Electronic Transactions on Numerical Analysis.
Volume 38, pp. 69-97, 2011.

ISSN 1068-9613.

ETNA
Kent State University
http://etna.math.kent.edu

STEADY–STATE ANALYSIS OF GOOGLE–LIKE STOCHASTIC MATRICES WITH BLOCK ITERATIVE METHODS*

TUĞRUL DAYAR† AND GÖKÇE N. NOYAN‡

Abstract. A Google–like matrix is a positive stochastic matrix given by a convex combination of a sparse, nonnegative matrix and a particular rank one matrix. Google itself uses the steady–state vector of a large matrix of this form to help order web pages in a search engine. We investigate the computation of the steady–state vectors of such matrices using block iterative methods. The block partitionings considered include those based on block triangular form and those having triangular diagonal blocks obtained using cutsets. Numerical results show that block Gauss–Seidel with partitionings based on block triangular form is most often the best approach. However, there are cases in which a block partitioning with triangular diagonal blocks is better, and the Gauss–Seidel method is usually competitive.

Key words. Google, PageRank, stochastic matrices, power method, block iterative methods, partitionings, cutsets, triangular blocks

AMS subject classifications. 60J10, 65F10, 65F50, 65B99

1. Introduction.

We consider positive stochastic matrices of the form

$$S = R + uv,$$

where $R \in \mathbb{R}^{n\times n}$ is a nonnegative and sparse square matrix, possibly reducible with some zero rows, $u \in \mathbb{R}^{n\times 1}$ is a nonnegative column vector, and $v \in \mathbb{R}^{1\times n}$ is a nonnegative row vector [30]. The reason behind v representing a row vector will soon become clear. Such stochastic matrices arise in the page ranking algorithm, PageRank [5], of Google (hence, the term Google–like). The objective therein is to compute the steady–state probability distribution row vector $\pi \in \mathbb{R}^{1\times n}$ of S in

$$\pi S = \pi, \ \pi e = 1,$$

where e is the column vector of ones. The first difficulty lies in that although S does not have any zero elements, one must make every effort to avoid fill–in and work in sparse storage since R is a sparse matrix and uv is an outer product. The second difficulty is related to the reducibility of R, since an arbitrary partitioning of a reducible matrix will not yield irreducible diagonal blocks, and hence, care must be exercised when employing block iterative solvers as we shall see. These rule out direct methods such as Gaussian elimination (GE) and iterative methods which require relatively large memory such as preconditioned Krylov subspace methods.

Now, let $P \in \mathbb{R}^{n\times n}$ be the transition probability matrix associated with the hyperlink structure of the web of pages to be ranked and $\alpha \in (0, 1)$ be the convex combination parameter used to obtain a positive stochastic matrix S so that it is ergodic and therefore can be analyzed for its steady–state [35]. In the PageRank algorithm,

$$R = \alpha P, \ u = e - Re,$$

*Received April 12, 2010. Accepted for publication January 17, 2011. Published online March 28, 2011. Recommended by M. Benzi.

†Department of Computer Engineering, Bilkent University, TR–06800 Bilkent, Ankara, Turkey (tugrul@cs.bilkent.edu.tr).

‡Undersecretariat for Defence Industries, Ziyabey Caddesi, 21 Sokak, No.4, TR–06520 Balgat, Ankara, Turkey (gnnoyan@ssm.gov.tr).

and v is the nonnegative personalization probability distribution row vector satisfying $ve = 1$. Note that P may have zero rows corresponding to dangling nodes; i.e., web pages without any outgoing hyperlinks. An equivalent formulation and an extensive discussion on PageRank can be found in [24]. The PageRank algorithm computes π iteratively using the power method. And ranking pages corresponds to sorting the pages (i.e., states of the underlying discrete–time Markov chain, DTMC) according to their steady–state probabilities. The problem is introduced during the Stanford Digital Library Technologies project (now known as The Stanford WebBase Project [38]). Since web matrices are extremely large and always changing, computation lasts long and needs to be repeated.

The rate of convergence of the power method depends on the subdominant eigenvalue of S. This eigenvalue is equal to α for reducible P and strictly less than α otherwise [18]. Convergence takes place at the rate by which powers of α approach zero. Thus, convergence is faster as α becomes smaller. However, the smaller α is, the higher the contribution of the second term uv and the lesser the hyperlink structure of the web in R influences page rankings. Slightly different α values can produce very different PageRank vectors, and as α approaches one, sensitivity issues begin to arise [33]. Brin and Page, the founders of Google, use $\alpha = 0.85$ (in other words, a teleportation probability, $(1 - \alpha)$, of 0.15), and for tolerance levels measured by residual norms ranging from 10^{-3} to 10^{-7}, they report convergence within 50 to 100 power iterations [5]. We remark that, normally $v = e^T/n$ (i.e., the uniform distribution) is used. However, when the web surfer has preferences and is therefore biased, a personalization vector other than the uniform distribution must be used. Hence, ideally the problem needs to be solved multiple times for different personalization vectors.

A number of improvements are made over the power method for the PageRank algorithm. Here, we mention some that are relevant to our work in this paper. The work in [1] suggests using the most recently computed values of the approximate solution in the same iteration as in the Gauss–Seidel (GS) method. This approach, which is classified under sequential updates by the framework in [27], is shown to bring in savings of about one half in the number of iterations with respect to the power method. The power method is also improved in [20], this time using quadratic extrapolation, but the improvement is fairly sensitive to how frequently the extrapolation strategy is invoked. A restarted variant of the Arnoldi algorithm is investigated in [17] for computing PageRank. Although timing results and exact memory requirements are not provided, the results promise considerable computational savings over the power method at the expense of relatively large memory requirements (since a relatively large number of solution vectors of length n need to be stored). A more recent study [16] shows that improvements in number of matrix–vector multiplications (which is essentially the number of power iterations) are possible with inner–outer iterations with modest memory requirements.

Another group of work relates to the way in which the web pages of interest are obtained by crawling. By sorting the pages according to their addresses and then parsing the addresses into separate fields, the authors in [21] have looked into block partitionings of web matrices in which each diagonal block represents the hyperlinks among pages within a domain. Domains in turn can be partitioned into hosts, thus resulting in the view of nested blocks and a method based on iteratively analyzing the diagonal blocks in isolation, aggregating them, and solving the aggreated system. This approach, which is classified under reiterated updates by the framework in [27], is shown to yield savings of about one half in the number of iterations with respect to the power method applied to the original order of pages, although the savings in time do not compare as favorably with respect to the power method applied to the sorted order of pages. An approach based on aggregation is also followed in [6], where a fixed

ETNA
Kent State University
http://etna.math.kent.edu

solution is assumed for the diagonal blocks associated with hosts, thereby resulting in a faster but approximative method.

In this paper, we do not assume any knowledge about addresses associated with web pages, take a sparse matrix view, and present the results of numerical experiments with a software tool [10, 11] for the steady–state analysis of Google–like matrices in a sequential setting; i.e., on a computer with a single computational core. The objective is to systematically compare and contrast different sparse iterative solvers. The tool can also be used to analyze irreducible DTMCs as in [34] for their steady–state distribution by setting $\alpha = 1$. There are eight solvers available. These are power (POWER), power with quadratic extrapolation (QPOWER), Jacobi over–relaxation (JOR), successive over–relaxation (SOR), block JOR (BJOR), block SOR (BSOR), iterative aggregation–disaggregation (IAD) with BJOR disaggregation step (IAD_BJOR), and IAD with BSOR disaggregation step (IAD_BSOR). The JOR and SOR solvers become respectively the Jacobi (J) and GS solvers for value 1 of the relaxation parameter. The motivation of the study is to identify those sparse solvers that decrease the iteration counts and solution times with respect to POWER without increasing the memory requirements too much. The contribution of the results to the literature are in the understanding of the type of partitionings to be recommended with block iterative solvers for Google–like stochastic matrices and the benefits obtained by employing them.

It is known that Tarjan's algorithm [36] can be used to symmetrically permute a matrix of order n with a zero–free diagonal to block triangular form in which the diagonal blocks are irreducible [14]. Its time complexity is $O(n) + O(\tau)$, where τ is the number of nonzero off–diagonal elements. In the context of web matrices, this permutation is first noted in [1]. The permutation is later pursued in [31] on parts of two web matrices and some preliminary results have been obtained with an IAD_BJ like method. However, the implementation has not been done in sparse storage and only iteration counts are reported, whereas, timing results are vital for this kind of study. The study in [13] is another one which considers symmetric permutations of web matrices to accelerate convergence of solution methods and is the most relevant one to the work in this paper. Therein, the effect of using breadth first traversal of the nodes of the web graph to generate a permutation of the corresponding matrix to block triangular form is investigated together with sorting the nodes for decreasing/increasing in–/out–degrees. Experiments are conducted on one web matrix with one value of α using power, Jacobi, GS, backward GS, and block GS (BGS) methods. The setup times to obtain the permutations used are not reported. Nevertheless, results on the web matrix suggest savings of about a half with BGS in the number of iterations and substantial improvements in solution time with respect to the power method. These two approaches could be classified under reiterated updates by the framework in [27] as well.

In this paper, we use the sparse implementation of Tarjan's algorithm in [37] to obtain a block triangular form and partitionings having triangular diagonal blocks that follow from there. To the best of our knowledge, efficient algorithms that search for the strongly connected components of graphs (which correspond to irreducible diagonal blocks of matrices in block triangular form) use depth first traversal of the nodes as in Tarjan's algorithm. Through a systematic study on multiple benchmark matrices with different values of α, we investigate the merit of various block partitionings. We do not consider any optimization in our code, and treat dangling nodes by considering the hyperlinks to them and not by penalizing [15], recursively eliminating [25], or aggregating [19] them. Hence, the timing results in our work could be improved further. Nevertheless, our results support those in [13], but also show that there are cases which benefit significantly from triangular diagonal blocks, and GS is also competitive.

In the next section, we discuss the solvers. In Section 3, we introduce the partitionings

ETNA
Kent State University
http://etna.math.kent.edu

considered with the block iterative solvers. Section 4 is about the benchmark matrices and their properties. Section 5 provides the results of numerical experiments. In Section 6, we conclude.

2. The solvers. Consider the following example web matrix in [24].

EXAMPLE 2.1. P corresponds to the web graph of 6 pages in Figure 2.1, where probabilities of outgoing links are uniformly distributed and page 2 does not have any outgoing links; i.e., it is a dangling node. P is reducible with the state space partitions $\{1,3\}$, $\{2\}$, $\{4,5,6\}$ forming irreducible subsets of states as in Figure 2.1.

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} \;\;1\;\; & \;\;2\;\; & \;\;3\;\; & \;\;4\;\; & \;\;5\;\; & \;\;6\;\; \end{array} \\ \left[\begin{array}{cccccc} 0 & 1/2 & 1/2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1/3 & 1/3 & 0 & 0 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 1/2 & 1/2 \\ 0 & 0 & 0 & 1/2 & 0 & 1/2 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{array}\right] \end{array}$$

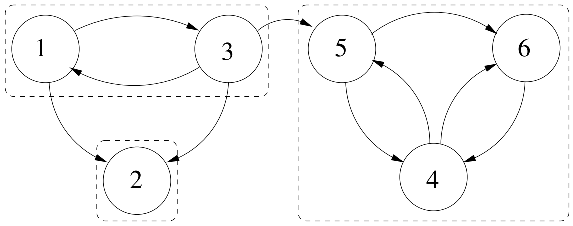


FIG. 2.1. *Web graph of the example.*

2.1. Power method. Given $\pi^{(0)} > 0$ and $\pi^{(0)}e = 1$, the POWER iteration

$$\pi^{(k+1)} = \pi^{(k)}R + \pi^{(k)}uv \quad \text{for } k = 0, 1, \ldots$$

can be implemented with one vector–matrix multiplication using R and two level–1 operations; i.e., dot–product and saxpy — multiplication of a vector with a scalar and the addition of another vector. Convergence takes place at rate by which α^k goes to 0. The smaller α is, the lesser the effect of the hyperlink structure of the web.

It is reported that by periodically applying quadratic extrapolation for values of α close to 1, the convergence of POWER can be accelerated [20]. However, the improvement is fairly sensitive to how frequently the extrapolation strategy is invoked, and therefore we do not consider QPOWER further in this paper.

2.2. Block iterative methods. Let

$$B = R^T - I$$

and E be a permutation matrix so that

$$A = E(B + v^T u^T)E^T, \; x = E\pi^T,$$

and consider the block splitting $A = D - L - U = M - N$ for $Ax = 0$ (i.e., $E(S^T - I)E^T E\pi^T = 0$), where

$$D = \text{diag}(A_{1,1}, A_{2,2}, \ldots, A_{J,J}),$$

ETNA
Kent State University
http://etna.math.kent.edu

$$-L = \begin{bmatrix} & & & \\ A_{2,1} & & & \\ A_{3,1} & A_{3,2} & & \\ \vdots & \vdots & \ddots & \\ A_{J,1} & A_{J,2} & \cdots & A_{J,J-1} \end{bmatrix}, \quad -U = \begin{bmatrix} & A_{1,2} & A_{1,3} & \cdots & A_{1,J} \\ & & A_{2,3} & \cdots & A_{2,J} \\ & & & \ddots & \vdots \\ & & & & A_{J-1,J} \\ & & & & \end{bmatrix},$$

$\mathrm{diag}(\cdot)$ denotes a diagonal matrix with its argument appearing along the diagonal, J, the number of blocks along the diagonal satisfies $1 < J \leq n$, $A_{i,j} \in \mathbb{R}^{n_i \times n_j}$ for $i, j = 1, 2, \ldots, J$ so that $n = \sum_{j=1}^{J} n_j$, $A_{j,j}$ has nz_j nonzero elements, and M is nonsingular. We remark that A is a symmetric permutation of $S^T - I$, and that E is neither explicitly generated nor stored but rather obtained using integer permutation vectors as we shall see.

Given $x^{(0)} > 0$, $e^T x^{(0)} = 1$, and the relaxation parameter $\omega \in (0, 2)$, the iteration

$$Mx^{(k+1)} = Nx^{(k)} \quad \text{for} \quad k = 0, 1, \ldots,$$

where

$$M = D/\omega, \; N = (1 - \omega)D/\omega + L + U$$

is (block) Jacobi over–relaxation, (B)JOR, and

$$M = D/\omega - L, \; N = (1 - \omega)D/\omega + U$$

is (block) successive over–relaxation, (B)SOR. These become point methods when $J = n$. The convergence for under–relaxation (i.e., $\omega \in (0, 1)$) is well known; in this case, it is also monotonic since the iteration matrices are positive. This is due a result in [2, pp. 270–271] and is proved in a more general setting by Theorem 4.16 in [7]. For $\omega = 1$, the iteration matrices are nonnegative. The Jacobi iteration matrix has a zero diagonal and the GS iteration matrix has a zero first column; otherwise, all other elements of these two iteration matrices are positive. Since both iteration matrices have the solution vector as their fixed point (and therefore, an eigenvalue of 1), a sufficient condition for convergence is to show that both iteration matrices do not have other eigenvalues of magnitude 1. Indeed they are as such: the condition for the Jacobi iteration matrix follows for $n > 2$ from the positivity of its off–diagonal; the condition for the GS iteration matrix follows from the fact that it has a positive submatrix of order $(n - 1)$ which is accessible from the excluded first row.

