normalizing $\mathbf{y}$ in such a way $||\mathbf{z}||_1 = 1$. □

Summarizing, we have shown that in order to compute the PageRank vector $\mathbf{z}$ we can solve the linear system $R\mathbf{y} = \mathbf{v}$, and then normalize $\mathbf{y}$ to obtain the PageRank vector $\mathbf{z}$. This means that the rank one matrix D in the PageRank

model accounting for dangling pages plays a role only in the scaling factor γ. Moreover, the computation of γ is not necessary as well, since generally we are only interested in the relative ranks of the Web pages. We point out that the matrix used by Arasu and al. [1] is also sparse due to the way they deal with the dangling nodes, but the PageRank obtained don't ranks the node in a natural way (see Example 1). Instead, our approach guarantees a more natural ranking and handles the density of S by transforming a dense problem in one which uses the sparse matrix R.

Moreover, when solving a linear system, particular attention must be devoted to the conditioning of the problem. It is easy to show [20, 17], that the condition number in the 1-norm of S is $\mathrm{cond}_1(S) = \frac{1+\alpha}{1-\alpha}$, which means that the problem tends to become ill-conditioned as α goes to one. On the other hand, as we will show in the full paper, $\mathrm{cond}_1(R) \leq \mathrm{cond}_1(S)$. However, on small synthetic matrices as well as on the web crawler of 24 million nodes used in the experimentation, we observed that the system involving R is much better conditioned than the one involving S for α going to 1. However, one can always construct a graph and a matrix P for which the condition numbers of R and S are the same, but it can be proved that the above inequality is strict if the Web graph has at least a dangling node for each connected component. Bianchini and al. in [2] prove that the iterative method derived by (2) and involving $\widehat{P}$ produces the same sequence of vectors of the Jacobi method applied to matrix R. In a recent paper [11] another way to deal with dangling nodes has been proposed. That approach is however different from ours. In particular, they assign separatelly a rank to dangling and non dangling pages and their algorithm requires the knowlege of a complete strongly connected subgraph of the web.

4 Exploiting the Web Matrix Permutations

In the previous section we have shown how to transform the linear system involving the matrix S into an equivalent linear system, where the coefficient matrix R is as sparse as the Web Graph. To solve the linear system $R\mathbf{y} = \mathbf{v}$ convenient strategies are Jacobi, Gauss-Seidel [25], because they use space comparable to that used by the Power method. These methods are convergent if and only if the spectral radius of the iteration matrix is strictly lower than one. Moreover, the more the spectral radius of the iteration matrix is close to zero the faster the convergence. Since R is an M-matrix, it is possible to show that both Jacobi and Gauss-Seidel methods are convergent and moreover that Gauss-Seidel method applied to R is always faster than Jacobi method [25].

When solving a sparse linear system a common practice [10] is to look for a reordering scheme that reduces the (semi)bandwidth in order to increase data locality and hence the time spent for each iteration. For some methods, we also have a change in the spectral radius of the iteration matrix when applied to the permuted matrix. This is not the case for Jacobi method since the spectral radius of the iteration matrix is invariant under permutation. In the same way, the convergence of the Power method is also independent of matrix reordering.

However, in [19] is given a method to reorder the matrix in such a way the Urls are sorted lexicographically. This may help to construct a better starting vector for the Power method and to improve data locality. A much challenging perspective is reordering the Web matrix for the Gauss-Seidel method, where opportune permutations can lead both to an increase in data locality and to an iteration matrix with a reduced spectral radius.

