SIAM J. SCI. COMPUT.
Vol. 21, No. 5, pp. 1691–1705

COMPARISON OF PARTITIONING TECHNIQUES FOR TWO-LEVEL ITERATIVE SOLVERS ON LARGE, SPARSE MARKOV CHAINS*

TUĞRUL DAYAR† AND WILLIAM J. STEWART‡

Abstract. Experimental results for large, sparse Markov chains, especially the ill-conditioned nearly completely decomposable (NCD) ones, are few. We believe there is need for further research in this area, specifically to aid in the understanding of the effects of the degree of coupling of NCD Markov chains and their nonzero structure on the convergence characteristics and space requirements of iterative solvers. The work of several researchers has raised the following questions that led to research in a related direction: How must one go about partitioning the global coefficient matrix into blocks when the system is NCD and a two-level iterative solver (such as block SOR) is to be employed? Are block partitionings dictated by the NCD form of the stochastic one-step transition probability matrix necessarily superior to others? Is it worth investigating alternative partitionings? Better yet, for a fixed labeling and partitioning of the states, how does the performance of block SOR (or even that of point SOR) compare to the performance of the iterative aggregation-disaggregation (IAD) algorithm? Finally, is there any merit in using two-level iterative solvers when preconditioned Krylov subspace methods are available? We seek answers to these questions on a test suite of 13 Markov chains arising in 7 applications.

Key words. Markov chains, near-complete decomposability, partitioning, block SOR, iterative aggregation-disaggregation, Krylov subspace methods, preconditioning

AMS subject classifications. 60J10, 60J27, 65F50, 65F10, 65B99, 65U05

PII. S1064827598338159

1. Introduction.

Solving for the stationary distribution of an irreducible Markov chain amounts to computing a positive solution vector to a homogeneous system of linear equations with a singular coefficient matrix subject to a normalization constraint. That is, the $(n \times 1)$ unknown stationary vector x in

$$Ax = 0, \quad \|x\|_1 = 1 \tag{1}$$

is to be found. Here $A = I - P^T$ is an $n \times n$ singular M-matrix [6] and P is a one-step stochastic transition probability matrix.

Of special interest are nearly completely decomposable (NCD) Markov chains [23]. An NCD Markov chain may be symmetrically permuted to the form

$$P_{n\times n} = \begin{array}{c} \begin{matrix} n_1 & n_2 & \cdots & n_N \end{matrix} \\ \begin{pmatrix} P_{11} & P_{12} & \cdots & P_{1N} \\ P_{21} & P_{22} & \cdots & P_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ P_{N1} & P_{N2} & \cdots & P_{NN} \end{pmatrix} \end{array} \begin{matrix} n_1 \\ n_2 \\ \vdots \\ n_N \end{matrix} \tag{2}$$

in which the nonzero elements of the off-diagonal blocks are small compared with those of the diagonal blocks. The subblocks P_{ii} are square and of order n_i, with

*Received by the editors May 1, 1998; accepted for publication (in revised form) February 21, 1999; published electronically April 28, 2000.
http://www.siam.org/journals/sisc/21-5/33815.html
†Department of Computer Engineering, Bilkent University, 06533 Bilkent, Ankara, Turkey (tugrul@cs.bilkent.edu.tr).
‡Department of Computer Science, North Carolina State University, Raleigh, NC 27695-8206 (billy@csc.ncsu.edu).

$n = \sum_{i=1}^{N} n_i$. Let $P = \text{diag}(P_{11}, P_{22}, \ldots, P_{NN}) + E$. The quantity $\|E\|_\infty$ is referred to as the degree of coupling and is taken to be a measure of the decomposability of the matrix.

Despite recent advances in iterative methods, practicing performance analysts generally use methods based on splittings when they want to compare the performance of newly devised algorithms against existing ones or when they need candidate solvers to evaluate the performance of a systems model at hand. Experimental results for large, sparse Markov chains, especially the ill-conditioned NCD ones, are few. We believe there is a need for further research in this area, specifically to aid in the understanding of the effects of the degree of coupling of NCD Markov chains and their nonzero structure on the convergence characteristics and space requirements of iterative solvers.

The work of several researchers [25, 17, 18, 16, 8, 24, 19] has raised important and interesting questions that led to research in a related direction. These questions are the following: How must one go about partitioning the global coefficient matrix A in (1) into blocks when the system is NCD and a two-level iterative solver (such as block successive overrelaxation (SOR)) is to be employed? Are block partitionings dictated by the NCD form of P necessarily superior to others? Is it worth investigating alternative partitionings? Better yet, for a fixed labeling and partitioning of the states, how does the performance of block SOR (or even that of point SOR) compare to the performance of the iterative aggregation-disaggregation (IAD) algorithm [30]? Finally, is there any merit in using two-level iterative solvers when preconditioned Krylov subspace methods [3, 27, 14, 26, 12, 28] are available?

Four block partitioning techniques are considered. The first one results from the near-complete decomposability test (*ncdtest*) of the Markov chain analyzer (MARCA) [31]. It determines the strongly connected components of the transition probability matrix by ignoring the nonzeros less than a prespecified decomposability parameter. Then symmetric permutations are performed to put the matrix into the form in which the diagonal blocks form the strongly connected components. In a recent paper [9], it is shown that the *ncdtest* algorithm may fail to produce a correct NCD partitioning of the state space. The same paper highlights an improved NCD partitioning algorithm, which has the same run-time complexity as that of *ncdtest*. We name this new NCD partitioning algorithm *newncd* and experiment with it. Also two straightforward partitionings are investigated. The *equal* partitioning forms (approximately) equal-order blocks. The second straightforward partitioning, *other*, uses blocks of order 1,2,3,..., respectively. Finally, the threshold parameterized block ordering (TPABLO) algorithm [8] is considered on some of the test problems.

When seeking answers to these questions, we did not consider two-level solvers of the inner-outer iteration type [24] but attempted to solve diagonal blocks (and the coupling matrix [23] in IAD) directly by Gaussian elimination. The memory needed to solve the coupling matrix is set aside at the beginning and what is left is used for diagonal blocks. Blocks of orders 1 and 2 are treated separately. We obtain the LU factorizations of as many diagonal blocks as possible given available memory and do this in such a way that smaller blocks are treated first, leaving the big blocks to be solved using point SOR when there is insufficient memory. Currently, we use a fairly large tolerance (i.e., 10^{-3}), a relaxation parameter of 1.0 (hence, Gauss–Seidel), and a maximum number of iterations of 100 with the point SOR algorithm when solving diagonal blocks. Furthermore, the block Gauss–Seidel correction [34] in the disaggregation step of IAD is replaced by block SOR.

Preconditioned Krylov subspace methods [29, 13] are state-of-the-art iterative solvers, developed mostly in the last fifteen years, that may be used, among other things, to solve for the stationary distribution of Markov chains [32]. A concise discussion on popular Krylov subspace methods and the motivation behind preconditioning may be found in [4]. In this study, we consider the methods generalized minimum residual (GMRES), direct quasi-GMRES (DQGMRES), biconjugate gradient (BCG), conjugate gradient squared (CGS), biconjugate gradient stabilized (BCGStab), and quasi-minimal residual (QMR) with incomplete LU (ILU) factorization preconditioning. Chapter 4 of [32] presents some of these methods for Markov chains.