Block iterative methods can be viewed as preconditioned power iterations, where the preconditioning matrix is M [35]. When combined with aggregation steps, they become iterative aggregation–disaggregation (IAD) with BJOR disaggregation step (IAD_BJOR) and BSOR disaggregation step (IAD_BSOR). Because there is an (outer) iteration specified by k, and for each value of k, one needs to solve J subsystems of linear equations directly or iteratively, methods discussed in this subsection are sometimes viewed as being two–level (see also [29]). Although we have implemented and experimented with IAD type methods, they do not yield competitive solvers for Google–like stochastic matrices. Hence, we do not discuss them further in this paper.

In the next section, we present various partitionings that can be used with block iterative solvers.

3. Partitionings for block iterative solvers. Although $S > 0$, one must work in sparse storage since R is sparse and uv is an outer product; i.e., rank–1 update on R. When R (hence, P) is reducible, an arbitrary partitioning will not yield irreducible diagonal blocks in D. In order to devise effective block iterative solvers, the objective must be to obtain a

ETNA
Kent State University
http://etna.math.kent.edu

partitioning of B in which J is relatively low and it is relatively easy to solve the diagonal blocks.

EXAMPLE 3.1. Let us now consider the nonzero structure of the matrix P of six web pages in Section 2, where an X represents a nonzero value. Recall that $B = \alpha P^T - I$. However, scaling with α does not change the nonzero structure of P^T, and we have

$$P = \begin{array}{c} 1\\2\\3\\4\\5\\6 \end{array} \left[\begin{array}{cccccc} & X & X & & & \\ & & & & & \\ X & X & & & X & \\ & & & & X & X \\ & & & X & & X \\ & & & X & & \end{array} \right], \quad B = \begin{array}{c} 1\\2\\3\\4\\5\\6 \end{array} \left[\begin{array}{cccccc} X & & X & & & \\ X & X & X & & & \\ X & & X & & & \\ & & & X & X & X \\ & & X & X & X & \\ & & & X & X & X \end{array} \right].$$

Now, without loss of generality, let us assume that B is permuted to block lower–triangular form having nb irreducible diagonal blocks using Tarjan's algorithm [14] as in

$$F = \begin{bmatrix} F_{1,1} & & & \\ F_{2,1} & F_{2,2} & & \\ \vdots & \vdots & \ddots & \\ F_{nb,1} & F_{nb,2} & \cdots & F_{nb,nb} \end{bmatrix}.$$

We remark that the irreducible diagonal blocks $F_{k,k}$ for $k = 1, 2, \ldots, nb$ have zero–free diagonals due to the definition of B and the states incident on each irreducible diagonal block are only fixed up to a permutation.

EXAMPLE 3.1. (Continued) After applying Tarjan's algorithm to obtain a block lower–triangular matrix for our example, we have the permutation vector $[1\ 3\ 2\ 4\ 5\ 6]^T$. If we symmetrically permute B with the permutation matrix $[e_1\ e_3\ e_2\ e_4\ e_5\ e_6]^T$, where e_l denotes the lth principal axis vector (i.e., lth column of I), we get the nonzero structure

$$F = \begin{array}{c} 1\\3\\2\\4\\5\\6 \end{array} \left[\begin{array}{cc|c|ccc} X & X & & & & \\ X & X & & & & \\ \hline X & X & X & & & \\ \hline & & & X & X & X \\ & X & & X & X & \\ & & & X & X & X \end{array} \right],$$

where $nb = 3$. The irreducible diagonal blocks correspond to state space partitions $\{1, 3\}$, $\{2\}$, and $\{4, 5, 6\}$, and they are respectively of order two, one, and three.

It is possible to compute a permutation matrix $Q_{k,k}$ for each irreducible diagonal block $F_{k,k}$ [9] such that

$$Q_{k,k} F_{k,k} Q_{k,k}^T = \begin{array}{c} \begin{array}{cc} n_{C_{k,k}} & n_{T_{k,k}} \end{array} \\ \begin{bmatrix} C_{k,k} & Y_{k,k} \\ Z_{k,k} & T_{k,k} \end{bmatrix} \end{array} \begin{array}{c} \\ n_{C_{k,k}} \\ n_{T_{k,k}} \end{array},$$

where $C_{k,k} \in \mathbb{R}^{n_{C_{k,k}} \times n_{C_{k,k}}}$, $T_{k,k} \in \mathbb{R}^{n_{T_{k,k}} \times n_{T_{k,k}}}$, and $T_{k,k}$ is triangular. Note that $T_{k,k}$ is necessarily nonsingular. It is clear that the smaller the order of submatrix $C_{k,k}$ is, the larger the order of submatrix $T_{k,k}$ becomes. Since a triangular block can be solved exactly by using substitution (together with the Sherman–Morrison formula [28] since the block becomes

ETNA
Kent State University
http://etna.math.kent.edu

positive due to the addition of the outer product term), it is useful to obtain a larger triangular block. We remark that, under the same permutation, the off–diagonal block $F_{k,l}$ gets permuted as

$$Q_{k,k}F_{k,l}Q_{l,l}^T = \begin{array}{c c} \begin{array}{cc} n_{C_{l,l}} & n_{T_{l,l}} \end{array} & \\ \left[\begin{array}{cc} C_{k,l} & Y_{k,l} \\ Z_{k,l} & T_{k,l} \end{array} \right] & \begin{array}{c} n_{C_{k,k}} \\ n_{T_{k,k}} \end{array} \end{array},$$

where $C_{k,l} \in \mathbb{R}^{n_{C_{k,k}} \times n_{C_{l,l}}}$, $T_{k,l} \in \mathbb{R}^{n_{T_{k,k}} \times n_{T_{l,l}}}$. Note that $T_{k,l}$ for $k \neq l$ have nothing to do with triangularity.

Minimizing $n_{C_{k,k}}$ can be posed as the minimum cutset (or feedback vertex set) problem which is known to be NP–complete for general graphs [22]; therefore, non–optimal solutions need to be considered. Fortunately, a polynomial time algorithm called Cutfind due to Rosen exists [32]. The algorithm runs in linear time and space and finds cutsets of graphs. Although cutsets computed with Cutfind may not be minimum, [9] shows that it is a fast algorithm for large graphs compared to other approximation algorithms and the size of the cutset computed is generally satisfying. We remark that it is again Tarjan's algorithm which is used to find the symmetric permutation that triangularizes the diagonal block associated with the states in the cutset's complement; i.e., $T_{k,k}$. Thus, for a block triangular matrix having irreducible diagonal blocks with zero–free diagonals, such a (2×2) block partitioning can be computed for each diagonal block and substitution can be used for solving the triangular diagonal blocks at each (outer) iteration, while the solution of the remaining diagonal blocks can be approximated with some kind of (inner) point iteration. This approach alleviates the fill–in problem associated with factorizing diagonal blocks in block iterative solvers up to a certain extent.

We consider five partitionings which can be used with block iterative solvers. The first three are based on the idea of symmetrically permuting B to block triangular form and using cutsets to obtain triangular diagonal blocks. The last two partitionings have been already used in the context of MCs before [12] and do not utilize Tarjan's algorithm. They are used to determine whether any improvement over the block iterative process results from employing the first three partitionings. The parameters of the partitionings can be expressed as a Cartesian product of four sets. Let set $\mathcal{B} = \{\texttt{y,n}\}$ denote whether Tarjan's algorithm is used or not, set $\mathcal{C} = \{\texttt{y,n}\}$ denote whether Rosen's Cutfind algorithm is used or not, set $\mathcal{R} = \{\texttt{y,n}\}$ denote whether the number of diagonal blocks is restricted to 2 or not, and set $\mathcal{O} = \{\texttt{l,u}\}$ denote whether a block lower– or upper–triangular orientation is desired with Tarjan's algorithm. Then experiments with partitionings take as parameters elements from proper subsets of $\mathcal{B} \times \mathcal{C} \times \mathcal{R} \times \mathcal{O}$. Experiments performed on web matrices using partitionings 1 and 2 can utilize elements of $\{\texttt{y}\} \times \mathcal{C} \times \mathcal{R} \times \mathcal{O}$. Those using partitioning 3 (in which, through a recursive application of the Tarjan and Cutfind algorithms, all diagonal blocks are made to be triangular) can utilize $\{\texttt{y}\} \times \{\texttt{y}\} \times \{\texttt{n}\} \times \mathcal{O}$. Since partitionings 4 and 5 do not use Tarjan's and Rosen's algorithms, the concept of orientation does not apply, and we arbitrarily set the last parameter to u and say these partitionings utilize the parameters $\{\texttt{n}\} \times \{\texttt{n}\} \times \{\texttt{n}\} \times \{\texttt{u}\}$.

Recall that nb is the number of diagonal blocks returned by Tarjan's algorithm for B and let nb_2 be of those that are of order two. Without loss of generality let us assume that the symmetric permutation is to block lower–triangular form, states incident on each diagonal block are ordered increasingly according to index, and the first state in each diagonal block of order two is placed in the cutset. Keep in mind that it does not make sense to run the Cutfind algorithm on diagonal blocks of order one and two since a diagonal block of order one is already triangular, and either state in a diagonal block of order two forms a triangular diagonal block.

ETNA
Kent State University
http://etna.math.kent.edu

3.1. Partitioning 1. In this partitioning, diagonal blocks of order one with no off–diagonal row elements are placed consecutively in the first diagonal block $T_{0,0}$. When Cutfind is used, lower–triangular diagonal blocks $T_{1,1}, T_{2,2}, \ldots, T_{K,K}$ follow this; the remaining diagonal blocks, which include states of cutsets, are ordered as $C_{K,K}$, $C_{K-1,K-1}$, …, $C_{1,1}$. Diagonal blocks of order one, which have some off–diagonal row elements, are grouped into the last block $C_{K+1,K+1}$ as in

$$E_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{l})} B E^T_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{l})} =$$

$$\begin{array}{rl}
 & \begin{array}{cccccccc} n_{T_{0,0}} & n_{T_{1,1}} & \cdots & n_{T_{K,K}} & n_{C_{K,K}} & \cdots & n_{C_{1,1}} & n_{C_{K+1,K+1}} \end{array} \\
\begin{array}{c} n_{T_{0,0}} \\ n_{T_{1,1}} \\ \vdots \\ n_{T_{K,K}} \\ n_{C_{K,K}} \\ \vdots \\ n_{C_{1,1}} \\ n_{C_{K+1,K+1}} \end{array} &
\left[\begin{array}{cccc|c|c|c|c}
T_{0,0} & & & & & & & \\
T_{1,0} & T_{1,1} & & & & & Z_{1,1} & Z_{1,K+1} \\
\vdots & \vdots & \ddots & & & \iddots & \vdots & \vdots \\
T_{K,0} & T_{K,1} & \cdots & T_{K,K} & Z_{K,K} & \cdots & Z_{K,1} & Z_{K,K+1} \\ \hline
Y_{K,0} & Y_{K,1} & \cdots & Y_{K,K} & C_{K,K} & \cdots & C_{K,1} & C_{K,K+1} \\ \hline
\vdots & \vdots & \iddots & & & \ddots & \vdots & \vdots \\ \hline
Y_{1,0} & Y_{1,1} & & & & & C_{1,1} & C_{1,K+1} \\ \hline
Y_{K+1,0} & Y_{K+1,1} & \cdots & Y_{K+1,K} & C_{K+1,K} & \cdots & C_{K+1,1} & C_{K+1,K+1}
\end{array}\right]
\end{array},$$

so that $nb = n_{T_{0,0}} + K + n_{C_{K+1,K+1}}$ and $J = 1_{\sum_{k=0}^{K} n_{T_{k,k}} > 0} + K + 1_{n_{C_{K+1,K+1}} > 0}$, where 1_f is the indicator function evaluating to 1 when f is true, 0 otherwise, and $E_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{l})}$ is the corresponding permutation matrix. Note that the diagonal block having $T_{0,0}$,$T_{1,1}$, …,$T_{K,K}$ along its diagonal is lower–triangular and that it is only this block which is guaranteed to be triangular.

An option for partitioning 1 is to let $J = 2$ as in (`y`,`y`,`y`,`l`) so that one has a (2×2) block partitioning in which the second diagonal block is comprised of

$$C_{K,K}, C_{K-1,K-1}, \ldots, C_{1,1}, C_{K+1,K+1}$$

along its diagonal. Note that it is not meaningful to restrict the number of diagonal blocks to 2 if the Cutfind algorithm is not used in partitioning 1. Hence, we do not consider partitioning 1 with parameters (`y`,`n`,`y`,`l`) and (`y`,`n`,`y`,`u`). A remark must be made at this point about orientation. When a block upper–triangular form is desired with Tarjan's algorithm, diagonal blocks of order one should be checked for zero off–diagonal elements in columns rather than rows and $T_{1,1}$,$T_{2,2}$,…,$T_{K,K}$ should be upper–triangularized if Cutfind is used.

EXAMPLE 3.1. (Continued.) Now, let us consider partitioning 1 on our web matrix of 6 pages. Since state 2 is in a state space partition by itself and has nonzero off–diagonal elements in its row, it will be at the end of the permutation. For the first irreducible diagonal block incident on $\{1, 3\}$, state 1 is an element of the cutset and state 3, being in the cutset's complement, is placed in the first triangular block. After applying the Cutfind algorithm to the irreducible diagonal block incident on $\{4, 5, 6\}$ of order three, we obtain its cutset as $\{4\}$ and therefore the cutset's complement as $\{5, 6\}$. Lower–triangularization of the diagonal block incident on $\{5, 6\}$ using Tarjan's algorithm yields the permutation vector $[5\ 6]^T$. So, the permutation vector becomes $[3\ 5\ 6\ 4\ 1\ 2]^T$ and the order of the triangular block is obtained as three. Hence, the symmetrically permuted matrix has the nonzero structure in

ETNA
Kent State University
http://etna.math.kent.edu

$$E_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{l})} B E^T_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{l})} =$$
$$\begin{array}{c} 3\\5\\6\\4\\1\\2 \end{array} \left[\begin{array}{ccc|c|c|c} X & & & & X & \\ X & X & & X & & \\ & X & X & X & & \\ \hline & X & X & X & & \\ \hline X & & & & X & \\ \hline X & & & & X & X \end{array}\right], \quad \begin{array}{c} 3\\5\\6\\4\\1\\2 \end{array} \left[\begin{array}{ccc|ccc} X & & & & X & \\ X & X & & X & & \\ & X & X & X & & \\ \hline & X & X & X & & \\ X & & & & X & \\ X & & & & X & X \end{array}\right],$$

where $E_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{l})} = [e_3\ e_5\ e_6\ e_4\ e_1\ e_2]^T$. In this example, $K = 2$ and $n_{T_{0,0}} = 0$, $n_{T_{1,1}} = 1$, $n_{T_{2,2}} = 2$, $n_{C_{2,2}} = 1$, $n_{C_{1,1}} = 1$, $n_{C_{3,3}} = 1$. There are four diagonal blocks; the first one is lower–triangular and of order three, the other three are of order one each. Restricting the number of diagonal blocks as in $(\mathtt{y},\mathtt{y},\mathtt{y},\mathtt{l})$ so that we have a (2×2) block partitioning in which the first diagonal block is lower–triangular, we obtain the partitioning on the right.

