NONSYMMETRIC SKYLINE VERSUS COMPRESSED SPARSE ROW FORMAT IN DIRECT METHODS FOR MARKOV CHAINS

TUĞRUL DAYAR[1]

Abstract. This paper investigates the implementation of Gaussian Elimination (GE) and Grassmann–Taksar–Heyman (GTH) algorithms for Markov chains in nonsymmetric skyline (NSK) and compressed sparse row (CSR) formats. Numerical experiments are carried out using three real life applications. Implementations that allocate space for both lower– and upper–triangular factors result in faster GTH solvers. Specifically, the CSR implementation of the GTH algorithm for the nontransposed linear system of equations gives the smallest run–time among different GTH implementations considered. For GE, the experiments suggest the CSR implementation for the transposed system of equations as the appropriate choice.

Key words. Markov chains, sparsity schemes, skyline storage, decomposability, Gaussian elimination, Grassman–Taksar–Heyman method, stationary probability

1 Introduction

Markovian modeling and analysis is extensively used in many disciplines in evaluating the performance of existing systems and in analyzing and designing systems to be developed. The stationary behavior of Markovian systems is uncovered by solving

$$\boldsymbol{\pi}\mathbf{P} = \boldsymbol{\pi}, \ \|\boldsymbol{\pi}\|_1 = 1, \tag{1.1}$$

where $\mathbf{P}$ is the one-step stochastic transition probability matrix (*viz.* discrete–time Markov chain — DTMC) and $\boldsymbol{\pi}$ is the unknown stationary probability distribution vector of the system under consideration. By definition, rows of $\mathbf{P}$ and elements of $\boldsymbol{\pi}$ sum up to 1.

In what follows, boldface capital letters denote matrices, boldface lowercase letters denote column vectors, italic lowercase and uppercase letters denote scalars. The typewriter style font is used for arrays. $\mathbf{o}$ represents a row or column vector of all zeros depending on the context. The convention of representing probability distributions by row vectors is adopted.

Equation (1.1) may also be viewed as a homogeneous linear system with singular coefficient matrix $\mathbf{A} = \mathbf{I} - \mathbf{P}^T$ and unknown vector $\boldsymbol{x} = \boldsymbol{\pi}^T$ as in

$$\mathbf{A}\boldsymbol{x} = \mathbf{o}, \ \|\boldsymbol{x}\|_1 = 1. \tag{1.2}$$

$\mathbf{A}$ is a singular M–matrix (see [2]) with 0 column sums, and the unique null vector of unit 1–norm is sought. Solving (1.2) is crucial in computing performance measures for Markovian systems. For queueing systems, these measures may be the average number of customers, the mean waiting time, or the blocking probability for a specific queue. In communication systems, they may be the total packet loss rate, the probability of an empty system, or any other relevant measure.

Of special interest to us are nearly completely decomposable (NCD) Markov chains [4], [11], [19], irreducible stochastic matrices that can be ordered so that the matrix of transition probabilities has a block structure in which the nonzero elements of the off-diagonal blocks are small compared with those of the diagonal blocks. Such matrices often arise in queueing network analysis, large scale economic modeling, and computer systems performance evaluation. If $\mathbf{P}$ is the

[1]Department of Computer Engineering and Information Science, Bilkent University, 06533 Bilkent, Ankara, Turkey (tugrul@bilkent.edu.tr).

stochastic transition probability matrix of a NCD Markov chain, then it may be written in the form $\mathbf{P} = \mathrm{diag}(\mathbf{P}_{1,1}, \mathbf{P}_{2,2}, \ldots, \mathbf{P}_{N,N}) + \mathbf{E}$, where $\mathbf{P}_{i,i}$'s are the square diagonal blocks of $\mathbf{P}$ and N is the number of NCD blocks. The quantity $\|\mathbf{E}\|_\infty$ is referred to as the degree of coupling and it is taken to be a measure of the decomposability of the matrix (see [8]). If it were zero, then $\mathbf{P}$ would be reducible. Small perturbations in the transition probabilities of NCD chains may lead to considerable changes in the stationary probabilities; hence, these chains are known to be ill–conditioned. NCD Markov chains that appear in applications are quite large and sparse, possibly having more than tens of thousands of states. For such large chains, direct methods can generate immense fill–in during the triangularization phase of the solution process, and due to storage limitations and very long execution times they become impractical. Extensive numerical experiments suggest iterative aggregation–disaggregation (IAD) as the method of choice for large NCD Markov chains (see [10], [3], [14], [18], [16], [19], [5]).

In [5], a modified version of Gaussian elimination (GE) has been used to enforce stability and to improve accuracy in the aggregation and disaggregation phases of the IAD algorithm. The only, yet significant, drawback of the so–called Grassman–Taksar–Heyman (GTH) method [7], [9], [15], [12], [5] seems to be the extra time it takes to execute in a compressed sparse row (CSR) format [6], [13], [1], [19] (also known as the Harwell–Boeing format). As indicated in [5], p. 294, the difficulty with the CSR format lies in the fact that unlike GE it is not possible to implement the GTH method for the transposed form of the equations in (1.2) using delayed row updates. The aim of this paper is to introduce the nonsymmetric skyline (NSK) sparse storage format [6], [13], [1] to the Markov chain domain and to show that the implementation of the GTH method in the NSK format mitigates the problems associated with the CSR format by performing the reduction phase 'in place'. The NSK format is suitable for the GTH method because provision for fill–in is made at the outset with no need for expanding and compacting rows that are updated during the reduction phase.

The paper is organized as follows. The next section provides a background on the CSR and NSK formats. The third section gives the GE and GTH algorithms for the NSK format and discusses related implementation issues. The real life applications investigated appear in section four with the results of the numerical experiments in section five. The last section summarizes the results.

2 CSR and NSK storage formats for Markov chains

The choice of the CSR format for Markovian solvers (see the MARkov Chain Analyzer, MARCA, in [17], for instance) is motivated by its simplicity, generality, and widespread use. The CSR format makes no assumptions about the nonzero structure of the matrix under consideration and it stores only the requisite elements. Space for the diagonal elements of $\mathbf{A}$ in equation (1.2) is always spared since they are all nonzero in exact arithmetic. If n is the number of states in $\mathbf{A}$ (*viz.* order of $\mathbf{A}$) and nz the number of nonzero elements, then only nz floating–point and $nz + n + 1$ integer storage locations are used with the CSR storage format. For sparse matrices, this implies considerable savings over n^2 floating–point storage locations of a two–dimensional array (see [1], pp. 58–59).

In the CSR format, nonzero elements of the coefficient matrix are stored in a floating–point array of length nz row by row, elements of row i preceeding those of row $i + 1$. The column index of each nonzero element is stored in the same location as its floating–point value, this time in an integer array of length nz. Finally, in order to have access to rows, the starting location of each row of $\mathbf{A}$ in the floating–point array is stored in an integer array of length n. To facilitate the computation of the number of nonzero elements in the last row, an extra element whose value is equal to $nz + 1$ is appended to this integer array effectively making its length $n + 1$. Details of the

CSR format for Markov chains may be found in [19], pp. 151–155.

On the other hand, the NSK sparse storage format is a special type of variable–band storage format composed of two substructures (see [13], pp. 7–8). In this format, the lower–triangular part of a nonsymmetric coefficient matrix is stored in row–oriented skyline (*viz.* profile, envelope) format (see [6], pp. 149–150, 204–205) and the upper–triangular part is stored in column–oriented skyline format (i.e., its transpose stored in row–oriented skyline format). The lower– and upper–triangular parts may be linked together through the diagonal elements as suggested in [13], p. 8, though there are a multitude of ways by which this can be accomplished.

