Factored approximate inverse preconditioners with dynamic sparsity patterns [*]

Eun-Joo Lee and Jun Zhang[†]
E-mail: elee3@csr.uky.edu, jzhang@cs.uky.edu
Laboratory for High Performance Scientific Computing & Computer Simulation,
Department of Computer Science, University of Kentucky,
773 Anderson Hall, Lexington, KY 40506-0046, USA

October 9, 2007

Abstract

We propose two sparsity pattern selection algorithms for factored approximate inverse preconditioners on solving general sparse matrices. The sparsity pattern is adaptively updated in the construction phase by using combined information of the inverse and original triangular factors of the original matrix. In order to determine the sparsity pattern, our first algorithm uses the norm of the inverse factors multiplied by the largest absolute value of the original factors, and the second employs the norm of the inverse factors divided by the norm of the original factors. Experimental results show that these algorithms improve the accuracy and robustness of the preconditioners on solving general sparse matrices.

1 Introduction

Krylov subspace methods with some suitable preconditioners have focused attention on solving large sparse linear systems of the form:

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

where A is a matrix of order n [?, ?, ?, ?]. Of the many types of preconditioners, recently sparse approximate inverse (SAI) preconditioners [?, ?, ?, ?, ?, ?, ?], of which preconditioning process is just a (sparse) matrix-vector product operation, have become popular in solving many application problems due to their potential in parallel implementations. In addition to that, SAI preconditioners solve certain problems that are difficult to handle by the general purpose (conventional) incomplete LU (ILU) preconditioners [?]. Even though their computational efficiency and potential parallelism leads us to choose SAI preconditioners as an alternative to the conventional ILU preconditioners, the search for an optimal sparsity pattern to construct the SAI preconditioners should be taken with extreme care because the performance of SAI preconditioners depends on the sparsity pattern [?, ?].

In determining the sparsity pattern of SAI, there exists two main classes of methods called static and dynamic strategies. A static sparsity pattern strategy prescribes the sparsity pattern

[*]Technical Report No. 488-07, Department of Computer Science, University of Kentucky, Lexington, KY, 2007.

[†]This author's research work was supported in part by the U.S. National Science Foundation under grant CCF-0527967, in part by the National Institutes of Health under grant 1R01HL086644-01, in part by the Kentucky Science and Engineering Foundation under grant KSEF-148-502-06-186, and in part by the Alzheimer's Association under Grant NIGR-06-25460. URL: http://www.cs.uky.edu/∼jzhang.

before constructing a preconditioner, and the pattern remains unchanged until finishing the construction. Although this class of sparsity pattern selection strategy is usually efficient in terms of computational cost due to the use of a prescribed (fixed) pattern through the construction process, for general sparse matrices, the prescribed sparsity pattern is often inadequate for robustness [?]. On the contrary, a dynamic sparsity pattern strategy adjusts the pattern using some rules in the construction phase. Such a strategy usually computes more accurate and robust preconditioners than a static strategy [?]. Thus, a dynamic sparsity pattern strategy may be used to substitute for a static sparsity pattern strategy in solving difficult matrices.

In recent years, a few dynamic pattern strategies [?, ?] for SAI preconditioners have been developed. For example, Bollhöfer [?] showed that the norms of the inverse triangular factors have direct influence on the dropping strategy in computing a new ILU decomposition. Based on this insight, Bollhöfer [?] proposed an algorithm that manages the process of dropped entries with small absolute values by using the row norm of any row of the inverse factors, but the algorithm has a limitation on solving some ill-conditioned problems.

As a part of our continuous efforts in determining dynamic sparsity pattern, we introduce two enhanced algorithms, which extend the algorithm [?] mentioned above, using combined information of the norm of the inverse factors and either the largest absolute value of the original factors or the norm of the original factors. Here, a factored approximate inverse (FAPINV) [?], which is a sparse approximate inverse with a factored form, is utilized as an SAI.

The remainder of this paper is organized as follows. Section **??** describes the FAPINV algorithm used to characterize and justify our algorithms for dynamic sparsity pattern in Section **??**. In Section **??**, numerical results are presented to demonstrate the performance of the proposed algorithms over a static pattern based FAPINV. Concluding remarks are in Section **??**.

2 Factored sparse approximate inverses

In this section, we first review a general framework of a factored inverse and an incomplete factored inverse algorithms, and this framework will be utilized in Section **??** to justify our sparsity pattern algorithms.

