ALGORITHM FOR ELIMINATION OF SYSTEMS WITH SPARSE ASYMMETRIC REDUCIBLE MATRICES

Daniela Bittnerová

Faculty of Science, Humanities and Education
Department of Mathematics and Didactics of Mathematics
Studentská 2, 461 17, Liberec, Czech Republic
daniela.bittnerova@tul.cz

Abstract

In the paper, an algorithm for finding an optimal or almost optimal permutation for an ordering of elements of a matrix, which is sparse, asymmetric and reducible, is suggested. Using this algorithm we can solve large sparse systems of linear equations more efficiently. The algorithm is a modification of the algorithm presented in [6], therefore the same indications and symbols are use.

Keywords: Asymmetric reducible sparse matrix, algorithm.

Introduction

A matrix can be considered as sparse if the number of its zero elements is much greater than the number of nonzero ones. Sparse matrices occur in many fields of applied mathematics, for example at partial differential equations solving, but also in many other fields of science, like structural analysis, management analysis, power system analysis, surveying etc. Solving a practical problem we obtain large systems of linear equations with sparse matrices. Hand in hand with a development of a parallel programming for large quantity data, the importance of an optimization of sparse matrices increases. There exist lots of ways how to do it. Some of them go out from graph structures of associated matrices. In the paper, one such algorithm is suggested. It is a modification of the algorithm presented in [6], where symmetric sparse irreducible matrices were discussed, for generally Boolean asymmetric reducible sparse matrices. Using such algorithm we are able to solve large sparse systems of linear equations by some direct method. Indications and symbols are kept from the paper [6].

1. Basic Theory and Assumptions

Let $\mathbf{A} = (a_{i,j})$ be a sparse matrix of order *n*, $\mathbf{V} = (v_{i,j})$, $\mathbf{W} = (w_{i,j})$ be matrices of order *n*, where

$$v_{i,j} \in \{0;1\}, \quad \forall i, j = 1, \cdots n \;\; v_{i,j} = 1 \Leftrightarrow a_{i,j} \neq 0$$

$$w_{i,j} \in \{0;1\}, \quad \forall i, j = 1, \cdots n \;\; w_{i,j} = 1 \Leftrightarrow (a_{i,j} \neq 0 \;\vee\; a_{j,i} \neq 0).$$

The matrix **A** is called **Boolean symmetric** if **V** = **W**. In the opposite case, it is said **Boolean asymmetric**.

The graph $\left[\begin{array}{l}\mathbf{G}(\mathbf{A}) = (\mathbf{X}, \mathbf{E}) \\ \mathbf{G}^{\mathbf{0}}(\mathbf{A}) = (\mathbf{X}, \mathbf{E}^{\mathbf{0}})\end{array}\right.$ is called $\left[\begin{array}{l}\text{the } \textbf{non-oriented} \\ \qquad\quad \textbf{oriented}\end{array}\right.$ graph associated to the matrix **A**, if $\mathbf{X}=\{x_1,\cdots,x_n\}$ is a set of nodes of the graph $\left[\begin{array}{l}\mathbf{G}(\mathbf{A})\\ \mathbf{G^0}(\mathbf{A})\end{array}\right.$ such that the node x_i corresponds to the row *i* of **A**, and the set $\left[\begin{array}{l}\mathbf{E}\\ \mathbf{E^0}\end{array}\right.$ is a set of all $\left[\begin{array}{l}\text{non-oriented}\\ \text{oriented}\end{array}\right.$ edges of the graph $\left[\begin{array}{l}\mathbf{G}(\mathbf{A}),\\ \mathbf{G^0}(\mathbf{A})\end{array}\right.$

i.e.
$$\left[\begin{array}{l}\mathbf{E}=\{\{x_i;x_j\};\,x_i\in\mathbf{X}\wedge x_j\in\mathbf{X}\wedge w_{i,j}=1,\,i<j\}.\\ \mathbf{E^0}=\{[x_i;x_j];\,x_i\in\mathbf{X}\wedge x_j\in\mathbf{X}\wedge v_{i,j}=1,\,i\neq j\}\end{array}\right.$$

