Hindawi Publishing Corporation
Mathematical Problems in Engineering
Volume 2013, Article ID 403912, 7 pages
http://dx.doi.org/10.1155/2013/403912

Research Article

Partial Refactorization in Sparse Matrix Solution: A New Possibility for Faster Nonlinear Finite Element Analysis

Qi Song, Pu Chen, and Shuli Sun

Department of Mechanics and Engineering Science & LTCS, College of Engineering, Peking University, Beijing 100871, China

Correspondence should be addressed to Shuli Sun; sunsl@mech.pku.edu.cn

Received 8 July 2013; Accepted 7 August 2013

Academic Editor: Song Cen

This paper proposes a partial refactorization for faster nonlinear analysis based on sparse matrix solution, which is nowadays the default solution choice in finite element analysis and can solve finite element models up to millions degrees of freedom. Among various fill-in's reducing strategies for sparse matrix solution, the graph partition is in general the best in terms of resultant fill-ins and floating-point operations and furthermore produces a particular graph of sparse matrix that prevents local change of entries from wide spreading in factorization. Based on this feature, an explicit partial triangular refactorization with local change is efficiently constructed with limited additional storage requirement in row-sparse storage scheme. The partial refactorization of the changed stiffness matrix inherits a big percentage of the original factor and is carried out only on partial factor entries. The proposed method provides a new possibility for faster nonlinear analysis and is mainly suitable for material nonlinear problems and optimization problems. Compared to full factorization, it can significantly reduce the factorization time and can make nonlinear analysis more efficient.

1. Introduction

Nonlinearities [1, 2] occur in practical applications of many engineering fields. For example, in design of steel structures, elastoplastic analyses are necessary to compute the limit loads of truss, frame, or shell structures. In area of concrete structural design or soil mechanics, complicated nonlinear constitutive relations have to be involved for realistic descriptions. In case of cable design, geometrically nonlinear effects have to be considered for large displacements. In numerical simulation of rainfall effect on stability of slope, the changes in pore-water pressure will alter the elastoplastic behavior of geologic media. In analysis of seepage [3], the coefficient of permeability changes with capillary pressure and can be also treated as nonlinear problems.

The nonlinear equilibrium equations are typified by the nonlinear system of equations:

$$\mathbf{R}(\mathbf{x}) = 0. \tag{1}$$

Generally, this can be achieved in finite element methods by the so-called Newton-Raphson method, which involves the linearization of the equilibrium equations, given by

$$\begin{gathered}\mathbf{R}(\mathbf{x}_{k+1}) \approx \mathbf{R}(\mathbf{x}_k) + \mathbf{K}(\mathbf{x}_k)\,\mathbf{u} = 0; \quad \mathbf{x}_{k+1} = \mathbf{x}_k + \mathbf{u}, \\ \mathbf{K}(\mathbf{x}_k)\,\mathbf{u} = D\mathbf{R}(\mathbf{x}_k)[\mathbf{u}],\end{gathered} \tag{2}$$

where $D\mathbf{R}(\mathbf{x})[\mathbf{u}]$ indicates directional derivative of $\mathbf{R}$ at $\mathbf{x}$ in the direction of an increment $\mathbf{u}$ and $\mathbf{K}$ indicates the tangent matrix. In each iteration step k, a linear system of equations has to be solved. Nowadays, for up to about two million equations, direct methods may be still dominant for the solution, compared with iterative methods. As stated in [1], the advantage of direct solvers lies on their capability to solve even ill-conditioned and indefinite system of equations, which is especially interesting for nonlinear applications. Moreover, direct methods are generally robust.

The convergence behavior of Newton-Raphson method is advantageous. Generally only a few iterations are needed to obtain the solution. However, in every iteration step, triangularization of the tangent matrix $\mathbf{K}(\mathbf{x}_k)$ in (2) involves extensive computational effort for large finite element models.

To reduce time consumption of triangular factorization of **K**, the modified Newton method is proposed in which **K** is not changed in several successive iteration steps. Therefore the number of refactorizations of **K** decreases and total computing time might be saves. In practice, this saving may be counterbalanced by the fact that modified Newton method converges much slower than Newton-Raphson method, while repetitive residuum load vector building, forward reduction, and back substitution in each iteration step become then time consuming.