If an upper–triangular orientation is used with partitioning 1, we will end up with the nonzero structure in

$$E_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{u})} B E^T_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{u})} =$$
$$\begin{array}{c} 2\\6\\5\\3\\1\\4 \end{array} \left[\begin{array}{cccc|c|c} X & & & & X & \\ & X & X & & & X \\ & & X & X & & X \\ & & & X & X & \\ \hline & & & X & X & \\ \hline & X & X & & & X \end{array}\right], \quad \begin{array}{c} 2\\6\\5\\3\\1\\4 \end{array} \left[\begin{array}{cccc|cc} X & & & & X & \\ & X & X & & & X \\ & & X & X & & X \\ & & & X & X & \\ \hline & & & X & X & \\ & X & X & & & X \end{array}\right],$$

where $E_{1(\mathtt{y},\mathtt{y},\mathtt{n},\mathtt{u})} = [e_2\ e_6\ e_5\ e_3\ e_1\ e_4]^T$, $K = 2$, $n_{T_{0,0}} = 1$, $n_{T_{1,1}} = 2$, $n_{T_{2,2}} = 1$, $n_{C_{2,2}} = 1$, $n_{C_{1,1}} = 1$, and $n_{C_{3,3}} = 0$. Restricting the number of diagonal blocks as in $(\mathtt{y},\mathtt{y},\mathtt{y},\mathtt{u})$ so that we have a (2×2) block partitioning in which the first diagonal block is upper–triangular, we obtain the partitioning on the right.

3.2. Partitioning 2. In this partitioning, diagonal blocks of order one are treated as in partitioning 1. Ordering of the remaining diagonal blocks is not changed except for those of order two. When Cutfind is used, for blocks of order two, (arbitrarily) the first state is moved to the first diagonal block $T_{0,0}$, which is lower–triangular, and the second state is moved to the last diagonal block $C_{K+1,K+1}$. While generating the overall permutation, consecutive processing of diagonal blocks is essential to ensure the lower–triangularity of $T_{0,0}$. When Cutfind is used, diagonal blocks $T_{1,1}, T_{2,2}, \ldots, T_{K,K}$ should be all lower–triangular. In the end, we have a partitioning of the form

$$E_{2(\mathtt{y,y,n,l})}BE^T_{2(\mathtt{y,y,n,l})} =$$

$$\begin{array}{c}
\begin{array}{ccccccccc} n_{T_{0,0}} & n_{C_{1,1}} & n_{T_{1,1}} & n_{C_{2,2}} & n_{T_{2,2}} & \cdots & n_{C_{K,K}} & n_{T_{K,K}} & n_{C_{K+1,K+1}} \end{array} \\
\left[\begin{array}{c|c|c|c|c|c|c|c|c}
T_{0,0} & & & & & & & & Z_{0,K+1} \\ \hline
Y_{1,0} & C_{1,1} & Y_{1,1} & & & & & & C_{1,K+1} \\ \hline
T_{1,0} & Z_{1,1} & T_{1,1} & & & & & & Z_{1,K+1} \\ \hline
Y_{2,0} & C_{2,1} & Y_{2,1} & C_{2,2} & Y_{2,2} & & & & C_{2,K+1} \\ \hline
T_{2,0} & Z_{2,1} & T_{2,1} & Z_{2,2} & T_{2,2} & & & & Z_{2,K+1} \\ \hline
\vdots & \vdots & \vdots & \vdots & \vdots & \ddots & & & \vdots \\ \hline
Y_{K,0} & C_{K,1} & Y_{K,1} & C_{K,2} & Y_{K,2} & \cdots & C_{K,K} & Y_{K,K} & C_{K,K+1} \\ \hline
T_{K,0} & Z_{K,1} & T_{K,1} & Z_{K,2} & T_{K,2} & \cdots & Z_{K,K} & T_{K,K} & Z_{K,K+1} \\ \hline
Y_{K+1,0} & C_{K+1,1} & Y_{K+1,1} & C_{K+1,2} & Y_{K+1,2} & \cdots & C_{K+1,K} & Y_{K+1,K} & C_{K+1,K+1}
\end{array}\right]
\end{array},$$

so that $nb = n_{T_{0,0}} + K + n_{C_{K+1,K+1}} - nb_2$ and $J = 1_{n_{T_{0,0}}>0} + 2K + 1_{n_{C_{K+1,K+1}}>0}$, where $E_{2(\mathtt{y,y,n,l})}$ is the corresponding permutation matrix. When Cutfind is not used as in (`y`,`n`,`n`,`l`), we still place the first states of diagonal blocks of order two after the block of states corresponding to states with zero off–diagonal row elements and the second states of diagonal blocks of order two at the end of the permutation but before the block of states corresponding to states with some off–diagonal row elements to see whether there is any merit in this permutation. Recall that partitioning 1 does not handle diagonal blocks of order two as such when Cutfind is not used.

When a block upper–triangular form is desired with Tarjan's algorithm in partitioning 2, diagonal blocks of order one should be checked for zero off–diagonal elements in columns rather than rows as in partitioning 1. Since we do not see any merit in restricting the number of diagonal blocks to 2 in partitioning 2, we do not consider the parameters (`y`,`y`,`y`,`l`), (`y`,`y`,`y`,`u`), (`y`,`n`,`y`,`l`), and (`y`,`n`,`y`,`u`).

EXAMPLE 3.1. (Continued.) Now, we consider partitioning 2 on our web matrix of 6 pages. We again place state 2 at the end of the permutation. For the irreducible diagonal block of order two, the first state is placed in the first diagonal block and the second state is placed in the last diagonal block. Since the result of the Cutfind algorithm on the diagonal block of order three is the same as in partitioning 1, state 4 is placed in the cutset and states 5 and 6 are permuted as $[5\ 6]^T$ to obtain a lower–triangular diagonal block. The permutation vector becomes $[1\ 4\ 5\ 6\ 3\ 2]^T$ and we have the four diagonal blocks given in

$$E_{2(\mathtt{y,y,n,l})}BE^T_{2(\mathtt{y,y,n,l})} = \begin{array}{c} 1 \\ 4 \\ 5 \\ 6 \\ 3 \\ 2 \end{array} \left[\begin{array}{c|c|cc|cc}
\mathrm{X} & & & & \mathrm{X} & \\ \hline
 & \mathrm{X} & \mathrm{X} & \mathrm{X} & & \\ \hline
 & \mathrm{X} & \mathrm{X} & & \mathrm{X} & \\
 & \mathrm{X} & \mathrm{X} & \mathrm{X} & & \\ \hline
\mathrm{X} & & & & \mathrm{X} & \\
\mathrm{X} & & & & \mathrm{X} & \mathrm{X}
\end{array}\right],$$

where $E_{2(\mathtt{y,y,n,l})} = [e_1\ e_4\ e_5\ e_6\ e_3\ e_2]^T$. In this example, $K = 1$ and $n_{T_{0,0}} = 1$, $n_{C_{1,1}} = 1$, $n_{T_{1,1}} = 2$, $n_{C_{2,2}} = 2$. Note that the first and the third blocks are lower–triangular.

If an upper–triangular orientation is used with partitioning 2, we will end up with the

ETNA
Kent State University
http://etna.math.kent.edu

nonzero structure in

$$E_{2(\mathtt{y,y,n,u})} B E^T_{2(\mathtt{y,y,n,u})} = \begin{array}{c} 2\\ 1\\ 4\\ 6\\ 5\\ 3 \end{array} \left[\begin{array}{cc|c|cc|c} X & X & & & & X \\ & X & & & & X \\ \hline & & X & X & X & \\ \hline & & X & X & X & \\ & & X & & X & X \\ \hline & X & & & & X \end{array}\right],$$

where $E_{2(\mathtt{y,y,n,u})} = [e_2\ e_1\ e_4\ e_6\ e_5\ e_3]^T$, $K = 1$, $n_{T_{0,0}} = 2$, $n_{C_{1,1}} = 1$, $n_{T_{1,1}} = 2$, and $n_{C_{2,2}} = 1$. Note that the first and the third blocks are upper–triangular.

Partitionings 1 and 2 do not guarantee that all diagonal blocks are triangular. The next subsection discusses how one can obtain such a partitioning.

3.3. Partitioning 3. This partitioning is obtained with a recursive procedure. A diagonal block considered at a particular recursive call (at the first call, B) is block triangularized using Tarjan's algorithm so that it has irreducible diagonal blocks along its diagonal. Diagonal blocks of order one and two are processed as before without calling the Cutfind algorithm, and the Cutfind algorithm is run on each irreducible diagonal block of order larger than two. In the (2×2) block partitionings obtained as such, the non–triangular diagonal blocks associated with the cutsets are grouped together and input to the next recursive call, while blocks associated with the cutset's complements are triangularized and grouped together into a larger triangular block. Recursion ends when each non–triangular diagonal block is of order less than two. We denote the permutation matrix corresponding to partitioning 3 as E_3, and remark that the permutation obtained for lower–triangular orientation is the reverse of that of the upper–triangular one.

The implementation of partitioning 3 uses the same data structures and routines as in partitionings 1 and 2 except that it requires three additional integer work arrays having lengths nz, $(n+1)$, and n, where nz denotes the number of nonzero elements of B. Note that nz less n is τ. The first two of these arrays are used for storing the nonzero pattern of the submatrix to be considered at that recursive call and the last one to record the permutation in between recursive calls.

EXAMPLE 3.1. (Continued.) For partitioning 3, in the first recursive call, after symmetrically permuting B according to the outcome of Tarjan's algorthm, we have three diagonal blocks, the first two of which are of order two and one. So, the Cutfind algorithm need not be executed on them. Therefore, states 1 and 2 are moved to the cutset's complement and state 3 is moved to the cutset. Application of Cutfind to the third irreducible diagonal block yields its cutset as $\{4\}$, and therefore the cutset's complement as $\{5, 6\}$. Lower–triangularization of the diagonal block incident on $\{5, 6\}$ using Tarjan's algorithm gives the permutation vector $[5\ 6]^T$. Hence, the overall permutation vector at the end of the first recursive call is obtained as $[3\ 4\ 1\ 2\ 5\ 6]^T$. Restricting the number of diagonal blocks so that we have a (2×2) block partitioning in which the second diagonal block of order four is lower–triangular, and in the second recursive call, applying Tarjan's algorithm to the first diagonal block of order two, we obtain two irreducible diagonal blocks of order one each. Therefore, the cutset is the empty set, cutset's complement becomes $\{3, 4\}$, and recursion ends. Note that now we have

ETNA
Kent State University
http://etna.math.kent.edu

a partitioning in which both diagonal blocks are lower–triangular

$$E_{3(\mathtt{y,y,n,l})} B E^T_{3(\mathtt{y,y,n,l})} = \begin{array}{c} 3\\ 4\\ 1\\ 2\\ 5\\ 6 \end{array} \left[\begin{array}{cc|cccc} \mathrm{X} & & \mathrm{X} & & & \\ & \mathrm{X} & & & \mathrm{X} & \mathrm{X} \\ \hline \mathrm{X} & & \mathrm{X} & & & \\ \mathrm{X} & & \mathrm{X} & \mathrm{X} & & \\ \mathrm{X} & \mathrm{X} & & & \mathrm{X} & \\ & \mathrm{X} & & & \mathrm{X} & \mathrm{X} \end{array}\right],$$

where $E_{3(\mathtt{y,y,n,l})} = [e_3\ e_4\ e_1\ e_2\ e_5\ e_6]^T$.

If an upper–triangular orientation is used with partitioning 3, we will end up with the nonzero structure in

$$E_{3(\mathtt{y,y,n,u})} B E^T_{3(\mathtt{y,y,n,u})} = \begin{array}{c} 6\\ 5\\ 2\\ 1\\ 4\\ 3 \end{array} \left[\begin{array}{cccc|cc} \mathrm{X} & \mathrm{X} & & & \mathrm{X} & \\ & \mathrm{X} & & & \mathrm{X} & \mathrm{X} \\ & & \mathrm{X} & \mathrm{X} & & \mathrm{X} \\ & & & \mathrm{X} & & \mathrm{X} \\ \hline \mathrm{X} & \mathrm{X} & & & \mathrm{X} & \\ & & & \mathrm{X} & & \mathrm{X} \end{array}\right],$$

where $E_{3(\mathtt{y,y,n,u})} = [e_6\ e_5\ e_2\ e_1\ e_4\ e_3]^T$ and both diagonal blocks are upper–triangular.

3.4. Partitionings 4 and 5. Two straightforward partitionings are considered from [12]. Partitioning *equal* (here, 4) has $\sqrt{n}$ diagonal blocks of order $\sqrt{n}$ if n is a perfect square. If $n \neq \lfloor\sqrt{n}\rfloor^2$, there is an extra diagonal block of order $n - \lfloor\sqrt{n}\rfloor^2$. Partitioning *other* (here, 5) uses diagonal blocks of order respectively $1, 2, 3, \ldots$. It has about $\sqrt{2n}$ blocks with the largest diagonal block being of order roughly $\sqrt{2n}$.

When partitionings 1 through 3 are employed with block iterative solvers, $S^T - I$ is symmetrically permuted using the corresponding permutation matrix E to obtain the coefficient matrix A. With partitioni Then A is scaled to have a diagonal of 1's and then transformed from point form to block form based on the partitioning chosen. A is transformed to point form and scaled back after the iterations are over.

In Table 3.1, we summarize the way in which the twelve of the fourteen partitionings we have experimented with can be obtained using the permutations suggested by partitionings 1 through 3.

In the next section, we introduce six web matrices, their properties, and the experimental framework.

4. Experimental framework. Different problems resulting from the reducible web matrices *Stanford*, *StanfordBerkeley*, *Eu2005*, *WebGoogle*, *In2004*, and *WebBase* are considered by letting $\alpha \in \{0.85, 0.9, 0.95, 0.97, 0.99\}$. Hence, altogether there are thirty problems that are analyzed. The *Stanford*, *StanfordBerkeley*, and *WebGoogle* matrices come from the University of Florida Sparse Matrix Collection [8]. All matrices model parts of the hyperlinked web of pages around the world and solving them would rank the pages. The matrices *Eu2005* and *In2004* are crawled by UbiCrawler [4, 26] and uncompressed using the WebGraph [3, 39] software. The *WebBase* matrix is obtained from the Stanford WebBase Project [38]. The six matrices we consider have orders ranging between 281,903 and 2,662,002, and possess dangling nodes. The matrices lack (some) diagonal elements.

