Fast PageRank Computation Via a Sparse Linear System*

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

[1]Dipartimento di Informatica, Università di Pisa
[2]IIT-CNR, Pisa

April 18, 2005

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

Abstract

Recently, the research community has devoted an increased attention to reduce the computational time needed by Web ranking algorithms. In particular, many techniques have been proposed to speed up the well-known PageRank algorithm used by Google. This interest is motivated by two dominant factors: (1) the Web Graph has huge dimensions and it is subject to dramatic updates in terms of nodes and links - therefore the PageRank assignment tends to became obsolete very soon; (2) many PageRank vectors need to be computed according to different choices of the personalization vectors or when adopting strategies of collusion detection.

In this paper, we show how the PageRank computation in the original random surfer model can be transformed in the problem of computing the solution of a sparse linear system. The sparsity of the obtained linear system makes it possible to exploit the effectiveness of Markov chain index reordering to speed up the PageRank computation. In particular, we rearrange the system matrix according to several permutations and we apply different scalar and block iterative methods to solve smaller linear systems. We tested our approaches on Web Graphs crawled from the net. The largest one accounts about 24 millions nodes and more than 100 million links. Upon this Web Graph, the cost for computing the PageRank is reduced of 65% in terms of Mflops and of 92% in terms of time respect to the Power method commonly used.

1 Introduction

The research community has devoted an increased attention to reduce the computation time needed by Web ranking algorithms. In fact, the Web changes very rapidly and in a week more than 25% of links are changed and 5% of "new content" is created [8]. This result indicates that search engines need to update link based ranking metrics very often and that a week-old ranking may not reflect very well the current importance of the pages.

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

*Partially supported by the GNCS-INDAM Project: "Problematiche Numeriche nel WEB"
†Work partially supported by the Italian Registry of ccTLD.it

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 of computation. Second, the need to assign *many* PageRank values to each Web page. This is necessary for PageRank's personalization [14, 15, 16] that was recently presented by Google as beta-service (see http://labs.google.com/personalized/) or for some heuristic for collusion-proof PageRank [26] algorithms which requires the computation of many different PageRank vectors for different choices of a parameter.

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]. A 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 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 3) 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 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 5). 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. Note that the phase of reordering a matrix according to a given permutation requires extra computational effort, but the time spent reordering the matrix is negligible respect to the time required to solve the system. A more important consideration is that the same reordering can be used to solve many PageRank problems with different personalization vectors.

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 65% in Mflops and of 92% in time with the respect of the Power method taken as reference method to compute the PageRank vector.

2 Definitions and Notations

In this section we 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 non-negative starting vector. 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, let outdeg(i) be 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".

3 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 most of the time one 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 one is due to the presence of dangling nodes which 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 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 4, 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 the 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 $\mathbf{v}$. 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, as suggested by Haveliwala in [13].

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 (4) opens the route to a large variety of numerical methods not completely investigated in the literature. In the next section we present a lightweight solution to handle the non-sparsity of S.

4 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 Kamvar et al. [19], a 2001 sample of the Web containing 290 million pages had only 70 million non-dangling nodes. This large amount of nodes without outlinks includes both pages which do not point to any other page, and also pages whose existence is inferred by hyperlinks but not yet reached by the crawler. Besides, a dangling node can represent pdf, ps, txt or any other file format gathered by a crawler but with no hyperlinks pointing outside. Page et al. [23], adopted

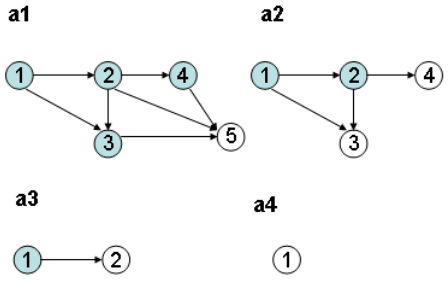


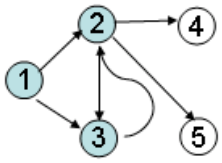
Figure 1: Removing the dangling node in figure (a1) generates new dangling nodes, which are in turn removed (a2, a3, a4). At the end of the process, no node receives a PageRank assignment.

the drastic solution of removing completely the dangling nodes. In this way, 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 in turn be removed (see fig 1). Therefore, the nodes with an assigned PageRank could be a fraction of the Web Graph.

Arasu et al. [1] handled dangling nodes in a different way respect to the natural model presented in Section 3. 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 4.1 *Consider the graph in Figure 2 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.*



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

Figure 2: An example of graph whose rank assignment differs if the dangling nodes are treated as in the model presented by Arasu et al. in [1].

From the above observations we believe that it is important to take into account the dangling nodes and we consider the natural model the one which better captures the behavior of a random surfer. The dense structure of the matrix S poses serious problems to the solution of the linear system (4). For this reason, the problem has been largely addressed as an eigenproblem, while we have seen very few attempts to solve the problem formulated as a linear system. Computing the PageRank vector with the Power method, allows to exploit the sparsity of the matrix P. In fact, it is common to implement the Power method in such a way 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 the equivalence of (4) to the solution of a system involving only the sparse matrix $R = I - \alpha P^T$. Next theorem makes use of a very powerful tool: the Sherman-Morrison formula (see [12], paragraph 2.1.3). It is well known that Sherman-Morrison formula is unstable, however we use it here only as an elegant technique for proving the theorem without any need of implementing it.