Luo [?] proposed an algorithm of a factored inverse in the form,

$$A^{-1} = LDU, \tag{2.2}$$

for a nonsingular matrix A of order n, where $L = [L_1, L_2, \cdots, L_n]$ is a lower triangular matrix of order n, and L_i is a column vector of length n with $L_{i,i} = 1, i = 1, 2, \cdots, n$. Similarly, $U = [U_1, U_2, \cdots, U_n]^T$ is an upper triangular matrix with a row vector U_i and $U_{i,i} = 1, i = 1, 2, \cdots, n$, and $D = \text{diag}[D_{1,1}, D_{2,2}, \cdots, D_{n,n}]$ denotes a diagonal matrix. The factored inverse algorithm that computes the inverse matrix, A^{-1}, of Equation (**??**) is presented in Algorithm **??**.

Algorithm **2.1** Factored Inverse Algorithm for Dense Matrices [?]

```
1.  Do $j = n \to 1$ with step (-1)
2.    Do $i = j+1, n$
3.      $w_i^{(j)} = a_{j,i} + \sum_{k=i+1}^{n} a_{j,k} * L_{k,i}$
4.    End Do
5.    Do $i = j+1, n$
6.      $U_{j,i} = -w_i^{(j)} * D_{i,i} - \sum_{k=i+1}^{n} w_k^{(j)} * D_{k,k} * U_{k,i}$
7.    End Do
8.    $D_{j,j} = 1/(a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j})$
9.    Do $i = j+1, n$
```

```
10.       $z_i^{(j)} = a_{i,j} + \sum_{k=i+1}^{n} U_{i,k} * a_{k,j}$
11.    End Do
12.    Do $i = j+1, n$
13.       $L_{i,j} = -z_i^{(j)} * D_{i,i} - \sum_{k=i+1}^{n} z_k^{(j)} * D_{k,k} * L_{i,k}$
14.    End Do
15. End Do
```

Note that n refers to the order of the original matrix A, and a_{ij} denotes a nonzero element in row i and column j, where $i, j = 1, \cdots, n$. The upper inverse factor U and the lower inverse factor L are computed in lines 2–7 and lines 9–14, respectively, and the diagonal inverse is constructed in line 8. Although Algorithm **??** is written to compute the inverse of a dense matrix, it can be easily modified to compute the inverse of a sparse matrix [**?**].

For a sparse matrix, we can see that the L and U matrices computed by Algorithm **??** can be too dense, and the computational cost might be high [**?**]. Thus, an incomplete inverse method that drops elements with small absolute values of the inverse is usually employed to reduce the cost and maintain the sparsity of the inverse factors. For example, Zhang [**?**] introduced an incomplete factored sparse approximate (FAPINV) algorithm that applies a static dropping strategy in two parts (four places) of the computation process to improve Algorithm **??**. Specifically, for a given dropping tolerance $\tau > 0$, the conditions $|w_i^{(j)}| \leq \tau$ and $|z_i^{(j)}| \leq \tau$ determine the loop involving lines 2–4 and 9–11 to be skipped, respectively. After processing lines 5–7 and 12–14, $U_{j,i}$ and $L_{i,j}$ are dropped if their absolute values are smaller than or equal to τ.

According to Maijerink and van der Vorst [**?**], a conventional incomplete LU factorization can be executed in an exact factorization, and the computed pivots are strictly positive when A is a nonsingular M-matrix. This holds true in FAPINV associated with the inverse computed from Algorithm **??**, and in the following, Proposition **??** establishes a proof that is similar to Proposition 3.1 in [**?**] where no breakdown in the incomplete process occurs if A is an M-matrix.

Proposition 2.1 *Let A be an M-matrix. Then*

$$U_{j,i} \geq 0, \quad L_{i,j} \geq 0 \quad \textit{and} \quad D_{j,j} > 0 \tag{2.3}$$

for $1 \leq i, j \leq n$.

Proof. We will prove the inequalities in (**??**) by using induction. For $j = n$, the inequalities are obviously true. In fact,

$$L_{n,n} = U_{n,n} = 1 \text{ and } D_{n,n} = 1/a_{n,n} \geq {a_{n,n}}^{-1} > 0.$$

Now, the following inductive assumptions are considered for $j \leq n-1$ and $j+2 \leq i \leq n$.

$$U_{j+1,i} = -w_i^{(j+1)} * D_{i,i} - \sum_{k=j+2}^{n} w_k^{(j+1)} * D_{k,k} * U_{k,i} \geq 0, \tag{2.4}$$

$$L_{i,j+1} = -z_i^{(j+1)} * D_{i,i} - \sum_{k=j+2}^{n} z_k^{(j+1)} * D_{k,k} * L_{i,k} \geq 0, \quad \text{and} \tag{2.5}$$

$$D_{j+1,j+1} = 1/(a_{j+1,j+1} + \sum_{k=j+2}^{n} U_{j+1,k} * a_{k,j+1}) > 0. \tag{2.6}$$