Section 2 includes a detailed description of the implementation framework. The results of the numerical experiments are analyzed in section 3. Appendices A–G in [10] provide a detailed explanation of each test problem, the nonzero plots of the underlying matrices, information about the matrices and the partitionings, and the results. In section 4, we draw some general conclusions.

2. Implementation framework. In this study, we experiment with the (point) SOR method [32, 4], which is a stationary iterative method, two types of two-level iterative methods, block SOR (BSOR) [32, 29, 24, 13] and IAD [30, 34, 32, 11], and the Krylov subspace methods GMRES, DQGMRES, BCG, CGS, BCGStab, and QMR (see [4, 29, 13] and the references therein).

2.1. Partitioning techniques. When applied to NCD Markov chains, one possibility is to order and partition the state space so that the stochastic matrix of transition probabilities has the form in (2). Obviously a zero degree of coupling (i.e., $\|E\|_\infty = 0$) implies a completely decomposable matrix. In NCD systems, there are eigenvalues close to 1. The poor separation of the unit eigenvalue results in a slow rate of convergence for standard matrix iterative methods. Two-level iterative methods, in general, do not suffer from this limitation which makes them suitable for such systems.

Four block partitioning techniques are considered. The first one is the *ncdtest* partitioning algorithm in MARCA. This algorithm searches for the strongly connected components (SCCs) of the directed graph (digraph) associated with the matrix obtained by zeroing the elements of P that are less than a user specified decomposability parameter γ, a real number between 0 and 1. The subset(s) of states output by the SCC search algorithm are identified as forming the NCD blocks P_{ii}. If the matrix is not already in the form (2), then symmetric permutations are performed to put it into the form in which the diagonal blocks form the SCCs. The *ncdtest* algorithm may fail to produce a correct NCD partitioning of the state space due to the possibility of having nonzeros greater than or equal to γ in the off-diagonal blocks. This is simply because the algorithm zeroes out the elements that are smaller than γ, but *not* those that are larger. The example in [9] shows how this can happen and presents an improved NCD partitioning algorithm, which has the same run-time complexity as that of *ncdtest*. We name this new NCD partitioning algorithm *newncd* and experiment with it. For clarity, we use γ' to denote the decomposability parameter of the *newncd* algorithm. Also two straightforward partitionings are investigated. The *equal* partitioning has $\sqrt{n}$ blocks of order $\sqrt{n}$ if n is a perfect square. If $n \neq \lfloor\sqrt{n}\rfloor^2$, there is an extra block of order $n - \lfloor\sqrt{n}\rfloor^2$. The second straightforward partitioning, *other*, has nb blocks of order, respectively, $1, 2, \ldots, nb$ if $n = \sum_{i=1}^{nb} i$ (and possibly an extra block of order $n - \sum_{i=1}^{nb} i$ if the difference is positive). This last partitioning ensures that there are about $\sqrt{2n}$ blocks and the largest block solved is of order roughly $\sqrt{2n}$.

We have also experimented with the TPABLO algorithm [8] on some of the test problems. The original parameterized block ordering (PABLO) algorithm presented in [25] aims at obtaining dense diagonal blocks by performing symmetric permutations of a given sparse coefficient matrix using two input parameters. The first parameter $\alpha > 0$ is used to ensure that the addition of a new state to a diagonal block will keep the ratio of the percentage of nonzero elements in that block to the percentage of nonzero elements in that block if the state were not added above α. The second parameter $0 \leq \beta \leq 1$ is used to ensure that each state in a diagonal block is adjacent to at least a certain proportion, i.e., β, of the states inside the diagonal block. TPABLO has three other parameters requiring a total of five parameters. The third parameter, $\gamma \geq 0$, either makes sure the permuted matrix does not have any elements in the off-diagonal blocks that are larger than γ in absolute value, or it makes sure all elements in the diagonal blocks are above γ in absolute value with the possibility that some elements in the off-diagonal blocks are also larger than γ in absolute value. The fourth and fifth parameters *minbs* and *maxbs* are used to control the minimum and maximum permissible order of diagonal blocks, respectively.

2.2. Preconditioners. The main idea behind preconditioning is to accelerate convergence by transforming the linear system so that the difference between the dominant and the subdominant eigenvalue of the preconditioned coefficient matrix is larger than what it used to be in the original system. The need for a preconditioner becomes vital when dealing with NCD systems. To provide effective solvers, Krylov subspace methods are used with preconditioners.

Consider the system of linear equations in (1) which can be transformed into the (left-)preconditioned equivalent system

$$M^{-1}Ax = 0,$$

where the preconditioner matrix M (also called preconditioner) has the property that it is a cheap approximation of A. The more M^{-1} resembles the group inverse $A^{\sharp}$ [22], the faster the method converges [32, p. 143]. The system is solved based on imposing the necessary convergence criteria on the preconditioned residual vector $r = -M^{-1}Ax$. The matrix M^{-1} need not be formed explicitly since the preconditioned residual vector may be computed by solving the system $Mr = -Ax$.

Various types of preconditioners have been (and are still being) developed (see [29, 5, 13]). Their efficiency is highly dependent on the system to be solved, and it is quite difficult to forecast which preconditioner is the best for a given system. In this study, we consider only preconditioners obtained from incomplete LU factorizations (ILU). We should remark that the SOR method and its block version are preconditioned power iterations (see [32, p. 144] and [13, pp. 26, 147–149]) and therefore can also be used with Krylov subspace methods as preconditioners. Saad points out in [27, p. 467] that "in addition to incomplete factorization preconditioning one can also use the more traditional relaxation methods such as the SOR, or SSOR iteration, as preconditioners. Our experience in [10] (reference [26] of this paper) with these techniques on real Markov chain problems is that they are not as efficient as the ILU type preconditioners. For further details see [10]." Nevertheless, we considered BSOR with *equal* partitioning as a preconditioner for BCGStab. We return to this point in the third section.

In computing ILU preconditioners, first an LU factorization of the coefficient matrix A is initiated. Throughout the factorization, nonzero elements are omitted according to different rules. These rules characterize the ILU type. Thus, instead of

ending up with an exact LU factorization, what we obtain is of the form

$$A = \tilde{L}\tilde{U} + F, \tag{3}$$

where F, called the remainder, is expected to be small in some sense. The incomplete LU factors $\tilde{L}$ and $\tilde{U}$ are lower and upper triangular matrices, respectively.

Recall that the coefficient matrices appearing in the systems of interest are irreducible singular M-matrices. It has been shown that ILU factorizations exist for such matrices [7] (in exact arithmetic) and that they are at least as stable as the complete LU factorization without partial pivoting (see [21, p. 152]). We should stress that not much work has been done in studying what constitutes a good incomplete factorization for Markov chain models [27, 26, 28]. Further studies are still needed.