The basic row–oriented skyline format is a straightforward approach to storing the triangular parts of $\mathbf{A}$. Each row of the lower–triangular part and the corresponding column of the upper-triangular part are stored contiguously in a floating–point array `a`. When storing row i of the lower–triangular part, one starts with the first (i.e., lowest column–indexed) nonzero element in the row and stores all elements up to and including the diagonal element $a_{i,i}$. Right after row i, column i of the upper–triangular part is stored starting from the first (i.e., lowest row–indexed) nonzero element in the column up to, but excluding, $a_{i,i}$. Consequently, the floating–point array `a` contains row 1 of the lower–triangular part followed by column 1 of the upper-triangular part, followed by row 2 of the lower–triangular part followed by column 2 of the upper-triangular part, etc. Note that column 1 of the upper-triangular part is necessarily empty and is not stored. All floating–point coefficients stored, whether they are zero or nonzero, form the profile of $\mathbf{A}$. Any zeros within the variable band are stored explicitly because this part of the matrix usually fills in totally during the triangularization phase. In fact, if each row in the lower–triangular part and each column in the upper–triangular part have a nonzero ahead of the diagonal, all elements in between the nonzero elements and the diagonal fill in. However, unlike the CSR format, column indices of entries in the profile need not be stored. As we shall see, the column indices may be derived easily by an indirect addressing step. Nevertheless, an integer array `ia` of length $2n+1$ needs to be allocated. The first $n+1$ elements of `ia` are pointers to the starting locations of rows in `a` in the same vein with the CSR format. The remaining n elements contain the locations of the diagonal elements of $\mathbf{A}$ in `a`.

The following example shows the contents of the floating–point array `a` and the integer array `ia` of the NSK sparse storage format for a 5×5 NCD Markov chain.

Example 2.1 Let

$$\mathbf{A} = \mathbf{I} - \mathbf{P}^T = \left(\begin{array}{ccc|cc} 0.8 & -0.4 & 0 & 0 & -0.05 \\ -0.75 & 0.7 & -0.55 & 0 & 0 \\ 0 & -0.3 & 0.6 & 0 & 0 \\ \hline -0.05 & 0 & -0.05 & 0.5 & 0 \\ 0 & 0 & 0 & -0.5 & 0.05 \end{array}\right).$$

Here $\mathbf{P}$ is a NCD Markov chain with 2 NCD blocks and a degree of coupling 0.05; the profile of $\mathbf{A}$ for the NSK format is given by

$$\text{Profile}(\mathbf{A}) = \begin{pmatrix} \text{X} & \text{X} & & & \text{X} \\ \text{X} & \text{X} & \text{X} & & \text{X} \\ & \text{X} & \text{X} & & \text{X} \\ \text{X} & \text{X} & \text{X} & \text{X} & \text{X} \\ & & & \text{X} & \text{X} \end{pmatrix},$$

where an X represents an entry for which storage is allocated in the nonsymmetric skyline representation. For this example, the contents of the real array `a` and the integer array `ia` are given by

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
a	$a_{1,1}$	$a_{2,1}$	$a_{2,2}$	$a_{1,2}$	$a_{3,2}$	$a_{3,3}$	$a_{2,3}$	$a_{4,1}$	$a_{4,2}$	$a_{4,3}$	$a_{4,4}$	$a_{5,4}$	$a_{5,5}$	$a_{1,5}$	$a_{2,5}$	$a_{3,5}$	$a_{4,5}$

	1	2	3	4	5	6	7	8	9	10	11
ia	1	2	5	8	12	18	1	3	6	11	13

The first n (=5) elements of `ia` are pointers to the starting locations of rows of $\mathbf{A}$ in `a`, and `ia`$(n+1)-1$ (=17) gives the amount of floating–point storage allocated in `a`. The last n elements of `ia` provide pointers to the diagonal elements of $\mathbf{A}$ in `a`. Observe that explicit zeros need to be stored in `a(15)`, `a(16)`, and `a(17)` since the first nonzero element in column 5 of $\mathbf{A}$ appears in row 1 and $a_{2,5}, a_{3,5}, a_{4,5}$ are all zero. Analogously, a zero is stored in `a(9)` because the first nonzero element in row 4 is to the left of $a_{4,2}$. Hence, an integer array of length $2n+1(=11)$ and a floating–point array of length `ia`$(n+1)-1$ are required. Here `ia`$(n+1)-1$ is greater than or equal to nz (=14) with equality rarely being achieved.

The column index of the first nonzero element in row i of $\mathbf{A}$ is given by

$$i - (\texttt{ia}(n+1+i) - \texttt{ia}(i)), \tag{2.1}$$

whereas the row index of the first nonzero element in column j may be computed as

$$j - (\texttt{ia}(j+1) - 1 - \texttt{ia}(n+1+j)). \tag{2.2}$$

In the latter, `ia`$(j+1)-1$ is the location in `a` of the last element above the diagonal in column j. If all elements above the diagonal in column j are zero, then $j - (\texttt{ia}(j+1) - 1 - \texttt{ia}(n+1+j)) = j$ as expected. For instance, in Example 2.1, the column index of the first nonzero element in row 3 is 3-(6-5)=2 and the row index of the first nonzero element in column 4 is 4-(12-1-11)=4.

The next section provides a brief overview of the GE and GTH algorithms for Markov chains in the CSR format. It also discusses the motivation behind choosing the NSK sparse storage format and presents the GE and the GTH algorithms for the NSK format comparing and contrasting with the CSR format along the way.

3 GE and GTH algorithms for the NSK format

The CSR implementations of GE and the GTH method are available for both the linear system in equation (1.2) and its nontransposed version (see [5]). From this point on, we refer to the form of the system in (1.2) 'transposed' and the system

$$\boldsymbol{\pi}\mathbf{B} = \mathbf{o}, \ \|\boldsymbol{\pi}\|_1 = 1, \tag{3.1}$$

where $\mathbf{B} = \mathbf{I} - \mathbf{P}$, 'nontransposed'. Equation (1.2) represents a homogeneous linear system with singular coefficient matrix $\mathbf{A}$ that may be written as the product of a nonsingular lower–triangular matrix and a singular upper–triangular matrix. The CSR implementations of GE for equations (1.2) and (3.1) in MARCA perform row reductions with delayed updates. Since the last row of the upper-triangular factor is necessarily zero, the reduction phase of GE needs to be carried up to row $n-1$; there is no need for row n to be eliminated. With the transposed system, only the upper–triangular factor needs to be kept.

The notation in Table 1 is used in Algorithms 1, 2, and 3 to enhance the readability of the pseudo–code. The portion on the left of the first two algorithms is at a higher level of abstraction.

TABLE 1
Notation for Algorithms 1, 2, and 3

$\mathbf{A}_{i_1:i_2,j_1:j_2}$	Submatrix of $\mathbf{A}$ confined in between rows i_1, i_2 and columns j_1, j_2, where $1 \le i_1 \le i_2 \le n$ and $1 \le j_1 \le j_2 \le n$. Whenever $i_1 = i_2$ ($j_1 = j_2$), we drop i_2 (j_2).
k	Step number in the reduction phase; for convenience it starts from 2.
k_{row_first}	Column index of the first nonzero element in row k.
j_{col_first}	Row index of the first nonzero element in column j.
m_{first}	Maximum of k_{row_first} and j_{col_first}.
k_{col_first}	Row index of the first nonzero element in column k.
i_{row_first}	Column index of the first nonzero element in row i.
n_{first}	Maximum of k_{col_first} and i_{row_first}.