Our permutation strategies are obtained by combining different elementary operations. A very effective reordering scheme, denoted by $\mathcal{B}$, is the one given by permuting the nodes of the Web graph according to the order induced by a BFS visit. The BFS visit makes it possible to discover reducibility of the Web Matrix, since this visit assigns contiguous permutation indices to pages pointed by the same source. Therefore, this permutation produces lower block triangular matrices. It has been observed [9] that the BFS strategy for reordering sparse symmetric matrices produces a reduced bandwidth when the children of each node are inserted in order of decreasing degree. For this reason, we examine other reordering schemes which are obtained by sorting the nodes in terms of their degree. Since the Web matrix is not symmetric, we consider the permutation which reorders the pages for decreasing out-degree; denoting this scheme as $\mathcal{O}$ while the permutation $\mathcal{Q}$ sorts the pages of the Web matrix for increasing out-degree. Note that these permutations list the dangling pages in the last and in the first rows of the Web matrix respectively. We experimented also the permutations obtained reordering the matrix by increasing and decreasing in-degree; denoted by $\mathcal{X}$ and $\mathcal{Y}$ respectively.

Since $R = I - \alpha P^T$, our algorithms needs to compute the transpose of the Web matrix. We denote the transposition of the Web matrix by $\mathcal{T}$. The various operations can be combined obtaining 15 different possible reordering of the Web matrix as shown in Figure 2. In accordance with the taxonomy in Figure 2, we denote, for instance, by $R_{\mathcal{XTB}} = I - \alpha\Pi(P)$, where the permuted matrix $\Pi(P)$ is obtained applying first the $\mathcal{X}$ permutation, then transposing the matrix and applying finally the $\mathcal{B}$ permutation on the matrix reordered. The first column in figure 2 gives rise to full matrices, while the second and third columns produce block triangular matrices due to the BFS's order of visit. In Figure 3, we show a plot of the structure of a Web matrix rearranged according to each item of the above taxonomy. We adopted ad hoc numerical methods for dealing with

Full	Lower Block Triangular	Upper Block Triangular
$\mathcal{T}$	$\mathcal{TB}$	$\mathcal{BT}$
$\mathcal{OT}$	$\mathcal{OTB}$	$\mathcal{OBT}$
$\mathcal{QT}$	$\mathcal{QTB}$	$\mathcal{QBT}$
$\mathcal{XT}$	$\mathcal{XTB}$	$\mathcal{XBT}$
$\mathcal{YT}$	$\mathcal{YTB}$	$\mathcal{YBT}$

Fig. 2. Web Matrix Permutation Taxonomy

the different shapes of matrices in Figure 3. In particular, we compared Power method, and Jacobi iterations with Gauss-Seidel, and Reverse Gauss-Seidel. We

recall that the Gauss-Seidel method computes $y_i^{(k+1)}$, the $i-$th entry of the vector at the $(k+1)-$th iteration step as a linear combination of $y_j^{(k+1)}$ for $j = 1, \ldots, i-1$ and of $y_j^{(k)}$ for $j = i+1, \ldots, n$. On the contrary, the Reverse Gauss-Seidel method computes the entries of the vector $\mathbf{y}^{(k+1)}$ bottom up, that is it computes $y_i^{(k+1)}$ for $i = n, \ldots, 1$ as a linear combination of $y_j^{(k+1)}$ for $j = n, \ldots, i+1$ and of $y_j^{(k)}$ for $j = 1, \ldots, i-1$.

Note that $R_{\mathcal{OT}} = JR_{\mathcal{QT}}J^T$ and $R_{\mathcal{XT}} = JR_{\mathcal{YT}}J^T$ where J is the anti-diagonal matrix, that is $J_{ij} = 1$ iff $i + j = n + 1$. This means that applying Gauss-Seidel to $R_{\mathcal{OT}}$ ($R_{\mathcal{XT}}$) is the same that applying Reverse Gauss-Seidel to $R_{\mathcal{QT}}$ ($R_{\mathcal{YT}}$).