Fortunately, it can be noticed that between two adjacent iteration steps, nonlinear phenomenon usually appears *locally*. For example, stress concentration often occurs around one or several points; plastic deformation usually starts from local elements and then spreads around; in analysis of seepage, only elements around the interface of liquid need to change the coefficient of permeability, and so forth. All these features are equivalent to local change of tangent matrix **K**; that is to say, only a small or very small percentage (but the number might be large such as five thousand rows) of stiffness coefficients will be changed in each linearized step.

In light of this, a new algorithm is proposed in this study to refactorize tangent matrix with local change. Our goal is to save time of triangular factorization, to make Newton-Raphson method more efficient and thus to provide a new possibility for faster nonlinear analysis. Generally speaking, local change occurs frequently in material nonlinear problems and optimization problems; thus the proposed algorithm is mainly suitable for them. In the worst case, a global change causes that the proposed algorithm degrades to full factorization.

An outline of this paper follows. In Section 2, background on sparse direct solution techniques is given. In Section 3, we discuss the effect propagation of local change in the matrix factor and conclude that only part of matrix factor needs to be recalculated. Based on all preparations, a new algorithm for refactorization is described in Section 4. The cost of computation is predictably reduced, and the precision is unaffected. In Section 5, numerical examples are given to illustrate performance of the proposed algorithm. Finally in Section 6, it is concluded that the algorithm proposed in this paper is significantly efficient and suitable for local change of structures. This method can be applied to a wide range of engineering problems and could be the foundation of faster nonlinear finite element analysis.

2. Sparse Direct Solution Techniques

Since 1990s, direct solution techniques of finite element analysis [4] have developed from conventional variable bandwidth [5] or frontal solutions [6] to various sparse solutions [5, 7–11], such as backward-reference (left-looking), forward-reference (right-looking), and multifrontal [12, 13] methods. They have yielded a breakthrough in terms of solution speed and storage requirement in finite element analysis. It is quite safe to conclude that solvers based on various row pivoting sparse storage schemes are, in terms of the solution time, memory requirement and scale of problems, much more efficient than those based on variable bandwidth or skyline storage schemes. With the achievements in hardware, the analysis of 10,000 to 500,000 nodes finite element calculations on microcomputers becomes popular. Since 1995, commercial FEA packages have turned to sparse direct solvers and gained a speedup of more than 10 times. The scale of problems that can be solved on certain machines increased about 3 to 10 times.

Sparse solution has two essential features. One is sparse index storage scheme, also called compact storage scheme, storing nonzero entries of matrix in a row or column compact form, which significantly improves the efficiency of time and space because only nonzero entries are considered in triangular factorization. The other is fill-in's reducing, an optimization procedure before numerical factorization, which finds an optimized pivoting sequence by permuting rows and columns of matrix, so that fill-ins as well as float-point operations of factorization will be decreased greatly.

Generally, a stiffness matrix of finite element analysis can be considered as an adjacency list of graph, in which a vertex presents an equation and a nonzero off-diagonal entry implies that the two corresponding vertices are adjacent. At present, one of practical fill-reducing algorithms is the so-called multilevel graph partition [14–16], a special case of graph partition. It is a divide and conquer algorithm, which partitions global optimization cost functions into some unrelated cost functions of subproblems and additional cost function between these subproblems. Approximately, the global cost function is minimized if the additional cost function is minimized and the scales of subproblems are roughly equal. The division guarantees that local change on one side does not spread to the other side. In other words, the graph partition prevents local change from wide spreading.

Taking advantage of both row-sparse solution and fill-reducing, a new algorithm to refactorize the tangent matrix for local change is designed. Only a part of rows of the matrix factor involves recalculation, and therefore the computational cost decreases. It is to note that no approximation is assumed in the presented algorithm; therefore accuracy is guaranteed as the same of full refactorization.

For simplicity, we assume in following discussions that a pivoting sequence of fill-in's reducing of a positive definite global stiffness matrix has been already found through a graph partition algorithm, such as METIS [19].

3. Effect of Local Change in Sparse Solution

In each iteration step of Newton-Raphson method, nonlinear finite element analysis yields to a system of linear equations as (2) or closely

$$\mathbf{K}(\mathbf{x}_k)\,\mathbf{u} = -\mathbf{R}(\mathbf{x}_k), \tag{3}$$

where $\mathbf{K}(\mathbf{x}_k)$ denotes the tangent stiffness matrix, which is symmetric positive definite generally in finite element method, $-\mathbf{R}(\mathbf{x}_k)$ the residuum vector, and **u** the displacement increment vector.