ALGORITHM 1
Row Doolittle decomposition in the NSK format for the system in (1.2)

For $k := 2$ to n do

- If $k < n$, update $\mathbf{A}_{k,1:k-1}$
  - If $k < n$,
    - $k_{row\_first} := k - (\mathtt{ia}(n+1+k) - \mathtt{ia}(k))$
    - For $j := 1$ to $k-1$ do
      - $j_{col\_first} := j - (\mathtt{ia}(j+1) - 1 - \mathtt{ia}(n+1+j))$
      - $m_{first} := \max(k_{row\_first}, j_{col\_first})$
      - $a_{k,j} := a_{k,j} - \sum_{l=m_{first}}^{j-1} a_{k,l} a_{l,j}$
      - $a_{k,j} := a_{k,j} / a_{j,j}$
- Update $\mathbf{A}_{1:k,k}$ (except $a_{n,n}$)
  - $k_{col\_first} := k - (\mathtt{ia}(k+1) - 1 - \mathtt{ia}(n+1+k))$
  - For $i := k_{col\_first}$ to k do
    - If ($k < n$ or $i < k$),
      - $i_{row\_first} := i - (\mathtt{ia}(n+1+i) - \mathtt{ia}(i))$
      - $n_{first} := \max(k_{col\_first}, i_{row\_first})$
      - $a_{i,k} := a_{i,k} - \sum_{l=n_{first}}^{i-1} a_{i,l} a_{l,k}$

The code fragment to the right of a curly left bracket is the detailed (i.e., blown–up) version of the abstract statement pointed to by the tip of the curly left bracket. We should remark that $\mathbf{A}$ is supplied to Algorithms 1 and 2 in the NSK format and is overwritten during the triangularization process. The indirect addressing step (see Equations (2.1) and (2.2)) employed in accessing the elements of $\mathbf{A}$ is not shown in order not to complicate the code further. The back solve corresponding to Algorithms 1 and 2 is given by Algorithm 3.

The GE implementation in the NSK format is provided solely for comparison purposes. As described in [5], p. 293, the CSR format GE for the nontransposed system has no discernible advantage over the transposed implementation; therefore the nontransposed implementation is not included in the numerical experiments in §5. Besides, rather than obtaining the upper–triangular factor through delayed row updates, this time we use row Doolittle decomposition (see [6], p. 204) that seems to be better suited to the NSK format (see Algorithm 1). With this form of the

implementation, we update $\mathbf{A}_{k,1:k-1}$ and $\mathbf{A}_{1:k,k}$ at step k. Since, the upper-triangular factor is singular, the last row in $\mathbf{A}$ need not be updated. The two if–statements in Algorithm 1 are used to this effect. Moreover, both of the inner products may be programmed so that they commence by multiplying two nonzero entries. See how m_{first} and n_{first} are computed in Algorithm 1. Yet, another advantage of the row Doolittle variant is that the first inner product is vectorizable [6], p. 202.

If the first inner product in Algorithm 1 is not vectorized, we would expect Algorithm 1 to take slightly more time than the CSR format GE with delayed row updates for the tranposed system. There are two reasons for this. First, even though the inner products are guaranteed to commence with the multiplication of two nonzero elements, in the early steps of the Doolittle decomposition (i.e., for small k) some of the other multiplications in the inner products may involve zero operands. Second, the extra indirect addressing step used to access the column elements in the inner products (i.e., $a_{l,j}$ and $a_{l,k}$, respectively) is likely to slow down the algorithm further. Since rows are not expanded and compacted, we do not employ a temporary floating–point work array (of length n) in Algorithm 1. We believe row Doolittle decomposition in the NSK format should still result in a competitive solver.

The GTH method is essentially GE with pivot corrections. For the transposed system, the pivot element at step k is computed by taking the negated sum of the column elements below the pivot. This stems from the fact that, at step k the unreduced square matrix $\mathbf{A}_{k:n,k:n}$ is also an M–matrix with 0 column sums just like the original coefficient matrix. As indicated in [5], p. 294, the GTH method achieves high relative accuracy but poses a significant problem for the CSR format. With the tranposed system, $\mathbf{A}_{k:n,k}$ must have been updated at the beginning of step k. Hence, delayed row updates cannot be used. The solution suggested in [5] that avoids storing the lower–triangular factor introduces substantial overhead by expanding, updating, and compacting rows in temporary floating–point storage. On the other hand, one needs to store both lower– and upper–triangular factors if delayed row updates are used in the GTH reduction of the nontranposed system. Drawing from this line of study, we conclude that the execution time of the GTH method in the CSR format is not likely to improve over the implementation provided for the nontransposed system in MARCA.

The rationale behind choosing the NSK format as a suitable sparse representation for the GTH method is that it does not require any assumptions regarding the nonzero structure of the coefficient matrix other than it be sparse. Most importantly, contrary to banded storage formats, the NSK format does not impose an upper limit on the maximum bandwidth of the coefficient matrix. Thus, space to store the nonzero multipliers and the upper–triangular factor is spared at the outset extenuating the problems associated with the GTH algorithm for both transposed and nontransposed systems.

Algorithm 2 gives the GTH reduction in the NSK format for the tranposed system and employs the additional notation in Table 2.

Step k of Algorithm 2 starts by storing the pivot row $\mathbf{A}_{k-1,k:n}$ in $\mathtt{y}$, a temporary floating–point array (of maximum length $n-1$). Then the lower–triangular part (including the diagonal elements) and the strictly upper-triangular part of the square submatrix $\mathbf{A}_{k:n,k:n}$ are updated. Accessing the elements of the pivot row involves extra indirect addressing (see Equations (2.1) and (2.2)). For that

TABLE 2
Additional notation for Algorithm 2

$\mathtt{y}$	Temporary floating–point array used to store the pivot row $\mathbf{A}_{k-1,k:n}$ at step k.
$\mathtt{z}$	Temporary floating–point array used to store the multiplier column $\mathbf{A}_{k:n,k-1}$ at step k.
σ	Floating–point variable used to compute the corrected diagonal $a_{k,k}$ at step k.

ALGORITHM 2

Row GTH reduction in the NSK format for the system in (1.2)

- Correct $a_{1,1}$ { • $a_{1,1} = -\sum_{i \neq 1} a_{i,1}$
- For $k := 2$ to $n-1$ do
  - ◇ Store pivot row $\mathbf{A}_{k-1,k:n}$ in $\mathtt{y}$ { ◇ For $j := k$ to n do
    - ○ $\mathtt{y}(j) := a_{k-1,j}$
  - ◇ $\sigma := 0$
  - ◇ For $i := k$ to n do
    - ○ Update multiplier $a_{i,k-1}$; store $a_{i,k-1}$ in $\mathtt{z}$ { ○ $a_{i,k-1} := a_{i,k-1}/a_{k-1,k-1}$
    - ○ $\mathtt{z}(i) := a_{i,k-1}$
    - ○ Update $\mathbf{A}_{i,k:i}$ (except $a_{i,i}$) { ○ If ($i \neq k$ and $\mathtt{z}(i) \neq 0$),
      - ▷ For $j := k$ to i do
        - · $a_{i,j} := a_{i,j} - \mathtt{z}(i)\mathtt{y}(j)$
        - · If ($i > k$ and $j = k$) $\sigma := \sigma + a_{i,j}$
  - ◇ For $j := k+1$ to n do
    - ○ Update $\mathbf{A}_{k:j-1,j}$ { ○ If $\mathtt{y}(j) \neq 0$,
      - ▷ $a_{i,j} := a_{i,j} - \sum_{i=k}^{j-1} \mathtt{y}(j)\mathtt{z}(i)$
  - ◇ Correct $a_{k,k}$ { ◇ $a_{k,k} := -\sigma$