The shapes of some matrices in Figure 3, encourage to exploit the matrix reducibility experimenting with block methods. In particular, we note that the matrices permuted according to the BFS visit have a block triangular structure. Moreover, also the matrix $R_{\mathcal{OT}}$ is lower block triangular, since it separates non-dangling nodes from dangling nodes. A natural way to handle block triangular matrices is to use forward/backward block substitution. For instance, on the lower block triangular system

$$\begin{bmatrix} R_{11} & & \\ R_{21} & R_{11} & \\ \vdots & & \ddots \\ R_{m1} & \cdots & R_{mm} \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_m \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \\ \vdots \\ \mathbf{v}_m \end{bmatrix},$$

the solution can be computed as follows

$$\begin{cases} \mathbf{y}_1 = R_{11}^{-1}\mathbf{v}_1, \\ \mathbf{y}_i = R_{ii}^{-1}\left(\mathbf{v}_i - \sum_{j=1}^{i-1} R_{ij}\mathbf{y}_j\right) & \text{for } i = 2, \ldots, m. \end{cases}$$

This requires the solution of m smaller linear systems, where the coefficient matrices are the diagonal blocks in the order they appear. As solving methods for the diagonal block systems, we tested both Gauss-Seidel and Reverse Gauss-Seidel methods. We denote by `LB` and `UB` the methods obtained using Gauss-Seidel as solver of the diagonal blocks on lower or upper block structures respectively. `LBR` and `UBR` use instead Reverse Gauss-Seidel to solve the diagonal linear systems. Summing up, we have the taxonomy of solution strategies reported in figure 4. In Section 5 we report the experimental results obtained by applying each method in Figure 4 to all the suitable matrices in Figure 2.

5 Experimental Results

We tested the approaches discussed in previous sections using three Web Graphs of different sizes. We report only the results on the largest one, which is a crawling of 24 million Web pages with about 100 million hyperlinks and containing approximately 3 million dangling nodes. This data set was donated to us by the Nutch project http://www.nutch.org/. We run our experiments on a PC with a Pentium IV 3GHz, 2.0GB of memory and 512Mb of L1 cache.

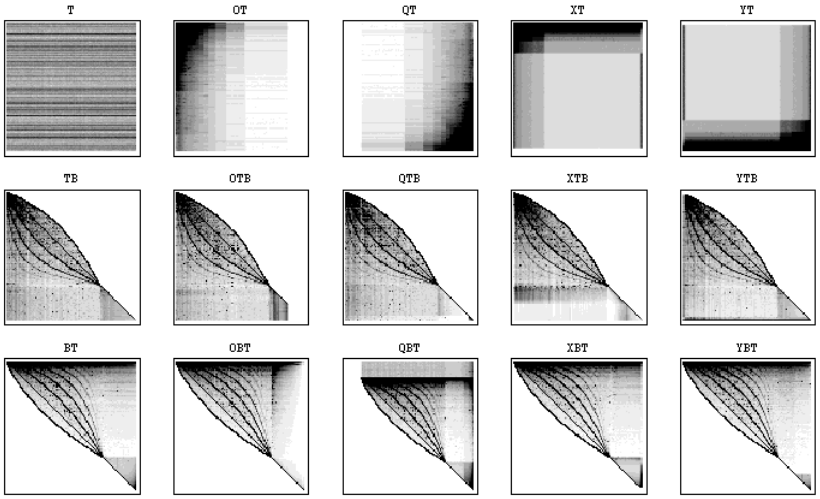


Fig. 3. Different shapes obtained by rearranging the Web matrix P in accordance to the taxonomy. First row represents full matrices; second and third lower and upper block triangular matrices respectively. Web Graph is made of 24 million nodes and 100 million links.

Scalar methods	shapes	Block methods	shapes
`PM`	all	`LB`	$R_{*\mathcal{TB}}$ and $R_{\mathcal{OT}}$
`Jac`	all	`LBR`	$R_{*\mathcal{TB}}$ and $R_{\mathcal{OT}}$
`GS`	all	`UB`	$R_{*\mathcal{BT}}$
`RGS`	all	`UBR`	$R_{*\mathcal{BT}}$

Fig. 4. Numerical Methods Taxonomy. `PM` is the Power method, `Jac` denotes the Jacobi method, `GS` and `RGS` are the Gauss-Seidel and Reverse Gauss-Seidel, respectively. All of them can be applied to each transformation of the matrix according to the taxonomy 2. Among block-methods we have `LB` and `LBR` which can be applied to all lower block triangular matrices and use `GS` or `RGS` to solve each diagonal block. Similarly, `UB` and `UBR` refer to the upper block triangular matrices.