At present, general solution procedure of finite element equation (3) is $\mathbf{LDL}^{\mathrm{T}}$, which decomposes **K** into product

```
do row $j = 1, n$
    do $k$ = all appropriate rows ($u_{kj} \neq 0$)
        RowTask ($k, j$)
    end do
    RowTask ($j, j$)
end do
```

Algorithm 1: Simple *jki* form of $\mathbf{LDL}^{\mathrm{T}}$ factorization [17, 18].

of lower triangular matrix $\mathbf{L}$, diagonal matrix $\mathbf{D}$, and upper triangular matrix $\mathbf{L}^{\mathrm{T}}$, as

$$\mathbf{K} = \mathbf{LU} = \mathbf{LDL}^{\mathrm{T}}. \quad (4)$$

With the result of $\mathbf{LDL}^{\mathrm{T}}$, the displacement $\mathbf{u}$ can be obtained easily by forward reduction and back substitution. In the process, factorization of stiffness matrix $\mathbf{K}$ is the most time consuming. In nonlinear analysis, repetitive factorizations are carried out at each iteration step. It is to point out that, in two adjacent iteration steps, the tangent stiffness matrices differ frequently only from each other on a small portion of elements. Besides, the difference is frequently local.

Suppose $\mathbf{K} = (k_{ij}) \in \mathbb{R}^{n\times n}$, the factors $\mathbf{L} = (l_{ij}) \in \mathbb{R}^{n\times n}$, and $\mathbf{D} = \mathrm{diag}(d_i) \in \mathbb{R}^{n\times n}$, which are a lower triangular matrix with unit diagonal and a diagonal matrix, respectively, where n is the number of equations. With regard to memory management, $\mathbf{U}$ shares the same memory locations as $\mathbf{L}^{\mathrm{T}}$ and $\mathbf{D}$ together, since the unit diagonal of $\mathbf{L}^{\mathrm{T}}$ does not necessarily occupy any storage space. Practically, we denote $\mathbf{K}$ before factorization and $\mathbf{U}/\mathbf{DL}^{\mathrm{T}}$ after factorization as $\mathbf{A} = (a_{ij}) \in \mathbb{R}^{n\times n}$. The bulk of the work in the $\mathbf{LDL}^{\mathrm{T}}$ factorization occurs in a triple nested loop around a single statement [20]:

$$a_{ji} = a_{ji} - l_{ik} l_{jk} d_{kk} = a_{ji} - l_{ik} u_{kj}. \quad (5)$$

There are six different forms [17] to implement $\mathbf{LDL}^{\mathrm{T}}$ factorization, which are six permutations of the indices i, j, k. Row-sparse factorization [21] and its improvement [8, 18] are foundations of the present sparse solution implementation, in which *jki* form and its variations are used. The simple *jki* form can be expressed in terms of tasks [18, 22]:

(1) RowTask (k, j): reduction of the target jth row by a multiple of the kth row, $k < j$, that is, elimination of a_{jk};

(2) RowTask (j, j): division of the off-diagonals of the target jth row by its diagonal.

The RowTask (k, j) itself involves an i-loop from j to n. We can symbolically write this procedure as shown in Algorithm 1.

As stiffness matrix is sparse, most entries in $\mathbf{K}$ as well as in $\mathbf{L}$ or $\mathbf{U}$ vanish, and therefore, only nonzero entries are actually involved in calculation. Besides nonzero entries in $\mathbf{K}$, the factor $\mathbf{L}$ or $\mathbf{U}$ contains additional nonzero entries generated by factorization. Thus, it is necessary to predetermine symbolically, not numerically, which entries will become nonzero in $\mathbf{L}$ or $\mathbf{U}$. This step is called symbolic analysis. After this, it can be analyzed by (5) how the variation of entries of stiffness matrix $\mathbf{K}$ will affect the factors $\mathbf{L}$ or $\mathbf{U}$ and $\mathbf{D}$ and how the effect will propagate in the $\mathbf{L}$ or $\mathbf{U}$.

