Commun.Fac.Sci.Univ.Ank.Series A1
Volume 58, Number 2, Pages 1–10 (2009)
ISSN 1303–5991

AN EFFICIENT STORAGE FORMAT FOR LARGE SPARSE MATRICES

AIYOUB FARZANEH, HOSSEIN KHEIRI AND MEHDI ABBASPOUR SHAHMERSI

ABSTRACT. In this paper we consider linear system $Ax = b$ where A is a large sparse matrix. A new efficient, simple and inexpensive method for storage of coefficient matrix A was presented. The purpose of this method is to reduce the storage volume of large non-symmetric sparse matrices. The results shows that the proposed method is very inexpensive in comparison with current methods such as Coordinate format, Compressed Sparse Row (CSR) format and Modified Sparse Row (MSR) format.

1. INTRODUCTION

Consider a linear system

$$Ax = b \tag{1.1}$$

where A is a large random nonsingular sparse matrix of the order $n \times n$, and b is given column vector of order n. Such systems of linear equations are frequently encountered in almost all scientific and engineering applications. For instance, sparse matrices appear in various applications including structural analysis, computational fluid dynamics, economic modeling, financial analysis, numerical optimization, statistical modeling, power network analysis, electromagnetic, meteorology, medical imaging, data mining, finite-element simulations, decision support systems in management science, circuit simulations, information retrieval and many more. A number of significant advancements in sparse matrix computations have been made in recent years [1, 8].

The irregular nature of sparse matrix-vector multiplication, $Ax = b$, has led to the development of a variety of compressed storage formats, which are widely used because they do not store any unnecessary elements. In this paper we introduce a new method for storage of matrix A which we called it Compressed Sparse Vector (CSV) format. We show that storage volume and computational cost in CSV

Received by the editors April 06, 2009; Accepted: July 07, 2009.
1991 *Mathematics Subject Classification.* 65F50;65F50; 68P20;68P20.
Key words and phrases. Sparse matrix; Storage schemes; Data structures .

format is less than other existent methods such as Coordinate, CSR and MSR storage formats. The CSV format can be used for all arbitrary sparse matrices such as non-square matrices.

The outline of the paper is as follows. First, three popular storage format (Coordinate, CSR and MSR formats) illustrated in the next section. In section 3, the CSV format for storing the matrix A is given. In section 4, advantages of CSV storage method is described briefly, and finally, numerical examples are given to illustrate performance and effectiveness of the new method in section 5. We present a case study of a 5×5 sparse matrix to show the data structures and the algorithm to storage coefficient matrix of $Ax = b$ using the CSV format.

2. Storage Schemes

In many scientific computations the manipulation of sparse matrices is considered the crux of the design. Generally the non-zero elements in a sparse matrix constitutes a very small percentage of data. This irregular nature of sparse matrix problems has led to the development of a variety of compressed storage formats [1, 3 - 6, 9-11]. There are more than thirteen different storage formats for coefficient matrix A [2]. which are widely used. The Coordinate format, Compressed Sparse Row (CSR) and Modified Sparse Row (MSR) formats are three important storage methods which have been widely used in most sources [2 - 11].

2.1. **The Coordinate Storage Format.** The simplest storage scheme for sparse matrices is the so-called Coordinate format. The data structure consists of three arrays: (1) AA a real array containing all the real or complex values of the nonzero elements of A in any order; (2) JR an integer array containing their row indices; and (3) JC a second integer array containing their column indices. All three arrays are of length Nz, the number of nonzero elements [3].

For example let A be an square matrix of the order 5×5

$$A = \begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

This matrix will be represented (for example) by:

AA	12.	9.	7.	5.	1.	2.	11.	3.	6.	4.	8.	10.
JR	5	3	3	2	1	1	4	2	3	2	3	4
JC	5	5	3	4	1	4	4	1	1	2	4	3

In this example, the elements are listed in an arbitrary order. But, they are usually listed by row or columns.