Three types of incomplete LU factorizations are considered. The first imposes on the computed preconditioner the same nonzero structure as the original matrix and is called ILU0. The idea of ILU0 is to drop all fill-in elements which occur during the LU factorization (recall that a fill-in element refers to a nonzero element introduced in the matrix which holds the LU factors in a location where there was initially a zero element in the original matrix).

The second is called ILUTH and is a threshold-based approach. In ILUTH, the factorization takes place in a row-by-row manner. The dropping rule of this preconditioning technique is to zero out all elements having an absolute value less than a prespecified threshold. The only exception is that the dropping rule does not apply to the diagonal elements which are kept no matter how small they become. The dropping rule is applied just after the multipliers are formed, once, and applied one more time right after the reduction of a row is over.

The third type of ILU preconditioner forces the computed factors to have at most a prespecified fixed number of nonzero elements per row and is called ILUK. This approach enables the user to control the amount of fill-in. Therefore, it is especially suitable for those cases where there is only a fixed amount of memory available to store the incomplete factors $\tilde{L}$ and $\tilde{U}$. Each time a row has been reduced, a search is conducted to find the K largest elements in absolute value, a timewise costly process. All other elements in the row are annihilated. As for ILUTH, the diagonal elements are preserved regardless of their magnitude.

2.3. Implementation issues. Since we are dealing with large sparse systems,[1] we need to work in sparse storage. We use the compact sparse row (CSR) Harwell–Boeing format, which requires three arrays: one real and one integer of size nz (i.e., number of nonzero elements in the coefficient matrix) and one integer of size $n + 1$. Unless otherwise specified, by reductions we mean row-reductions. This strategy is used to take full advantage of the row-by-row storage of the CSR format. We would like to remark that we generate and store all test matrices using the MARCA package. Later these files are used as input to the solvers.

All code is written in Fortran and compiled in double precision with $g77$ on a SUN Sparcstation 4 with 64 Mb RAM running Solaris 2.5. The numerical experiments are timed using a C function that reports CPU time. SOR, the two-level iterative solvers, the four partitioning algorithms, and the three ILU preconditioners are part of the MARCA software package version 3.0. The Krylov subspace methods are implemented

[1] The average order of the seven problems we experimented with is $33,278$; the largest and the smallest matrices are of order $104,625$ and $8,258$, respectively.

using two one-dimensional arrays defined at the beginning of the driver program to hold double precision and integer values.

In two-level iterative methods, we attempt to solve diagonal blocks and the coupling matrix in IAD directly by Gaussian elimination. The memory needed to solve the coupling matrix is set aside at the beginning and what is left is used for the diagonal blocks. If there is not enough space for solving the coupling matrix, the method fails. Blocks of orders 1 and 2 are treated separately. We obtain the LU factorizations of as many diagonal blocks as possible given available memory and do this in such a way that smaller blocks are treated first, leaving the big blocks to be solved using SOR when there is insufficient memory. In order to accelerate this process we use a fairly large tolerance 10^{-3}, a maximum number of iterations of 100, and a relaxation parameter of 1.0 (hence, Gauss–Seidel) with the SOR algorithm when solving the remaining diagonal blocks. Furthermore, the block Gauss–Seidel correction [34] in the disaggregation step is replaced by BSOR. The results reported are always those that are obtained using the optimal relaxation parameter ω with one significant digit after the decimal point.

The *newncd* partitioning algorithm is implemented so that if there are states that are left in singletons after the NCD partition corresponding to a decomposability parameter is determined, they are grouped into a single subset which forms the last NCD block. When choosing decomposability parameters γ for *ncdtest*, we report the smallest and largest values of γ (as 0.10 times a power of 10) for which there are at least two blocks in the partition. On the other hand, when experimenting with *newncd*, we had to work on a finer scale with γ' since there were not as many possibilities as γ of *ncdtest*. Hence, we report the smallest and largest values of γ' (in two decimal digits of precision times a power of 10) for which there are at least two blocks in the partition (see [10, appendices A–G]).

The dimension of the Krylov subspace we used for (restarted) GMRES is 20 (i.e., $m = 20$). The number of vectors kept in DQGMRES is 20 (i.e., $k = 20$) in all but one of the applications (*qn*), where we had to limit k to 7 or 9 depending on the preconditioner used. With each Krylov subspace solver, we used three different thresholds for the ILUTH preconditioner: 10^{-2}, 10^{-3}, and 10^{-5}. Due to the amount of fill-in and the computation time, it is futile to experiment with a threshold value of 10^{-5} in two of the applications (*qn*, *mutex*). In ILUK, we allowed a maximum of 10 nonzero elements per row of the preconditioned matrix (i.e., $K = 10$). In all the Krylov subspace methods implemented, we use left-preconditioning and take the ILU preconditioner as $M = \tilde{L}\tilde{U}$ (see (3) in section 2.3).

In order to regulate the amount of fill-in produced, ILUTH is implemented in such a way that before the reduction of a given row, the number of free entries in the double precision work array is divided by the number of remaining rows to be reduced. This gives us the maximum number of allowable nonzero elements that can be stored for the current row. If the reduction gives a higher number of nonzero elements than the allowable maximum, the threshold is multiplied by 10 and the dropping rule is applied again. This is repeated until the number of nonzero elements in a given row becomes less than or equal to the allowable maximum. The first row of the matrix is not reduced. As suggested by Axelsson in [2, p. 259] it is possible to "... modify the pivot entry by adding a positive (usually small) number to it, when required, before the elimination takes place to guarantee that pivot entries will have sufficient size...." We have not considered modified ILU preconditioners as such, and have let the ILU factorization proceed naturally until we obtain a reduced diagonal element

with magnitude less than 10^{-300} in which case the computation is forced to fail. This is done to avoid division by zero due to underflow of small pivot elements, and failure happens only in one of our applications (*leaky*) which is a pathological case.

In ILUK, the Kth largest value in magnitude, say max, in the reduced row is determined. Then all elements having an absolute value less than max are set to zero. If the number of nonzero elements in the row is still higher than K, the reduced row is scanned from left to right and elements having an absolute value equal to max are set to zero until the number of nonzero elements becomes K. As in ILUTH, the reduction does not include the first row of the matrix and the method fails if any reduced diagonal element is found to be less than 10^{-300}.

In order to reduce the possibility of underflow and overflow, each row of the coefficient matrix is multiplied by the inverse of the largest value in magnitude in that row (i.e., absolute value of the diagonal element). This is a scaling operation and it transforms the system to a more suitable form without altering the global solution. Normalizing the solution vector at each iteration is an alternative way to limit the effect of underflow and overflow and up to a certain extent control the irregular convergence behavior of some iterative methods. The drawback of this strategy is that it may lead to considerable loss of precision due to rounding errors that occur at each iteration or to even divergence. In SOR, BSOR, and IAD, the coefficient matrix is scaled and the solution vector is normalized at each iteration. As for the Krylov subspace methods we implemented, the coefficient matrix is not scaled and the solution vector is normalized only upon convergence.