A stopping criterion of 10^{-7} is imposed on the absolute difference between the vectors computed in two successive iterations. In Figure 5 we report the running time in seconds, the number of iterations and the Mflops, for each combination of solving and reordering methods described in Figure 4 and 2. The cost of each iteration is obtained as the ratio between the number of Mflops and the number of iterations. Some cells are empty since there are methods suitable only on particular shapes. Moreover, in Figure 5 the results in the last two columns are relative to `LB` and `LBR` methods for lower block triangular matrices and `UB` or `UBR` for upper block triangular matrices. We do not report in the table the behavior of `Jac` method, since it has always worst performance than `GS` method. Since the diagonal entries of R are equal to 1, Jacobi method is essentially equivalent to the Power method. In our case, the only difference is that `Jac` is applied to R

while PM works on $\widehat{P}$, which incorporates the rank one modification accounting for dangling nodes. We implemented PM using the optimizations suggested in [23, 13]. Using the results of Theorem 1, we get a reduction in Mflops of about 3%. For the increased data locality, the running time of Jac benefits from matrix reordering, and we have a reduction up to 23% over the Power iterations. We

Name	PM	GS	RGS	LB/UB	LBR/UBR
$\mathcal{T}$	3454, 33093	1903, 19957	2141, 20391	−, −	−, −
$\mathcal{OT}$	2934, 33093	1660, 20825	1784, 19957	1570, 19680	1515, 18860
$\mathcal{XT}$	3315, 33309	1920, 21259	1944, 19957	−, −	−, −
$\mathcal{TB}$	1386, 32876	731, 21910	717, 18439	485, 16953	438, 15053
$\mathcal{OTB}$	1383, 33093	705, 21476	705, 18439	480, 17486	446, 15968
$\mathcal{QTB}$	1353, 32876	743, 23645	618, 16920	520, 18856	**385, 13789**
$\mathcal{XTB}$	1361, 33309	682, 21259	715, 19090	484, 17196	472, 16414
$\mathcal{YTB}$	1392, 32876	751, 22343	**625, 16270**	501, 17972	390, 13905
$\mathcal{BT}$	1394, 33093	628, 18439	879, 22560	464, 15545	570, 19003
$\mathcal{OBT}$	1341, 33309	605, 18873	806, 21693	470, 15937	543, 18312
$\mathcal{QBT}$	1511, 33093	702, 18873	922, 21693	427, 15128	493, 17387
$\mathcal{XBT}$	1408, 33093	667, 19306	860, 21693	474, 16075	541, 18265
$\mathcal{YBT}$	1351, 33093	600, 18439	806, 21693	461, 15564	541, 18310

Fig. 5. Experimental Results: in columns are listen the numerical methods analyzed and the rows describe the permutations applied to the matrix R. Each cell represents the running time in seconds, the number of Mflops. Note that the results in the last two columns account either for the cost of the LB and LBR methods, applied to lower block triangular matrices, or for the cost of UB and UBR methods, applied to upper block triangular matrices. In bold we highlight our best results for scalar and block methods. For block methods we had a cost of only 42% in terms of Mflops with respect to the Power method, commonly used to compute the PageRank.

now compare the proposed methods versus PM applied to the original matrix since this is the common solving method to compute PageRank vector. Other comparisons can be obtained from Figure 5. As one can expect, the use of GS and RGS on the original matrix already accounts for a reduction of about 40% in the number of Mflops and of about 45% in running time. These improvements are striking when the system matrix is permuted. The best performance of scalar methods is obtained using the $\mathcal{YTB}$ combination of permutations on RGS method. This yields a Mflops reduction of 51% with respect to PM and a further reduction of 18% with respect to the GS both applied to the full matrix. The running time is reduced of 82%.