If **G** is a connected graph associated to the matrix **A**, it implies the matrix **A** is irreducible. Let us denote:

$$\mathbf{N}(x)=\{y\in\mathbf{X};\{x;y\}\in\mathbf{E}\}\cup\{x\}$$
$$\mathbf{N_1}(x)=\{y\in\mathbf{X};[x;y]\in\mathbf{E^0}\}\cup\{x\}$$
$$\mathbf{N_2}(x)=\{y\in\mathbf{X};[y;x]\in\mathbf{E^0}\}\cup\{x\}$$
$$\mathbf{D}(x)=\{\{y;z\}\notin\mathbf{E};\,y\in\mathbf{N}(x)\wedge z\in\mathbf{N}(x)\wedge y\neq z\}$$
$$\mathbf{D_1}(x)=\{[y;z]\notin\mathbf{E^0};\,y\in\mathbf{N_1}(x)\wedge z\in\mathbf{N_1}(x)\wedge y\neq z\}$$
$$\mathbf{D_2}(x)=\{[y;z]\notin\mathbf{E^0};\,y\in\mathbf{N_2}(x)\wedge z\in\mathbf{N_2}(x)\wedge y\neq z\}$$

Evidently,

N is the set of all successors (neighbours) of the node *x* including *x* in the graph **G**,

$\left[\begin{array}{l}\mathbf{N_1}(x)\\ \mathbf{N_2}(x)\end{array}\right.$ is the set of all $\left[\begin{array}{l}\text{heads (ended nodes)}\\ \text{tails (starting nodes)}\end{array}\right.$ of all oriented edges $\left[\begin{array}{l}\text{from}\\ \text{to}\end{array}\right.$ *x* in $\mathbf{G^0}$ including *x*.

The sets **D**(*x*), $\mathbf{D_1}(x)$, $\mathbf{D_2}(x)$ describe relations of successors in the graph $\mathbf{G^0}$ each other, in relation to the fill in of matrices **L** and **U** in the **LU**-decomposition of the matrix **A** (and/or to the fill in of matrices **L**, **D** in the $\mathbf{LDL^T}$-decomposition). The numbers of elements of the sets $\mathbf{N}(x_i)$, $\mathbf{N_1}(x_i)$, $\mathbf{N_2}(x_i)$, $\mathbf{D}(x_i)$, $\mathbf{D_1}(x_i)$, $\mathbf{D_2}(x_i)$ are denoted by the symbols $n_i(\mathbf{G})$, $n_i^1(\mathbf{G^0})$, $n_i^2(\mathbf{G^0})$, $d_i(\mathbf{G})$, $d_i^1(\mathbf{G^0})$, $d_i^2(\mathbf{G^0})$, respectively. The number $d_i(\mathbf{G})$ (called "fill in") determines a number of new edges after elimination of the node x_i.

The graph $\left[\begin{array}{l}\mathbf{G}_{x_i}=(\mathbf{X}-\{x_i\},\ \mathbf{E}(\mathbf{X}-\{x_i\})\cup\mathbf{D}(x_i))\\ \mathbf{G}^0_{x_i}=(\mathbf{X}-\{x_i\},\ \mathbf{E^0}(\mathbf{X}-\{x_i\})\cup\mathbf{D}_1(x_i)\cup\mathbf{D}_2(x_i))\end{array}\right.$ is called the graph produced from the graph $\left[\begin{array}{l}\mathbf{G}\\ \mathbf{G^0}\end{array}\right.$ by an elimination of the node x_i.

With respect to the found algorithm, let us denote $\mathbf{G_0}=\mathbf{G}$, $\mathbf{G^0_0}=\mathbf{G^0}$,

$\left[\begin{array}{l}\mathbf{G}_k\\ \mathbf{G^0}_k\end{array}\right.$ be a graph produced from $\left[\begin{array}{l}\mathbf{G}_{k-1}\\ \mathbf{G^0}_{k-1}\end{array}\right.$ by an elimination of the node x_i, $k=1,2,\cdots,n-1$.

2. Algorithms

Our goal is to find an optimal, respective almost optimal, permutation to change the given ordering of rows and columns of the matrix **A** so that the elimination of a system with the matrix **A** is as efficient as possible. In [6], two algorithms are presented. They are defined for irreducible sparse symmetric systems. The following algorithms are determined for reducible sparse generally asymmetric systems. The Algorithm 1 deals with matrices as if symmetric, Algorithm 2 deals with asymmetric matrices.