matter, it is convenient to keep the pivot row in $\mathtt{y}$ while $\mathbf{A}_{k:n,k:n}$ is updated. Since $\mathbf{A}$ is stored in the NSK format, we choose to update the lower–triangular part of $\mathbf{A}_{k:n,k:n}$ row by row and its upper–triangular part column by column. When the multiplier column $A_{k:n,k-1}$ in the lower–triangular part is updated, the multipliers are stored in $\mathtt{z}$, a temporary floating–point array (of maximum length $n-1$), so that they are readily available during the update of the upper–triangular part. The floating–point arrays $\mathtt{y}$ and $\mathtt{z}$ are used to speed up the algorithm in the NSK format by evading the overhead of the indirect addressing step. The elements of $\mathbf{A}_{k+1:n,k}$ are summed up in σ. The pivot element $a_{k,k}$ is corrected using -σ at the end of step k, and therefore excluded from the update (see the second if–statement in Algorithm 2). Observe that unlike GE, the elements in $\mathbf{A}_{n,k:n-1}$ should be updated, because they contribute to the computation of pivot elements in future steps. The last diagonal element $a_{n,n}$ may be skipped since it eventually turns out to be zero. The test $\mathtt{z}(i) \neq 0$ in the first if–statement and the test $\mathtt{y}(i) \neq 0$ in the third if–statement of Algorithm 2 check for a nonzero multiplier and for a nonzero pivot row element, respectively.

We expect Algorithm 2 to execute faster than the GTH reduction of the transposed system in the CSR format and to execute timewise comparatively with the GTH reduction of the nontransposed system in the CSR format. An intuitive explanation is the following. Since the implementation in the CSR format for the nontransposed system needs to accomodate the lower–triangular factor and the upper–triangular fill–in, it spends time with bookkeeping operations to find out if there is sufficient space. Such bookkeeping operations are nonexistent in Algorithm 2. Moreover, even though delayed row updates are used with the GTH method for the nontransposed system in the CSR format, expanding, updating, and compacting rows of both triangular factors in the format is

TABLE 3
Additional notation for Algorithm 3

p	Temporary floating–point array used to store the stationary vector.
s	Floating–point variable that holds a running sum of the stationary vector elements.

ALGORITHM 3
Back solve in the NSK format for Algorithms 1 and 2

- $\mathtt{p}(n) := \mathtt{1.0d} - \mathtt{16}$;
- $s := \mathtt{p}(n)$;
- For $j := n$ downto 2 do
  - $\diamond$ $j_{col\_first} := j - (\mathtt{ia}(j+1) - 1 - \mathtt{ia}(n+1+j))$;
  - $\diamond$ For $i := j_{col\_first}$ to $j-1$ do
    - $\circ$ $\mathtt{p}(i) := \mathtt{p}(i) - a_{i,j} * \mathtt{p}(j)$;
  - $\diamond$ $\mathtt{p}(j-1) := \mathtt{p}(j-1)/a_{j-1,j-1}$;
  - $\diamond$ $s := s + \mathtt{p}(j-1)$;
- For $j := 1$ to n do
  - $\diamond$ $\mathtt{p}(j) := \mathtt{p}(j)/s$;

---

most likely to have a detrimental effect on the execution time of the algorithm. On the other hand, the indirect addressing expressed by equations (2.1) and (2.2) seems to be the most significant drawback of the NSK format. The indirect addressing involved in the CSR format is not as complicated, frequent, and therefore not as time–consuming as that of the NSK format.

The back solve corresponding to Algorithms 1 and 2 for the NSK format is given in Algorithm 3 (see Table 3 for additional notation). The back solve in Algorithm 3 nicely fits to the NSK format. At the beginning, the last element in the temporary floating–point array (i.e., $\mathtt{p}(n)$) is set to a value on the order of machine epsilon in double precision. The running sum s is initialized to $\mathtt{p}(n)$; then the back solve starts executing. At step j, column j of the upper–triangular factor (excluding the diagonal elements) is accessed starting from the first nonzero entry (see how j_{col_first} is computed) and multiplied with $\mathtt{p}(j)$. Following this, $\mathtt{p}(j-1)$ is obtained through the division of this element by $a_{j-1,j-1}$. After all steps are over, the temporary floating-point array p is normalized to give the stationary distribution.

In the next section we consider three real life applications arising in the Markov modeling area and that are NCD.

4 Applications

This section includes a description of the applications we used in our experiments. The fact that the GTH algorithm provides a relatively more accurate solution is extensively dealt with in the literature [7], [9], [15], [12], [5]. For that matter, we have not sought instances of the applications that are extremely ill–conditioned. Rather the intent is to show that the NSK implementation of the GTH algorithm results in a competitive solver compared with the CSR implementations considered in [5] for a variety of Markov chains with different average bandwidths and nonzero structures.

The first application is chosen from the high–speed networking area [20]. The second application is the interactive computer system that comes up quite often and that is recently investigated in [5], p. 298. The third application is from the telecommunications area. Applications two and three appear in [18], pp. 444, 446, respectively, and we do not wish to consider the details of them further. However, we think the first application needs to be explained more and the rest of this section is devoted to that.

Broadband Integrated Services Digital Networks (B–ISDNs) are to support multiple types of traffic such as voice, video, and data. The Asynchronous Transfer Mode (ATM) is the transport technique of choice for B–ISDNs by the standards committees. In this mode of operation, all information is carried using fixed size packets (called 'cell's) so as to share the network among multiple classes of traffic. Since multiclass traffic will be carried on B–ISDNs, different quality of service requirements will be imposed by different applications.

One type of congestion control for ATM networks deals with discarding cells at ATM buffers in order to guarantee a prespecified cell loss rate. One bit in each ATM cell header is reserved to assign the space priority of cells. This bit indicates whether the given cell is high priority or low priority. Priority cell discarding is a buffer management scheme in which higher priority cells are favored in receiving buffer space. An efficient technique for determining the cells to be discarded when congestion occurs is the complete buffer sharing scheme with pushout thresholds [20].

The analysis and simulation results of the complete buffer sharing scheme with pushout thresholds is given in [20]. In the system under consideration, there are two classes of traffic arriving to an ATM buffer of size K. Time is divided into fixed size slots of length equal to one cell transmission time. The arrival of traffic class l (= 1,2) to the buffer is modelled as a Bernoulli process with probability of cell arrival p_l in a slot.