2.2. **The CSR Storage Format.** The CSR format was originally suggested by A. Brameller [3] and D. J. Rose [4]. This format is the most popular scheme for storing large sparse matrices. In the above example (coordinate format), if the elements were listed by row, the array JC which contains redundant information might be replaced by an array which points to the beginning of each row instead. This would involve non-negligible saving in storage. Storing given matrix A with a CSR scheme requires three one-dimensional arrays AA, JR, and JC of length Nz, Nz, and $n+1$ respectively, where n is the number of rows and Nz is the total number of nonzero elements in the matrix A.

The array AA contains the non-zero elements of A stored row-by-row, JR contains the column indices which correspond to the non-zero elements in the array AA, and JC contains $n+1$ pointers which delimit the rows of non-zero elements in the array AA, as illustrated below.

Consider matrix (2), this matrix will be represented by:

AA	1. 2. 3. 4. 5. 6. 7. 8. 9. 10. 11. 12.
JR	1 4 1 2 4 1 3 4 5 3 4 5
JC	1 3 6 10 12 13

2.3. **The MSR Storage Format.** The Modified Sparse Row (MSR) format has only two parallel arrays of equal length (Nz+1): A real array AA and an integer array JA. The first n position in AA contains the diagonal elements of the matrix in order. The position $n+1$ of the array AA is not used, but may sometimes be used to carry other information concerning the matrix. Starting at position $n+2$, the non-zero elements of AA excluding its diagonal elements, are stored by row. For each element AA(k), the integer JA(k) represents its column index on the matrix. The $n+1$ first position of JA contains the pointer to the beginning of each row in AA and JA.

Thus for Matrix (2), the two arrays will be as follows:

AA	1. 4. 7. 11. 12. $*$ 2. 3. 5. 6. 8. 9. 10.
JA	7 8 10 13 14 14 4 1 4 1 4 5 3

The star denotes an unused location. Notice that JA(n) =JA($n+1$) = 14, indicating that the last row is a zero row, once the diagonal element has been removed [2]. The restriction of MSR method is that principal diagonal element of coefficient matrix must be non-zero [2].

3. The Compressed Sparse Vector (CSV) Storage Format

Now we introduce a new efficient method which like MSR format has two arrays but can be used for storing sparse matrices with arbitrary sparsity patterns. First we consider the following method which is the main idea for the Compressed Sparse Vector (CSV) format.
Suppose A is a non-square $m \times n$ large sparse matrix. We consider two arrays as AA and IA of length $Nz + 1$, where Nz is the number of non-zero elements in A. The first n position in the array AA contains the non-zeroes of A stored row-by-row and the first n position in IA contains the indices of non-zero elements that results from row counting indexing which assigns a number for any element and only saves the indices of non-zero elements in IA and non-zero elements in AA; and m and n are the last elements in AA and IA respectively.

For instance consider matrix (2), the arrays AA and IA are as follows:

AA	1	2	3	4	5	6	7	8	9	10	11	12
IA	1	4	6	7	9	11	13	14	15	18	19	25

In this version, by increasing dimensions of matrix A, the index value increases rapidly necessitating large amount of space assignments for storing indexes.

In order to overcome this problem, we implement new version of indexing called Compressed Sparse Vector (CSV) format, in which we start indexing from the first element of matrix, a_{11}, by considering two states as follows:

Case 1. If $a_{11}= 0$, it takes No.1 as index and we go to the next element, and counting continues till the first non-zero element, then we store coupled non-zero element and its associated index, and indexing continues starting from No.1 from next element until the last non-zero element is received.

Case 2. If $a_{11} \neq 0$, it takes No.1 as index then we store this non-zero element and its related index as first coupled, and indexing continues from next element starting from the number 1 as described in Case 1 and Case 2.
The new data structure has two arrays with the following function:

$*$ AA a real array of the length $Nz + 1$, that the first Nz element in AA is set aside to store non-zero elements of matrix A of any order and the position $Nz + 1$ indicates the number of rows.