Algorithm 1

Step 0:

Put $\mathbf{S^0} = \emptyset,\ \mathbf{G_0} = \mathbf{G}$.

Step $k = 1, \cdots, n-1$:

1. Put $\mathbf{R^k} = \mathbf{S^{k-1}}$ for $\mathbf{S^{k-1}} \neq \emptyset$ else $\mathbf{R^k} = \{x_j;\, x_j \in \mathbf{G_{k-1}}\}$.
2. $\mathbf{S_1^k} = \left\{ x_j \in \mathbf{R^k};\, d_j(\mathbf{G_{k-1}}) = \min\limits_{x_q \in \mathbf{R^k}} d_q(\mathbf{G_{k-1}}) \right\}$.
3. $i_k = \min\limits_{x_j \in \mathbf{S_1^k}} \{j\}$.
4. Eliminate the node x_{i_k} from the graph $\mathbf{G_{k-1}}$ (i.e. the index i_k constructed in the permutation *p* is equal to the integer *k*).
5. $\mathbf{S^k} = \{x_j \in \mathbf{N}(x_{i_k}) \cap \mathbf{G_k}\}$

Algorithm 2

Step 0:

Put $\mathbf{S^0} = \emptyset,\ \mathbf{G_0^0} = \mathbf{G^0}$.

Step $k = 1, \cdots, n-1$:

1. Put $\mathbf{R^k} = \mathbf{S^{k-1}}$ for $\mathbf{S^{k-1}} \neq \emptyset$ else $\mathbf{R^k} = \{x_j;\, x_j \in \mathbf{G_{k-1}^0}\}$.
2. $\mathbf{S_1^k} = \left\{ x_j \in \mathbf{R^k};\, d_j(\mathbf{G_{k-1}^0}) = \min\limits_{x_q \in \mathbf{R^k}} d_q(\mathbf{G_{k-1}^0}) \right\}$.
3. $i_k = \min\limits_{x_j \in \mathbf{S_1^k}} \{j\}$.
4. Eliminate the node x_{i_k} from the graph $\mathbf{G_{k-1}^0}$ (i.e. the index i_k constructed in the permutation *p* is equal to the integer *k*).
5. $\mathbf{S^k} = \{x_j \in (\mathbf{N_1}(x_{i_k}) \cup \mathbf{N_2}(x_{i_k})) \cap \mathbf{G_k^0}\}$

Sometimes, Boolean symmetric matrices of linear systems contain a row or column, of which the number of non-zero elements approximates to *n*.

Then it is suitable to define a maximal permissible number of elements $d_{\max}$ of the set $\mathbf{D}(x_i)$, and transport nodes x_i, for them $d_i(\mathbf{G}) > d_{\max}$, resp. $d_i(\mathbf{G^0}) > d_{\max}$, in the desired permutation to the last positions. We will eliminate these nodes from the graph **G**, resp. $\mathbf{G^0}$, and Algorithm 1, resp. Algorithm 2, deals with remaining nodes.

Example 1:

Let

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 1 & 0 & 0 & 2 \\ 3 & 0 & -1 & 0 & 1 \\ 4 & 0 & 0 & 2 & 0 \\ 5 & 2 & 1 & 0 & 3 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 1 & -1 & 2 & 0 & 3 \\ -4 & 1 & 0 & 0 & 0 \\ 0 & 0 & 3 & 0 & 4 \\ 2 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 4 \end{bmatrix}$$

be matrices. Applying the Algorithm 1 above, we have got the permutation matrix **P** = (2, 5, 3, 1, 4) and then

$$\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{P^T} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix},$$

$$\mathbf{PAP^T} = \begin{bmatrix} 1 & 2 & 0 & 2 & 0 \\ 2 & 3 & 1 & 5 & 0 \\ 0 & 1 & -1 & 3 & 0 \\ 2 & 5 & 3 & 1 & 4 \\ 0 & 0 & 0 & 4 & 2 \end{bmatrix}, \quad \mathbf{PBP^T} = \begin{bmatrix} 1 & 0 & 0 & -4 & 0 \\ 1 & 4 & 0 & 0 & 0 \\ 0 & 4 & 3 & 0 & 0 \\ -1 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 2 & 1 \end{bmatrix}.$$