Since the matrices differ on a small portion of elements, consider a change of row j of $\mathbf{U}$ which is caused by previous eliminations or entries change of $\mathbf{K}$ directly. It will cause change of row i that $u_{ji} \neq 0$ as shown in Algorithm 1. Take (6), for example. Change of row 1 generally causes change of row 3 and row 7, since change of u_{13} and u_{17} spreads to d_3 and d_7. Change of row 3 spreads changes to row 5 and row 7:

$$\begin{pmatrix} d_1 & & u_{13} & & & & u_{17} \\ & d_2 & u_{23} & & & & u_{27} \\ & & d_3 & & u_{35} & & u_{37} \\ & & & d_4 & & u_{46} & \\ & & & & d_5 & u_{56} & u_{57} \\ & & & & & d_6 & u_{67} \\ & & & & & & d_7 \end{pmatrix}. \quad (6)$$

In general, change of diagonal entry directly affects value of all nonzero entries of this row in the matrix, and therefore, for convenience, let a row be the smallest unit of change, corresponding to an equation or a general displacement of the structure. Through the previous analysis, the rule of direct effect of change is summarized as follows.

Rule. In $\mathbf{LDL}^{\mathrm{T}}$ factorization, change of a row in stiffness matrix $\mathbf{K}$ directly affects values of this row in factorization results $\mathbf{U}(\mathbf{L}^{\mathrm{T}})$ and $\mathbf{D}$, and then affects rows corresponding to the columns where nonzero entries of this row are.

Consider (6) again. The change of row 1 will affect rows 3 and 7. The effect will propagate in the matrix, as row 3 will affect rows 5 and 7, and row 5 will affect rows 6 and 7. Due to sparseness of stiffness matrix, the number of nonzero entries in each row is finite. As a result, change of one row is able to affect only part of the matrix. In this case, change of row 1 will eventually affect rows 3, 5, 6, and 7, whereas rows 2 and 4 remain unchanged.

In practice, fill-ins' reducing, which minimizes the number of fill-ins (additional nonzero entries produced by factorization) through equation reordering, is always required before numerical factorization. A typical optimized result delivered by the divide and conquer graph partition algorithm is as follows:

$$\mathbf{K} = \begin{pmatrix} \mathbf{K}_{11} & & \mathbf{K}_{13} & & & & \mathbf{K}_{17} \\ & \mathbf{K}_{22} & \mathbf{K}_{23} & & & & \mathbf{K}_{27} \\ \mathbf{K}_{31} & \mathbf{K}_{32} & \mathbf{K}_{33} & & & & \mathbf{K}_{37} \\ & & & \mathbf{K}_{44} & & \mathbf{K}_{46} & \mathbf{K}_{47} \\ & & & & \mathbf{K}_{55} & \mathbf{K}_{56} & \mathbf{K}_{57} \\ & & & \mathbf{K}_{64} & \mathbf{K}_{65} & \mathbf{K}_{66} & \mathbf{K}_{67} \\ \mathbf{K}_{71} & \mathbf{K}_{72} & \mathbf{K}_{73} & \mathbf{K}_{74} & \mathbf{K}_{75} & \mathbf{K}_{76} & \mathbf{K}_{77} \end{pmatrix}. \quad (7)$$

In this case, if stiffness of group 2 is changed, only groups 2, 3, and 7 are necessary to be recalculated; similarly, if stiffness of group 6 is changed, only groups 6 and 7 are affected.

Now it is concluded that if the stiffness matrix changes locally, based on fill-ins' reducing, only part of values in $\mathbf{L}$

```
Step 1. Input original stiffness matrix K, upper triangular matrix U and diagonal matrix D;
Step 2. Create array CHANGE(n) with the initial value zero, where n is the number of
equations. CHANGE(i) = 0 indicates row i is not changed, while CHANGE(i) = 1 indicates
row i is changed;
Step 3. Input the change of each element stiffness matrix and assemble them to original
total stiffness matrix. Then let CHANGE(i) be 1 if row i is changed;
Step 4. According to Rule, spread the effect of changes over the whole matrix and mark
all the changed rows:
Step 5. Assemble matrix to the original factor. If CHANGE(i) = 0, fill in row i with
original factorization result; if CHANGE(i) = 1, fill in row i with entries of new stiffness matrix;
Step 6. Numerically factorize the matrix, only recalculating entries in changed rows
(CHANGE(i) = 1) according to (5). Only changed row is added to the elimination tree, in
implementation being represented by the linked list.
Step 7. The solution procedure after factorization stays the same.
```

Algorithm 2: Sparse direct methods of refactorization.

```
do $j = 1, n$
    if (CHANGE($j$) = 1) then
        do $i = j + 1, n$
            if ($u_{ji}$ is nonzero) then CHANGE($i$) = 1
        end do
    end if
end do
```

Algorithm 3: Spread the effect of modification in *jki* factorization.

and **D** need to be recalculated. The cost of computation is predictably reduced, and the precision is unaffected.