From the inductive assumptions (??) and (??), we get $U_{i,k} \geq 0$ and $L_{k,i} \geq 0$ for $i+1 \leq k \leq n$. Also, $a_{j,i}$ and $a_{i,j}$ are nonpositive since A is an M-matrix. Hence, $w_i^{(j)}$ and $z_i^{(j)}$ can be expressed by

$$w_i^{(j)} = a_{j,i} + \sum_{k=i+1}^{n} a_{j,k} * L_{k,i} \leq 0 \quad \text{and} \quad z_i^{(j)} = a_{i,j} + \sum_{k=i+1}^{n} U_{i,k} * a_{k,j} \leq 0, \tag{2.7}$$

where $j+1 < i < n$. Using the updating formula given in Algorithm ?? for $U_{j,i}, L_{i,j}$, and $D_{j,j}$, we have

$$U_{j,i} = -w_i^{(j)} * D_{i,i} - \sum_{k=j+1}^{i-1} w_k^{(j)} * D_{k,k} * U_{k,i} \geq 0,$$

$$L_{i,j} = -z_i^{(j)} * D_{i,i} - \sum_{k=j+1}^{i-1} z_k^{(j)} * D_{k,k} * L_{k,i} \geq 0,$$

and

$$D_{j,j} = 1/(a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j}) > 0.$$

Therefore, no component of $U_{j,i}$ and $L_{i,j}$ can become negative, and $D_{j,j}$ remains positive. ■

Incompleteness arises from FAPINV due to the dropping of nonzero fill-ins and causes the algorithm to produce smaller or equal absolute entries than those of the inverse factorization from Algorithm ??. In accordance with such facts and the result of Proposition ??, we can see that FAPINV also generates a nonnegative factored approximate inverse when A is an M-matrix.

Proposition 2.2 *Let A be an M-matrix, and $A^{-1} = LDU$ be an inverse matrix of A obtained by Algorithm ??, where $L, D,$ and U are lower, diagonal, and upper inverse factors of A, respectively. If $G = \tilde{L}\tilde{D}\tilde{U}$ is a factored approximate inverse computed by FAPINV [?], where $\tilde{L}, \tilde{D},$ and $\tilde{U}$ are lower, diagonal, and upper inverse factors, respectively. Then*

$$D_A^{-1} \leq G \leq A^{-1}, \tag{2.8}$$

where D_A is the diagonal part of A.

Proof. Since the elements of $\tilde{L}$, and $\tilde{U}$ are obtained from L and U by dropping elements if the absolute value of the elements are smaller than a dropping tolerance τ, then

$$\tilde{L} \leq L, \quad \tilde{D} \leq D, \text{ and } \tilde{U} \leq U.$$

Hence, we can easily get $G \leq A^{-1}$.

From Proposition ??, we know that

$$D_{j,j} > 0, \quad a_{j,j} > 0, \quad U_{j,k} \geq 0, \text{ and } a_{k,j} \leq 0.$$

It follows that

$$0 < a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j} \leq a_{jj}.$$

Therefore,

$$D_{j,j} \geq 1/a_{j,j}.$$

■

As the results of Proposition ?? and Proposition ??, we have found that both the exact factored inverse computed by Algorithm ?? and FAPINV [?] construct nonnegative inverses when A is an M-matrix.

3 Dynamic sparsity pattern based factored approximate inverse preconditioners

We now introduce two algorithms that determine dynamic sparsity patterns for FAPINV preconditioners on solving general sparse matrices. In determining the sparsity pattern, we exploit information of the inverse and original factors by the following two supporting reasons that (1) the entries of FAPINV are computed by the original and previously computed inverse triangular factors [?], and (2) the norm of the inverse factor is strongly related with the dropping tolerance of FAPINV [?]. From that point of view, (1) our first algorithm, Norm-Largest-Dynamic sparsity pattern in Algorithm **??**, computes FAPINV with the dynamic sparsity pattern using the norm of the inverse factors multiplied by the largest absolute value of the original factors, and (2) the second, Norm-Norm-Dynamic sparsity pattern Algorithm **??**, employs the norm of the inverse factors divided by the norm of the original factors.