The states of the corresponding queueing system may be represented by the ordered pair (i, j), where i and j are the number of class 1 and class 2 cells in the buffer, respectively. Let $k(= i + j)$ denote the total number of cells in the buffer at state (i, j). Then, a natural state space ordering that places the states with the same number of total cells in the buffer (i.e., k) consecutively, gives rise to a block matrix with $\sum_{k=0}^{K}(k+1) = (K+1)(K+2)/2$ states. The first block consists of the state (0,0) (i.e., the state in which the buffer is empty), the second block has states (0,1), (1,0), the third block has states (0,2), (1,1), (2,0), and so on. The kth block has $k + 1$ states. That is, we have the following ordering:

$$(0,0) \prec (0,1) \prec (1,0) \prec (0,2) \prec (1,1) \prec (2,0) \prec (0,3) \prec (1,2) \prec (2,1) \prec (3,0) \prec \cdots \prec (K,0)$$

During a time slot, no cells, one cell, or two cells may arrive. If one or two cells arrive, this happens at the beginning of a slot. A cell departure occurs by the end of the slot if the buffer has at least one cell at the beginning of the slot. Hence, an arriving cell cannot be transmitted before the end of the next slot. With these assumptions, a cell is discarded iff two cells arrive to a full buffer. The pushout threshold for class 2 cells is given by T_2 and the pushout threshold of class 1 cells is given by $T_1(= K - T_2)$. If two cells arrive to a full buffer (i.e., $i + j = K$), then a class 2 cell is discarded if $j > T_2$, otherwise a class 1 cell is discarded if $j < T_2$. When $j = T_2$, the lower priority traffic class cell is discarded. One may view the system as if there is temporary space to store up to two arrivals while the buffer is full and a decision as to which class of cell will be discarded is made. The state transitions corresponding to space priority buffer management with pushout thresholds in ATM networks are given in Table 4.

To simplify the model, we assume that the head of the queue (i.e., the cell that will be leaving the buffer at the end of the current time slot — if there was one to begin with) is a type 1 cell with probability $i/(i + j)$ and it is a type 2 cell with probability $j/(i + j)$.

TABLE 4
State transitions for the threshold pushout scheme in ATM networks

Block transition	State transition from (i,j) to	Condition	Event	Probability
$k \to k-1$	$(i-1,j)$	$i>0$	No arrivals, class 1 departure	$\frac{i}{i+j}(1-p_1)(1-p_2)$
	$(i,j-1)$	$j>0$	No arrivals, class 2 departure	$\frac{j}{i+j}(1-p_1)(1-p_2)$
$k \to k$	$(i-1,j+1)$	$i>0$	Class 2 arrival, class 1 departure	$\frac{i}{i+j}(1-p_1)p_2$
	$(i-1,j+1)$	$i>0, j<T_2, i+j=K$	Class 1, 2 arrivals, class 1 departure	$\frac{i}{i+j}p_1p_2$
	$(i-1,j+1)$	$i>0, j=T_2, i+j=K, T_1<T_2$	Class 1, 2 arrivals, class 1 departure	$\frac{i}{i+j}p_1p_2$
	(i,j)	$i=0, j=0, K\geq 1$	Class 2 arrival, no departure	$(1-p_1)p_2$
	(i,j)	$i>0$	Class 1 arrival, class 1 departure	$\frac{i}{i+j}p_1(1-p_2)$
	(i,j)	$j>0$	Class 2 arrival, class 2 departure	$\frac{j}{i+j}(1-p_1)p_2$
	(i,j)	$i>0, j>T_2, i+j=K$	Class 1, 2 arrivals, class 1 departure	$\frac{i}{i+j}p_1p_2$
	(i,j)	$i>0, j=T_2, i+j=K, T_1\geq T_2$	Class 1, 2 arrivals, class 1 departure	$\frac{i}{i+j}p_1p_2$
	(i,j)	$0<j<T_2, i+j=K$	Class 1, 2 arrivals, class 2 departure	$\frac{j}{i+j}p_1p_2$
	(i,j)	$j>0, j=T_2, i+j=K, T_1<T_2$	Class 1, 2 arrivals, class 2 departure	$\frac{j}{i+j}p_1p_2$
	(i,j)	$i=0, j=0, K=0$	Always	1
	$(i+1,j-1)$	$j>0$	Class 1 arrival, class 2 departure	$\frac{j}{i+j}p_1(1-p_2)$
	$(i+1,j-1)$	$j>T_2, i+j=K$	Class 1, 2 arrivals, class 2 departure	$\frac{j}{i+j}p_1p_2$
	$(i+1,j-1)$	$j>0, j=T_2, i+j=K, T_1\geq T_2$	Class 1, 2 arrivals, class 2 departure	$\frac{j}{i+j}p_1p_2$
$k \to k+1$	$(i,j+1)$	$i=0, j=0, K\geq 1$	Class 2 arrival, no departure	$(1-p_1)p_2$
	$(i,j+1)$	$i>0, i+j<K$	Class 1, 2 arrivals, class 1 departure	$\frac{i}{i+j}p_1p_2$
	$(i,j+1)$	$i=0, j=0, K=1$ $T_2=1$	Class 1, 2 arrivals, no departure	p_1p_2
	$(i+1,j)$	$i=0, j=0, K\geq 1$	Class 1 arrival, no departure	$p_1(1-p_2)$
	$(i+1,j)$	$j>0, i+j<K$	Class 1, 2 arrivals, class 2 departure	$\frac{j}{i+j}p_1p_2$
	$(i+1,j)$	$i=0, j=0, K=1$ $T_2=0$	Class 1, 2 arrivals, departure	p_1p_2
$k \to k+2$	$(i+1,j+1)$	$i=0, j=0, K>1$	Class 1, 2 arrivals no departure	p_1p_2

The DTMC corresponding to these assumptions is block tridiagonal (with the exception of the first row of blocks) where each diagonal block is tridiagonal and has a different block size. Depending on the selected threshold, the nonzero elements in the last row of blocks change making it very difficult to apply analytical solution techniques to such a system with control.

The performance measures of interest for this application are the drop probabilities and the average number of type 1 and type 2 cells in the buffer at steady state. These measures may be computed once we obtain $\boldsymbol{\pi}$.

Results of the numerical experiments for all three applications are given in the next section.

5 Numerical Experiments

Experiments with GE and GTH algorithms are carried out in CSR and NSK storage formats on a SUN SPARC 1000. Some of the routines used are part of the software package MARCA (see [17]). All routines are written in Fortran and compiled in both double–precision and quadruple–precision floating–point arithmetic. For each problem solved, we employ five different solvers (see Table 5). GTH_{ncsr} uses the nontransposed system of equations given in (3.1); all other solvers use the transposed system of equations. GTH_{csr} shifts the contents of the work arrays when expanding/compacting rows; GTH_{ncsr} stores the multiplier matrix. We remark that the timing of the GTH_{ncsr} algorithm has been improved over the implementation discussed in [5], p. 294. This is achieved by reordering the loop that corrects the pivot element in the reduction phase.

For each problem solved, we provide the measures in Table 6 to help with the assessment of the results. The decomposability parameter γ may be taken as an approximation of the degree of coupling; it is used to determine the strongly connected components in the transition probability matrix $\mathbf{P}$ by simply ignoring the elements that are less than the suggested value. For each problem, we give the smallest possible γ and the corresponding number of blocks, N. The matrices supplied to the solvers GE_{nsk} and GTH_{nsk} are converted to the NSK format from the CSR format (see T_{map} in Table 6), which is our default format for storing matrices. One way to get around this conversion is to generate the Markov chain in the NSK format (however unnatural it is). Coefficient matrices input to the solvers are already in transposed form. Therefore, the solver GTH_{ncsr} needs to retranspose the input matrix back to the original nontransposed form in equation (3.1). Moreover, the relative error in the solution is computed as $\|\boldsymbol{\pi} - \tilde{\boldsymbol{\pi}}\|_2 / \|\boldsymbol{\pi}\|_2$, where $\boldsymbol{\pi}$ is the quadruple–precision solution and $\tilde{\boldsymbol{\pi}}$ is the double–precision solution. Both $\boldsymbol{\pi}$ and $\tilde{\boldsymbol{\pi}}$ are normalized so that their 1–norms are unity. We do not provide the residual error (i.e., $Err_{res} = \|\boldsymbol{\pi} - \tilde{\boldsymbol{\pi}}\|_2$) because in all problems each solver provides a solution with a residual error on the order of machine epsilon (i.e., 10^{-16}) or less.