Each matrix is obtained from a file which stores in compact sparse row format the matrix columnwise without (some) diagonal entries. In other words, each file stores the nonzero structure of P^T in compact sparse row format and must be read as such. The missing diagonal

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 3.1
Obtaining partitionings 1 through 3 on B.

Partitioning		Steps
1 (y,y,n,l)	(1)	Apply Tarjan's algorithm to obtain block lower–triangular form.
	(2)	Move states of diagonal blocks of order one with zero off–diagonal row elements to beginning, those with some off–diagonal row elements to end of permutation.
	(3)	Apply the Cutfind algorithm to each diagonal block of order larger than two; move states in cutset's complement towards beginning and states in cutset towards end of permutation; lower–triangularize diagonal block of states in cutset's complement; for diagonal blocks of order two, move first state towards end and second state towards beginning of permutation.
	(4)	All states moved towards beginning of permutation form first (lower–triangular) diagonal block; all other subsets of states moved towards end of permutation form remaining diagonal blocks.
1 (y,y,n,u)		As in 1 (y,y,n,l) except 'lower' and 'row' are replaced with 'upper' and 'column'.
1 (y,y,y,l)		As in 1 (y,y,n,l) except step (4) is changed such that all other subsets of states moved towards end of permutation form second diagonal block.
1 (y,y,y,u)		As in 1 (y,y,n,u) except step (4) is changed such that all other subsets of states moved towards end of permutation form second diagonal block.
1 (y,n,n,l)		As in 1 (y,y,n,l) except step (3) is omitted.
1 (y,n,n,u)		As in 1 (y,y,n,u) except step (3) is omitted.
2 (y,y,n,l)	(1)	As in step (1) of 1 (y,y,n,l).
	(2)	As in step (2) of 1 (y,y,n,l).
	(3)	For diagonal blocks of order two, move first state towards beginning of permutation and second state towards end of permutation.
	(4)	Apply the Cutfind algorithm to each diagonal block of order larger than two; move states in cutset's complement towards beginning and states of cutset to end of states in diagonal block; lower–triangularize diagonal block of states in cutset's complement.
	(5)	All states moved towards beginning of permutation form first (lower–triangular) diagonal block; all other subsets of states form remaining diagonal blocks.
2 (y,y,n,u)		As in 2 (y,y,n,l) except 'lower' and 'row' are replaced with 'upper' and 'column'.
2 (y,n,n,l)		As in 2 (y,y,n,l) except step (4) is omitted and step (5) is changed such that only states moved to beginning of permutation in step (2) form first (lower–triangular) diagonal block; all other subsets of states form remaining diagonal blocks.
2 (y,n,n,u)		As in 2 (y,y,n,u) except step (4) is omitted and step (5) is changed such that only states moved to beginning of permutation in step (2) form first (upper–triangular) diagonal block; all other subsets of states form remaining diagonal blocks.
3 (y,y,n,l)		Recursively:
	(1)	Apply Tarjan's algorithm to obtain block lower–triangular form.
	(2)	Process diagonal blocks of order one and two without calling the Cutfind algorithm.
	(3)	Apply the Cutfind algorithm to all other diagonal blocks.
	(4)	Lower–triangularize diagonal blocks associated with cutsets' complements and group them together into a larger triangular block.
	(5)	If there is a cutset with more than one state, group states associated with cutsets together and make a recursive call with corresponding diagonal block.
3 (y,y,n,u)		As in 3 (y,y,n,l) except 'lower' is replaced with 'upper'.

elements are inserted into the corresponding data structures for all solvers except POWER so as to facilitate the construction of $B = R^T - I$. Here we remark that it is possible to implement the point solvers J and GS without forming B as discussed in [13]. A software implementation in the form of a Java code which has done so can be found at [23]. Now, note that the particular sparse format in the files favors POWER in two ways. Since there is no need to transpose P^T to work on the rows of P to handle diagonal elements at the outset for POWER, memory for missing diagonal elements and transposition need not be allocated. This is not the case for the other solvers and is also mentioned in [16]. That is, POWER can work by postmultiplying $R^T = \alpha P^T$ with a column vector at each iteration after P^T is read into the compact sparse row format data structure at the outset. This translates to

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 4.1
Properties of web matrices.

Matrix	n	nz'	dangling nodes	missing diagonal elements	nb	order of largest diagonal block
Stanford	281,903	2,312,497	20,315	281,903	29,914	150,532
StanfordBerkeley	683,446	7,583,376	68,062	683,446	109,238	333,752
Eu2005	862,664	19,235,140	71,675	361,237	90,768	752,725
WebGoogle	916,428	5,105,039	176,974	916,428	412,479	434,818
In2004	1,382,908	16,917,053	86	1,005,498	367,675	593,687
WebBase	2,662,002	44,843,770	574,863	2,662,002	954,137	1,390,621

savings in memory and time for POWER. For all the other solvers, we allocate a floating–point array to store the diagonal elements of B, scale the rows so that the diagonal elements are all 1 and unscale them back at the end as it is done in the MC analysis software tool MARCA [34]. This saves the flops due to diagonal elements at each iteration. Recall that the software tool [10, 11] also works with irreducible MCs when α is set to 1. Furthermore, each solver including POWER needs at least two floating–point arrays of length n to compute an approximate error norm to test for convergence and the residual norm upon stopping. GS (that is, SOR when the relaxation parameter ω is 1) can do with one floating–point array of length n during the iterative process but still needs the second floating-point array for the residual norm computation. However, certain optimizations that improve the memory requirements and timings of our code may be possible. Finally, the routine from [37] that implements Tarjan's algorithm is available for research purposes.

Information regarding the web matrices used in the experiments appears in Table 4.1. The first column provides the matrix name. Columns n and nz' give the order of the matrix and its number of nonzeros as read from the input file. Columns four and five provide its numbers of dangling nodes and missing diagonal elements. Hence, the number of nonzeros in B is given by $nz = nz' +$ missing diagonal elements. Column nb lists the number of diagonal blocks returned by Tarjan's algorithm on B. Finally, the last column gives the order of the largest one among the nb diagonal blocks. The order of the smallest diagonal block in each matrix is 1 since each matrix has at least one dangling node, and therefore is not listed in the table. Computation of n/nb reveals that the average order of diagonal blocks returned by Tarjan's algorithm is respectively 9.4, 6.3, 9.5, 2.2, 3.8, 2.8 in one decimal digit of precision for the matrices in the given order.

TABLE 4.2
Nonzero structure of the partitionings on B for the Stanford *matrix.*

Partitioning	J	$\min_j n_j$	$\max_j n_j$	n_1	nz_1	$E[n_j]$	$E[nz_j]$	nz_{off}	$\text{med}_j n_j$
1 (y,y,n,l)	3,520	1	159,037	159,037	452,823	80	332	1,425,977	2
1 (y,y,n,u)	3,520	1	179,180	179,180	517,281	80	339	1,401,385	2
1 (y,y,y,l)	2	122,866	159,037	159,037	452,823	140,952	650,834	1,292,733	140,952
1 (y,y,y,u)	2	102,723	179,180	179,180	517,281	140,952	608,351	1,377,698	140,952
1 (y,n,n,l)	3,520	2	150,532	172	172	80	673	224,891	6
1 (y,n,n,u)	3,520	2	150,532	20,315	20,315	80	668	244,614	6
2 (y,y,n,l)	4,776	1	100,162	1,303	1,356	59	241	1,441,346	6
2 (y,y,n,u)	4,776	1	100,162	21,446	21,569	59	237	1,461,074	6
2 (y,n,n,l)	3,520	1	150,532	172	172	80	673	226,848	6
2 (y,n,n,u)	3,520	1	150,532	20,315	20,315	80	667	246,646	6
3 (y,y,n,u)	42	1	185,332	185,332	555,177	6,712	16,845	1,886,918	52
4 (n,n,n,u)	531	530	1,003	530	534	531	538	2,308,602	530
5 (n,n,n,u)	751	1	750	1	1	375	380	2,308,739	375

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 4.3
Nonzero structure of the partitionings on B for the Stanford_Berkeley *matrix.*

Partitioning	J	$\min_j n_j$	$\max_j n_j$	n_1	nz_1	$E[n_j]$	$E[nz_j]$	nz_{off}	$\text{med}_j n_j$
1 (y,y,n,l)	6,750	1	360,788	360,788	1,013,364	101	601	4,208,017	3
1 (y,y,n,u)	6,750	1	424,115	424,115	1,195,425	101	553	4,534,222	3
1 (y,y,y,l)	2	322,658	360,788	360,788	1,013,364	341,723	2,348,303	3,570,216	341,723
1 (y,y,y,u)	2	259,331	424,115	424,115	1,195,425	341,723	1,975,004	4,316,815	341,723
1 (y,n,n,l)	6,750	2	333,752	4,735	4,735	101	1,087	926,824	6
1 (y,n,n,u)	6,750	2	333,752	68,062	68,062	101	1,021	1,371,763	6
2 (y,y,n,l)	10,116	1	227,157	6,426	6,484	68	398	4,239,810	5
2 (y,y,n,u)	10,116	1	227,157	69,753	69,870	68	354	4,685,016	5
2 (y,n,n,l)	6,750	1	333,752	4,735	4,735	101	1,087	928,567	6
2 (y,n,n,u)	6,750	1	333,752	68,062	68,062	101	1,021	1,373,832	6
3 (y,y,n,u)	163	2	458,614	458,614	1,719,076	4,193	12,946	6,156,597	10
4 (n,n,n,u)	827	826	1,170	826	4,340	826	4,931	4,188,520	826
5 (n,n,n,u)	1,169	1	1,168	1	1	585	3,465	4,216,462	585

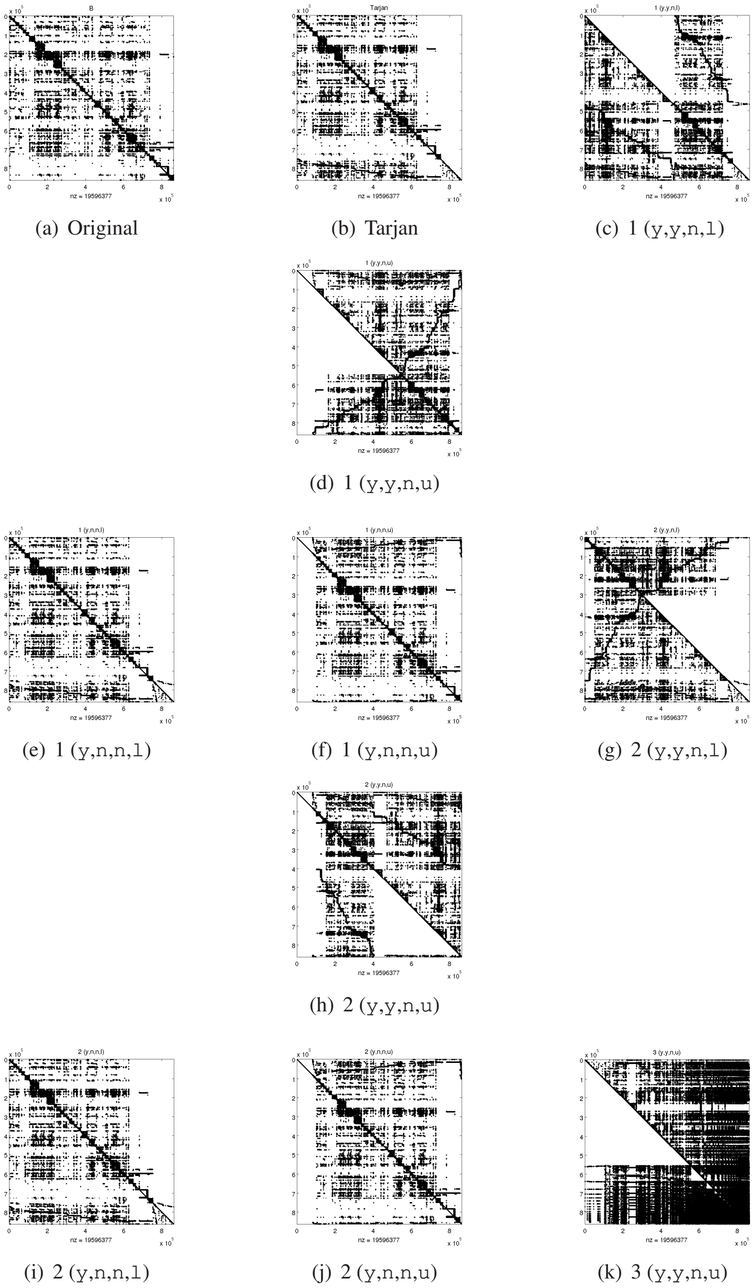
TABLE 4.4
Nonzero structure of the partitionings on B for the Eu2005 *matrix.*

Partitioning	J	$\min_j n_j$	$\max_j n_j$	n_1	nz_1	$E[n_j]$	$E[nz_j]$	nz_{off}	$\text{med}_j n_j$
1 (y,y,n,l)	1,162	1	465,433	465,433	1,354,607	742	9,087	9,028,274	2
1 (y,y,n,u)	1,163	1	546,874	546,874	1,875,258	742	9,452	8,603,667	2
1 (y,y,y,l)	2	397,231	465,433	465,433	1,354,607	431,332	5,618,481	8,359,415	431,332
1 (y,y,y,u)	2	315,790	546,874	546,874	1,875,258	431,332	5,516,566	8,563,246	431,332
1 (y,n,n,l)	1,162	2	752,725	752,725	18,201,086	742	15,867	1,158,774	3
1 (y,n,n,u)	1,163	2	752,725	81,441	81,441	742	15,841	1,173,377	3
2 (y,y,n,l)	1,588	1	451,763	368	382	543	6,654	9,029,473	2
2 (y,y,n,u)	1,588	1	451,763	81,809	82,067	543	6,645	9,043,989	2
2 (y,n,n,l)	1,162	1	752,725	752,725	18,201,086	742	15,867	1,158,974	3
2 (y,n,n,u)	1,163	1	752,725	81,441	81,441	742	15,841	1,173,734	3
3 (y,y,n,u)	378	1	555,044	555,044	1,942,183	2,282	6,713	17,058,750	1
4 (n,n,n,u)	929	928	1,480	928	11,808	929	7,829	12,322,997	928
5 (n,n,n,u)	1,314	1	1,313	1	1	657	5,424	12,469,660	656

Representative plots of the resulting nonzero structures under the assumed partitionings are provided in Figures 4.1 and 4.2 with the pertaining data interleaved in Tables 4.2–4.7. The nonzero plots of *Stanford* resemble those of *WebGoogle*, and the nonzero plots of *Stanford_Berkeley*, *In2004*, *WebBase* resemble those of *Eu2005*; hence, their nonzero plots are omitted. The number, location, and values of nonzeros in these partitionings have a significant effect on the number of iterations performed by the solvers and their respective solution times. Note, however that the nonzero structure of B, and hence those of its associated partitionings, do not change for different values of α, nor will they change for different personalization vectors v. Hence, time spent at the outset for computing the partitionings can very well be justified if they are not significantly more than the iteration times. In Tables 4.2–4.7, the column "Partitioning" indicates the block partitioning used and lists its parameters. Number of diagonal blocks, order of smallest and largest diagonal blocks, order and number of nonzeros of the first diagonal block, average order and average number of nonzeros of diagonal blocks (both rounded to the nearest integer), number of nonzero elements lying in off–diagonal blocks, and median order of diagonal blocks in the particular partitioning appear in columns J, $\min_j n_j$, $\max_j n_j$, n_1, nz_1, $E[n_j]$, $E[nz_j]$, nz_{off}, and $\text{med}_j n_j$, respectively. Note that for partitioning 3, orientation of the block triangular form is immaterial from a statistical point of view of the nonzero structure, and therefore, results only with upper–triangular orientation are reported.