The initial approximation chosen is the uniform distribution. The convergence criteria we use in SOR and the other solvers, respectively, are

$$\textbf{stop if} \quad k \geq maxit \quad \textbf{or} \quad \|r^{(k)}\|_\infty \leq stop\_tol$$

and

$$\textbf{stop if } k \geq maxit \quad \textbf{or} \quad \|r^{(k)}\|_\infty \leq stop\_tol \quad \textbf{or} \quad \left(\|r^{(k)}\|_\infty \leq stop\_tol_1 \quad \textbf{and} \quad |\|r^{(k)}\|_\infty - \|r^{(k-1)}\|_\infty| \leq stop\_tol_2\right).$$

Here k is the iteration number, $r^{(k)}$ is the residual vector at iteration k, $maxit$ is the maximum number of iterations the algorithm will be permitted to perform, and $stop\_tol$ is the user-specified stopping tolerance, which should be less than 1 and greater than machine epsilon. The stopping tolerance, $stop\_tol$, is set to 10^{-10}, meaning we consider the entries of A (our right-hand side is 0) to have errors in the range $\pm 10^{-10}\|A\|$. The use of $stop\_tol_1$ and $stop\_tol_2$ forces the solver to terminate when the norm of the residual vector is decreasing too slowly while the difference between two successive residuals is small enough. In the experiments, we set $stop\_tol_1$ and $stop\_tol_2$ to 10^{-6} and 10^{-12}, respectively. As for $maxit$, we use 100, 500, or 1,000, depending on the solver and the particular problem at hand.

For each problem solved, the true residual and the relative backward error in the solution (see [15, 2]) are computed. The true residual is computed as $\|A\hat{x}\|_\infty$, where $\hat{x}$ is the normalized approximate solution upon convergence. The relative backward error is computed as $\|A\hat{x}\|_\infty/(\|A\|_\infty\|\hat{x}\|_\infty)$. In SOR and two-level iterative solvers, we explicitly compute the residual vector at each iteration meaning there is an extra matrix-vector multiplication involved. On the other hand, the residual vector is a byproduct of all Krylov subspace methods (except GMRES and DQGMRES) from which $\|r^{(k)}\|_\infty$ can be easily computed. In GMRES, we use $\|r^{(k)}\|_2$ (and not $\|r^{(k)}\|_\infty$)

in the convergence test at each (inner) iteration. At the end of each restart, the residual vector is computed explicitly from the unnormalized current approximation and then $\|r^{(k)}\|_\infty$ compared with *stop_tol*. As for DQGMRES, it is the scalar gamma that is used in the convergence test at each iteration (see the algorithm in [29]). If BCGStab terminates due to the convergence test $\|s\|_\infty \leq$ *stop_tol* (see the BCGStab algorithm in [4, p. 27]), it is indicated in the results.

3. Overview of results. In this section, we overview the results of the numerical experiments performed on 13 test cases. We consider 7 models, 6 of which appear in [33] and one which is discussed in [1]. Details corresponding to each of the 7 applications can be found in [10, appendices A–G]. The applications are named *pushout*, 2*D*, *ncd*, *telecom*, *qn*, *leaky*, and *mutex*. All seven models arise in Markov chain applications. Three of these models (*pushout*, *ncd*, and *mutex*) are chosen and 2 more test problems for each one generated (namely, *medium*, *hard*, *ncd_alt*1, *ncd_alt*2, *mutex_alt*1, and *mutex_alt*2) giving us a total of 13 test problems. The original *pushout* test problem is given the name *easy* as it is somewhat easier to solve compared to the *medium* and *hard* test matrices.

Table 1 summarizes the characteristics of the 13 test matrices. The majority of the matrices would be ranked among the largest of the matrices considered in the Matrix Market [20]. The sym column indicates whether the test matrix has symmetric nonzero structure. Two of the problems (*ncd*, *mutex*) give test matrices with symmetric nonzero structure. We would single out *leaky*, *ncd_alt*2, *ncd*, *ncd_alt*1, and *telecom* as NCD test cases based on the smallest decomposability parameter that could be used with the *newncd* partitioning algorithm. These test cases have degree of coupling values (see column $\|E\|_\infty$) ranging (in the given order) from $0.2e-101$ to $0.9e-2$. The test cases *medium*, *qn*, *hard*, *easy*, and 2*D* have degree of coupling values between $0.7e+0$ and $0.1e+0$. Hence, they are not as NCD. The remaining test cases *mutex*, *mutex_alt*1, and *mutex_alt*2 lie somewhere in between, all with degree of coupling $0.2e-1$. The number of blocks in the partition followed by the order of the smallest and largest blocks corresponding to $\|E\|_\infty$ are given in the column Partition. We have noticed during the experiments that the *ncdtest* algorithm is unable to find partitionings in NCD form with small degree of coupling values except for the *ncd* test matrices. In fact, the degree of coupling values corresponding to various *ncdtest* partitionings in all the other test matrices are notoriously large (see the discussion in section 2.1). The symmetric nonzero structure of the *ncd* test problem seems to have helped the *ncdtest* algorithm. The (a_0, a_1) column in Table 1 gives the coefficient of asymmetry of each test matrix, where $a_0 = 0.5\|A + A^T\|_1, a_1 = 0.5\|A - A^T\|_1$, and $a_o \ll a_1$ imply high asymmetry for each test matrix. None of the matrices is highly asymmetric. Finally, the last two columns give, respectively, the lower and higher bandwidth of each test matrix excluding the diagonal.

The time spent for partitioning using *equal* and *other* is negligible. On the other hand, the time to partition a given test matrix using the *ncdtest* and *newncd* algorithms has two components: time spent to determine the partition and time to permute the coefficient matrix according to the ordering of states in the computed partition. The time taken by the *ncdtest* and *newncd* partitionings used in our experiments does not exceed, respectively, 1 and 1.8 seconds except for matrices generated from the *qn*, *leaky*, and *mutex* test problems. The time spent by *ncdtest* and *newncd* for the *qn* test matrix is no more than 5.4 and 6.8 seconds, respectively. The time spent by *ncdtest* and *newncd* for the *leaky* test matrix is no more than 2.8 and 8.5 seconds, respectively. Finally, the time spent by *ncdtest* and *newncd* for the *mutex*

TABLE 1
Characteristics of the 13 *test matrices.*

Matrix	n	nz	sym	$\|E\|_\infty$	Partition	(a_0, a_1)	lower	higher
easy	20,301	140,504	no	$0.1e+0$	$3: 22-20,048$	(2.97, 1.96)	201	201
medium	20,301	140,504	no	$0.7e+0$	$67: 4-7,430$	(1.90, 0.94)	201	201
hard	20,301	140,504	no	$0.3e+0$	$2: 12-20,289$	(1.97, 0.99)	201	201
$2D$	16,641	66,049	no	$0.1e+0$	$129: 129-129$	(2.00, 1.00)	65	129
ncd	23,426	256,026	yes	$0.2e-3$	$21: 528-5,456$	(2.00, 0.98)	460	460
*ncd_alt*1	23,426	256,026	yes	$0.2e-3$	$21: 528-5,456$	(2.00, 0.98)	460	460
*ncd_alt*2	23,426	256,026	yes	$0.8e-4$	$51: 1-1,326$	(2.00, 0.98)	460	460
telecom	20,491	101,041	no	$0.9e-2$	$3: 1-20,488$	(2.27, 1.14)	31	60
qn	104,625	593,115	no	$0.2e+0$	$15: 6,975-6,975$	(2.06, 1.06)	2,728	5,385
leaky	8,258	197,474	no	$0.2e-101$	$2: 1-8,257$	(2.46, 1.46)	191	435
mutex	39,203	563,491	yes	$0.2e-1$	$2: 16,384-22,819$	(1.64, 0.69)	13,495	13,495
*mutex_alt*1	39,203	563,491	yes	$0.2e-1$	$2: 16,384-22,819$	(1.61, 0.70)	13,495	13,495
*mutex_alt*2	39,203	563,491	yes	$0.2e-1$	$2: 16,384-22,819$	(1.61, 0.70)	13,495	13,495

test matrices is no more than 5.5 and 5.8 seconds, respectively.