The products **PA** and **PB** exchange rows of the matrix **A** and **B**, $\mathbf{AP^T}$ and $\mathbf{BP^T}$ exchange columns of **A** and **B**. Algorithms 1 manipulate with matrices as they are Boolean symmetric. It removes (almost all) non-zero elements of diagonals and produces (almost all) non-zero vectors perpendicular to it. Let us look at the structures of the matrices **A** and **B** (non-zero elements are indicated by the symbol **x**, the symbol $\bullet$ denotes non-zero elements which we must fill in **B** to obtain the symmetric matrix **B**):

$$\mathbf{A} = \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{x} & & & \mathbf{x} \\ \mathbf{x} & & \mathbf{x} & & \mathbf{x} \\ \mathbf{x} & & & \mathbf{x} & \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & & \mathbf{x} \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \mathbf{x} & \mathbf{x} & \mathbf{x} & \bullet & \mathbf{x} \\ \mathbf{x} & \mathbf{x} & & & \bullet \\ \bullet & & \mathbf{x} & & \mathbf{x} \\ \mathbf{x} & & & \mathbf{x} & \\ \bullet & \mathbf{x} & \bullet & & \mathbf{x} \end{bmatrix}$$

Then the matrix **B** has the same structure as **A**.

Using the algorithms 1, we have got

$$\mathbf{PAP^T} = \begin{bmatrix} \mathbf{x} & \mathbf{x} & & \mathbf{x} & \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \\ & \mathbf{x} & \mathbf{x} & \mathbf{x} & \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \\ & & & \mathbf{x} & \mathbf{x} \end{bmatrix}, \qquad \mathbf{PBP^T} = \begin{bmatrix} \mathbf{x} & \bullet & & \mathbf{x} & \\ \mathbf{x} & \mathbf{x} & \bullet & \bullet & \\ & \mathbf{x} & \mathbf{x} & \bullet & \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \bullet \\ & & & \mathbf{x} & \mathbf{x} \end{bmatrix}.$$

Example 2:

In the Table 1, we see how the almost optimal permutation is changing depending on the choice of the number $d_{\max}$.

Tab. 1. *The dependance of the permutation on* $d_{\max}$

Matrices A, B		**$d_{\max}$ = 0**			**$d_{\max}$ = 1**			**$d_{\max}$ = 2**		
i		D(*i*)	inv$_i$	per$_i$	D(*i*)	inv$_i$	per$_i$	D(*i*)	inv$_i$	per$_i$
1	1,2,3,4,5	[2,3]	**5**	2	[2,3], [2,3]	**5**	2	[2,3], [2,4], [3,4], [4,5]	**5**	2
2	1,2,5	∅	1	3	∅	1	5	∅	1	5
3	1,3,5	∅	2	4	∅	3	3	∅	3	3
4	1,4	∅	3	5	∅	4	4	∅	4	4
5	1,2,3,5	[2,3]	**4**	1	[2,3]	2	1	[2,3]	2	1

Source: Own based on computations

The highlighted node *i* is the node, for that the number of elements of the set D(*i*) is greater than the chosen number $d_{\max}$ and that is transport to last free positions in the permutation **P**.

If $d_{\max}$ = 2 then the permutation matrix **P** is of the form $\mathbf{P} = (2,5,3,4,1)$ and

$$\mathbf{PAP^T} = \begin{bmatrix} 1 & 2 & 0 & 0 & 2 \\ 2 & 3 & 1 & 0 & 5 \\ 0 & 1 & -1 & 0 & 3 \\ 0 & 0 & 0 & 2 & 4 \\ 2 & 5 & 3 & 4 & 1 \end{bmatrix}, \ \mathbf{PBP^T} = \begin{bmatrix} 1 & 0 & 0 & 0 & -4 \\ 1 & 4 & 0 & 0 & 0 \\ 0 & 4 & 3 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 \\ -1 & 3 & 2 & 0 & 1 \end{bmatrix}.$$