Some observations are due. Among the six web matrices, the nonzero plots of B for

ETNA
Kent State University
http://etna.math.kent.edu



(a) Original (b) Tarjan (c) 1 (y,y,n,l)

(d) 1 (y,y,n,u)

(e) 1 (y,n,n,l) (f) 1 (y,n,n,u) (g) 2 (y,y,n,l)

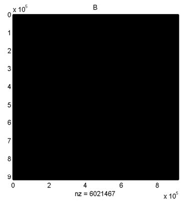
(h) 2 (y,y,n,u)

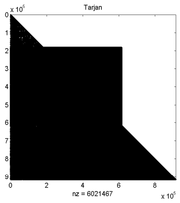
(i) 2 (y,n,n,l) (j) 2 (y,n,n,u) (k) 3 (y,y,n,u)

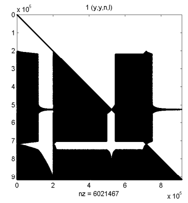
FIG. 4.1. *Nonzero plots of the partitionings on B for the* Eu2005 *matrix.*

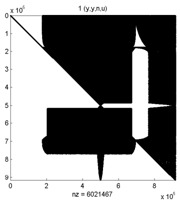
Stanford and *WebGoogle* (the latter given in Figure 4.2(a) indicate a more uniform distribution of their nonzeros across rows and columns. These two matrices are of different orders, but both look very dense. Note that neither the average order of the diagonal blocks returned by Tarjan's algorithm (9.4 and 2.2, respectively) nor the average number of nonzeros per row (9.2 and 6.6, respectively) is similar for these two matrices. For partitioning 3, the largest percentage of nonzeros within diagonal blocks appear in the *Stanford* and *WebGoogle* ma-

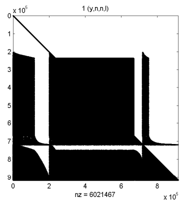
ETNA
Kent State University
http://etna.math.kent.edu

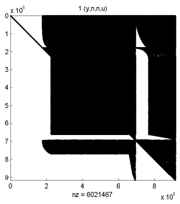

(a) Original

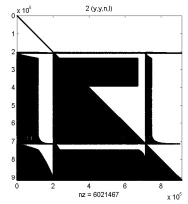

(b) Tarjan

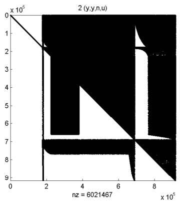

(c) 1 (`y,y,n,l`)

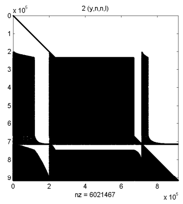

(d) 1 (`y,y,n,u`)

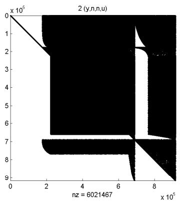

(e) 1 (`y,n,n,l`)


(f) 1 (`y,n,n,u`)


(g) 2 (`y,y,n,l`)


(h) 2 (`y,y,n,u`)


(i) 2 (`y,n,n,l`)


(j) 2 (`y,n,n,u`)

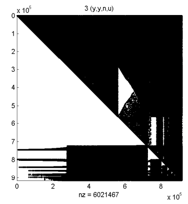

(k) 3 (`y,y,n,u`)

FIG. 4.2. *Nonzero plots of the partitionings on B for the* WebGoogle *matrix.*

trices as well, with about 27% and 32%, respectively. For this partitioning, it is also the same matrices for which the average order of diagonal blocks is largest with about 16,835 and 27,771, respectively. We will see in the next section that these two properties work in favor of partitioning 3 in all of the ten problems associated with these two matrices. On the other hand, the smallest number of nonzeros in the off–diagonal blocks appear in partitioning 1 with parameters (`y,n,n,l`) in all matrices except *WebGoogle* and *In2004* for which it

TABLE 4.5
Nonzero structure of the partitionings on B for the WebGoogle *matrix.*

Partitioning	J	$\min_j n_j$	$\max_j n_j$	n_1	nz_1	$E[n_j]$	$E[nz_j]$	nz_{off}	$\text{med}_j n_j$
1 (y,y,n,l)	12,876	1	524,850	524,850	1,057,698	71	234	3,013,372	1
1 (y,y,n,u)	12,876	1	499,941	499,941	966,216	71	228	3,079,757	1
1 (y,y,y,l)	2	391,578	524,850	524,850	1,057,698	458,214	1,624,343	2,772,781	458,214
1 (y,y,y,u)	2	416,487	499,941	499,941	966,216	458,214	1,783,650	2,454,168	458,214
1 (y,n,n,l)	12,876	2	434,818	201,883	201,883	71	371	1,243,063	4
1 (y,n,n,u)	12,876	2	434,818	176,974	176,974	71	371	1,242,875	4
2 (y,y,n,l)	17,412	1	280,066	206,052	216,382	53	163	3,189,056	2
2 (y,y,n,u)	17,412	1	280,066	181,143	182,884	53	163	3,189,011	2
2 (y,n,n,l)	12,876	1	434,818	201,883	201,883	71	371	1,247,096	4
2 (y,n,n,u)	12,876	1	434,818	176,974	176,974	71	371	1,238,462	4
3 (y,y,n,u)	33	1	722,666	722,666	1,640,818	27,771	58,624	4,086,863	299
4 (n,n,n,u)	958	579	957	957	957	957	962	5,099,746	957
5 (n,n,n,u)	1,354	1	1,353	1	1	677	681	5,099,941	676

TABLE 4.6
Nonzero structure of the partitionings on B for the In2004 *matrix.*

Partitioning	J	$\min_j n_j$	$\max_j n_j$	n_1	nz_1	$E[n_j]$	$E[nz_j]$	nz_{off}	$\text{med}_j n_j$
1 (y,y,n,l)	16,644	1	929,778	929,778	2,573,755	83	641	7,259,360	4
1 (y,y,n,u)	16,644	1	635,092	635,092	1,745,311	83	619	7,618,442	4
1 (y,y,y,l)	2	453,130	929,778	929,778	2,573,755	691,454	5,427,036	7,068,479	691,454
1 (y,y,y,u)	2	635,092	747,816	635,092	1,745,311	691,454	5,408,008	7,106,534	691,454
1 (y,n,n,l)	16,644	2	593,687	294,780	294,780	83	992	1,413,736	7
1 (y,n,n,u)	16,644	2	593,687	94	94	83	1,002	1,239,060	7
2 (y,y,n,l)	26,978	1	384,611	297,934	305,915	51	375	7,809,139	4
2 (y,y,n,u)	26,978	1	384,611	3,248	3,297	51	381	7,663,801	4
2 (y,n,n,l)	16,644	1	593,687	294,780	294,780	83	992	1,413,651	7
2 (y,n,n,u)	16,644	1	593,687	94	94	83	1,003	1,230,381	7
3 (y,y,n,u)	474	1	986,091	986,091	2,965,885	2,918	7,584	14,327,904	11
4 (n,n,n,u)	1,176	1,175	2,283	1,175	31,832	1,176	9,404	6,863,176	1,175
5 (n,n,n,u)	1,663	1	1,662	1	1	832	6,615	6,922,088	832

is a contender. It is clear that this partitioning with Tarjan's algorithm employed to obtain a block lower–triangular form seems to concentrate the largest number of nonzeros within diagonal blocks. We will also see that this is something useful for block iterative methods. Finally, partitioning 4 returns balanced block sizes and always a reasonable number of blocks as intended.

In the next section, we provide the results of numerical experiments on the benchmark matrices.

5. Numerical results. We compare the performance of sparse solvers in the software tool [11] with an emphasis on the memory used, number of iterations taken, accuracy achieved, time for preprocessing and solution. The numerical experiments are performed on a 2.66 GHz Pentium IV processor with 4 GB main memory under the Linux operating system using the o3 optimization level in compiling the code. We provide the results of experiments with three solvers: POWER, GS, and BGS.

The stopping criteria used by all solvers is

$$k \geq maxit \ \texttt{or} \ \|x^{(k)} - x^{(k-1)}\|_\infty \leq stoptol,$$

where k is the iteration number, $maxit$ is the maximum number of iterations to be performed, and $stoptol$ is the stopping tolerance. The solver BGS also uses the additional criteria

$$\|x^{(k)} - x^{(k-1)}\|_\infty \leq stoptol_1 \ \texttt{and} \ \left| \|x^{(k)} - x^{(k-1)}\|_\infty - \|x^{(k-1)} - x^{(k-2)}\|_\infty \right| \leq stoptol_2.$$

TABLE 4.7
Nonzero structure of the partitionings on B for the WebBase *matrix.*

Partitioning	J	$\min_j n_j$	$\max_j n_j$	n_1	nz_1	$E[n_j]$	$E[nz_j]$	nz_{off}	$\text{med}_j n_j$
1 (y,y,n,l)	8,584	1	1,635,593	1,635,593	5,085,941	310	2,030	30,079,285	1
1 (y,y,n,u)	8,584	1	1,854,996	1,854,996	6,049,826	310	2,104	29,445,974	1
1 (y,y,y,l)	2	1,026,409	1,635,593	1,635,593	5,085,941	1,331,001	9,542,739	28,420,294	1,331,001
1 (y,y,y,u)	2	807,006	1,854,996	1,854,996	6,049,826	1,331,001	11,578,808	24,348,156	1,331,001
1 (y,n,n,l)	8,584	2	1,390,621	355,460	355,460	310	4,541	8,523,737	3
1 (y,n,n,u)	8,584	2	1,390,621	574,863	574,863	310	4,528	8,634,908	3
2 (y,y,n,l)	10,188	1	1,054,569	358,949	362,330	261	1,655	30,647,129	4
2 (y,y,n,u)	10,188	1	1,054,569	578,352	585,200	261	1,644	30,755,389	4
2 (y,n,n,l)	8,584	1	1,390,621	355,460	355,460	310	4,541	8,524,636	3
2 (y,n,n,u)	8,584	1	1,390,621	574,863	574,863	310	4,528	8,636,363	3
3 (y,y,n,u)	404	1	2,225,698	2,225,698	8,197,002	6,589	21,962	38,633,180	10
4 (n,n,n,u)	1,632	1,631	1,841	1,631	50,035	1,631	11,114	29,368,144	1,631
5 (n,n,n,u)	2,307	1	2,306	1	1	1,154	7,675	29,799,021	1,154

The use of $stoptol_1$ and $stoptol_2$ forces the solver to terminate when the norm of the residual is decreasing too slowly, while the differences between two successive iterates is small enough. We let $stoptol = 10^{-10}$, $stoptol_1 = 10^{-6}$, and $stoptol_2 = 10^{-12}$, respectively, whereas $maxit = 5,000$.

For the block iterative methods, the triangular diagonal blocks are solved exactly at each outer iteration with the help of the Shermann–Morrison formula. The remaining diagonal blocks are solved approximately using the corresponding point iterative method. BGS has two additional parameters indicating maximum number and tolerance of inner iterations to be performed with GS for the solution of diagonal blocks. The values of these parameters are respectively elements of the sets $\{3, 5\}$ and $\{10^{-3}, 10^{-5}, 10^{-10}\}$. Hence, six experiments are carried out with each of the partitioning parameters except those of partitioning 3, which makes altogether seventy–four experiments for each problem with the BGS solver.

Now, we present our results and then make a general summary. Tables 5.1 through 5.6 provide results from the experiments for each web matrix with $\alpha \in \{0.85, 0.9, 0.95, 0.97, 0.99\}$. In each table, besides POWER and GS, we have also indicated the BGS solver which gives the minimum total solution time for each partitioning. The column "Solver" indicates the name of the solver and, for BGS solvers, the partitioning used with its parameters. The column "MB" provides the memory requirement of the solver in megabytes. Note that this column includes memory for nonzeros in the matrix and (except POWER) its transpose. The columns "Iterations" and "Residual" give the number of iterations performed and the infinity norm of the residual vector (i.e., $\pi^{(k)} - \pi^{(k)}S$ or $-Ax^{(k)}$ depending on the solver) upon stopping. The setup time and the total solution time in seconds (s) are given in the next two columns. The setup time includes time for reading the web matrix from the input file, allocating and setting the necessary data structures, and wherever applicable, scaling the coefficient matrix, computing the partitioning, transforming the sparse point representation of the coefficient matrix to a sparse block representation, transforming and scaling it back after the iterations are over. The total solution time is the sum of setup and iteration times. Finally, the last column gives the ratio of the total partitioning time to one iteration time of POWER, rounded to the nearest nonnegative integer. In other words, the column "Ratio" indicates the number of POWER iterations that can be executed during the time the respective partitioning is computed. The bold number shows the best overall total timing result in each table. The identities of winning solvers according to minimum total solution time coincide with those of minimum iteration time in all problems except *WebBase* with $\alpha = 0.85$, where it will become BGS with partitioning 1 (y,n,n,l) rather than GS if we consider minimum iteration time. We must remark

ETNA
Kent State University
http://etna.math.kent.edu

that it is acceptable for timing results of an experiment obtained by different runs on the same platform to differ by 10–15% due to various effects. The conclusions we derive take this into account.