We should also remark that in all problems considered the number of states are between 500 and 5,500. For problems with smaller state spaces, the run–time patterns are not as revealing as they are for larger problems. Furthermore, in each problem we give the amount of storage allocated for the NSK and CSR formats at the outset so that we have an appreciation for the memory requirements of the implementations.

TABLE 5
Solvers used

GE_{csr}	Gaussian elimination in CSR
GTH_{csr}	Grassmann–Taksar–Heyman algorithm in CSR
GTH_{ncsr}	Grassmann–Taksar–Heyman algorithm for (3.1) in CSR
GE_{nsk}	Gaussian elimination in NSK
GTH_{nsk}	Grassmann–Taksar–Heyman algorithm in NSK

TABLE 6
Notation for parameters of numerical methods

n	Order of the stochastic transition matrix, $\mathbf{P}$
nz	Number of nonzero elements in the matrix
γ	(Smallest) decomposability parameter
N	Number of blocks corresponding to γ
bw_{min}	Minimum bandwidth of $\mathbf{I} - \mathbf{P}^T$
bw_{avg}	Average bandwidth of $\mathbf{I} - \mathbf{P}^T$
bw_{max}	Maximum bandwidth of $\mathbf{I} - \mathbf{P}^T$
S	Solver used
T_{solver}	Time spent in S, (in CPU seconds)
T_{map}	Time spent to convert $\mathbf{P}^T$ from the CSR format to the NSK format, (in CPU seconds)
nzu	Number of nonzero elements in the upper–triangular factor
$Mflops$	Number of Megaflops in S
Err_{rel}	$\|\boldsymbol{\pi} - \tilde{\boldsymbol{\pi}}\|_2 / \|\boldsymbol{\pi}\|_2$

5.1 Application 1

The first application we consider is the complete buffer sharing scheme with pushout thresholds for ATM networks discussed in §4.

(a) In this part we use the following parameters:

$$K = 35,\ p_1 = 0.99,\ p_2 = 0.15,\ T_2 = 5.$$

The NSK format requires 31,749 double–precision and 1,333 integer storage locations.
The CSR format requires 4,379 double–precision and 5,046 integer storage locations.
The GTH_{ncsr} solver ends up with 15,575 nonzero elements in the lower–triangular factor (bandwidth of $\mathbf{I} - \mathbf{P}$ ranges between 5 and 72; its average bandwidth is 48.6).

TABLE 7
Results for Application 1(a):
$n = 666,\ nz = 4,379,\ \gamma = 10^{-4},\ N = 2\ ,[bw_{min} - bw_{max}] = [2 - 72],\ bw_{avg} = 46.8$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	0.25		16,240	0.44	$0.36E-15$
GTH_{csr}	4.82		16,242	0.66	$0.15E-14$
GTH_{ncsr}	0.38		15,578	0.66	$0.15E-14$
GE_{nsk}	0.34	0.41	16,835	0.45	$0.37E-15$
GTH_{nsk}	0.72	0.41	16,835	0.47	$0.16E-14$

(b) In this part we use the following parameters:

$$K = 75,\ p_1 = 0.9,\ p_2 = 0.9,\ T_2 = 10.$$

The NSK format requires 295,529 double–precision and 5,853 integer storage locations.
The CSR format requires 19,879 double–precision and 22,806 integer storage locations.
The GTH_{ncsr} solver ends up with 146,375 nonzero elements in the lower–triangular factor (bandwidth of $\mathbf{I} - \mathbf{P}$ ranges between 5 and 152; its average bandwidth is 102.0).

TABLE 8
Results for Application 1(b):
$n = 2,926,\ nz = 19,879,\ \gamma = 10^{-2},\ N = 644\ , [bw_{min} - bw_{max}] = [2 - 152],\ bw_{avg} = 100.1$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	4.67		149,300	8.44	$0.93E-15$
GTH_{csr}	100.21		149,302	12.72	$0.11E-14$
GTH_{ncsr}	6.73		146,378	12.72	$0.11E-14$
GE_{nsk}	6.38	8.04	152,075	8.58	$0.88E-15$
GTH_{nsk}	14.15	8.04	152,075	8.80	$0.13E-14$

(c) In this part we use the following parameters:

$$K = 100,\ p_1 = 0.9,\ p_2 = 0.9,\ T_2 = 10.$$

The NSK format requires 691,954 double–precision and 10,303 integer storage locations.
The CSR format requires 35,254 double–precision and 40,406 integer storage locations.
The GTH_{ncsr} solver ends up with 343,500 nonzero elements in the lower–triangular factor (bandwidth of $\mathbf{I} - \mathbf{P}$ ranges between 5 and 202; its average bandwidth is 135.3).

TABLE 9
Results for Application 1(c):
$n = 5,151,\ nz = 35,254,\ \gamma = 10^{-3},\ N = 23\ , [bw_{min} - bw_{max}] = [2 - 202],\ bw_{avg} = 133.3$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	14.59		348,650	26.24	$0.38E-14$
GTH_{csr}	330.69		348,652	39.50	$0.22E-14$
GTH_{ncsr}	20.71		343,503	39.50	$0.22E-14$
GE_{nsk}	19.88	25.21	353,600	26.57	$0.34E-14$
GTH_{nsk}	45.32	25.21	353,600	27.08	$0.22E-14$

5.2 Application 2

The second application we consider is the interactive computer system discussed recently in [5], pp. 296–298.

(a) In this part we use the following parameters:

$$\eta_t + \eta = 15,\ \lambda = (10^{-4})\eta_t,\ p_1\mu_0(\eta) = 100(\eta/128)^{1.5},\ p_2\mu_0(\eta) = 0.05,$$

$$1 - p_1 - p_2 = 0.002,\ \mu_1 = 0.2,\ \mu_2 = 1/30.$$

Here $\eta_t + \eta$ is the fixed number of processes in the computer system (i.e., closed queueing network).
The NSK format requires 62,032 double–precision and 1,633 integer storage locations.
The CSR format requires 4,896 double–precision and 5,713 integer storage locations.
The GTH_{ncsr} solver ends up with 30,608 nonzero elements in the lower–triangular factor.

TABLE 10
Results for Application 2(a):
$n = 816,\ nz = 4,896,\ \gamma = 10^{-4},\ N = 5\ , [bw_{min} - bw_{max}] = [2 - 103],\ bw_{avg} = 75.9$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	0.77		31,423	1.35	$0.11E-11$
GTH_{csr}	17.53		31,425	1.68	$0.79E-12$
GTH_{ncsr}	0.96		31,423	1.68	$0.79E-12$
GE_{nsk}	1.02	0.60	31,423	1.35	$0.11E-11$
GTH_{nsk}	1.55	0.60	31,423	1.38	$0.79E-12$

(b) In this part we use the same parameters as in part (a) except

$$\eta_t + \eta = 20.$$

The NSK format requires 222,211 double–precision and 3,543 integer storage locations.
The CSR format requires 11,011 double–precision and 12,783 integer storage locations.
The GTH_{ncsr} solver ends up with 110,220 nonzero elements in the lower–triangular factor.