Theorem 4.2 *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 (see [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 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 which accounts for the dangling pages plays a role only in the scaling factor γ. Moreover, the computation of γ is not always necessary and this step can be occasionally be omitted.

Note 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 2). 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.

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. Recently Eiron et al. [11] proposed another way to deal with dangling nodes. They assign separately a rank to dangling and non dangling pages and their algorithm requires the knowledge of a complete strongly connected subgraph of the web.

4.1 The conditioning of the problem in the new formulation

When solving a linear system, particular attention must be devoted to the conditioning of the problem, the magic number accounting for the "hardness" of solving a linear system. The condition number of a matrix A is defined as $\mathrm{cond}(A) = \|A\|\,\|A^{-1}\|$ for any matrix norm. It is easy to show, as proved by Kamvar and Haveliwala [17], that the condition number in the 1-norm of S is $\mathrm{cond}(S) = \frac{1+\alpha}{1-\alpha}$, which means that the problem tends to become ill-conditioned as α goes to one. For the conditioning of R we can prove the following theorem.

Theorem 4.3 *The condition numbers expressed in the 1-norm of matrices S and R are such that*

$$\mathrm{cond}(R) \leq \mathrm{cond}(S).$$

Moreover, the inequality is strict if and only if from every node there is a direct path to a dangling node.

Proof. In order to prove the theorem we have to show that $\mathrm{cond}(R) \leq \frac{1+\alpha}{1-\alpha}$. We have

$$\|R\|_1 = \max_{j=1,\ldots,n} \sum_{i=1}^{n} |r_{ij}| = \max_{j=1,\ldots,n} \left(1 + \alpha \sum_{i=1, i\neq j}^{n} p_{ji}\right).$$

Then, if $P \neq O$, $\|R\|_1 = 1 + \alpha$. Note that R^{-1} is a non negative matrix, hence

$$\|R^{-1}\|_1 = \|\mathbf{e}^T R^{-1}\|_1 = \|R^{-T}\mathbf{e}\|_\infty.$$

Moreover,

$$R^{-T} = (I - \alpha P)^{-1} = \sum_{i=0}^{\infty} \alpha^i P^i. \tag{6}$$

Since every entry of the vector $P\mathbf{e}$ is less or equal 1, we have $P^i\mathbf{e} \leq \mathbf{e}$, hence

$$\|R^{-1}\|_1 = \|\sum_{i=0}^{\infty} \alpha^i P^i \mathbf{e}\| \leq \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha},$$

which proves that $\mathrm{cond}(R) \leq \mathrm{cond}(S)$.

Let us now prove that the inequality is strict if from every page it is possible to reach a dangling node with Markov chain state transitions. Since P has dangling nodes, P is reducible. Let k, $k \geq 1$ be the number of strongly connected components of the Web Graph. We can permute rows and columns of P grouping together the nodes belonging to the same connected component and listing the dangling nodes in the last rows. Therefore, P can be expressed in this reduced form

$$P = \begin{bmatrix} P_{11} & P_{1,2} & \cdots & \cdots & P_{1,k} \\ & P_{2,2} & \cdots & & P_{2,k} \\ & & \ddots & & \vdots \\ & & & P_{k-1,k-1} & P_{k-1,k} \\ O & & & & O \end{bmatrix},$$

where the diagonal blocks P_{ii} are irreducible. By hypothesis, from every web page we can reach a dangling node with a finite number of clicks. In terms of the matrix P, this means that, for every i, $1 \leq i \leq k$, each block P_{ii} has at least a row whose sum is strictly less than 1. An extension of the Gershgorin circle Theorem [25][Theorem 1.7, page 20] ensures that the spectral radius of every diagonal block P_{ii} is less than 1. Since the eigenvalues of P are the eigenvalues of the diagonal blocks this guarantees $\rho(P) < 1$. Let us consider the iterative method $\mathbf{x}_{i+1} = P\mathbf{x}_i$, since $\rho(P) < 1$ this method is convergent to the zero vector, for every choice of the starting vector $\mathbf{x}_0$. Then, choosing $\mathbf{x}_0 = \mathbf{e}$, there exists an integer i such that $P^i\,\mathbf{e} < \mathbf{e}$. From (6) we have

$$\|R^{-T}\|_1 = \|\sum_{i=0}^{\infty} \alpha^i P^i \mathbf{e}\|_\infty < \|\sum_{i=0}^{\infty} \alpha^i \mathbf{e}\| = \frac{1}{1-\alpha},$$

which proves the "if" part. To prove the "only if" part, assume by contradiction that there is at least a page from which it is not possible to reach a dangling page, and assume this page belongs to the h-th connected component. Since P_{hh} is irreducible, this means that from every node in P_{hh} it is not possible to reach a dangling page. Hence, there is at least an index i, $1 \leq i \leq k-1$, such

that the strip corresponding to the i-th block is zero, except the diagonal block P_{ii}. This means, the diagonal block P_{ii} has an eigenvalue equal to one, hence $\rho(P) = 1$ and $P_{ii}\mathbf{e}_i = \mathbf{e}_i$, which implies $\text{cond}(R) = (1+\alpha)/(1-\alpha)$. ∎

Theorem 4.3 proves that the condition number of matrix R is always less or equal the condition number of S. This means that computing the PageRank vector solving the system with matrix R is never worst than computing it solving the system involving S. In fact, the system in R is less sensitive to the errors due to the finite representation of the numbers appearing in P than the system involving S.