TABLE 5.1
Results for the Stanford *matrix.*

α	Solver	MB	Iterations	Residual	Setup (s)	Total (s)	Ratio
0.85	POWER	38	103	8.4e-11	1.2	10.1	
	GS	50	45	5.9e-11	1.3	5.9	
	BGS 1 (y,n,n,l) 3, 10^{-10}	86	18	9.2e-11	2.5	5.5	15
	BGS 2 (y,y,n,l) 3, 10^{-3}	110	42	6.4e-11	3.0	5.4	21
	BGS 3 (y,y,n,l)	122	44	5.3e-11	3.1	**4.9**	22
	BGS 4 (n,n,n,u) 3, 10^{-3}	86	45	5.9e-11	1.8	6.2	7
	BGS 5 (n,n,n,u) 5, 10^{-3}	86	45	5.9e-11	1.8	6.2	7
0.9	POWER	38	154	7.6e-11	1.3	14.3	
	GS	50	65	7.5e-11	1.4	8.0	
	BGS 1 (y,n,n,l) 3, 10^{-10}	86	26	7.1e-11	2.5	6.9	14
	BGS 2 (y,y,n,l) 3, 10^{-3}	110	62	7.2e-11	3.0	6.6	20
	BGS 3 (y,y,n,l)	122	63	7.0e-11	3.1	**5.7**	21
	BGS 4 (n,n,n,u) 3, 10^{-3}	86	65	7.5e-11	1.8	8.1	6
	BGS 5 (n,n,n,u) 3, 10^{-3}	86	65	7.5e-11	1.6	8.0	4
0.95	POWER	38	288	9.1e-11	1.2	25.6	
	GS	50	124	8.6e-11	1.4	13.9	
	BGS 1 (y,n,n,l) 3, 10^{-10}	86	45	7.6e-11	2.5	10.3	15
	BGS 2 (y,y,n,l) 3, 10^{-3}	110	118	8.2e-11	2.9	9.8	17
	BGS 3 (y,y,n,l)	122	117	8.8e-11	3.2	**8.0**	24
	BGS 4 (n,n,n,u) 3, 10^{-3}	86	124	8.6e-11	1.8	13.9	7
	BGS 5 (n,n,n,u) 5, 10^{-3}	86	124	8.6e-11	1.8	13.9	7
0.97	POWER	38	475	9.6e-11	1.2	41.4	
	GS	50	201	9.0e-11	1.4	21.7	
	BGS 1 (y,n,n,l) 5, 10^{-10}	86	43	8.6e-11	2.4	14.2	14
	BGS 2 (y,y,n,l) 3, 10^{-3}	110	192	9.0e-11	3.1	14.1	22
	BGS 3 (y,y,n,l)	122	191	9.3e-11	3.2	**11.1**	24
	BGS 4 (n,n,n,u) 5, 10^{-3}	86	201	9.0e-11	1.7	21.3	6
	BGS 5 (n,n,n,u) 3, 10^{-3}	86	201	9.0e-11	1.7	21.4	6
0.99	POWER	38	1,319	9.9e-11	1.2	112.5	
	GS	50	574	5.6e-11	1.4	59.3	
	BGS 1 (y,y,y,l) 3, 10^{-3}	108	566	5.4e-11	3.0	35.0	21
	BGS 2 (y,y,n,u) 3, 10^{-3}	110	588	9.6e-11	3.0	36.2	21
	BGS 3 (y,y,n,u)	122	552	7.4e-11	3.1	**25.8**	23
	BGS 4 (n,n,n,u) 3, 10^{-3}	86	574	5.6e-11	1.8	57.7	7
	BGS 5 (n,n,n,u) 5, 10^{-3}	86	574	5.6e-11	1.7	57.7	6

5.1. Empirical observations. The results show that we can solve the largest and most difficult problem in our suite of thirty problems in less than four minutes on the given platform. The winning solver according to iteration time is always BGS. It is BGS with partitioning 1 in 17 problems, BGS with partitioning 3 in 10 problems, and BGS with partitioning 4 in 3 problems. In general, the time to compute partitionings 1 through 3 are only a fraction of the iteration time if the time to read the matrix and prepare the data structures is excluded from the setup time. That is, partitioning time is roughly equal to the setup time of BGS with the respective partitioning minus the setup time of POWER.

BGS with partitioning 3 and lower–triangular orientation is winner in the *Stanford* matrix with $\alpha \in \{0.85, 0.9, 0.95, 0.97\}$. As reported in Table 5.1, it is also the winner for $\alpha = 0.99$, but with upper–triangular orientation. BGS with partitioning 3 is also the winner in the *WebGoogle* matrix with lower–triangular orientation for $\alpha \in \{0.85, 0.9\}$ and upper–triangular orientation for $\alpha \in \{0.95, 0.97, 0.99\}$. For these two web matrices, there seems

ETNA
Kent State University
http://etna.math.kent.edu

TABLE 5.2
Results for the Stanford_Berkeley *matrix.*

α	Solver	MB	Iterations	Residual	Setup (s)	Total (s)	Ratio
0.85	POWER	115	93	7.7e-11	3.1	9.5	
	GS	152	55	6.3e-11	3.4	7.8	
	BGS 1 (y,n,n,l) 3, 10^{-10}	254	20	5.3e-11	4.0	**7.6**	13
	BGS 2 (y,n,n,l) 3, 10^{-5}	254	44	6.2e-11	4.1	8.7	15
	BGS 3 (y,y,n,u)	357	42	7.2e-11	7.0	11.7	57
	BGS 4 (n,n,n,u) 3, 10^{-3}	254	55	6.3e-11	3.7	8.5	9
	BGS 5 (n,n,n,u) 3, 10^{-3}	254	55	6.3e-11	3.7	8.5	9
0.9	POWER	115	143	8.5e-11	3.4	13.2	
	GS	152	78	7.1e-11	3.4	9.7	
	BGS 1 (y,n,n,l) 3, 10^{-10}	254	27	7.7e-11	4.2	**9.2**	12
	BGS 2 (y,n,n,l) 3, 10^{-5}	254	65	8.7e-11	4.2	10.8	12
	BGS 3 (y,y,n,u)	357	64	7.5e-11	6.8	14.0	50
	BGS 4 (n,n,n,u) 3, 10^{-3}	254	78	7.1e-11	3.6	10.4	3
	BGS 5 (n,n,n,u) 3, 10^{-3}	254	78	7.1e-11	3.8	10.7	6
0.95	POWER	115	292	9.5e-11	3.0	23.0	
	GS	152	143	8.5e-11	3.4	15.0	
	BGS 1 (y,n,n,l) 5, 10^{-10}	254	31	1.2e-10	4.2	**12.7**	18
	BGS 2 (y,y,n,l) 3, 10^{-3}	320	119	7.4e-11	4.6	16.5	23
	BGS 3 (y,y,n,u)	357	128	8.5e-11	6.9	21.3	57
	BGS 4 (n,n,n,u) 3, 10^{-5}	254	142	8.6e-11	3.9	16.4	13
	BGS 5 (n,n,n,u) 5, 10^{-5}	254	137	9.3e-11	3.6	15.8	9
0.97	POWER	115	490	9.6e-11	3.1	36.5	
	GS	152	239	9.1e-11	3.4	22.5	
	BGS 1 (y,n,n,l) 5, 10^{-10}	254	52	1.5e-10	4.1	**17.7**	15
	BGS 2 (y,y,n,u) 5, 10^{-3}	320	186	9.0e-11	4.6	23.0	22
	BGS 3 (y,y,n,l)	357	209	9.1e-11	6.7	30.1	53
	BGS 4 (n,n,n,u) 3, 10^{-10}	254	169	9.2e-11	3.7	22.4	9
	BGS 5 (n,n,n,u) 3, 10^{-10}	254	173	9.0e-11	3.8	22.8	10
0.99	POWER	115	1,454	9.8e-11	3.1	101.7	
	GS	152	719	9.7e-11	3.4	61.2	
	BGS 1 (y,n,n,l) 5, 10^{-10}	254	138	1.1e-10	4.1	**40.9**	15
	BGS 2 (y,y,n,u) 3, 10^{-3}	320	524	9.8e-11	4.8	56.8	25
	BGS 3 (y,y,n,l)	357	559	9.7e-11	6.7	69.3	53
	BGS 4 (n,n,n,u) 3, 10^{-10}	254	454	9.7e-11	3.9	53.4	12
	BGS 5 (n,n,n,u) 3, 10^{-10}	254	411	9.5e-11	4.2	49.6	16

to be a value of α beyond which the BGS solver favors upper–triangular orientation with partitioning 3 as the problem becomes more difficult to solve. We remark that we use forward BGS and the block lower–triangular part of A as the preconditioning matrix M. It is that part which multiplies the values of the current approximation. Interestingly, it is only the *Stanford* and *WebGoogle* matrices in which partitioning 3 accompanies a winning solver. Not all web matrices need to look the same, and the *Stanford* matrix may not be a good representative most probably due to its relative smallness. But, partitioning 3 also yields a winning solver with BGS in *WebGoogle*, which has an order of about a million and significantly different statistical features than *Stanford* as pointed out in section 4. When BGS is a winner with partitioning 3, the improvements in iteration time compared to the second best solver, GS, and POWER are respectively more than 21%, 47%, and 74%. These correspond to speedups of more than 1.3, 1.9, and 3.8. In Figure 5.1(a), we plot the % change in iteration time of BGS 3 (y,y,n,u) over that of GS for the ten experiments carried out with the *Stanford* and *WebGoogle* matrices. The graph shows that the improvement in iteration time is at least 45% for this set of experiments. In general, the time to compute partitioning 3 becomes relatively large compared to those of other partitionings as the order of the matrix increases, due to the relatively large number of recursive calls performed. However, it is still dependent on the

TABLE 5.3
Results for the Eu2005 *matrix.*

α	Solver	MB	Iterations	Residual	Setup (s)	Total (s)	Ratio
0.85	POWER	256	90	7.9e-11	7.6	20.3	
	GS	340	48	4.5e-11	8.6	16.0	
	BGS 1 (y,n,n,l) 5, 10^{-10}	542	11	5.4e-11	9.5	**15.5**	13
	BGS 2 (y,n,n,l) 5, 10^{-5}	542	43	7.1e-11	9.6	18.0	14
	BGS 3 (y,y,n,l)	746	41	7.1e-11	19.1	29.5	81
	BGS 4 (n,n,n,u) 3, 10^{-5}	542	44	5.3e-11	8.8	16.0	9
	BGS 5 (n,n,n,u) 5, 10^{-5}	542	44	5.6e-11	9.0	16.3	10
0.9	POWER	256	137	8.1e-11	7.4	26.7	
	GS	340	71	4.0e-11	8.3	19.2	
	BGS 1 (y,n,n,l) 5, 10^{-10}	542	15	5.8e-11	9.6	**18.5**	16
	BGS 2 (y,n,n,l) 3, 10^{-3}	542	72	1.8e-11	9.6	22.2	16
	BGS 3 (y,y,n,l)	746	63	7.1e-11	18.9	34.9	82
	BGS 4 (n,n,n,u) 3, 10^{-10}	542	52	7.0e-11	8.6	18.8	9
	BGS 5 (n,n,n,u) 3, 10^{-10}	542	53	8.0e-11	8.8	19.1	10
0.95	POWER	256	269	9.4e-11	7.7	45.2	
	GS	340	140	1.7e-11	8.5	29.9	
	BGS 1 (y,n,n,l) 3, 10^{-10}	542	47	8.2e-11	9.6	29.1	14
	BGS 2 (y,y,n,l) 3, 10^{-3}	663	132	6.7e-11	10.5	33.6	20
	BGS 3 (y,y,n,l)	746	127	8.7e-11	18.6	50.8	78
	BGS 4 (n,n,n,u) 3, 10^{-10}	542	96	9.0e-11	8.9	**27.2**	9
	BGS 5 (n,n,n,u) 3, 10^{-10}	542	98	8.1e-11	9.1	27.8	10
0.97	POWER	256	434	9.6e-11	7.3	67.5	
	GS	340	228	2.3e-11	8.6	43.6	
	BGS 1 (y,n,n,u) 3, 10^{-10}	542	85	5.6e-11	9.8	43.0	18
	BGS 2 (y,y,n,l) 3, 10^{-3}	663	214	7.7e-11	10.5	47.9	23
	BGS 3 (y,y,n,l)	746	210	9.1e-11	18.5	71.8	81
	BGS 4 (n,n,n,u) 3, 10^{-10}	542	168	9.2e-11	8.9	**40.1**	12
	BGS 5 (n,n,n,u) 3, 10^{-10}	542	171	9.3e-11	9.9	42.0	19
0.99	POWER	256	1,258	9.9e-11	7.4	182.5	
	GS	340	634	4.8e-11	8.3	105.9	
	BGS 1 (y,n,n,l) 3, 10^{-10}	542	267	7.0e-11	9.9	115.1	18
	BGS 2 (y,y,n,l) 3, 10^{-3}	663	590	8.3e-11	10.5	113.3	22
	BGS 3 (y,y,n,l)	746	594	9.5e-11	18.9	169.5	83
	BGS 4 (n,n,n,u) 3, 10^{-10}	542	502	9.6e-11	8.9	**100.9**	11
	BGS 5 (n,n,n,u) 3, 10^{-10}	542	511	9.7e-11	8.8	102.8	10

value of J, the number of blocks in A, that comes up as a result of the computation. When J is relatively small, the partitioning time will also be relatively small. Note that J equals 42 and 33 with partitioning 3 for the *Stanford* and *WebGoogle* matrices, respectively. Although the Cutfind algorithm shortens time per outer iteration, it does so at the expense of an increased number of iterations, and this sets back the total solution time. In any case, partitioning 3 is to be recommended for web matrices having nonzero plots as those of *Stanford* and *WebGoogle*, and the larger partitioning time will be clearly offset when the same web matrix is used multiple times.