TABLE 11
Results for Application 2(b):
$n = 1,771,\ nz = 11,011,\ \gamma = 10^{-4},\ N = 7\ , [bw_{min} - bw_{max}] = [2 - 170],\ bw_{avg} = 125.4$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	4.26		111,990	7.89	$0.15E-11$
GTH_{csr}	116.90		111,992	9.46	$0.16E-11$
GTH_{ncsr}	5.24		111,990	9.46	$0.16E-11$
GE_{nsk}	5.98	2.85	111,990	7.89	$0.13E-11$
GTH_{nsk}	8.10	2.85	111,990	8.00	$0.16E-11$

(c) In this part we use the same parameters as in part (a) except

$$\eta_t + \eta = 25.$$

The NSK format requires 613,912 double–precision and 6,553 integer storage locations.
The CSR format requires 20,826 double–precision and 24,103 integer storage locations.
The GTH_{ncsr} solver ends up with 305,318 nonzero elements in the lower–triangular factor.

TABLE 12
Results for Application 2(c):
$n = 3,276,\ nz = 20,826,\ \gamma = 10^{-4},\ N = 9\ , [bw_{min} - bw_{max}] = [2 - 253],\ bw_{avg} = 187.3$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	17.76		308,593	32.39	$0.74E-12$
GTH_{csr}	545.43		308,595	37.75	$0.54E-12$
GTH_{ncsr}	20.87		308,593	37.75	$0.54E-12$
GE_{nsk}	23.77	10.13	308,593	32.38	$0.85E-12$
GTH_{nsk}	31.32	10.13	308,593	32.69	$0.54E-12$

(d) In this part we use the same parameters as in part (a) except

$$\eta_t + \eta = 30.$$

The NSK format requires 1,428,416 double–precision and 10,913 integer storage locations.
The CSR format requires 35,216 double–precision and 40,673 integer storage locations.
The GTH_{ncsr} solver ends up with 711,480 nonzero elements in the lower–triangular factor.

TABLE 13
Results for Application 2(d):
$n = 5,456,\ nz = 35,216,\ \gamma = 10^{-5},\ N = 2\ ,[bw_{min} - bw_{max}] = [2 - 353],\ bw_{avg} = 261.7$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	57.31		716,935	104.95	$0.19E-11$
GTH_{csr}	1,989.25		716,937	119.82	$0.29E-11$
GTH_{ncsr}	67.22		716,935	119.82	$0.29E-11$
GE_{nsk}	77.84	28.84	716,935	104.94	$0.19E-11$
GTH_{nsk}	96.18	28.84	716,935	105.65	$0.29E-11$

5.3 Application 3

The last application we consider is from the telecommunications area (see [18], p. 343 for details).

(a) In this part we use the following parameters:

$$K1 = 25,\ K2 = 50,\ a = 0.6,\ \mu = 1,\ \tau = 0.05,\ h = 0.85,\ \lambda = 5.$$

Here $K1$ and $K2$ are the maximum number of customers permitted into stations S1 and S2 respectively.
The NSK format requires 47,801 double–precision and 2,653 integer storage locations.
The CSR format requires 6,451 double–precision and 7,778 integer storage locations.
The GTH_{ncsr} solver ends up with 23,400 nonzero elements in the lower–triangular factor (bandwidth of $\mathbf{I} - \mathbf{P}$ ranges between 2 and 75; its average bandwidth is 36.0).

TABLE 14
Results for Application 3(a):
$n = 1,326,\ nz = 6,451,\ \gamma = 10^{-2},\ N = 826\ ,[bw_{min} - bw_{max}] = [2 - 77],\ bw_{avg} = 36.1$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	0.33		24,725	0.51	$0.40E-12$
GTH_{csr}	8.51		24,727	1.39	$0.32E-12$
GTH_{ncsr}	0.68		24,400	1.39	$0.32E-12$
GE_{nsk}	0.41	1.27	24,725	0.51	$0.40E-12$
GTH_{nsk}	2.10	1.27	24,725	0.54	$0.32E-12$

(b) In this part we use the same parameters as in part (a) except

$$K1 = 10,\ K2 = 220.$$

The NSK format requires 54,296 double–precision and 4,863 integer storage locations.
The CSR format requires 11,681 double–precision and 14,113 integer storage locations.
The GTH_{ncsr} solver ends up with 25,960 nonzero elements in the lower–triangular factor (bandwidth of $\mathbf{I} - \mathbf{P}$ ranges between 2 and 30; its average bandwidth is 22.3).

TABLE 15
Results for Application 3(b):
$n = 2,431,\ nz = 11,681,\ \gamma = 10^{-2},\ N = 371\ , [bw_{min} - bw_{max}] = [2 - 32],\ bw_{avg} = 22.3$

S	T_{solver}	T_{map}	nzu	$Mflops$	Err_{rel}
GE_{csr}	0.26		28,390	0.35	$0.32E-12$
GTH_{csr}	21.75		28,392	3.31	$0.46E-12$
GTH_{ncsr}	1.37		28,335	3.30	$0.46E-12$
GE_{nsk}	0.31	4.21	28,390	0.35	$0.32E-12$
GTH_{nsk}	6.29	4.21	28,390	0.38	$0.46E-12$

Our first observation is related to the correlation between the decomposability parameter γ and the relative errors of GE and the GTH method. In all problems considered γ is larger than 10^{-6} and in none of the problems can GTH beat GE in terms of relative error. A similar observation along these lines also appears in [5], p. 301. As it is mentioned at the beginning of §4, a comparison of relative errors is not the aim of this paper; we therefore digress from the issue of accuracy to the topic of interest.

Due to the symmetric (or almost symmetric) nonzero structure of the coefficient matrices in the problems considered, the double–precision storage allocated for the NSK and GTH_{ncsr} implementations are approximately the same and twice as much as that allocated for GE_{csr} and GTH_{csr}. The total number of double precision entries allocated for NSK and GTH_{ncsr} implementations divide almost evenly between the lower– and upper–triangular factors; remember that the lower–triangular factor gets stored in these implementations. As expected (see §2), the NSK format consumes much less integer storage than the CSR format.

The coefficient matrices in the first application have nonsymmetric nonzero structure (see nzu in Tables 7, 8, and 9). A slightly larger number of nonzero elements is allocated for the upper–triangular factor in the NSK format. On the other hand, a slightly smaller number of nonzero elements is allocated for the upper–triangular factor in GTH_{ncsr}. In all three parts of this application, GTH_{nsk} ends up taking approximately twice as much time as GTH_{ncsr}.

In the second application, the nonzero structure of the coefficient matrices are symmetric. Interestingly, the same number of nonzero elements gets allocated for the upper–triangular factors in the CSR and NSK storage formats (see nzu in Tables 10, 11, 12, and 13). In other words, the same amount of fill–in is generated by the reduction phase. In all four parts of this application, the run–time of GTH_{nsk} is approximately one and a half times that of GTH_{ncsr}.

The coefficient matrices in both parts of the third application have relatively smaller average bandwidths compared with the first two applications. The smaller coefficient matrix in the first part of this application has the larger average bandwidth. Just like the first application, we have coefficient matrices with nonsymmetric nonzero structure. The run–time of GTH_{nsk} is 3–4 times that of GTH_{ncsr} for this application (see Tables 14–15).