We present the fastest 5 solvers according to total solution time for the 13 test matrices in Table 2. Beside the name of a Krylov subspace solver, we either write ILU0 or the threshold of the ILUTH preconditioner employed. For two-level iterative solvers, the value in parentheses beside the solver's name is the decomposability parameter used in the *ncdtest*/*newncd* partitioning algorithm, underlined in the latter case. The figures in the P.T. and T. columns, respectively, denote partitioning/preconditioning and solution times. The relaxation parameter given by ω is the optimal one. The figures in the #it column represent the number of iterations taken to convergence and those in the Bk.Error column give the corresponding relative backward error in the computed solution. A superscript s in the #it column indicates BCGStab convergence due to $\|s\|_\infty$, whereas a superscript dagger (†) in the same column indicates convergence due to the difference between the last two residuals (see section 2.3). For two-level solvers, we denote by Blocks the number of diagonal blocks solved iteratively, followed after a colon by the number of blocks in the partitioning and inside the parentheses by the order of the smallest and largest blocks, respectively. For ILU preconditioned Krylov subspace solvers, we denote by *nzlu* the sum of the number of nonzeros in $\tilde{L}$ and $\tilde{U}$ (see (3) in section 2.2).

For $2D$, the fastest two-level solver is IAD with *newncd* $\gamma = 0.57e-1$ taking partitioning time 0.7 sec and solution time 7.5 sec. For *ncd_alt*1, the fastest Krylov subspace solver is BCGStab with ILUTH(10^{-5}) taking preconditioning time 19.7 sec and solution time 4.5 sec. The chosen ILU preconditioners fail to be computed for the *leaky* test matrix.

Numerical experiments show that two-level iterative solvers are in general very competitive with ILU preconditioned Krylov subspace solvers. Out of 10 test matrices for which two-level iterative solvers are winners, BSOR is the fastest solver for 8 test matrices, 4 times with *equal* (*medium*, *ncd_alt*2, *mutex_alt*1, *mutex_alt*2), twice with *other* (*easy*, *ncd_alt*1), and twice with *newncd* (*telecom*, *leaky*). IAD is the fastest solver for 2 test matrices (*hard*, *ncd*) both with *newncd*. CGS is the fastest solver for 2 test matrices (*qn*, *mutex*) both with ILU0 and BCGStab is the fastest solver for 1 test matrix ($2D$) with ILUTH(10^{-5}). Obviously, these results depend on the particular implementations and should be observed with care. For instance, among the 65 iterative solvers appearing in Table 2, only 35 are two-level solvers. Now we inspect the results in more detail, then draw general conclusions in the next section.

It is noticed that the more balanced, in terms of the order of blocks, is the partitioning, the better two-level iterative solvers take advantage of the divide-and-conquer notion, and hence the faster they converge. For the two-level solvers appearing in Ta-

TABLE 2
Fastest solvers of the 13 *test matrices.*

Matrix	#	Solver	ω	P.T.	T.	#it	Bk.Error	Blocks/$nzlu$
easy	1	BSOR, *other*	1.0	0.0	2.3	5	$0.3e-11$	$0:\ 201(\ 1-\ 201)$
	2	CGS, (ILU0)	—	0.5	2.4	4	$0.1e-13$	$140,504$
	3	BCGStab, (ILU0)	—	0.5	2.6	4	$0.3e-15$	$140,504$
	4	GMRES, (ILU0)	—	0.5	2.6	7	$0.7e-16$	$140,504$
	5	BSOR, $(0.10e-2)$	1.0	0.7	2.6	5	$0.5e-11$	$0:\ 4,060(\ 1-\ 162)$
medium	1	BSOR, *equal*	1.0	0.0	4.3	1	$0.6e-15$	$0:\ 143(137-\ 142)$
	2	IAD, *equal*	1.0	0.0	5.3	1	$0.4e-15$	$0:\ 143(137-\ 142)$
	3	IAD, *other*	1.0	0.0	5.8	10	$0.4e-10$	$0:\ 201(\ 1-\ 201)$
	4	BCGStab, (10^{-3})	—	6.4	3.2	4	$0.8e-16$	$275,253$
	5	CGS, (10^{-3})	—	6.4	4.0	5	$0.4e-16$	$275,253$
hard	1	IAD, $(0.20e+0)$	1.4	1.2	33.7	51	$0.2e-08$	$0:\ 201(\ 3-4,860)$
	2	BCGStab, (10^{-3})	—	12.4	22.7	14	$0.3e-08$	$860,386$
	3	GMRES, (10^{-3})	—	12.4	35.2	40	$0.4e-10$	$860,386$
	4	IAD, *other*	1.3	0.0	49.0	127	$0.3e-08$	$0:\ 201(\ 1-\ 201)$
	5	CGS, (10^{-3})	—	12.4	37.2	23	$0.5e-09$	$860,386$
$2D$	1	BCGStab, (10^{-5})	—	3.3	1.0	2^s	$0.6e-09$	$250,897$
	2	CGS, (10^{-5})	—	3.3	1.3	2	$0.9e-11$	$250,897$
	3	GMRES, (10^{-5})	—	3.3	1.6	4	$0.1e-11$	$250,897$
	4	BCGStab, (10^{-3})	—	2.7	2.2	5	$0.4e-10$	$138,392$
	5	CGS, (10^{-3})	—	2.7	2.4	5	$0.4e-11$	$138,392$
ncd	1	IAD, $(0.10e+0)$	1.0	1.4	7.9	4	$0.8e-12$	$0:\ 1,221(\ 2-\ 326)$
	2	IAD, $(0.10e-3)$	1.0	1.0	19.7	3	$0.2e-14$	$0:\ 51(\ 1-\ 1,326)$
	3	BSOR, $(0.10e-3)$	1.0	1.0	20.4	11	$0.4e-12$	$0:\ 51(\ 1-\ 1,326)$
	4	BSOR, $(0.10e-2)$	1.0	1.8	20.3	11	$0.4e-12$	$0:\ 51(\ 1-\ 1,326)$
	5	BCGStab, (10^{-5})	—	19.8	3.8	4	$0.3e-11$	$282,825$
*ncd_alt*1	1	BSOR, *other*	1.0	0.0	2.9	6	$0.6e-15$	$0:\ 216(\ 1-\ 215)$
	2	IAD, $(0.10e-3)$	1.0	0.9	4.8	2	$0.8e-16$	$0:\ 1,326(\ 1-\ 51)$
	3	IAD, *equal*	1.0	0.0	5.8	$5^\dagger$	$0.7e-08$	$0:\ 154(\ 17-\ 153)$
	4	IAD, *other*	1.0	0.0	5.9	6	$0.2e-15$	$0:\ 216(\ 1-\ 215)$
	5	BSOR, $(0.10e+0)$	1.0	1.4	4.8	9	$0.5e-11$	$0:\ 1,221(\ 2-\ 326)$
*ncd_alt*2	1	BSOR, *equal*	1.0	0.0	1.5	1	$0.3e-13$	$0:\ 154(\ 17-\ 153)$
	2	BSOR, $(0.10e+0)$	1.1	1.4	1.9	1	$0.4e-15$	$0:\ 1,221(\ 2-\ 326)$
	3	IAD, $(0.10e+0)$	1.0	1.4	4.1	1	$0.3e-16$	$0:\ 1,221(\ 2-\ 326)$
	4	IAD, $(0.10e-1)$	1.2	1.4	10.8	6	$0.4e-12$	$0:\ 1,273(\ 2-\ 58)$
	5	BCGStab, (10^{-5})	—	18.3	13.9	18^s	$0.3e-11$	$241,259$