Note that if the condition which guarantees the strict inequality between the condition number of S and R is not satisfied, there exists a reordering of R which allows to split the original problem into two disjoint subproblems. As we will see in Section 5, this is computationally convenient and moreover, the conditioning of the two subproblems should be compared with the conditioning of the two subproblems regarding S.

5 Exploiting the Web Matrix Permutations

In Section 4 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. The new sparse formulation allows to exploit the effectiveness of other iterative procedures to compute the PageRank vector, which were not applicable when dealing with S. To solve the linear system $R\mathbf{y} = \mathbf{v}$ two convenient strategies are Jacobi, Gauss-Seidel methods [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 rate of convergence depends on the spectral radius and the lower is the radius the faster is the convergence. Since $R = I - \alpha P$ where P is a nonnegative matrix, R is a so called M-matrix, and it is well known that both Jacobi and Gauss-Seidel methods are convergent and that Gauss-Seidel method applied to M-matrices 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 for increasing data locality and hence the time spent for each iteration. Moreover, if the matrix is reducible, the problem can be split into smaller linear systems which can be solved in cascade.

The simpler permutation scheme is the one which separates dangling from non-dangling pages. In this case the matrix system involving R has a very simple shape

$$\begin{bmatrix} I - \alpha P_1^T & O \\ -\alpha P_2^T & I \end{bmatrix} \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \end{bmatrix} = \begin{bmatrix} \mathbf{v}_1 \\ \mathbf{v}_2 \end{bmatrix}, \tag{7}$$

and once solved the system $(I - \alpha P_1^T)\mathbf{y}_1 = \mathbf{v}_1$, the vector $\mathbf{y}_2$ is computed with only a matrix vector multiplication as $\mathbf{y}_2 = \mathbf{v}_2 + \alpha P_2^T \mathbf{y}_1$. Two recent papers, one by Eiron et al. [11] and the other by Langville and Meyer [20], arrive, with a different reasoning, to the same formulation of the problem, observing that a much smaller problem than the initial one has to be solved. The problem involving P_1, is still solved using the Power method, and they did not further investigate other reordering schemes which can increase the benefits of permutation strategies.

Note that once reordered R into a block triangular matrix, a natural way to solve the system 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

$$\begin{cases} \mathbf{y}_1 = & R_{11}^{-1}\mathbf{v}_1, \\ \mathbf{y}_i = & R_{ii}^{-1}\left(\mathbf{v}_i - \sum_{j=1}^{i-1} R_{ij}\mathbf{y}_j\right) \quad \text{for } i = 2, \ldots, m. \end{cases} \tag{8}$$

This requires the solution of m smaller linear systems, where the coefficient matrices are the diagonal blocks in the order they appear. Note that this strategy is always better than applying an iterative method to the whole matrix.

Moreover, for some iterative methods, it may happen that a permutation in the matrix reduces the spectral radius of the iteration matrix and hence the number of iterations needed to reach convergence. 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 rate of convergence of the Power method is also independent of matrix reordering, since it depends only on the spectral properties of the matrix and on the starting vector. A very nice attempt in this direction is given in [19] where it is shown how, reordering the matrix sorting the URLs lexicographically, 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.

The permutation strategies we propose have two different purposes. The first goal is to increase data locality and decrease, when possible, the spectral radius of the iteration matrix. The second one is to discover a block triangular structure in the Web matrix in order to apply block methods as described in equation (8).

Applying a permutation to a sparse matrix as well as finding the permutation which satisfies some desiderata, is a costly operation. However, it is often convenient to spend more efforts to decrease the running time of the solving method when the same matrix is used many times to solve different PageRank problems, as required for personalized web search [14] or in the case of heuristic for collusion-proof PageRank [26].

The permutations we considered are obtained by combining different elementary operations. A very effective reordering scheme, denoted by $\mathcal{B}$, is the one obtaine 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 by Cuthill and McKee [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, even if P is not symmetric, we examine other reordering schemes which are obtained by sorting the nodes in terms of their degree. In particular we consider the permutation which reorders the pages for decreasing out-degree; denoting this scheme as $\mathcal{O}_{\mathrm{d}}$ while

Full	Lower Block Triangular	Upper Block Triangular
$\mathcal{T}$	$\mathcal{T}\mathcal{B}$	$\mathcal{B}\mathcal{T}$
$\mathcal{O}_{\rm d}\mathcal{T}$	$\mathcal{O}_{\rm d}\mathcal{T}\mathcal{B}$	$\mathcal{O}_{\rm d}\mathcal{B}\mathcal{T}$
$\mathcal{O}_{\rm a}\mathcal{T}$	$\mathcal{O}_{\rm a}\mathcal{T}\mathcal{B}$	$\mathcal{O}_{\rm a}\mathcal{B}\mathcal{T}$
$\mathcal{I}_{\rm d}\mathcal{T}$	$\mathcal{I}_{\rm d}\mathcal{T}\mathcal{B}$	$\mathcal{I}_{\rm d}\mathcal{B}\mathcal{T}$
$\mathcal{I}_{\rm a}\mathcal{T}$	$\mathcal{I}_{\rm a}\mathcal{T}\mathcal{B}$	$\mathcal{I}_{\rm a}\mathcal{B}\mathcal{T}$