$*$ IA an integer array of the length $Nz + 1$, that the first Nz element contains the indices which correspond to the non-zero elements in the array AA and the element $Nz + 1$ stands for the number of columns.
For example compressed format of matrix (2) using CSV format is as bellow:

AA	1	2	3	4	5	6	7	8	9	10	11	12
IA	1	3	2	1	2	2	2	1	1	3	1	6

The pseudocode of the CSV storage method is given as follows:

3.1. **ALGORITHM1. The CSV Method.**
Start.
Step1 :
Define index=0, IA as integer array and AA as real array.
Step2:
for i=1,...,n do
for j=1,...,m do
(i)index=index+1.
(ii) if $a_{i,j} \neq 0$ do
(I)add $a_{i,j}$ to AA array
(II) add index to IA array.
(III) index=0.
END DO.
END DO.
END DO.

3.2. **CSV Codes with Matlab.**
function ms=edited(a)
bp=[size(a,1) size(a,2)];
bs=[0 0]; index=0;
for i=1:size(a,1)
for j=1:size(a,2)
index=index+1;
if a(i,j) =0
bs=[index a(i,j)];
bp=[bp;bs];
index=0;
end
end
end
ms=bp;

4. Advantages of CSV format

The CSV storage method with a very simple algorithm has special advantages. Here we illustrate some advantages of this format.

4.1. **Less Storage Volume.** Considering the construction process of arrays AA and IA, storing volume of this method has been reduced considerably in comparison with other methods. Restarting indices values in the CSV, after passing each non-zero element has great effect in reducing storage volume of array IA. For instance in example 2, the results of Table 2 show that for a hepta-diagonal matrix of order 2000×2000, 7.62 MB of space is needed if the matrix stored with all its zero and non-zero elements, if we use coordinate, CSR and MSR storage methods, the file sizes reduces to 148.0 KB, 98.1 KB and 89.1 KB respectively, but a considerable decrease in required volume appears using CSV storage format, with storage volume of only 60.5 KB.
More illustrated examples have been given in numerical results.

4.2. **Ease of Transpose Matrix Calculation.** Calculating of transpose matrix A in current methods has difficulties and needs more computation. For example Coordinate format needs $3 \times Nz$ operations to calculate transpose matrix from the compressed format. But in CSV format we only replace the elements $Nz+1$ in both arrays AA and IA with together, and after that we change counting method. This can be done by changing the place of the last elements in AA and IA and counting in the opposite way of the one that has been used to create the compressed matrix; meaning that if we used row counting for the compressed matrix we should use column approach for the transpose matrix.

For example consider matrix (2), A^T can be calculated as follows:

$$\begin{pmatrix} 1 & 0 & 0 & 2 & 0 \\ 3 & 4 & 0 & 5 & 0 \\ 6 & 0 & 7 & 8 & 9 \\ 0 & 0 & 10 & 11 & 0 \\ 0 & 0 & 0 & 0 & 12 \end{pmatrix}$$

$\Downarrow$

Representing with CSV method (by row counting)

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

$\Downarrow$

Replacing last elements in both arrays

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

$\Downarrow$

recalling by column counting

$$\begin{pmatrix} 1 & 3 & 6 & 0 & 0 \\ 0 & 4 & 0 & 0 & 0 \\ 0 & 0 & 7 & 10 & 0 \\ 2 & 5 & 8 & 11 & 0 \\ 0 & 0 & 9 & 0 & 12 \end{pmatrix}$$

4.3. **High Speed.** Indexing algorithm of the CSV method shows that this method is very simple, needing a decreased number of operations in the computing and retrieving processes which makes it a quick-yielding method.

4.4. **Broad Range for Storage Sparse Matrices.** Considering the size of the compressed matrix in the CSV storage method, broader range of matrices can be represented by this method. Results in Table 4 show that the performance of the CSV method in dense sparse matrices is better than the others. For instance, consider matrix No.3 of the order 200×200 in the Table 4, storage volume of the compressed matrix of all the methods that used here, except the CSV method, are more than the storage volume of the original matrix.