4. Algorithm Design for Local Change of Sparse Matrix

Algorithm for refactorization involves sparse storage scheme, which takes extremely advantage of sparseness of numerical part. However, index can be compressed furthermore, and indirect addressing of data access can be reduced substantially [5]. Row-sparse factorization [21] and its improvement [18] are the foundation of present sparse solution, which consist of four steps: *symbolic assembly*, *symbolic factorization*, *numerical assembly*, and *numerical factorization*. Instead of the elimination tree, an updated linked list is used in factorization.

In this refactorization algorithm, structural topology is unchanged, so symbolic assembly and symbolic factorizationinherit the original results. Thus, three steps need to be redesigned.

(a) *Modification analysis*: determines which rows will be changed in upper triangular matrix **U**.

(b) *Numerical assembly*: reassembles entries of matrix in rows which are changed.

(c) *Numerical factorization*: modifies rows of upper triangular matrix **U** and recalculates the value of corresponding entries.

The detailed steps of this algorithm are shown as Algorithm 2.

For further illustration steps 4 and 6 are refined as follows in Algorithms 3 and 4.

In step 6 (Algorithm 4), only parts of factor are recalculated for elimination, and as a result, computation efficiency compared to full refactorization is improved.

5. Numerical Examples and Analysis

Two building structures are selected to show the efficiency of proposed algorithm, in size of 321,210 and 442,331 equations or DOFs (degrees of freedom), in which up to 5000 DOFs are changed. Both numerical examples are tested on Windows 7 system with Intel Xeon CPU E5-2620 (2.00 GHz, 2×6 cores, but only one core used) and 32 GB memory. The codes have not integrated in nonlinear solver, and therefore only refactorization part is presented. Moreover, only computational effort and elapsed time of factorization are discussed, since the algorithm proposed in this paper involves no approximation and the solution is as accurate as a full recalculation.

Example 1. A multitower building, as shown in Figure 1(a), is calculated. The number of DOFs or equations is 321,210; the number of nonzero entries in stiffness matrix is 17,169,579, and that in factor is 62,281,728.

Table 1: Statistic results of refactorization of Example 1.

Changed elements	Changed DOFs		Affected DOFs				Number of operations (MFLOP)				Factorization time (sec)			
Full refactorization	321210		321210				30630				10.71			
Partial refactorization	Average		Min	Max	Average		Min	Max	Average		Min	Max	Average	
1	8	0.003%	159	3219	2283	0.71%	72	4654	2894	9.45%	0.2	2.19	1.45	13.54%
2	13	0.004%	159	3936	2431	0.76%	72	4837	2930	9.57%	0.18	2.22	1.47	13.71%
5	26	0.01%	1977	4512	2834	0.88%	2524	4916	3252	10.62%	1.33	2.19	1.60	14.94%
10	51	0.02%	2193	5295	3317	1.03%	2524	4953	3416	11.15%	1.36	2.23	1.64	15.33%
20	95	0.03%	2247	5466	3526	1.10%	2524	4955	3426	11.18%	1.27	2.33	1.66	15.53%
50	227	0.07%	2487	10101	3975	1.24%	2528	6246	3507	11.45%	1.25	2.55	1.67	15.62%
100	358	0.11%	2541	10452	4124	1.28%	2528	6252	3511	11.46%	1.28	2.63	1.68	15.67%
200	533	0.17%	2622	10548	4560	1.42%	2528	7911	3677	12.00%	1.3	3.35	1.72	16.11%
500	1175	0.37%	3714	15978	7947	2.47%	3357	10655	6237	20.36%	1.6	3.88	2.62	24.44%
1000	2167	0.67%	4770	19347	13351	4.16%	3371	14167	9370	30.59%	1.56	5.43	3.69	34.43%
2000	4949	1.54%	12462	29079	17262	5.37%	5314	14005	10234	33.41%	2.12	5.34	4.06	37.94%

Table 2: Statistic results of refactorization of Example 2.