The common intuition is that Gauss-Seidel method behaves better on a quasi-lower triangular while Reverse Gauss-Seidel is faster when applied to quasi-upper triangular matrices. However, in this case the intuition turns to be misleading. In fact, for our Web matrix RGS works better on lower block triangular matrices and GS works better on upper matrices.

Even better results are obtained by block methods. LB applied to $R_{\mathcal{OT}}$ achieves a reduction of 41% in Mflops with respect to PM. Adopting this solving

method, we explore just the matrix reducibility due to dangling nodes. As depicted in Figure 6, the best result is obtained for the $\mathcal{QTB}$ permutation when the `LBR` solving method is applied. In this case, we have a reduction of 58% in Mflops and of 89% in the running time. This means that our solving algorithm computes the PageRank vector in about a tenth of the running time and with less than half operations of the Power method. The results given in Figure 5 do not take into account the effort spent in reordering the matrix. However, the most costly reordering scheme is the BFS visit of the Web graph, which can be efficiently implemented in semi-external memory as reported in [6, 22]. The running time spent for doing the BFS are comparable to those reported in [5], where less of 4 minutes are taken on a Web Graph with 100 million nodes. The effort spent in applying permutation operators is largely repaid from the speedup achieved on the solving methods. Moreover, in case of personalized PageRank the permutations can be applied only once and reused for all personalized vectors $\mathbf{v}$. An intuitive picture of the gain obtained by combining permutation strategies with the scalar and block solving methods is shown in Figure 6.

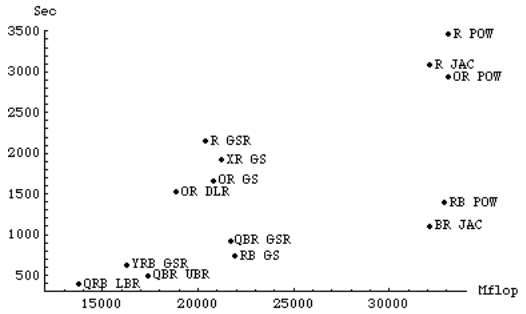


Fig. 6. A plot of some of the results of Figure 5. On the x-axis the number of Mflops and on the y-axis the running time in seconds. Each point is labeled with the permutation applied and the solving method used.

6 Conclusion

The ever-growing size of Web graph implies that the value and importance of fast methods for Web ranking is going to rise in the future. The problem of PageRank computation can be easily be viewed as a dense linear system. We showed how to handle the density of this matrix by transforming the original problem in one which uses a matrix as sparse as the Web Graph itself. On the contrary of what done in [1], we achieved this result without altering the original model. This result allows to efficiently consider the PageRank computation as a sparse linear system, in alternative to the eigenpairs interpretation. Dealing with a sparse linear system opens the way to exploit the Web Matrix reducibility by composing opportunely some Web matrix permutations to speedup the PageRank computation. We showed that permuting the Web matrix according to a combination of in-degree or out-degree and sorting the pages following the

order of the BFS visit, can effectively increase data locality and reduce the running time when used in conjunction with numerical method such as lower block solvers. Our best result achieves a reduction of 58% in Mflops and of 89% in terms of seconds required compared to the Power method commonly used to compute the PageRank. This means that our solving algorithm requires almost a tenth of the time and much less than half in terms of Mflops. The previous better improvement over the Power method is due to [21] where a reduction of 80% in time is achieved on a data set of roughly 400.000 nodes. In light of the experimental results, our approach for speeding up PageRank computation appears much promising.

Acknowledgment

We thank Daniel Olmedilla of Learning Lab Lower Saxony, Doug Cutting and Ben Lutch of the Nutch Organization, Sriram Raghavan and Gary Wesley of Stanford University. They provided to us some Web graphs and a nice Web crawler. We thank Luigi Laura and Stefano Millozzi for their COSIN library. We also thank Paolo Ferragina for useful discussions and suggestions.