5. Numerical Examples

In this section we tested the general tridiagonal, hepta-diagonal, random and dense sparse matrices with different dimensions. The CSV method has been compared with the Coordinate, CSR and MSR methods. Table 1 and Table 2 contain results obtained using these methods for tridiagonal and hepta-diagonal matrices. Also in Table 3, we examined these methods on random matrices and Table 4 is for dense matrices created by MATLAB [1].

In the Tables 1-4 the second column is for dimensions of matrices, the next column represent the number of non-zero elements, and "*primary*" column is for size of matrix that stored with its zero and non-zero elements, Furthermore the columns of "*Coordinate*", "*CSR*"[2], "*MSR*" and "*CSV*", represent the storage volume of the compressed matrices of relevant storage methods.

Example 1.

In this example three tridiagonal matrices of orders 1000, 2000, 3000 were compressed with Coordinate, CSR, MSR and CSV storage formats. Table 1 shows that the CSV method has a significant decline of storage volume in comparison with Coordinate, CSR and MSR methods.

[1] All the matrices has been saved with function dlwrite in MATLAB.

[2] Three values in CSR column are for storage of AA, JR and JC arrays respectively.

*	n	$\#zeros$	*Primary*	*coordinate*	*CSR*	*MSR*	*CSV*
1	1000	2998	1.90 MB	28.6 KB	(5.85+11.3+4.52)KB	18.3 KB	13.6 KB
2	2000	5998	7.62 MB	63.8 KB	(11.7+26.0+9.40)KB	38.8 KB	29.2 KB
3	3000	8998	17.1 MB	98.9 KB	(17.5+40.6+14.2)KB	59.3 KB	43.9 KB

Table 1: Results for Tridiagonal Matrices

Example 2.

In this example three hepta-diagonal matrices of orders 1000, 2000, 3000 were compressed with Coordinate, CSR, MSR and CSV storage formats. In this example like example 1, capability of CSV method appears in Table 2.

*	n	$\#zeros$	*Primary*	*coordinate*	*CSR*	*MSR*	*CSV*
1	1000	6988	1.90 MB	66.8 KB	(13.6+26.5+4.73)KB	41.3 KB	29.2 KB
2	2000	13988	7.62 MB	148.0 KB	(27.3+60.7+10.1)KB	89.1 KB	60.5 KB
3	3000	20988	17.1 MB	230.0 KB	(40.9+94.9+16.0)KB	137.0 KB	90.7 KB

Table 2: Results for Hepta-diagonal Matrices

Example 3.

Here, again, we used Coordinate, CSR, MSR and CSV storage methods for compressing sparse matrices, which have been tested on the following three randomly selected matrices. The systems of linear equations with the size of 5000, 6000 and 7000 were considered. The results are reported in Table 3.

*	n	$\#zeros$	*Primary*	*coordinate*	*CSR*	*MSR*	*CSV*
4	5000	165777	47.6 MB	202.0 KB	(47.1+77.3+25.6)KB	125.0KB	123.0 KB
5	6000	25865	68.6 MB	320.0 KB	(73.6+124.0+30.4)KB	201.0KB	187.0 KB
6	7000	30176	93.4 MB	371.0 KB	(85.9+142.0+38.2)KB	233.0KB	220.0 KB

Table 3: Results for random Matrices

Example 4.

We, also, tested Coordinate, CSR, MSR and CSV storage methods for compressing dense sparse matrices on the following three dense random matrices with approximately 50 percent of non-zero elements. The systems of linear equations with the size of 100, 150, and 200 were considered. Results of this example show that the CSV storage format keeps its effectiveness for storage of dense matrices in comparison with above formats. The results are reported in Table 4.