Figure 3: Web Matrix Permutation Taxonomy

the permutation $\mathcal{O}_{\rm a}$ sorts the pages of the Web matrix for ascending 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 ascending and descending in-degree; denoted by $\mathcal{I}_{\rm a}$ and $\mathcal{I}_{\rm d}$ respectively. Since $R = I - \alpha P^T$, our algorithms needs to compute the transpose of the Web matrix, we denote this operation by $\mathcal{T}$. The various operations can be combined obtaining different reorderings of the Web matrix as shown in Figure 3. In accordance with the taxonomy in Figure 3, we denote, for instance, by $R_{\mathcal{I}_{\rm d}\mathcal{T}\mathcal{B}} = I - \alpha\Pi(P)$, where the permuted matrix $\Pi(P)$ is obtained applying first the $\mathcal{I}_{\rm d}$ permutation, then transposing the matrix and applying finally the $\mathcal{B}$ permutation on the matrix reordered. The first column in Figure 3 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 4, we show a plot of the structure of a Web matrix rearranged according to each item of the above taxonomy. It is important to observe that on non symmetric matrices the BFS order of visit transforms the matrix in a lower block triangular form, with a number of diagonal blocks which is greater or equal to the number of strongly connected components. However, the number of diagonal blocks detected with a BFS depends very much on the starting node of the visit. For example in Figure 5 starting from node 1 we detect just a component, while starting from node 4 we get four separate components.

In order to have diagonal blocks of smaller size, and split the problem in smaller subproblems, we investigate other permutations further exploiting the reducibility of the matrix. Let $\mathcal{J}$ denote the reverse permutation, that is the permutation which assigns the index $n - i$ to node i-th. To some of the shapes in Figure 4, we can apply another $\mathcal{B}$ operation, sometimes after a reversion of the matrix with the operator $\mathcal{J}$. We obtain the shapes in Figure 6.

Note that the smallest size of the largest diagonal block is achieved for $R_{\mathcal{O}_{\rm a}\mathcal{B}\mathcal{T}\mathcal{J}\mathcal{B}}$. The size of the largest component in $R_{\mathcal{O}_{\rm a}\mathcal{B}\mathcal{T}\mathcal{J}\mathcal{B}}$ is something more than 13 Million, while for the permutations in Figure 3 which uses only a BFS, we were able to reach a size of around 17 Million.

We adopted ad hoc numerical methods for dealing with the different shapes of matrices in Figure 4 and 6. 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 $R_{\mathcal{O}_{\rm d}\mathcal{T}} = JR_{\mathcal{O}_{\rm a}\mathcal{T}}J^T$ and $R_{\mathcal{I}_{\rm d}\mathcal{T}} = JR_{\mathcal{I}_{\rm a}\mathcal{T}}J^T$ where J is the anti-diagonal matrix, that is $J_{ij} = 1$ iff $i + j = n + 1$. This means

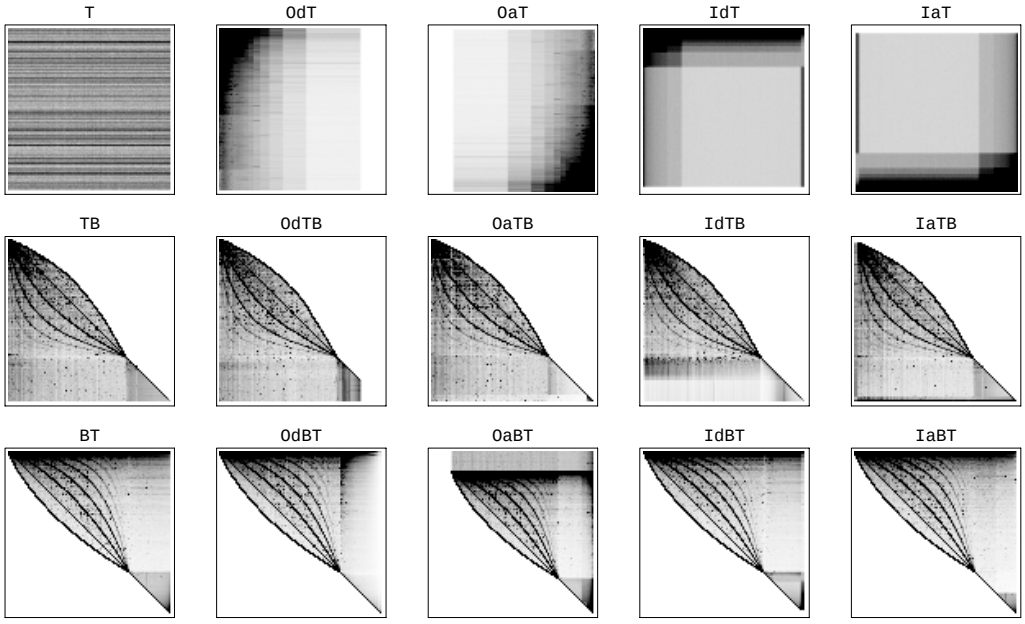
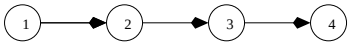


Figure 4: Different shapes obtained rearranging the Web matrix P in accordance to the taxonomy. First row represents full matrices; second and third lower and upper block triangular matrices. Web Graph is made of 24 million nodes and 100 million links.