Changed elements	Changed DOFs		Affected DOFs				Number of operations (MFLOP)				Factorization time (sec)			
Full refactorization	442331		442331				141251				48.87			
Partial refactorization	Average		Min	Max	Average		Min	Max	Average		Min	Max	Average	
1	9	0.002%	1662	5472	4410	1.00%	7330	21280	16521	11.70%	3.54	8.76	6.34	12.96%
2	15	0.003%	1674	6018	4901	1.11%	7330	21515	17389	12.31%	3.17	8.75	6.82	13.96%
5	30	0.01%	1701	8760	5344	1.21%	7330	22401	17705	12.53%	3.01	8.78	6.71	13.72%
10	54	0.01%	1761	12972	5906	1.34%	7330	38960	18622	13.18%	3.05	14.95	7.08	14.48%
20	98	0.02%	1803	13734	6558	1.48%	7330	39431	19774	14.00%	3.09	12.89	7.32	14.98%
50	198	0.04%	1893	18411	7185	1.62%	7330	39616	20212	14.31%	3.42	14.3	7.61	15.57%
100	345	0.08%	2016	19326	7553	1.71%	7330	39810	20259	14.34%	3.55	13.84	7.56	15.48%
200	597	0.14%	2103	23763	8185	1.85%	7330	40123	20360	14.41%	3.07	13.57	7.65	15.65%
400	2472	0.56%	6525	25506	10995	2.49%	16097	40123	21653	15.33%	5.88	14.92	8.07	16.51%
1000	3288	0.74%	20766	42306	26598	6.01%	29494	42004	35659	25.25%	10.06	15.76	12.81	26.20%

```
do row $j = 1, n$
    if (CHANGE($j$) = 1) then
        do $k$ = all appropriate row
            RowTask ($k$, $j$)
        end do
        RowTask ($j$, $j$)
    end if
end do
```

Algorithm 4: Simple *jki* form of $\mathbf{LDL}^{\mathrm{T}}$ refactorization.

Consider that nonlinear phenomenon occurs between two adjacent iteration steps. For example, part of steel structures reaches plastic phase. For simplicity, in each test one group's element stiffness is changed to 90% of standard value, and one group here normally consists of a certain number (from 1 to 2000) of well-connected elements in one floor which means a local change.

The performance of the proposed algorithm is shown in Table 1. The first row of data shows the result of full refactorization, a violent method to factorize the new matrix, introduced as reference of comparison. The rest of the rows show the result of partial refactorization using the proposed algorithm. For each case (change of certain number of elements) about 30 samples are selected randomly. For statistical purpose, minimum, maximum, and average affected DOFs, (million floating-point operations), MFLOP and computing time are given. The percentage next to average values is the ratio between this item and corresponding value of full refactorization.

The numerical results indicate that the local change, up to 2% DOFs in this example, will only spread to a little bit large percentage of DOFs, up to 10% here. The elapsed time or computational effort is usually less than half of full recalculation, which demonstrates that this method is more efficient than full recalculation. The MFLOP ratio of this algorithm and full recalculation is always much larger than the ratio of affected DOFs and total DOFs, because reduction of some DOFs or equations refers a large number of DOFs, both changed and unchanged. Also, it is shown that the MFLOP ratio is coincident with elapsed time ratio.

Comparing the cases of different numbers of changed elements, it is concluded that computational effort of refactorization depends slightly on the number of modified/affected