*	n	$\#zeros$	*Primary*	*coordinate*	*CSR*	*MSR*	*CSV*
1	100	4997	29.3 KB	52.9 KB	(14.6+20.0)KB+630 B	33.1KB	29.3 KB
2	150	11109	65.9 KB	125.0 KB	(32.5+46.4)KB+951 B	68.2 KB	65.1 KB
3	200	19953	117.0 KB	232.0 KB	(58.5+86.6+1.30)KB	130.5KB	116.0 KB

Table 4: Results for dense random Matrices

6. Conclusion

A new storage method for large sparse matrices was presented in this paper. This new method which we called it Compressed Sparse Vector (CSV) format, for storage of coefficient matrix A of linear system (1), has been based on row counting indexing, in CSV method, growing rate of indices values has been controlled by restarting indices after passing each non-zero element. In this work we considered the case when matrix A is multi-diagonal (tri- and hepta-diagonal) and also the case of random matrices. Results show that storage compaction in this new method is better than other methods. Also, we showed that calculating of transpose of matrix A is very simple without any computation cost. Furthermore, we can conclude that application of CSV method for representing sparse matrices not only reduces the storage volume of the compressed matrix, but also it increases the speed of the computers in practice. Also, using this method is suitable for dense sparse matrices, therefore, a broad range of sparse matrices could be compressed. Thus, since memory is an issue, the method's low storage requirements provide a means to tackle very large problems which would otherwise be out of reach.

ÖZET: Bu çalışmada, A bir geniş seyrek matris olmak üzere, $Ax = b$ lineer sistemi ele alınarak katsayılar matrisi olan A nın, depolanmasına ilişkin etkin, basit ve kolay bir yöntem verilmiştir. Bu yöntemin amacı, simetrik olmayan, geniş seyrek matrislerin depolanma hacmini küçültmektir. Elde edilen sonuçlar, önerilen yöntemin, Koordinat Formatı, Sıkıştırılmış Seyrek Satır (CSR) Formatı ve Uyarlanmış Seyrek Satır (MSR) Formatı gibi, halen kullanılmakta olan formatlarla kıyaslandığında çok daha kolay olduğunu göstermektedir

References

[1] Esmond G. Ng and Daniel J. Pierce, Introduction to the Special Section on Sparse and Structured Matrices and Their Applications, SIAM. J. Matrix Anal. and Appl. 20, 887 (1999).
[2] Y. Saad, Modified from SPARSKIT: a basic tool kit for sparse matrix computations,(June 6, 1994).
[3] Y. Saad, Iterative Methods For Sparse Linear Systems, 2nd edition with corrections.(Jan. 3RD, 2000).
[4] A.Brameller, R.N. Allan and Y.M. Hamam, Sparsity, Pitman, New York, (1976).
[5] D.J. Rose and R.N. Willoughby, Sparse Matrix and its Applicatios, Plenum Pres, New York , (1972).
[6] R.C. Mittal and A.H. Al-Kurdi, LU-decomposition and numerical structure for solving large sparse nonsymmetric linear systems, Computers Math. Applic. 42, 131-155, (2001).

[7] R.C. Mittal and A.H. Al-Kurdi, An efficient method for constructing an ILU preconditioner for solving large systems by the GMRES method, Computers Math. Applic. 45, 1757-1772, (2003).
[8] A. Pinar and V. Vassilevska, Finding Nonoverlapping Substructures of a Sparse Matrix, Electronic Transactions on Numerical Analysis (ETNA), Volume 21, 107-124, 2005.
[9] J. Dongarra, Sparse matrix storage formats, in: Z. Bai, et al. (Eds.), Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide, SIAM, Philadelphia, 2000. Electronic version available at: http://www.cs.utk.edu/ dongarra/etemplates/node372.html.
[10] Matrix Market, Electronic version available at: http://math.nist.gov/MatrixMarket/.
[11] E. Montagne, A. Ekambaram, An optimal storage format for sparse matrices, Information Processing Letters. 90, 87-92, (2004).

Current address: Faculty of Mathematical Sciences, University of Tabriz, Tabriz - Iran
E-mail address: ayub_farzaneh@yahoo.com