GS is producing minimum total solution time in the *WebBase* matrix with $\alpha = 0.85$, but is a runner–up in eleven other problems. However, if minimum iteration time is considered, it is never a winner, and a runner–up only in eight problems. GS consistently reduces the number of iterations with a factor of about 2 over that of POWER and does not have much overhead associated with preprocessing, which make it competitive as observed in the literature before. BGS with partitioning 4 is producing minimum solution time in three problems, namely *Eu2005* with $\alpha \in \{0.95, 0.97, 0.99\}$, and performs relatively well in all other problems. When only Tarjan's algorithm is employed, BGS with partitioning 1 is producing minimum total solution time in the remaining sixteen problems (seventeen if minimum itera-

TABLE 5.4
Results for the WebGoogle *matrix.*

α	Solver	MB	Iterations	Residual	Setup (s)	Total (s)	Ratio
0.85	POWER	97	92	8.0e-11	3.2	32.8	
	GS	127	42	5.7e-11	3.9	19.4	
	BGS 1 (y,n,n,l) $5, 10^{-10}$	225	11	4.0e-11	8.0	18.6	15
	BGS 2 (y,n,n,l) $3, 10^{-3}$	225	42	7.4e-11	8.0	21.2	15
	BGS 3 (y,y,n,l)	324	41	6.8e-11	9.8	**17.5**	21
	BGS 4 (y,n,n,u) $3, 10^{-3}$	225	42	5.7e-11	5.3	20.3	7
	BGS 5 (n,n,n,u) $3, 10^{-3}$	225	42	5.7e-11	5.3	20.2	7
0.9	POWER	97	142	8.4e-11	3.4	48.7	
	GS	127	63	5.8e-11	3.8	26.9	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	225	24	7.1e-11	8.0	24.5	14
	BGS 2 (y,y,n,u) $3, 10^{-3}$	295	60	7.5e-11	10.1	26.9	21
	BGS 3 (y,y,n,l)	324	61	7.9e-11	10.0	**21.4**	21
	BGS 4 (n,n,n,u) $3, 10^{-5}$	225	63	5.8e-11	5.2	27.7	6
	BGS 5 (n,n,n,u) $3, 10^{-3}$	225	63	5.8e-11	5.3	27.8	6
0.95	POWER	97	291	9.1e-11	3.3	96.2	
	GS	127	121	7.7e-11	3.9	48.4	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	225	46	8.2e-11	8.0	41.3	15
	BGS 2 (y,y,n,u) $3, 10^{-3}$	295	118	8.4e-11	10.3	43.2	22
	BGS 3 (y,y,n,u)	324	127	8.1e-11	10.0	**33.7**	21
	BGS 4 (n,n,n,u) $3, 10^{-3}$	225	121	7.7e-11	5.2	48.5	6
	BGS 5 (n,n,n,u) $3, 10^{-3}$	225	121	7.7e-11	5.3	48.5	6
0.97	POWER	97	489	9.6e-11	3.3	159.3	
	GS	127	195	8.2e-11	3.8	75.7	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	225	74	8.7e-11	8.1	60.9	15
	BGS 2 (y,y,n,u) $3, 10^{-3}$	295	191	9.1e-11	10.1	63.4	21
	BGS 3 (y,y,n,u)	324	204	8.9e-11	9.9	**48.0**	21
	BGS 4 (n,n,n,u) $5, 10^{-3}$	225	195	8.2e-11	5.4	75.0	7
	BGS 5 (n,n,n,u) $5, 10^{-3}$	225	195	8.2e-11	5.3	75.0	6
0.99	POWER	97	1481	9.8e-11	3.2	472.9	
	GS	127	528	9.8e-11	3.9	197.9	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	225	198	9.8e-11	8.0	138.5	15
	BGS 2 (y,y,n,l) $5, 10^{-3}$	295	532	9.8e-11	9.8	158.1	21
	BGS 3 (y,y,n,u)	324	553	9.2e-11	9.9	**112.9**	21
	BGS 4 (n,n,n,u) $3, 10^{-5}$	225	527	9.8e-11	5.3	193.4	7
	BGS 5 (n,n,n,u) $3, 10^{-3}$	225	528	9.8e-11	5.3	193.5	7

tion time is considered). In fifteen (respectively, sixteen) of those problems, it is the winner with lower–triangular orientation, the exception being *In2004* with $\alpha = 0.99$. Whenever BGS is the winner with partitioning 1 and parameters (y,n,n,l) or (y,n,n,u), it yields the smallest number of iterations among the solvers in the tables. In all cases for the winning solvers, the stopping tolerance for the inner GS iteration for non–triangular diagonal blocks turns out to be 10^{-10}. When BGS 1 (y,n,n,l) $5, 10^{-10}$ is a winner, it reduces the number of iterations over that of POWER with a factor ranging between 7.8 (*In2004* with $\alpha = 0.95$) to 10.5 (*Stanford_Berkeley* with $\alpha = 0.99$). When BGS 1 (y,n,n,l) $3, 10^{-10}$ is a winner, it reduces the number of iterations over that of POWER with a factor ranging between 4.6 (*Stanford_Berkeley* with $\alpha = 0.85$) to 7.0 (*WebBase* with $\alpha = 0.97$). When BGS is a winner with partitioning 1 (y,n,n,l), the improvements in iteration time compared to POWER are between 44% and 63%. This corresponds to a speedup between 1.8 and 2.7. In Figures 5.1(b) and 5.1(c), we plot the % changes in iteration times of respectively BGS 1 (y,n,n,l) $3, 10^{-10}$ and BGS 1 (y,n,n,l) $5, 10^{-10}$ over that of GS for the thirty experiments performed. The two graphs show that the improvement in iteration time is at least 20% in 60% of the experiments and at least 15% in 70% of the experiments.

We remark that BGS 1 (y,n,n,l) $3, 10^{-10}$ is faster than BGS 1 (y,n,n,l) $3, 10^{-3}$ and

TABLE 5.5
Results for the In2004 *matrix.*

α	Solver	MB	Iterations	Residual	Setup (s)	Total (s)	Ratio
0.85	POWER	252	100	3.8e-10	6.9	21.3	
	GS	332	57	6.4e-11	8.9	18.3	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	551	21	4.9e-11	9.3	**17.0**	17
	BGS 2 (y,y,n,l) $3, 10^{-3}$	690	44	6.9e-11	9.7	18.8	19
	BGS 3 (y,y,n,u)	771	44	2.0e-11	29.7	39.8	158
	BGS 4 (n,n,n,u) $3, 10^{-5}$	551	52	7.5e-11	8.1	17.5	8
	BGS 5 (n,n,n,u) $3, 10^{-10}$	551	40	1.4e-11	8.0	17.3	8
0.9	POWER	252	151	4.8e-10	7.1	28.8	
	GS	332	84	7.2e-11	7.9	21.8	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	551	30	6.5e-11	9.4	**20.2**	16
	BGS 2 (y,y,n,u) $5, 10^{-3}$	690	63	7.3e-11	10.1	23.3	21
	BGS 3 (y,y,n,u)	771	65	3.2e-11	30.0	44.8	159
	BGS 4 (n,n,n,u) $3, 10^{-10}$	551	58	1.8e-11	8.7	22.0	11
	BGS 5 8n,n,n,u) $5, 10^{-5}$	551	58	1.8e-11	8.3	21.8	8
0.95	POWER	252	313	2.6e-10	6.9	51.6	
	GS	332	159	6.1e-11	7.7	34.0	
	BGS 1 (y,n,n,l) $5, 10^{-10}$	551	40	6.9e-11	9.2	**28.4**	16
	BGS 2 (y,y,n,u) $5, 10^{-3}$	690	124	8.5e-11	10.1	36.0	22
	BGS 3 (y,y,n,l)	771	128	8.9e-11	30.6	60.0	166
	BGS 4 (n,n,n,u) $3, 10^{-10}$	551	110	2.1e-11	8.3	33.3	10
	BGS 5 (n,n,n,u) $3, 10^{-10}$	551	110	2.3e-11	8.1	33.3	8
0.97	POWER	252	526	1.3e-10	6.9	82.1	
	GS	332	253	8.7e-11	7.8	49.7	
	BGS 1 (y,n,n,l) $5, 10^{-10}$	551	54	9.0e-11	9.0	**36.1**	15
	BGS 2 (y,n,n,l) $3, 10^{-10}$	551	145	9.4e-11	9.1	51.9	15
	BGS 3 (y,y,n,l)	771	207	9.2e-11	30.2	77.5	163
	BGS 4 (n,n,n,u) $3, 10^{-10}$	551	173	3.1e-11	8.3	47.4	10
	BGS 5 (n,n,n,u) $3, 10^{-10}$	551	173	3.3e-11	8.4	47.7	10
0.99	POWER	252	1,379	1.8e-10	7.0	204.9	
	GS	332	757	9.9e-11	7.9	133.0	
	BGS 1 (y,n,n,u) $3, 10^{-10}$	551	222	9.2e-11	9.2	**91.9**	15
	BGS 2 (y,n,n,l) $3, 10^{-10}$	551	387	9.7e-11	9.4	126.4	17
	BGS 3 (y,y,n,l)	771	556	9.3e-11	30.2	157.3	162
	BGS 4 (n,n,n,u) $3, 10^{-10}$	551	449	9.7e-11	8.6	108.0	11
	BGS 5 (n,n,n,u) $3, 10^{-10}$	551	448	9.8e-11	8.4	108.4	10

BGS 1 (y,n,n,l) $3, 10^{-5}$ in all of the thirty experiments. The improvement in iteration time with BGS 1 (y,n,n,l) $3, 10^{-10}$ over both solvers is at least 25% in 50% of the thirty experiments (results not shown). On the other hand, BGS 1 (y,n,n,l) $5, 10^{-10}$ is slower than BGS 1 (y,n,n,l) $5, 10^{-3}$ and BGS 1 (y,n,n,l) $5, 10^{-5}$ in less than 10% of the thirty experiments. The improvement in iteration time with BGS 1 (y,n,n,l) $5, 10^{-10}$ over both solvers is at least 25% in 50% of the thirty experiments (results not shown). If the performances of BGS 1 (y,n,n,l) $3, 10^{-10}$ and BGS 1 (y,n,n,l) $5, 10^{-10}$ are compared, they come across as being similar.

Among the three solvers we consider in this paper, POWER has the minimum memory requirement, whereas BGS has the highest memory requirement when partitioning 3 is employed. The memory requirement of BGS with partitioning 1 using only Tarjan's algorithm is about twice that of POWER, the memory requirement of BGS with partitioning 3 is about three times that of POWER, and the memory requirement of GS is comparable to that of POWER.

Another set of experiments are conducted by considering symmetric permutations of the six web matrices under study. To this end, ten random permutations for each web matrix is generated using Matlab, thus giving us altogether sixty experiments for $\alpha = 0.85$. All of

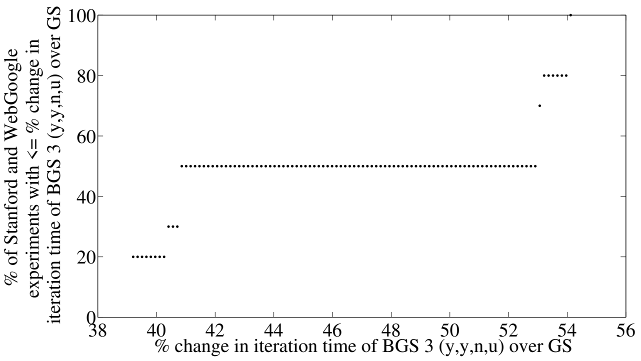
TABLE 5.6
Results for the WebBase *matrix.*

α	Solver	MB	Iterations	Residual	Setup (s)	Total (s)	Ratio
0.85	POWER	625	96	7.2e-11	21.3	53.4	
	GS	826	44	7.4e-11	20.5	**37.1**	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	1,341	18	6.8e-11	24.3	39.8	9
	BGS 2 (y,n,n,u) $5, 10^{-3}$	1,341	46	6.7e-11	23.7	45.0	7
	BGS 3 (y,y,n,l)	1,858	42	6.1e-11	38.1	63.8	50
	BGS 4 (n,n,n,u) $3, 10^{-5}$	1,341	44	6.3e-11	20.9	39.8	0
	BGS 5 (n,n,n,u) $5, 10^{-3}$	1,341	45	6.2e-11	21.0	40.2	0
0.9	POWER	625	147	8.3e-11	18.6	67.8	
	GS	826	65	7.9e-11	21.3	45.9	
	BGS 1 (y,n,n,l) $5, 10^{-10}$	1,341	17	7.9e-11	22.8	**44.1**	13
	BGS 2 (y,n,n,u) $5, 10^{-3}$	1,341	68	7.0e-11	22.8	54.0	13
	BGS 3 (y,y,n,l)	1,858	62	7.5e-11	37.4	75.2	56
	BGS 4 (n,n,n,u) $3, 10^{-3}$	1,341	66	7.4e-11	21.6	49.4	9
	BGS 5 (n,n,n,u) $5, 10^{-5}$	1,341	66	7.9e-11	20.8	49.1	7
0.95	POWER	625	300	9.4e-11	17.7	119.4	
	GS	826	126	8.3e-11	20.5	69.1	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	1,341	44	9.0e-11	24.2	**64.4**	19
	BGS 2 (y,y,n,u) $5, 10^{-3}$	1,657	122	9.0e-11	26.2	82.0	25
	BGS 3 (y,y,n,l)	1,858	121	7.4e-11	39.3	112.9	64
	BGS 4 (n,n,n,u) $3, 10^{-3}$	1,341	127	8.3e-11	20.6	73.6	9
	BGS 5 (n,n,n,u) $3, 10^{-5}$	1,341	127	8.7e-11	21.4	76.5	11
0.97	POWER	625	503	9.5e-11	18.5	185.8	
	GS	826	205	9.2e-11	21.0	98.7	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	1,341	72	9.2e-11	23.1	**88.3**	14
	BGS 2 (y,y,n,u) $5, 10^{-3}$	1,657	197	9.3e-11	26.1	115.9	23
	BGS 3 (y,y,n,l)	1,858	196	8.6e-11	38.0	156.8	59
	BGS 4 (n,n,n,u) $3, 10^{-5}$	1,341	206	9.2e-11	20.9	110.9	7
	BGS 5 8n,n,n,u) $5, 10^{-3}$	1,341	206	9.4e-11	20.6	108.3	6
0.99	POWER	625	1,511	9.9e-11	18.1	522.1	
	GS	826	583	9.7e-11	21.4	242.0	
	BGS 1 (y,n,n,l) $3, 10^{-10}$	1,341	221	9.8e-11	23.1	**222.5**	15
	BGS 2 (y,y,n,u) $3, 10^{-3}$	1,657	559	9.7e-11	25.5	279.8	22
	BGS 3 (y,y,n,l)	1,858	563	9.8e-11	37.4	378.3	58
	BGS 4 (n,n,n,u) $3, 10^{-3}$	1,341	587	9.8e-11	22.1	266.5	12
	BGS 5 (n,n,n,u) $3, 10^{-5}$	1,341	583	9.8e-11	20.8	265.0	8

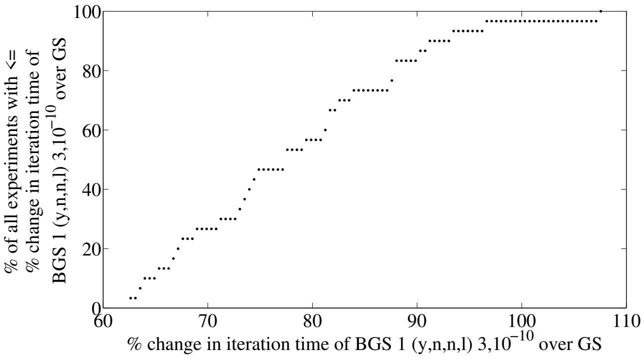
the sixty symmetrically permuted matrices are inspected for their nonzero structures and observed to possess nonzero plots that look uniformly distributed (as in the original versions of the *Stanford* and *WebGoogle* matrices). Furthermore, the performance of BGS with partitioning 3 turns out to be insensitive to random symmetric permutations of the input matrix and is much better than that of GS under the same symmetric permutation. It is noticed that GS favors the ordering of web pages in the original versions of the *Stanford_Berkeley*, *Eu2005*, *In2004*, and *WebBase* matrices. Results of numerical experiments in Figure 5.2 show that BGS with partitioning 3 is always better than BGS 1 (y,n,n,l) $3, 10^{-10}$, which in turn is always better than GS, and the % change in iteration time of BGS with partitioning 3 over that of GS is at least 45%.