References

1. A. Arasu, J. Novak, A. Tomkins, and J. Tomlin. PageRank computation and the structure of the Web: Experiments and algorithms. In *Proceedings of the Eleventh International World Wide Web Conference*, 2002.
2. M. Bianchini, M. Gori, and F. Scarselli. Inside PageRank. *ACM Transactions on Internet Technology*, 2004. to appear.
3. P. Boldi and S. Vigna. WebGraph framework i: Compression techniques. In *Proceedings of the Thirteenth International WWW Conference*, 2004.
4. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.
5. A. Z. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. L. Wiener:. Graph structure in the Web. *Computer Networks*, 33:309–320, 2000.
6. Adam L. Buchsbaum, Michael Goldwasser, Suresh Venkatasubramanian, and Jeffery Westbrook. On external memory graph traversal. In *Symposium on Discrete Algorithms*, pages 859–860, 2000.
7. Y. Chen, Q. Gan, and T. Suel. I/o-efficient techniques for computing Pagerank. In *Proceedings of the Eleventh International WWW Conference*, 2002.
8. J. Cho and S. Roy. Impact of Web search engines on page popularity. In *Proceedings of the Thirteenth International WWW Conference*, 2004.
9. E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *In Proc. 24th Nat. Conf. ACM*, pages 157–172, 1969.
10. C. C. Douglas, J. Hu, M. Iskandarani, M. Kowarschik, U. Rüde, and C. Weiss. Maximizing cache memory usage for multigrid algorithms. In *Multiphase Flows and Transport in Porous Media: State of the Art*, pages 124–137. Springer, Berlin, 2000.
11. N. Eiron, S. McCurley, and J. A. Tomlin. Ranking the web frontier. In *Proceedings of the Thirteenth International WWW Conference*, 2004.

12. G. H. Golub and C. F. Van Loan. *Matrix Computations*. The John Hopkins University Press, Baltimore, 1996. Third Edition.
13. T. Haveliwala. Efficient computation of PageRank. Technical report, Stanford University, 1999.
14. T. Haveliwala. Topic-sensitive PageRank. In *Proceedings of the Eleventh International WWW Conference*, 2002.
15. T. Haveliwala, S. Kamvar, and G. Jeh. An analytical comparison of approaches to personalizing PageRank. Technical report, Stanford University, 2003.
16. G. Jeh and J. Widom. Scaling personalized Web search. In *In Proceedings of the Twelfth International WWW Conference*, 2002.
17. S. Kamvar and T. Haveliwala. The condition number of the pagerank problem. Technical report, Stanford University, 2003.
18. S. Kamvar, T. Haveliwala, C. Manning, and G. Golub. Extrapolation methods for accelerating PageRank computations. In *Proceedings of the Twelfth International WWW Conference*, 2003.
19. S. D. Kamvar, T. H. Haveliwala, C. Manning, and G. H. Golub. Exploiting the block structure of the Web for computing PageRank. Technical report, Stanford University, 2003.
20. A. N. Langville and C. D. Meyer. Deeper inside PageRank. *Internet Mathematics*, 2004. to appear.
21. C. P. Lee, G. H. Golub, and S. A. Zenios. A fast two-stage algorithm for computing PageRank. Technical report, Stanford University, 2003.
22. K. Mehlhorn and U. Meyer. External-memory breadthfirst search with sublinear I/O. In *European Symposium on Algorithms*, pages 723–735, 2002.
23. L. Page, S. Brin, R. Motwani, and T. Winograd. The PageRank citation ranking: Bringing order to the Web. Technical report, Stanford Digital Library Technologies Project, 1998.
24. W. S. Stewart. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1995.
25. R. S. Varga. *Matrix Iterative Analysis*. Prentice-Hall, Englewood Cliffs, 1962.