ble 2, none of the diagonal blocks are solved iteratively. The IAD algorithm proves to be competitive with BSOR. Seventeen of the 35 two-level iterative solvers in Table 2 are IAD. When the coupling matrix is of reasonable size, IAD usually gives good performance. We especially recommend IAD for those cases that have a small degree of coupling. However, the drawback of IAD is that it may fail if the coupling matrix is reducible in floating point arithmetic or require an unreasonably long time to converge when the coupling matrix is large. Straightforward partitionings, especially *equal*, are very competitive with those of *newncd*. Out of 10 test matrices for which two-level iterative solvers provide winners, 4 are with *equal*, 2 are with *other*, and 4 are with *newncd* partitionings. Out of 35 two-level solvers in Table 2, 14 are using *newncd* partitionings, and 15 are using *equal* and *other* partitionings.

SOR does not give satisfactory results; it converges in less than 1,000 iterations in eight of the test matrices and appears only twice in Table 2. Interestingly, the optimal relaxation parameter for SOR and BSOR always happens to be equal to or larger than 1.0. For IAD, the optimal relaxation parameter turns out to be 0.9 for a few test matrices, otherwise it is larger. In most of the experiments, 1.0 is the optimal

TABLE 2 (CONTINUED)
Fastest solvers of the 13 *test matrices.*

Matrix	#	Solver	ω	P.T.	T.	#it	Bk.Error	Blocks/$nzlu$
telecom	1	BSOR, $(0.15e+0)$	1.9	1.0	2.9	1	$0.2e-11$	$0: 563(2-3,257)$
	2	IAD, $(0.10e-1)$	1.0	0.7	3.4	1	$0.2e-15$	$0: 1,981(1-31)$
	3	IAD, $(0.15e+0)$	1.7	1.0	3.4	1	$0.2e-11$	$0: 563(2-3,257)$
	4	BCGStab, (10^{-5})	—	2.3	2.5	3	$0.2e-11$	318, 749
	5	CGS, (10^{-5})	—	2.3	2.5	3	$0.2e-11$	318, 749
qn	1	CGS, (ILU0)	—	2.0	105.2	38	$0.7e-10$	593, 115
	2	IAD, *equal*	1.2	0.0	123.7	50	$0.2e-09$	$0: 324(296-323)$
	3	BSOR, $(0.10e+0)$	1.1	4.2	120.6	40	$0.9e-10$	$0: 91,800(1-450)$
	4	BCGStab, (ILU0)	—	2.0	125.3	45^s	$0.2e-09$	593, 115
	5	IAD, *other*	1.2	0.0	130.6	54	$0.1e-09$	$0: 457(1-456)$
leaky	1	BSOR, $(0.10e-14)$	1.0	4.5	12.4	1	$0.5e-16$	$0: 2(4-8,254)$
	2	IAD, $(0.10e-14)$	1.0	4.5	12.4	1	$0.7e-16$	$0: 2(4-8,254)$
	3	BSOR, $(0.64e+0)$	1.0	2.8	19.1	1	$0.2e-13$	$0: 64(4-8,005)$
	4	IAD, $(0.64e+0)$	1.0	2.8	19.4	1	$0.2e-13$	$0: 64(4-8,005)$
	5	BSOR, $(0.19e-101)$	1.0	8.5	20.9	1	$0.2e-15$	$0: 2(1-8,257)$
mutex	1	CGS, (ILU0)	—	2.5	9.8	5	$0.6e-10$	563, 491
	2	BCGStab, (ILU0)	—	2.5	9.9	5	$0.4e-10$	563, 491
	3	GMRES, (ILU0)	—	2.5	11.8	10	$0.3e-11$	563, 491
	4	SOR	1.1	—	15.3	18	$0.2e-11$	—
	5	BSOR, *equal*	1.1	0.0	16.2	13	$0.4e-11$	$0: 198(197-394)$
*mutex_alt*1	1	BSOR, *equal*	1.0	0.0	4.6	1	$0.8e-13$	$0: 198(197-394)$
	2	BCGStab, (ILU0)	—	2.6	5.4	3^s	$0.2e-10$	563, 491
	3	SOR	1.0	—	8.7	10	$0.8e-15$	—
	4	CGS, (ILU0)	—	2.6	6.3	3	$0.1e-11$	563, 491
	5	GMRES, (ILU0)	—	2.6	7.2	6	$0.7e-13$	563, 491
*mutex_alt*2	1	BSOR, *equal*	1.0	0.0	4.6	1	$0.1e-17$	$0: 198(197-394)$
	2	BCGStab, (ILU0)	—	2.6	5.4	3^s	$0.2e-14$	563, 491
	3	CGS, (ILU0)	—	2.6	6.3	3	$0.4e-15$	563, 491
	4	GMRES, (ILU0)	—	2.6	7.1	$6^\dagger$	$0.6e-16$	563, 491
	5	BSOR, *other*	1.0	0.0	11.6	$10^\dagger$	$0.4e-16$	$0: 280(1-279)$

choice.

Among the Krylov subspace methods of interest, it is clear that BCGStab performs the best. It converges for 12 test matrices with at least one preconditioner and appears 12 times among 28 Krylov subspace solvers in Table 2. Its total solution time is always the shortest or close to that of an outperforming Krylov subspace solver. CGS comes a close second appearing 10 times in Table 2, and GMRES third, the latter being more costly in terms of memory requirements and number of flops per iteration. QMR is also competitive in some cases; however, it almost always terminates due to the difference between the last two residuals. There are cases in which DQGMRES takes a smaller number of iterations than the corresponding GMRES solver, but even in those cases its solution time is almost always longer. BCG performs very poorly and converges only for a few test matrices.