$$R_1 = \begin{pmatrix} 1 & -\alpha & 0 & 0 \\ 0 & 1 & -\alpha & 0 \\ 0 & 0 & 1 & -\alpha \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R_4 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -\alpha & 1 & 0 & 0 \\ 0 & -\alpha & 1 & 0 \\ 0 & 0 & -\alpha & 1 \end{pmatrix}.$$

Figure 5: An example of a connected graph. Applying a BFS visit starting with node 1 one detect just a component and R_1 is in unreduced form, while starting with node 4 we get four different components and R_4 is fully reduced.

that applying Gauss-Seidel to $R_{\mathcal{O}_{\mathrm{d}}\mathcal{T}}$ ($R_{\mathcal{I}_{\mathrm{d}}\mathcal{T}}$) is the same that applying Reverse Gauss-Seidel to $R_{\mathcal{O}_{\mathrm{a}}\mathcal{T}}$ ($R_{\mathcal{I}_{\mathrm{a}}\mathcal{T}}$).

The shapes of some matrices in Figures 4 and 6, encourage to exploit the matrix reducibility experimenting with block methods. Moreover, also the matrix $R_{\mathcal{O}_{\mathrm{d}}\mathcal{T}}$ is lower block triangular, since it separates non-dangling nodes from dangling nodes. This matrix is a particular case of the one considered in equation (7).

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 7. In Section 6 we report the experimental results obtained by applying each method in Figure 7 to all the suitable matrices in Figure 4 and in Figure 6.

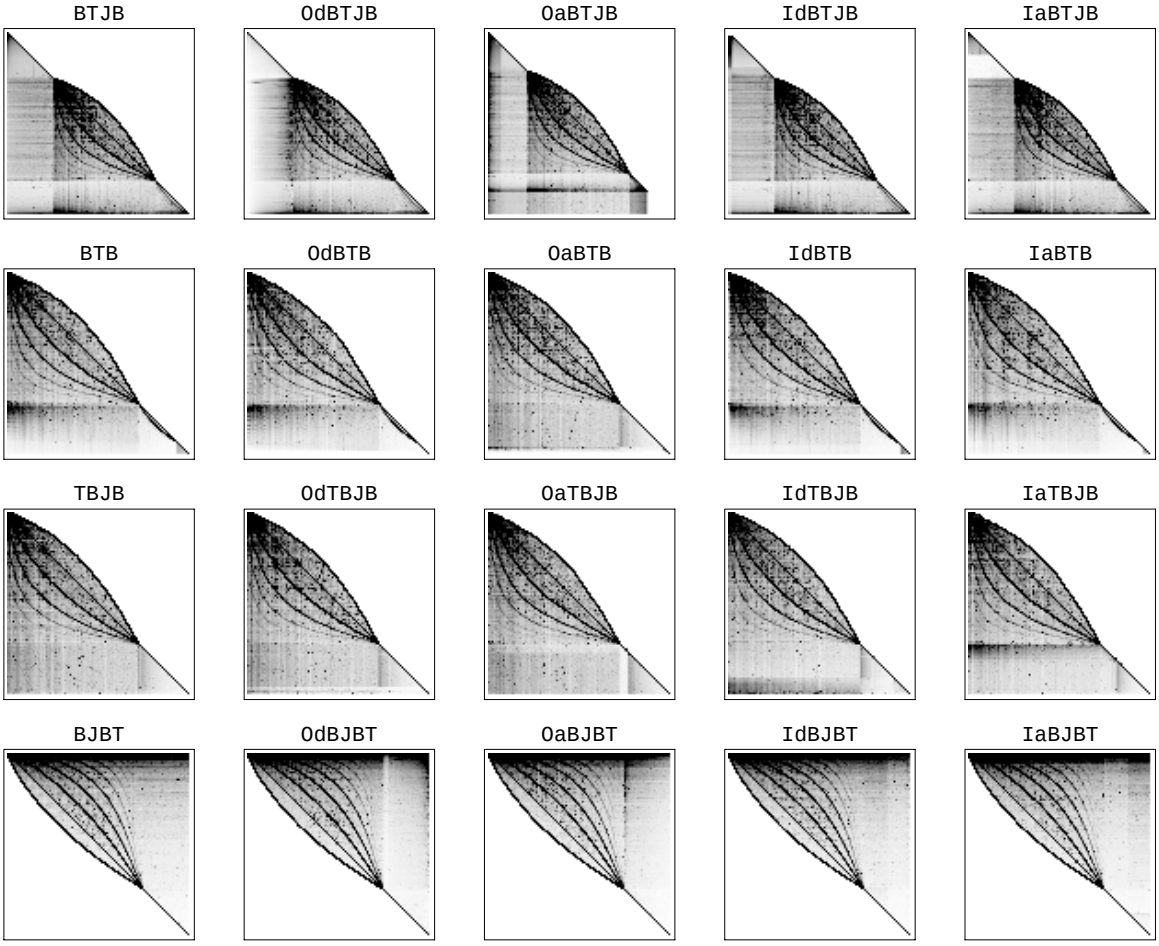


Figure 6: Different shapes obtained rearranging the Web matrix P made of 24 million nodes and 100 million links according to permutations involving two BFS visits.