To compare the performance of BGS with partitionings 1 (y,n,n,l) and 3 (y,y,n,l) and to test our hypothesis further, two sets of fifty sparse matrices of order 500,000 with densities of 0.625×10^{-5} and 1.25×10^{-5} are randomly generated using Matlab. The respective densities yield average numbers of nonzeros of about 4.1 and 7.2 per row including the diagonal element in each matrix. Furthermore, the sparsity patterns of these matrices follow the standard uniform distribution and look like those of *Stanford* and *WebGoogle*. Numerical experiments with our tool for $\alpha = 0.85$ show that BGS 3 (y,y,n,l) is always faster than

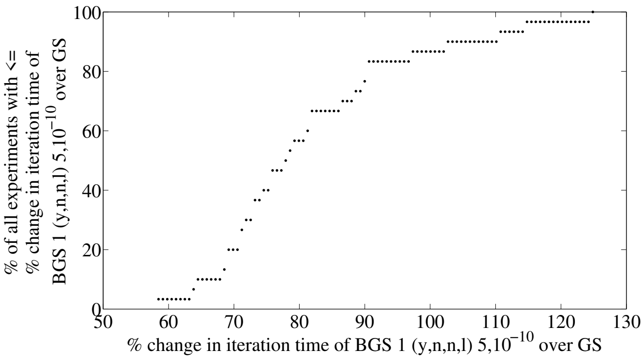
ETNA
Kent State University
http://etna.math.kent.edu



(a) BGS 3 (`y`,`y`,`n`,`u`) versus GS in $Stanford$ and $WebGoogle$ experiments



(b) BGS 1 (`y`,`n`,`n`,`l`) 3,10^{-10} versus GS in all experiments



(c) BGS 1 (`y`,`n`,`n`,`l`) 5,10^{-10} versus GS in all experiments

FIG. 5.1. *Plots for the % changes in iteration times of various BGS solvers over those of GS.*

BGS 1 (`y`,`n`,`n`,`l`) $3, 10^{-10}$ and the average improvement in iteration time in the first set of fifty sparse matrices is about 43% and that in the second set of fifty sparse matrices is about 42% (results not shown). These correspond to speedups of 1.8 and 1.7, respectively. More importantly, as depicted in Figures 5.3(a) and 5.3(b), in 90% of the experiments the % change in iteration time of BGS 3 (`y`,`y`,`n`,`l`) over those of POWER and GS are respectively at least 50% and 40% for the lower density case and 35% and 25% for the higher density case.

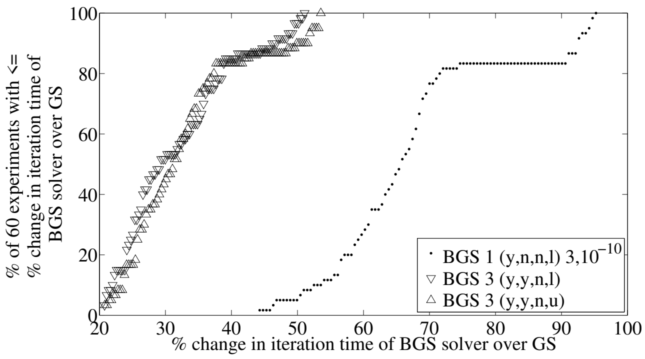
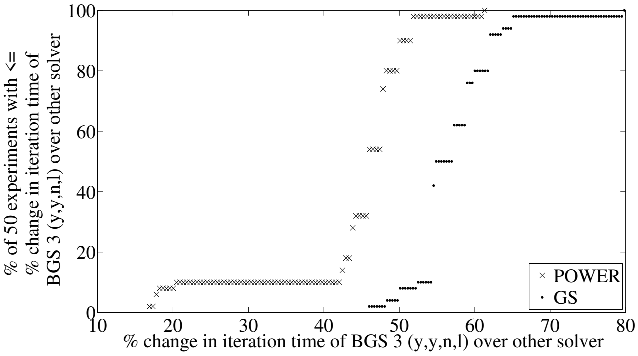


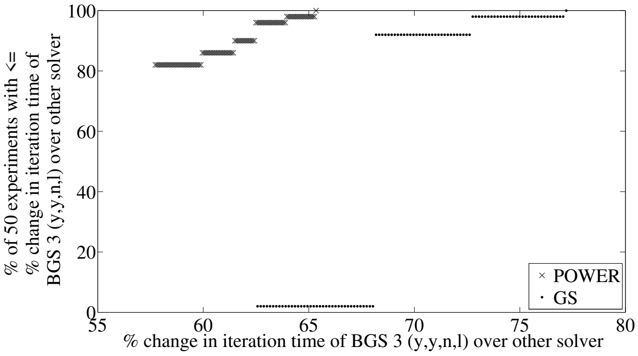
FIG. 5.2. *Plots for the % changes in iteration times of BGS solvers over those of GS on 60 experiments (10 symmetric random permutations of 6 web matrices each) with* $\alpha = 0.85$.

6. Conclusion. Various partitionings are considered with block iterative methods for the computation of the steady–state vector of Google–like stochastic matrices in a sequential setting. Some of these partitionings have triangular diagonal blocks, which can be solved directly. Effects of these block partitionings are analyzed through a set of numerical experiments with different values of the teleportation probability in which, among other things, memory requirements and solution times are measured. Time spent at the outset for computing the partitionings are relatively small and can very well be justified when the same web matrix is used multiple times. In general, block Gauss–Seidel with Tarjan's algorithm used to symmetrically permute the sparse term in the coefficient matrix of the system of linear equations to block lower–triangular form, with 3 to 5 point Gauss–Seidel iterations and a stopping tolerance of 10^{-10} to solve the diagonal blocks yields the fastest solver. Its memory requirement is about twice that of the power method and it decreases the number of iterations over that of the power method by a factor of more than 4.5. However, for matrices with nonzero plots that look uniformly distributed, a block partitioning with triangular diagonal blocks is to be favored with a lower–triangular orientation for values of α close to 0.85 and an upper–triangular orientation for values of α close to 1. Although requiring about three times memory as that of the power method and more time consuming to compute, such a partitioning when accompanying block Gauss–Seidel on such matrices yields iteration time improvements of over 20% compared to the second best solver. Furthermore, it is observed that the performance of block Gauss–Seidel with this partitioning is insensitive to random symmetric permutations of the input matrix (which also have nonzero plots that look uniformly distributed) and is much better than that of Gauss–Seidel under the same symmetric permutation. However, as has been repeatedly observed in the literature, Gauss–Seidel is to be recommended if memory is at a premium. It has comparable memory requirement to that of the power method and reduces the number of iterations over that of the power method by a factor of about 2.

Acknowledgements. The work of the first author is supported by the Turkish Academy of Sciences grant TÜBA–GEBİP and that of the second author is supported by The Scientific and Technological Research Council of Turkey. A preliminary version of this work is presented in the Dagstuhl Seminar 07071 "Web Information Retrieval and Linear Algebra Algorithms" in February 2007, and part of the work is carried out at Saarland University where the first author was on sabbatical leave. We thank the two anonymous referees for their remarks and



(a) BGS 3 (y,y,n,l) versus POWER and GS on 50 sparse matrices with densities 0.625×10^{-5} and $\alpha = 0.85$



(b) BGS 3 (y,n,n,l) versus POWER and GS on 50 sparse matrices with densities 1.25×10^{-5} and $\alpha = 0.85$

FIG. 5.3. *Plots for the % changes in iteration times of BGS 3 (*y,y,n,l*) over those of POWER and GS on 100 sparse matrices of order 500,000 with* $\alpha = 0.85$.

pointing out some of the references, and the second referee especially for encouraging us to consider random symmetric permutations, which led to an improved manuscript.

REFERENCES

[1] A. ARASU, J. NOVAK, A. S. TOMKINS, AND J. A. TOMLIN, *PageRank computation and the structure of the web: Experiments and algorithms*, in Proceedings of the 11th International Conference on World Wide Web, Honolulu, Hawaii, 2002, Poster Track. http://www2002.org/CDROM/poster/173.pdf

[2] R. BELLMAN, *Introduction to Matrix Analysis*, 2nd ed., SIAM, Philadelphia, Pennsylvania, 1997.

[3] P. BOLDI AND S. VIGNA, *The WebGraph Framework I: Compression Techniques*, in Proceedings of the 13th International Conference on World Wide Web, New York, ACM, 2004, pp. 595–601.

[4] P. BOLDI, B. CODENOTTI, M. SANTINI, AND S. VIGNA, *UbiCrawler: A Scalable Fully Distributed Web Crawler*, Software: Practice & Experience, 34 (2004), pp. 711–726.

[5] S. BRIN AND L. PAGE, *The anatomy of a large–scale hypertextual web search engine*, Computer Networks and ISDN Systems, 30 (1998), pp. 107–117.

[6] A. Z. BRODER, R. LEMPEL, F. MAGHOUL, AND J. PEDERSEN, *Efficient PageRank approximation via graph aggregation*, Inform. Retrieval, 9 (2006), pp. 123–138.

[7] P. BUCHHOLZ AND T. DAYAR, *On the convergence of a class of multilevel methods for large, sparse Markov chains*, SIAM J. Matrix Anal. Appl., 29 (2007), pp. 1025–1049

[8] T. DAVIS, *University of Florida Sparse Matrix Collection*, 2011, NA Digest, 92, no. 42, October 16, 1994, NA Digest, 96, no. 28, July 23, 1996, and NA Digest, 97, no. 23, June 7, 1997. http://www.cise.ufl.edu/research/sparse/matrices.

[9] T. DAYAR, *Obtaining triangular diagonal blocks using cutsets*, Technical Report BU–CE–0701, Department of Computer Engineering, Bilkent University, Ankara, Turkey, January 2007. http://www.cs.bilkent.edu.tr/tech-reports/2007/BU-CE-0701.pdf

[10] T. DAYAR AND G. N. NOYAN, *A software tool for the steady–state analysis of Google–like stochastic matrices*, in Proceeding of the Fourth International ICST Conference on Performance Evaluation Methodologies and Tools, Workshop on Tools for Solving Structured Markov Chains, Eds. B. Meini, A. Horvath, ICST, Brussels, Belgium, 2009, Article No. 14.

[11] ———, *Software for the steady–state analysis of Google–like stochastic matrices*, 2011. http://www.cs.bilkent.edu.tr/~tugrul/software.html

[12] T. DAYAR AND W. J. STEWART, *Comparison of partitioning techniques on two–level iterative solvers on large, sparse Markov chains*, SIAM J. Sci. Comput., 21 (2000), pp. 1691–1705.

[13] G. M. DEL CORSO, A. GULLI, AND F. ROMANI, *Fast PageRank computation via a sparse linear system*, Internet Math., 2 (2005), pp. 251–273.

[14] I. S. DUFF AND J. K. REID, *An implementation of Tarjan's algorithm for the block triangularization of a matrix*, ACM Trans. Math. Software, 4 (1978), pp. 137–147.

[15] N. EIRON, K. S. MCCURLEY, AND J. A. TOMLIN, *Ranking the web frontier*, in Proceedings of the 13th International Conference on World Wide Web, New York, ACM, 2004, pp. 309–318. http://research.yahoo.com/files/1p309.pdf

[16] D. F. GLEICH, A. P. GRAY, C. GREIF, AND T. LAU, *An inner–outer iteration for PageRank*, SIAM J. Sci. Comput., 32 (2010), pp. 349–371.

[17] G. H. GOLUB AND C. GREIF, *An Arnoldi-type algorithm for computing Page Rank*, BIT Numer. Math., 46 (2006), pp. 759–771.

[18] T. H. HAVELIWALA AND S. D. KAMVAR, *The second eigenvalue of the Google matrix*, Technical Report 2003–20, Stanford University, 2003. http://ilpubs.stanford.edu:8090/582/1/2003-20.pdf

[19] I. C. F. IPSEN AND T. M. SELEE, *PageRank computation, with special attention to dangling nodes*, SIAM J. Matrix Anal. Appl., 28 (2007), pp. 1281–1296.

[20] S. D. KAMVAR, T. H. HAVELIWALA, C. D. MANNING, AND G. H. GOLUB, *Extrapolation methods for accelerating PageRank computations*, in Proceedings of the 12th International Conference on World Wide Web, Budapest, Hungary, 2003, pp. 261–270.

[21] ———, *Exploiting the block structure of the web for computing PageRank*, Technical Report 2003–17, Stanford University, 2003. http://ilpubs.stanford.edu:8090/579/1/2003-17.pdf

[22] R. M. KARP, *Reducibility among combinatorial problems*, in Complexity of Computer Computations, R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, 1972, pp. 85–103.

[23] Laboratory of Web Algorithms Software, 2011. http://law.dsi.unimi.it/software.php

[24] A. N. LANGVILLE AND C. D. MEYER, *Deeper inside PageRank*, Internet Math., 1 (2004), pp. 335–380.

[25] ———, *A reordering for the PageRank problem*, SIAM J. of Sci. Comput., 27 (2006), pp. 2112–2120.

[26] Larbin home page, 2011. http://larbin.sourceforge.net/index-eng.html

[27] F. MCSHERRY, *A uniform approach to accelerated PageRank computation*, in the Proceedings of the 14th International Conference on the World Wide Web, Chiba, Japan, ACM, 2005, pp. 575–582.

[28] C. D. MEYER, *Matrix Analysis and Applied Linear Algebra*, SIAM, Philadelphia, 2000.

[29] V. MIGALLÓN, J. PENADÉS, AND D. B. SZYLD, *Block two-stage methods for singular systems and Markov chains*, Numer. Linear Algebra Appl., 3 (1996), pp. 413–426.

[30] G. N. NOYAN, *Steady–state analysis of Google–like matrices*, M.S. Thesis, Department of Computer Engineering, Bilkent University, Ankara, Turkey, September 2007.

[31] I. PULTAROVA, *Tarjan's algorithm in computing PageRank*, in the 13th Annual Conference Proceedings of Technical Computing Prague, C. Moler, A. Procházka, and B. Walden, eds., VŠCHT, Prague, Czech Republic, 2005.

[32] B. K. ROSEN, *Robust linear algorithms for cutsets*, J. Algorithms, 3 (1982), pp. 205–217.

[33] S. SERRA–CAPIZZANO, *Jordan canonical form of the Google matrix: A potential contribution to the PageRank computation*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 305–312.

[34] W. J. STEWART, *MARCA: Markov chain analyzer*, in Numerical Solution of Markov Chains, W. J. Stewart, ed., Marcel Dekker, New York, 1991, pp. 687–690.

[35] ———, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Princeton, 1994.

[36] R. E. TARJAN, *Depth–first search and linear graph algorithms*, SIAM J. Comput., 1 (1972), pp. 146–160.

[37] The HSL Mathematical Software Library, 2011. http://www.hsl.rl.ac.uk/

[38] The Stanford WebBase Project, 2011. http://diglib.stanford.edu:8091/~testbed/doc2/WebBase/

[39] Webgraph, 2011. http://webgraph.dsi.unimi.it