We should point out that the ILU0 preconditioner leads to better total solution time than all the other ILU preconditioners for five test matrices (*easy*, *qn*, *mutex*, *mutex_alt*1, *mutex_alt*2). As for the threshold preconditioners, ILUTH(10^{-3}) is the best preconditioner for two test matrices (*medium*, *hard*), whereas ILUTH(10^{-5}) is the best preconditioner for five test matrices ($2D$, *ncd*, *ncd_alt*1, *ncd_alt*2, *telecom*). There are cases which show that a denser preconditioner is not always the better preconditioner. The problem with ILUK is the long time overhead to form the preconditioner. The ILUTH(10^{-3}) and ILUTH(10^{-5}) preconditioners are superior to ILU0 for test matrices that are of medium order (around 20,000 states), have narrow bandwidth (such as $2D$), or are relatively more difficult to solve (such as *ncd_alt*2, *ncd*, *ncd_alt*1, *telecom*, *hard*, *medium*).

We considered BSOR with *equal* partitioning as a preconditioner for BCGStab using $w = 1.0$ as the relaxation parameter. In doing this, we transformed the block lower triangular part of the coefficient matrix A from sparse point format to sparse

TABLE 3
Solvers with TPABLO ordering, $\alpha = \beta = 0.5, minbs = 10, maxbs = 200$.

Matrix	Solver	ω	P.T.	T.	#it	Bk.Error	Blocks
easy	IAD, $\gamma = 0.10e-3$	1.0	6.3	10.3	8	$0.3e-12$	$0:\ 103(\ 8-200)$
medium	IAD, $\gamma = 0.10e-2$	1.2	6.5	7.6	2	$0.6e-10$	$0:\ 103(\ 8-200)$
$2D$	IAD, $\gamma = 0.10e+0$	1.1	9.2	5.9	17	$0.9e-10$	$0:\ 129(129-129)$
ncd	IAD, $\gamma = 0.10e+0$	1.0	6.6	16.5	5	$0.7e-13$	$0:1,045(\ 10-\ 52)$
telecom	IAD, $\gamma = 0.10e-1$	1.0	5.2	28.6	47	$0.3e-12$	$0:\ 196(\ 6-200)$

block format and factorized its diagonal blocks. The resulting sparse block matrix was then used in the preconditioner solves of BCGStab. Results of 13 test cases which we have not included show that the particular BSOR preconditioned BCGStab solver is inferior to ILU0 preconditioned BCGStab. Furthermore, the BSOR preconditioner computation time is larger than that of ILU0 with the exception of the *mutex* test matrices. An intuitive explanation is the following. ILU0 has no indirect addressing other than that involved to access A in sparse format. It is a highly sequential algorithm introducing no fill-in and therefore is capable of employing a static data structure to store the factors. The test cases that we consider have relatively small bandwidths, and factorizing the diagonal blocks in BSOR comprises a lot of the work done in computing ILU0, and maybe more due to fill-in. Furthermore, BSOR has the transformation overhead and extra indirection in the factorization of the diagonal blocks due to the sparse block format. Finally, the ILU0 preconditioner attempts at an "approximate" factorization of the complete A using its original sparsity pattern, whereas BSOR uses the LU factors of the diagonal blocks only.

We executed the TPABLO algorithm on five of the test matrices and recorded the computed partitionings for $\alpha = \beta = 0.5$, $minbs = 10$, $maxbs = 200$. Then we solved for the stationary distribution using both BSOR and IAD with the recorded block structure and the optimal threshold value γ of TPABLO (see section 2.1), which we picked from $\{0.10e+0, 0.10e-1, 0.10e-2, 0.10e-3\}$. When choosing the test matrices, we tried to form a representative set of problems with different degrees of difficulty and sparsity patterns. Unfortunately, it was not possible to use TPABLO with the *qn* and *mutex* test matrices. The winners of these experiments are given in Table 3.

Note that TPABLO gives mostly balanced partitionings for the chosen parameters, and it turns out to be the case that, when input to the two-level solvers of MARCA, all diagonal blocks in these partitionings are solved directly. Also, in the partitionings TPABLO computes, it can come up with blocks of order less than *minbs* (or larger than *maxbs*, something observed in our experiments). None of the TPABLO solvers considered provide a winner when compared with the results in Table 2 though IAD with TPABLO $\gamma = 0.10e+0$ is competitive with *ncdtest* $\gamma = 0.10e-3$ for the *ncd* test matrix. The partitioning provided by TPABLO for the $2D$ matrix is an *equal* partitioning, however, with a different ordering of the states. The solution time for $2D$ is better than its IAD with *equal* counterpart if we exclude the partitioning time. However, in both the *ncd* and $2D$ test matrices, there is a faster IAD solver with a *newncd* partitioning. Our conclusion regarding TPABLO is that it may give faster converging orderings, but with a set of five parameters, it is quite difficult to fine-tune.

4. Conclusion. Results of experiments on a test suite of 13 Markov chains show that the particular two-level iterative solvers BSOR and IAD are in general very com-

petitive with ILU preconditioned Krylov subspace solvers BCGStab, CGS, and GMRES. For these two-level iterative solvers, there are cases in which a straightforward partitioning of the coefficient matrix such as *equal* or *other* gives a faster solution than can be obtained using the *ncdtest* or *newncd* partitioning algorithms. However, in between *newncd* and *ncdtest*, the former gives faster converging iterations than the latter in a larger number of the test cases, and therefore should be more favorable.

It is clear that the ILU preconditioned Krylov subspace solvers under consideration are affected adversely with higher ill-conditioning (*leaky*, *ncd_alt*2, *ncd*, *ncd_alt*1) unless the matrix is very narrow banded (*telecom*). When the Markov chain is extremely ill-conditioned (*leaky*), incomplete LU factorization may even fail. For NCD matrices, we recommend IAD and BSOR with *newncd* partitioning and relaxation parameter 1.0. When using *newncd*, one should opt as much as possible for a balanced partitioning in terms of the number and order of blocks so that all blocks are solved directly. However, higher ill-conditioning does not always imply poorer performance. It is noticed in some cases that it may even help a solver, especially IAD (compare results of *ncd_alt*2 with those of *ncd_alt*1 and *ncd*), to converge faster. If a Krylov subspace solver needs to be used with an NCD matrix, the choice of the preconditioner should be ILUTH with a small threshold value.

For very narrow banded and well behaving matrices ($2D$), the ILUTH preconditioner is cheap to compute and very strong which makes Krylov subspace methods the solvers of choice. Furthermore, we see that ILU0 preconditioned BCGStab and CGS are very effective Krylov subspace solvers if the coefficient matrix is relatively large but highly sparse (*qn*) or wide-banded (*mutex*). For such matrices, we can also recommend BSOR *equal* and SOR with relaxation parameter 1.0 or slightly larger. One final remark is that a BSOR *equal* preconditioned BCGStab does not improve the situation over an ILU0 preconditioned BCGStab.