The following are other observations we made by inspecting the results of the experiments. In all cases, GTH_{ncsr} (GTH_{csr}) is the fastest (slowest) GTH implementation and GE_{csr} is the faster GE implementation. The ratio of the run–times of GE_{csr} and GE_{nsk} ranges between 1.19 and 1.44, the higher end corresponding to Application 2 and the lower end corresponding to Application 3. In each application considered, a relatively smaller average bandwidth implies a larger ratio of the run–times of GTH_{nsk} and GTH_{ncsr}. In other words, a smaller profile causes GTH_{nsk} to suffer more (see Table 15, for instance). The ratio of the average bandwidth to the number of states in Application 3.b is 0.009, the smallest of all problems; the ratio of the run–times of GTH_{nsk} and GTH_{ncsr} is 4.59, the largest of all problems. The smallest ratio of the run–times of GTH_{nsk} and GTH_{ncsr} is obtained in Application 2.d as 1.43 (see Table 13). We forecast that for dense matrices this ratio is likely to get closer to 1. In all parts of Applications 1 and 2, the ratio of the run–times of GTH_{nsk} and GTH_{ncsr} is between 1 and 2 (see Tables 7–13). A correlation also exists between the ratio of the run–times of GTH_{ncsr} and GE_{csr} and the average bandwidth for each application. The largest ratio of the run–times of GTH_{ncsr} and GE_{csr} is obtained in Application 3.b as 5.27 (see Table 15). The smallest ratio of the run–times of GTH_{ncsr} and GE_{csr} is 1.17 of Application 2.d (see Table 13).

The number of megaflops for GE_{nsk} and GE_{csr} are equal in Application 2 and close to one another in other applications. The number of megaflops for GE_{nsk} is the smallest in Application 2.c (see Table 12). The number of megaflops for GTH_{nsk} is slighty larger than that of GE_{nsk} in all problems. This is due to the correction of the pivot element at each step. On the other hand, the number of megaflops for GTH_{nsk} is always less than that of GTH_{ncsr} and GTH_{csr}. This happens because updates due to zero pivot row elements are accounted for in the CSR implementations whereas there is a possibility of skipping (some of) the same zero pivot row element updates in Algorithm 2. See the second if–statement which tests for a nonzero pivot row element.

6 Conclusion

Computing the stationary probability distribution of a Markov chain amounts to solving a homogeneous linear system of equations with singular coefficient matrix. A modified version of GE referred to as the GTH method may be used to enforce stability and to improve accuracy in the solution process. This paper addresses a significant drawback of the GTH method. The GTH algorithm tends to run slower than the GE algorithm for a given problem in a sparse storage implementation.

We experimented with NSK and CSR format implementations of GE and GTH on three applications from the Markov modelling area. The coefficient matrices arising from these applications suggest that Markov chains in real life tend to have a structure, which is influenced by the order of generating the states. In our problems, the coefficient matrices were sparse and had relatively small average bandwidths. Moreover, the matrices had, if not symmetric, almost symmetric nonzero structure.

The NSK format allocates space for both lower– and upper–triangular factors at the outset and performs the reduction 'in place'. Hence, if it is possible to generate/store a Markov chain in the NSK format, then it is guaranteed that sufficient space is allocated for the upper–triangular factor; GE and GTH algorithms cannot terminate due to insufficient work space during the triangularization phase. On the other hand, for each solver implemented in the CSR format, at the outset it is possible to symbolically compute the maximum amount of storage necessary for the solver to run to completion.

The CSR implementation of GE for the transposed system of equations, GE_{csr}, results in the faster GE solver though the NSK implementation, GE_{nsk}, is quite competitive never executing

longer than 1.5 times that of the CSR implementation. As for GTH, the CSR implementation for the nontransposed system of equations, which accommodates the lower–triangular factor, GTH_{ncsr}, appears to be the fastest GTH solver. The CSR implementation of GTH for the transposed system of equations, GTH_{csr}, which does not store the lower–triangular factor, performs drastically slow. The NSK implementation of GTH is not as competitive as its GE counterpart due to the more frequent (complicated) indirect addressing involved. In conclusion, space is traded off for time to improve the run–time of GTH; delayed row updates makes the faster CSR implementations superior. When we need to use GTH, then the algorithm we should choose is GTH_{ncsr}. When we can get away with GE, we should use GE_{csr}.

Future work may focus on the effects of using the object–oriented paradigm in the design and implementation of sparse formats for direct methods such as GTH.

Acknowledgements. The author is currently supported by the Scientific and Technical Research Council of Turkey (TÜBİTAK) grant EEEAG-161; he gratefully acknowledges Professor William J. Stewart for providing access to the software package MARCA and for the stimulating discussions on sparse storage implementations of the GTH algorithm.

References

[1] 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: Building Blocks for Iterative Methods*, SIAM Press, Philadelphia, 1994.

[2] A. Berman and R. J. Plemmons, *Nonnegative Matrices in the Mathematical Sciences*, Academic Press, New York, 1979.

[3] W. L. Cao and W. J. Stewart, *Iterative aggregation/disaggregation techniques for nearly uncoupled Markov chains*, J. Assoc. Comput. Mach., 32 (1985), pp. 702–719.

[4] P.-J. Courtois, *Decomposability: Queueing and Computer System Applications*, Academic Press, New York, 1977.

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

[6] I. S. Duff, A. M. Erisman and J. K. Reid, *Direct Methods for Sparse Matrices*, Oxford University Press, London, 1986.

[7] W. K. Grassmann, M. I. Taksar and D. P. Heyman, *Regenerative analysis and steady state distributions for Markov chains*, Oper. Res., 33 (1985), pp. 1107–1116.

[8] W. J. Harrod and R. J. Plemmons, *Comparison of some direct methods for computing the stationary distributions of Markov chains*, SIAM J. Sci. Statist. Comput., 5 (1984), pp. 453–469.

[9] D. P. Heyman, *Further comparisons of direct methods for computing stationary distributions of Markov chains*, SIAM J. Sci. Statist. Comput., 8 (1987), pp. 226–232.

[10] J. R. Koury, D. F. McAllister and W. J. Stewart, *Iterative methods for computing stationary distributions of nearly completely decomposable Markov chains*, SIAM J. Alg. Discrete Meth., 5 (1984), pp. 164–186.

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

[12] C. A. O'Cinneide, *Entrywise perturbation theory and error analysis for Markov chains*, Numer. Math., 65 (1993), pp. 109–120.

[13] Y. Saad, *SPARSKIT: A Basic Tool for Sparse Matrix Computation*, Tech. Report CSRD TR 1029, CSRD, University of Illinois, 1990.

[14] P. J. SCHWEITZER, *A survey of aggregation-disaggregation in large Markov chains*, in Numerical Solution of Markov Chains, W. J. Stewart, ed., Marcel Dekker, Inc., New York, 1991, pp. 63–88.

[15] G. W. STEWART AND G. ZHANG, *On a direct method for the solution of nearly uncoupled Markov chains*, Numer. Math., 59 (1991), pp. 1–11.

[16] G. W. STEWART, W. J. STEWART AND D. F. MCALLISTER, *A two-stage iteration for solving nearly completely decomposable Markov chains*, in The IMA Volumes in Mathematics and its Applications 60: Recent Advances in Iterative Methods, by G. H. Golub, A. Greenbaum, and M. Luskin, eds., Springer-Verlag, New York, 1994, pp. 201–216.

[17] W. J. STEWART, *MARCA: Markov Chain Analyzer, A software package for Markov modeling*, in Numerical Solution of Markov Chains, W. J. Stewart, ed., Marcel Dekker, Inc., New York, 1991, pp. 37–61.

[18] W. J. STEWART AND W. WU, *Numerical experiments with iteration and aggregation for Markov chains*, ORSA J. Comput., 4 (1992), pp. 336–350.

[19] W. J. STEWART, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, New Jersey, 1994.

[20] D. TIPPER, S. PAPPU, A. COLLINS AND J. GEORGE, *Space priority buffer management for ATM networks*, in Asynchronous Transfer Mode Networks, Yannis Viniotis and Raif O. Onvural, eds., Plenum Press, New York, 1993, pp. 157–166.