Scalar methods	shapes
PM	all
Jac	all
GS	all
RGS	all

Block methods	shapes
LB	$R_{*\mathcal{B}}$ and $R_{\mathcal{O}_{\mathrm{d}}\mathcal{T}}$
LBR	$R_{*\mathcal{B}}$ and $R_{\mathcal{O}_{\mathrm{d}}\mathcal{T}}$
UB	$R_{*\mathcal{T}}$
UBR	$R_{*\mathcal{T}}$

Figure 7: 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 3. 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.

Name	PM	GS	RGS	LB/UB	LBR/UBR
$\mathcal{T}$	3454 33093 152	1774 19957 92	1818 20391 94	– – –	– – –
$\mathcal{O}_{\mathrm{d}}\mathcal{T}$	2934 33093 152	1544 20825 85	1477 19957 104	1451 19680 96	1397 18860 92
$\mathcal{I}_{\mathrm{d}}\mathcal{T}$	3315 33309 153	1840 21259 98	1735 19957 92	– – –	– – –
$\mathcal{TB}$	1386 32876 151	606 21910 101	519 18439 85	401 16953 100	359 15053 82
$\mathcal{O}_{\mathrm{d}}\mathcal{TB}$	1383 33093 152	582 21476 99	505 18439 85	392 17486 97	369 15968 85
$\mathcal{O}_{\mathrm{a}}\mathcal{TB}$	1353 32876 151	610 23645 109	440 16920 78	424 18856 106	**315 13789 75**
$\mathcal{I}_{\mathrm{d}}\mathcal{TB}$	1361 33309 153	561 21259 98	511 19090 88	385 17196 98	380 16414 87
$\mathcal{I}_{\mathrm{a}}\mathcal{TB}$	1392 32876 151	619 22343 105	**450 16270 75**	400 17972 100	314 13905 75
$\mathcal{BT}$	1394 33093 152	522 18439 85	640 22560 104	379 15545 85	479 19003 104
$\mathcal{O}_{\mathrm{d}}\mathcal{BT}$	1341 33309 153	507 18873 87	579 21693 100	398 15937 87	466 18312 100
$\mathcal{O}_{\mathrm{a}}\mathcal{BT}$	1511 33093 152	591 18873 87	680 21693 100	357 15128 87	413 17387 100
$\mathcal{I}_{\mathrm{d}}\mathcal{BT}$	1408 33093 152	554 19306 89	626 21693 100	397 16075 88	450 18265 100
$\mathcal{I}_{\mathrm{a}}\mathcal{BT}$	1351 33093 152	497 18439 85	575 21693 100	386 15564 85	447 18310 100

Figure 8: Experimental Results: in columns are listen the numerical methods analyzed and the rows describe some of the permutations applied to the matrix R. Each cell represents the running time in seconds, the number of Mflops and the number of iterations taken by the solving methods. 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 in terms of Mflops for scalar and block methods.

6 Experimental Results

We tested the approaches discussed in previous sections using a Web Graphs obtained as 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 (see http://www.nutch.org/). We run our experiments on a PC with a Pentium IV 3GHz, 2.0GB of memory and 512Kb of L2 cache. A stopping criterion of 10^{-7} is imposed on the absolute difference between the vectors computed in two successive iterations. In order to have a fair comparison for the Power method the tolerance has been changed "a posteriori" to obtain the same absolute error of the other methods. In fact, in the case of Power method, the PageRank vector has 1-norm equal to one while for the other methods we do not scale the vector at each step.

In Figure 8 we report the running time in seconds, the Mflops and the number of iterations for each combination of solving and reordering methods described in Figure 7 on the matrices of Figure 4. We believe that the number of iterations and the number of floating point iterations of a method are more fair measures than the running time in seconds which is more implementation-dependent. However, the running time is the only factor accounting for an increase in data-locality when a permutation of the matrix does not change the number of iterations.

Some cells of equation 8 are empty since there are methods suitable only on particular shapes. Moreover, in Figure 8 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. However, the measure of the gain using GS rather than Jac can be obtained from Figure 10. 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. Although we implemented `PM` using the optimizations suggested in [23, 13] this method requires a slight greater number of operations. Using the results of Theorem 4.2, 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 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 8. 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 48% in running time. These improvements are striking when the system matrix is permuted. The best performance of scalar methods is obtained using the $\mathcal{I}_{\mathrm{a}}\mathcal{T}\mathcal{B}$ 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 87%.

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 out to be misleading. In fact, for our Web matrix `RGS` works better on lower block triangular matrices and `GS` works better on quasi-upper triangular matrices. Even better results are obtained using block methods. `LB` applied to $R_{\mathcal{O}_{\mathrm{d}}\mathcal{T}}$ 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 in equation (7). The best result is obtained for the $\mathcal{O}_{\mathrm{a}}\mathcal{T}\mathcal{B}$ permutation when the `LBR` solving method is applied. In this case, we have a reduction of 58% in Mflops and of 90% 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.

We applied each of the methods in 7 also on the shapes in Figure 6 obtained with two BFS visits. The results are in Table 9 From Figure 9 we see that we have a further gain when the matrix is split in a greater number of blocks. In fact we have a reduction up to 92/

The results given in Figure 8 and 9 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 by Broder et al. in [5], where less of 4 minutes are taken on a Web Graph with 100 million nodes and it is however 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 10 and 11.