Acknowledgments. We thank Wail Gueaieb for providing the framework to carry out the experiments with the Krylov subspace solvers, Yousef Saad for his comments on this work, and Daniel Szyld for supplying the TPABLO routines and his comments on an earlier version. We also thank the anonymous referee for the constructive report which led to an improved manuscript.

REFERENCES

[1] Ö. Aras and T. Dayar, *Complete buffer sharing with pushout thresholds in ATM networks under bursty arrivals*, in the Proceedings of the First Symposium on Computer Networks, İstanbul, Turkey, 1996, pp. 144–156.

[2] O. Axelsson, *Iterative Solution Methods*, Cambridge University Press, Cambridge, UK, 1994.

[3] V. A. Barker, *Numerical solution of sparse singular systems of equations arising from ergodic Markov chains*, Comm. Statist. Stochastic Models, 5 (1989), pp. 355–381.

[4] R. Barrett, M. Berry, T. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. van der Vorst, *Templates for the Solution of Linear Systems*, SIAM, Philadelphia, PA, 1994.

[5] M. Benzi and M. Tuma, *A comparision of some preconditioning techniques for general sparse matrices*, in Iterative Methods in Linear Algebra, II, IMACS Series in Computational and Applied Mathematics, Vol. 3, S. Margenov and P. Vassilevski, eds., IMACS, New Brunswick, NJ, 1996, pp. 191–203.

[6] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, SIAM, Philadelphia, PA, 1994.

[7] J. J. Buoni, *Incomplete factorization of singular M-matrices*, SIAM J. Alg. Discrete Methods, 7 (1986), pp. 193–198.

[8] H. Choi and D. B. Szyld, *Application of threshold partitioning of sparse matrices to Markov*

chains, in the Proceedings of the IEEE International Computer Performance and Dependability Symposium IPDS'96, Urbana, IL, 1996, pp. 158–165.

[9] T. Dayar, *Permuting Markov Chains to Nearly Completely Decomposable Form*, Tech. Report BU-CEIS-9808, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, August 1998; available via ftp from ftp://ftp.cs.bilkent.edu.tr/pub/tech-reports/1998/BU-CEIS-9808.ps.z.

[10] T. Dayar and W. J. Stewart, *Comparison of Partitioning Techniques for Two-Level Iterative Solvers on Large, Sparse Markov Chains*, Tech. Report BU-CEIS-9805, Department of Computer Engineering and Information Science, Bilkent University, Ankara, Turkey, April 1998; Available via ftp from ftp://ftp.cs.bilkent.edu.tr/pub/tech-reports/1998/BU-CEIS-9805.ps.z.

[11] T. Dayar and W. J. Stewart, *On the effects of using the Grassmann-Taksar-Heyman method in iterative aggregation-disaggregation*, SIAM J. Sci. Comput., 17 (1996), pp. 287–303.

[12] R. W. Freund and M. Hochbruck, *On the use of two QMR algorithms for solving singular systems and applications in Markov chain modelling*, Numer. Linear Algebra Appl., 1 (1994), pp. 403–420.

[13] A. Greenbaum, *Iterative Methods for Solving Linear Systems*, SIAM, Philadelphia, PA, 1997.

[14] D. Gross, B. Gu, and R. M. Soland, *The biconjugate gradient method for obtaining the steady-state probability distributions of Markovian multiechelon repairable item inventory systems*, in The Numerical Solution of Markov Chains, W. J. Stewart, ed., M. Dekker, New York, 1991, pp. 473–489.

[15] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, SIAM, Philadelphia, PA, 1996.

[16] S. T. Leutenegger and G. H. Horton, *On the utility of the multi-level algorithm for the solution of nearly completely decomposable Markov chains*, in Computations with Markov Chains: Proceedings of the Second International Workshop on the Numerical Solution of Markov Chains, W. J. Stewart, ed., Kluwer, Boston, MA, 1995, pp. 425–442.

[17] I. Marek and D. B. Szyld, *Iterative and semi-iterative methods for computing stationary probability vectors of Markov operators*, Math. Comp., 61 (1993), pp. 719–731.

[18] I. Marek and D. B. Szyld, *Local convergence of the (exact and inexact) iterative aggregation method for linear systems and Markov operators*, Numer. Math., 69 (1994), pp. 61–82.

[19] I. Marek and P. Mayer, *Convergence analysis of an iterative aggregation/disaggregation method for computing stationary probability vectors of stochastic matrices*, Numer. Linear Algebra Appl., 5 (1998), pp. 253–274.

[20] Matrix Market, *A Repository of Test Matrices at the National Institute of Standards and Technology*, http://math.nist.gov/MatrixMarket.

[21] J. A. Meijerink and H. A. van der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Math. Comp., 31 (1977), pp. 148–162.

[22] C. D. Meyer, *The role of the group generalized inverse in the theory of finite Markov chains*, SIAM Rev., 17 (1975), pp. 443–464.

[23] C. D. Meyer, *Stochastic complementation, uncoupling Markov chains, and the theory of nearly reducible systems*, SIAM Rev., 31 (1989), pp. 240–272.

[24] 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.

[25] J. O'Neil and D. B. Szyld, *A block ordering method for sparse matrices*, SIAM J. Sci. Statist. Comput., 11 (1990), pp. 811–823.

[26] B. Philippe, Y. Saad, and W. J. Stewart, *Numerical methods in Markov chain modelling*, Oper. Res., 40 (1992), pp. 1156–1179.

[27] Y. Saad, *Projection methods for the numerical solution of Markov chain models*, in The Numerical Solution of Markov Chains, W. J. Stewart, ed., M. Dekker, New York, 1991, pp. 455–471.

[28] Y. Saad, *Preconditioned Krylov subspace methods for the numerical solution of Markov chains*, in Computations with Markov Chains: Proceedings of the Second International Workshop on the Numerical Solution of Markov Chains, W. J. Stewart, ed., Kluwer, Boston, MA, 1995, pp. 49–64.

[29] Y. Saad, *Iterative Solution of Sparse Linear Systems*, PWS Publishing, New York, 1996.

[30] G. W. Stewart, W. J. Stewart, and D. F. McAllister, *A two-stage iteration for solving nearly completely decomposable Markov chains*, in Recent Advances in Iterative Methods, IMA Vol. Math. Appl. 60, G. H. Golub, A. Greenbaum, and M. Luskin, eds., Springer-Verlag, New York, 1994, pp. 201–216.

[31] W. J. Stewart, *MARCA: Markov Chain Analyzer. A software package for Markov modelling*, in The Numerical Solution of Markov Chains, W. J. Stewart, ed., M. Dekker, New York,

1991, pp. 37–62.
[32] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, Princeton, NJ, 1994.
[33] W. J. Stewart, *Marca Models: A Database of Markov Chain Models*, http://www.csc.ncsu.edu/faculty/WStewart/MARCA_Models/MARCA_Models.html.
[34] W. J. Stewart and W. Wu, *Numerical experiments with iteration and aggregation for Markov chains*, ORSA J. Comput., 4 (1992), pp. 336–350.