The structures of non-zero elements are:

$$\mathbf{PAP^T} = \begin{bmatrix} \mathbf{x} & \mathbf{x} & & & \mathbf{x} \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & & \mathbf{x} \\ & \mathbf{x} & \mathbf{x} & & \mathbf{x} \\ & & & \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} & \mathbf{x} \end{bmatrix}, \qquad \mathbf{PBP^T} = \begin{bmatrix} \mathbf{x} & \bullet & & \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{x} & \bullet & & \bullet \\ & \mathbf{x} & \mathbf{x} & & \bullet \\ & & & \mathbf{x} & \mathbf{x} \\ \mathbf{x} & \mathbf{x} & \mathbf{x} & \bullet & \mathbf{x} \end{bmatrix}.$$

The advantage of these algorithms is a less number of computing operations for a solving of systems of linear equations since we calculate with almost all non-zero elements of given matrices of systems.

Conclusion

In a science papers and books, there exist lots of algorithms and methods, how to solve large linear systems with a sparse matrix. Principles of computations are different. Some of them use graphic structures of matrices. The algorithm above is one of them. The other ways could be found, for example in [2] -[6].

Acknowledgements

The paper was supported by the project ESF, no. CZ.1.07/2.3.00/09.0155, „Constitution and Improvement of a Team for Demanding Technical Computations on Parallel Computers at TU Liberec.“

Literature

[1] Duff, I. S. – Erisman, A. M. – Reid, J. K.: *Direct Methods for Sparse Matrices*. Oxford, Clarendon Press 1990, pp. 341. ISBN 0-19-853421-3

[2] Ersavas, B. F.: *Sparse Matrix Ordering and Gaussian Elimination.* In: ECE 3652 Fundamentals of Computer Engineering 2002, pp. 1-12.

[3] Gilbert, A. - Li, Y. - Porat, E. - M. Strauss, M.: *Approximate Sparse Recovery: Optimizing time and measurements*. Manuscript, 2009.

[4] Gilbert, J. R. – Ng, E. G.: *Predicting Structure in Nonsymetric Sparse Matrix Factorizations*. New York 1993.

[5] Kapre, N. – DeHon, A.: *Parallelizing Sparse Matrix Solve for Spice Circuit Simulation Using FPGAS.* In: Proceedings of IEEE International Conference on Field-Programmable Technology (FPT 2009), 2009, pp. 1-9.

[6] Segethová, J.: *Elimination on Sparse Symmetric Systems of a Special Structure.* Aplikace matematiky 17, 6, 1972, pp. 447–460.

RNDr. Daniela Bittnerová, CSc.

ALGORITMUS PRO ELIMINACI SOUSTAV S ŘÍDKÝMI NESYMETRICKÝMI ROZLOŽITELNÝMI MATICEMI

V článku je uveden algoritmus určený k nalezení optimálního či skoro optimálního uspořádání prvků matice, která je řídká, nesymetrická a rozložitelná (reducibilní). Pomocí tohoto algoritmu můžeme efektivněji řešit velké řídké soustavy lineárních rovnic. Uvedený algoritmus je modifikací známého algoritmu pro symetrické nerozložitelné matice.

EIN ALGORITHMUS FÜR DIE ELIMINIERUNG DER SYSTEME MIT SELTENEN ASYMMETRISCHEN ZERLEGBAREN MATRIXEN

Im Artikel wird ein Algorithmus vorgestellt, der zu einer optimalen oder fast optimalen Anordnung der Matrixelemente führen soll. Diese Matrix ist selten, asymmetrisch und zerlegbar (reduzierbar). Mit Hilfe dieses Algorithmus können wir große seltene Systeme linearer Gleichungen effektiver lösen. Der angeführte Algorithmus ist eine Modifikation eines bekannten Algorithmus für eine symmetrische, nicht zerlegbare Matrix.

ALGORYTM DO ELIMINACJI UKŁADÓW Z RZADKIMI NIESYMETRYCZNYMI MACIERZAMI ROZKŁADALNYMI

W artykule przedstawiono algorytm przeznaczony do znalezienia optymalnego lub prawie optymalnego układu elementów macierzy, która jest rzadka, niesymetryczna i rozkładalna. Przy pomocy tego algorytmu można bardziej efektywnie rozwiązywać duże rzadkie układy równań liniowych. Podany algorytm stanowi modyfikację znanego algorytmu dla symetrycznych macierzy nierozkładalnych.