7 Conclusion

The ever-growing size of the Web graph implies that the value and the importance of fast methods for Web ranking is going to rise in the future. Moreover, the even growing interest toward personalized PageRank justifies an effort in "pre-processing" the Web graph matrix in order to compute faster the many PageRank vectors needed.

The problem of PageRank computation can be easily be viewed as a dense linear system. We

Name	GS	RGS	LB/UB	LBR/UBR
$\mathcal{TBJB}$	565 21476 99	488 18439 85	402 17534 99	327 13980 76
$\mathcal{O}_{\mathrm{d}}\mathcal{TBJB}$	613 23645 109	445 16920 78	426 18853 106	318 13790 76
$\mathcal{O}_{\mathrm{a}}\mathcal{TBJB}$	547 21476 99	478 18656 86	390 17891 99	362 16193 86
$\mathcal{I}_{\mathrm{d}}\mathcal{TBJB}$	568 22127 102	447 17354 80	407 18458 100	321 14326 78
$\mathcal{I}_{\mathrm{a}}\mathcal{TBJB}$	547 21476 99	487 18873 87	375 17175 99	368 16389 87
$\mathcal{BTJB}$	605 22560 104	443 16486 76	333 15905 101	**246 11617 73**
$\mathcal{O}_{\mathrm{d}}\mathcal{BTJB}$	577 22560 104	**415 16270 75**	336 15915 101	252 11779 73
$\mathcal{O}_{\mathrm{a}}\mathcal{BTJB}$	655 21693 100	544 18005 83	365 15896 99	280 12231 82
$\mathcal{I}_{\mathrm{d}}\mathcal{BTJB}$	623 22560 104	447 16270 75	328 15896 101	250 11921 74
$\mathcal{I}_{\mathrm{a}}\mathcal{BTJB}$	553 21693 100	**415 16270 75**	328 15919 99	245 11624 72
$\mathcal{BTB}$	510 21259 98	483 19957 92	382 17354 98	370 16410 91
$\mathcal{O}_{\mathrm{d}}\mathcal{BTB}$	519 21476 99	487 19957 92	383 16800 98	376 16403 91
$\mathcal{O}_{\mathrm{a}}\mathcal{BTB}$	551 21259 98	522 19957 92	370 17168 98	367 16404 91
$\mathcal{I}_{\mathrm{d}}\mathcal{BTB}$	515 21476 99	485 19957 92	368 16996 98	369 16410 91
$\mathcal{I}_{\mathrm{a}}\mathcal{BTB}$	509 21259 98	480 19957 92	366 16790 98	366 16396 91
$\mathcal{BJBT}$	529 20174 93	570 21693 100	413 16813 92	452 18286 100
$\mathcal{O}_{\mathrm{d}}\mathcal{BJBT}$	491 18656 86	566 21693 100	385 15545 85	450 18281 100
$\mathcal{O}_{\mathrm{a}}\mathcal{BJBT}$	501 19090 88	588 22560 104	399 16105 88	464 18841 103
$\mathcal{I}_{\mathrm{d}}\mathcal{BJBT}$	486 18439 85	563 21693 100	387 15565 85	449 18310 100
$\mathcal{I}_{\mathrm{a}}\mathcal{BJBT}$	504 19306 89	565 21693 100	398 16081 88	452 18269 100

Figure 9: Experimental Results on the shapes in figure 6. 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 and the number of iterations taken by the solving methods. In bold we highlight our best results in terms of Mflops for scalar and block methods. We omit the values obtained with the Jacobi method since they are almost unaffected by matrix permutations and can be found in Figure 8.