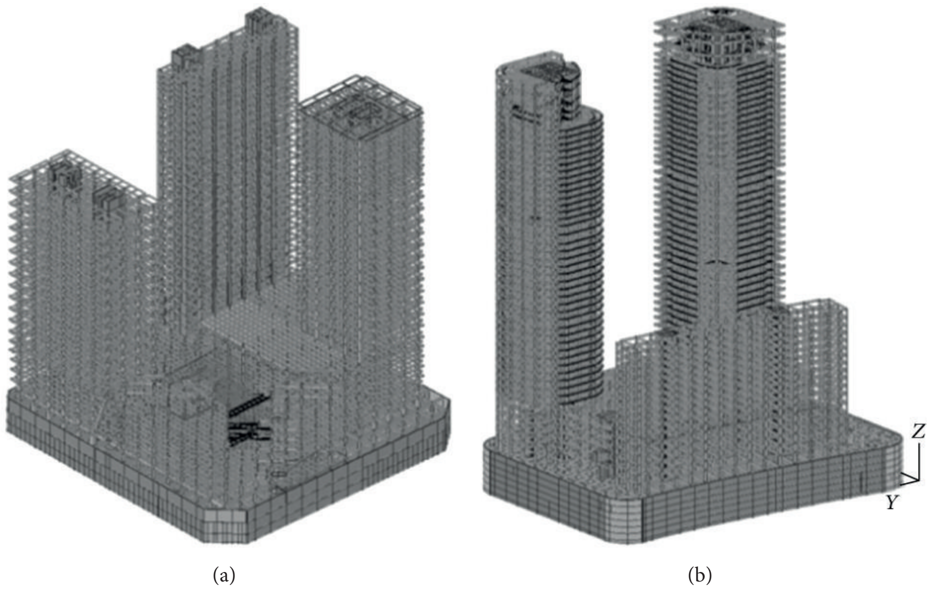

Figure 1: Schematic diagrams of two numerical examples.

DOFs and the relation is roughly monotonic but not necessarily linear. In this numerical example, changing less than 200 elements, the computational efforts of refactorization are almost the same, about 15%, and even changing 500 or more elements, the computational effort does not increase much. In fact, in the worst case, changing all DOFs, it degrades to full refactorization.

Example 2. A two-tower building, as shown in Figure 1(b), is calculated. The number of DOFs or equations is 442,331; the number of nonzero entries in stiffness matrix is 27,160,169 and that in factor is 152,414,429.

Like Example 1, this structure is changed by a certain number (from 1 to 1000) of well-connected elements. The results are shown in Table 2. For each case about 100 samples are selected randomly.

The results demonstrate again that the proposed algorithm is efficient and factorization time decreases. Furthermore, in each case of these two examples, maximum (affected DOFs, MFLOP, and computing time) is usually less than twice the average, which indicates that the algorithm is robust.

Summarizing the results of numerical tests yields *qualitatively* the relationship between computational effort ratio and ratio of changed DOFs in Figure 2. It shows clearly that the algorithm proposed in this paper is efficient most of the time.

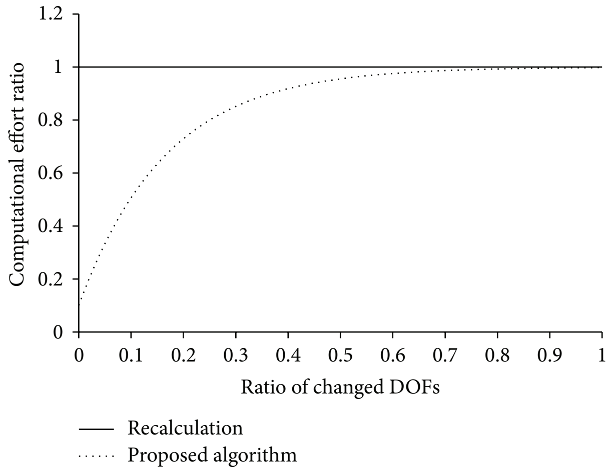

Figure 2: Schematic diagram of computational effort.

6. Conclusions

In this paper, a partial refactorization algorithm based on row-sparse solution and fill-in's reducing is proposed for nonlinear finite element analysis, especially material nonlinear problems and optimization problems. Instead of full refactorization in traditional nonlinear analysis, the proposed procedure finds the changed factor of the tangent matrix with much lower cost.

It is concluded that this algorithm can significantly improve the efficiency compared with full refactorization. There are several advantages as follows.

(a) The algorithm does not affect the precision of final results, since it involves no approximation, just skipping repetitive computation.

(b) A large number of elements or DOFs can be changed.

(c) The amplitude of change is not limited to being small, as usually being required by approximate approaches.

(d) Only one additional array CHANGE is required to perform the computation.

(e) The implementation is simple, if one understands row-sparse solution procedure.

We expect the proposed algorithm could be integrated in nonlinear analysis, especially Newton-Raphson method. It can significantly reduce the refactorization time and make Newton-Raphson method more efficient. In other words, it opens a new possibility for faster nonlinear analysis. The proposed algorithm can also be applied to many engineering problems, in which a series of linear systems of equations with step-by-step local change has to be solved, including structural optimization, progressive collapse analysis, and analysis of seepage.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (no. 10972005) and National Basic Research Program of China (2010CB731503). The authors thank Mr. XL Wang from YJK Building Software Co. who provided the data of examples.