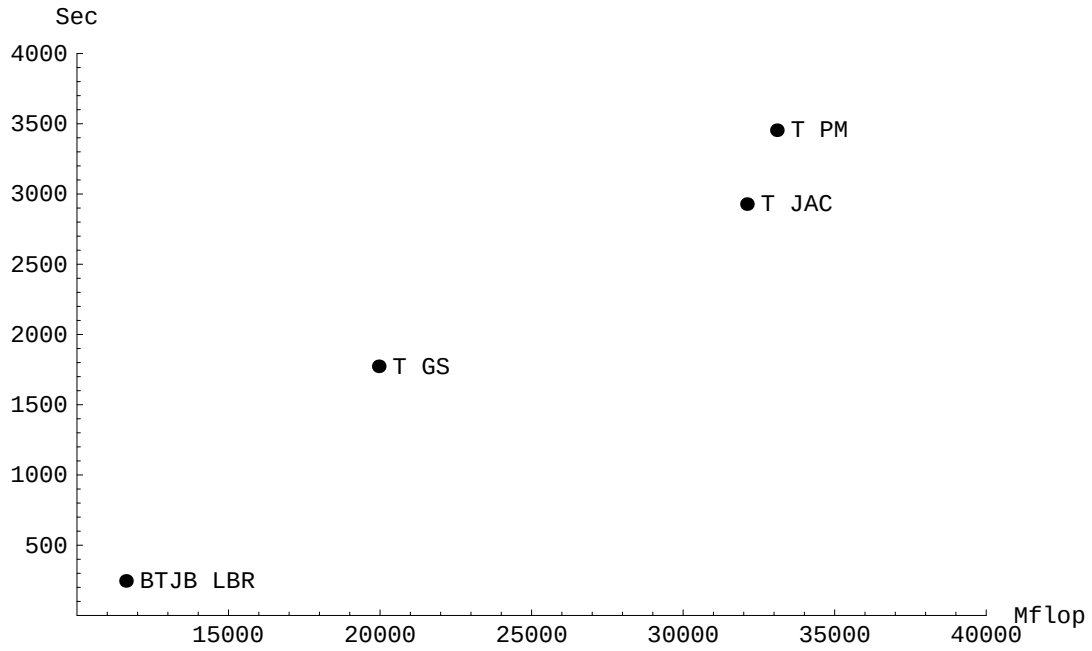


Figure 10: A plot of few results of Figure 8 and 9. On the x-axis the number of Mflops and on the y-axis the running time in seconds. Each point is labeled with the permutation applied and the solving method used. Power, Jacobi and Gauss-Seidel methods applying any permutation are compared with the best result.

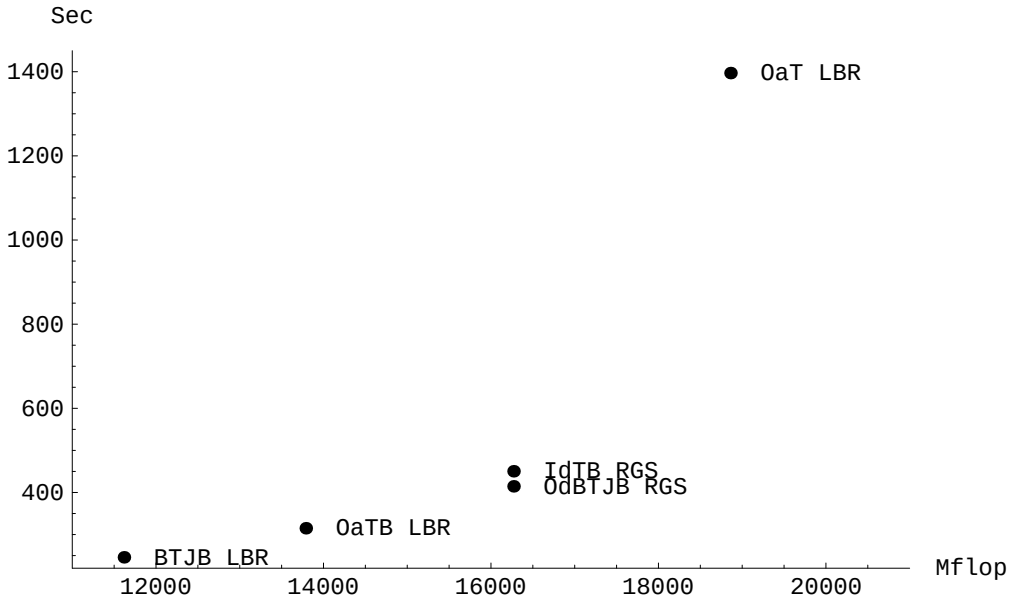


Figure 11: A plot of the more interesting results of Figure 8 and 9. On the x-axis the number of Mflops and on the y-axis the running time in seconds. Each point is labeled with the permutation applied and the solving method used. The best performance of a method, which permutes the node in dangling-nondangling without applying any BFS visit, is plotted with the best block and scalar methods which use one or two BFS visits.

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 by Arasu et al. 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 eigenpair interpretation. Dealing with a sparse linear system opens the way to exploit the reducibility of the web matrix by composing opportunely some permutations strategies 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 65% in Mflops and of 92% 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 of the Mflops used by the Power method. The previous better improvement over the Power method is due to Lee et al. in [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 especially when dealing with personalized PageRank.

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 *Proc. of the 11th WWW Conf.*, 2002.

[2] M. Bianchini, M. Gori, and F. Scarselli. Inside PageRank. *ACM Trans. on Internet Technology*, 2004. to appear.

[3] P. Boldi and S. Vigna. WebGraph framework I: Compression techniques. In *Proc. of the 23th Int. WWW Conf.*, 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] A. L. Buchsbaum, M. Goldwasser, S. Venkatasubramanian, and J. Westbrook. On external memory graph traversal. In *SODA*, pages 859–860, 2000.

[7] Y. Chen, Q. Gan, and T. Suel. I/O-efficient techniques for computing Pagerank. In *Proc. of the 11th WWW Conf.*, 2002.

[8] J. Cho and S. Roy. Impact of Web search engines on page popularity. In *Proc. of the 13th WWW Conf.*, 2004.

[9] E. Cuthill and J. McKee. Reducing the bandwidth of sparse symmetric matrices. In *Proc. 24th Nat. Conf. ACM*, pages 157–172, 1969.

[10] 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, 2000.

[11] N. Eiron, S. McCurley, and J. A. Tomlin. Ranking the web frontier. In *Proc. of 13th WWW Conf.*, 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 *Proc. of the 11th WWW Conf.*, 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 *Proc. of the 12th WWW Conf.*, 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 *Proc. of 12th. WWW Conf.*, 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. Technical report, 2004.

[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, 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.

[26] H. Zhang and A. Goel and R. Govindan and K. Mason and B. Van Roy. Making Eigenvector-Based Reputation System Robust to Collusion. In *Proceedings of Third International Workshop on Algorithms and Models for the Web-Graph, WAW*, pages 92–104, 2004.