References

[1] P. Wriggers, *Nonlinear Finite Element Methods*, Springer, Berlin, Germany, 2008.

[2] J. Bonet and R. D. Wood, *Nonlinear Continuum Mechanics for Finite Element Analysis*, Cambridge University Press, Cambridge, UK, 1997.

[3] L. Lam and D. G. Fredlund, "Saturated-unsaturated transient finite element seepage model for geotechnical engineering," in *Finite Elements in Water Resources*, pp. 113–122, Springer, Berlin, Germany, 1984.

[4] H. Zhou, S. Wu, and P. Chen, "Advances in direct solution technique of finite element analysis," *Advances in Mechanics*, vol. 37, no. 2, pp. 175–188, 2007 (Chinese).

[5] E. L. Wilson, K. Bathe, and W. P. Doherty, "Direct solution of large systems of linear equations," *Computers and Structures*, vol. 4, no. 2, pp. 363–372, 1974.

[6] B. M. Irons, "A frontal solution program for finite element analysis," *International Journal for Numerical Methods in Engineering*, vol. 2, pp. 5–32, 1970.

[7] P. Chen, D. Zheng, S. Sun, and M. Yuan, "High performance sparse static solver in finite element analyses with loop-unrolling," *Advances in Engineering Software*, vol. 34, no. 4, pp. 203–215, 2003.

[8] D. T. Nguyen, J. Qin, T. Y. Chang et al., "Efficient sparse equation solver with unrolling strategies for computational mechanics," in *Proceedings of the 4th International Conference on Concurrent Enterprising (ICES '97)*, pp. 676–681, San Jose, Costa Rica, August 1997.

[9] O. Schenk, K. Gärtner, W. Fichtner, and A. Stricker, "PARDISO: a high-performance serial and parallel sparse linear solver in semiconductor device simulation," *Future Generation Computer Systems*, vol. 18, no. 1, pp. 69–78, 2001.

[10] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "Efficienct management of parallelism in object oriented numerical software libraries," in *Modern Software Tools in Scientific Computing*, pp. 163–202, Birkhauser Press, Boston, Mass, USA, 1997.

[11] I. S. Duff and J. K. Reid, "MA47, a Fortran code for direct solution of indefinite sparse symmetric linear systems," Tech. Rep. RAL 95-001, Rutherford Appleton Laboratory, Oxford, UK, 1995.

[12] I. S. Duff and J. K. Reid, "The multifrontal solution of indefinite sparse symmetric linear equations," *ACM Transactions on Mathematical Software*, vol. 9, no. 3, pp. 302–325, 1983.

[13] J. W. H. Liu, "The multifrontal method for sparse matrix solution: theory and practice," *SIAM Review*, vol. 34, no. 1, pp. 82–109, 1992.

[14] B. Hendrickson and R. Leland, "A multilevel algorithm for partitioning graphs," Tech. Rep. SAND93-1301, Sandia National Laboratories, Albuquerque, NM, USA, 1993.

[15] A. Gupta, "Fast and effective algorithms for graph partitioning and sparse matrix ordering," Tech. Rep. RC, 20496 (90799), IBM T. J. Watson Research Center, Yorktown Heights, NY, USA, 1996.

[16] S. Wu, *Algebraic multigrid analysis and its application in finite element method [M.S. thesis]*, Peking University, Beijing, China, 2007, (Chinese).

[17] J. M. Ortega, "The *ijk* forms of factorization methods. I. Vector computers," *Parallel Computing*, vol. 7, no. 2, pp. 135–147, 1988.

[18] P. Chen and S. Sun, "A new high performance sparse static solver in finite element analysis with loop-unrolling," *Acta Mechanica Solida Sinica*, vol. 18, no. 3, pp. 248–255, 2005.

[19] G. Karypis and V. Kumar, *Metis: A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Ordering of Sparse Matrices Version 4. 0*, University of Minnesota, Department of Computer Science, Minneapolis, Minn, USA, 1998.

[20] G. H. Golub and C. F. V. Loan, *Matrix Computation*, The Johns Hopkins University Press, Baltimore, Md, USA, 3rd edition, 1996.

[21] S. Pissanetzky, *Sparse Matrix Technology*, Academic Press, London, UK, 1984.

[22] D. Zheng and T. Y. P. Chang, "Parallel cholesky method on MIMD with shared memory," *Computers and Structures*, vol. 56, no. 1, pp. 25–38, 1995.