ALGORITHM **3.1** FAPINV WITH NORM-LARGEST-DYNAMIC SPARSITY PATTERN

```
1.  Find the largest absolute values, $Large_L$ and $Large_U$, of the lower and upper parts of $A$
2.  **Do** $j = n, 1$ **with step (-1)**
3.      $\zeta_U(j) = \zeta_L(j) = 0$
4.      **Do** $i = j + 1, n$
5.          Compute $U_{j,i}$ by using FAPINV [?]
6.          **If** $(|U_{j,i}| > \zeta_U(j))$**, then** $\zeta_U(j) = |U_{j,i}|$
7.      **End Do**
8.      $\eta_U = \zeta_U(j) * Large_U$
9.      **If** $(\eta_U > 1)$**, then** $\tau = \tau/\eta_U$
10.     **Do** $i = j + 1, n$
11.         **If** $(|U_{j,i}| < \tau)$**, then** $U_{j,i} = 0$
12.     **End Do**
13.     $D_{j,j} = 1/(a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j})$
14.     **Do** $i = j + 1, n$
15.         Compute $L_{i,j}$ by using FAPINV [?]
16.         **If** $(|L_{i,j}| > \zeta_L(j))$**, then** $\zeta_L(j) = |L_{i,j}|$
17.     **End Do**
18.     $\eta_L = \zeta_L(j) * Large_L$
19.     **If** $(\eta_L > 1)$**, then** $\tau = \tau/\eta_L$
20.     **Do** $i = j + 1, n$
21.         **If** $(|L_{i,j}| < \tau)$**, then** $L_{i,j} = 0$
22.     **End Do**
23. **End Do**
```

Note that $Large_L$ and $Large_U$ refer to the largest absolute values of the lower and upper triangular factors of the original matrix, respectively. The $\zeta_U(j)$ and $\zeta_L(j)$ in lines 6 and 15 denote the largest absolute values of the columns U_j^T and L_j, respectively. The upper inverse factor U and the lower inverse factor L are computed in lines 4–12 and lines 14–22, respectively, and the diagonal inverse D is constructed in line 13. In lines 5 and 15, $U_{j,i}$ and $L_{i,j}$ can be obtained by using the FAPINV algorithm [?]. In lines 9 and 19, the dropping tolerance τ is determined by η_U and η_L, and the value of τ is updated for each j. Finally, in lines 10–12 and 20–22, if the absolute value of an element is smaller than the tolerance τ, the element is then dropped.

ALGORITHM **3.2** FAPINV WITH NORM-NORM-DYNAMIC SPARSITY PATTERN

```
1.  **Do** $i = 1, n$
2.     $U_\eta(i)$ = the largest absolute value in the row $i$ of the upper triangular factor of $A$.
3.     $L_\eta(i)$ = the largest absolute value in the row $i$ of the lower triangular factor of $A$.
4.  **End Do**
5.  **Do** $j = n, 1$ **with step (-1)**
6.     $\zeta_U(j) = \zeta_L(j) = 0$
7.     **Do** $i = j+1, n$
8.        Compute $U_{j,i}$ by using FAPINV [?]
9.        **If** $(|U_{j,i}| > \zeta_U(j))$**, then** $\zeta_U(j) = |U_{j,i}|$
10.    **End Do**
11.    $\eta_U = \zeta_U(j)/U_\eta(j)$
12.    **If** $(\eta_U > 1)$**, then** $\tau = \tau/\eta_U$
13.    **Do** $i = j+1, n$
14.       **If** $(U_{j,i} < \tau)$**, then** $U_{j,i} = 0$
15.    **End Do**
16.    $D_{j,j} = 1/(a_{j,j} + \sum_{k=j+1}^{n} U_{j,k} * a_{k,j})$
17.    **Do** $i = j+1, n$
18.       Compute $L_{i,j}$ by using FAPINV [?]
19.       **If** $(|L_{i,j}| > \zeta_L(j))$**, then** $\zeta_L(j) = |L_{i,j}|$
20.    **End Do**
21.    $\eta_L = \zeta_L(j)/L_\eta(j)$
22.    **If** $(\eta_L > 1)$**, then** $\tau = \tau/\eta_L$
23.    **Do** $i = j+1, n$
24.       **If** $(L_{i,j} < \tau)$**, then** $L_{i,j} = 0$
25.    **End Do**
26. **End Do**
```

As similar in Algorithm **??**, the upper inverse factor U and the lower inverse factor L are computed in lines 7–15 and lines 17–25, respectively, and the diagonal inverse D is constructed in line 16.

It should be noted that the computational procedures of our algorithms are different from that of Bollhöfer's [?], in mainly that we utilize the original triangular factors in determining the sparsity pattern. Further, we measured the performance between our algorithms and Bollhöfer's by presenting numerical comparisons in Section **??**.

Let G_1 and G_2 be two FAPINV preconditioners of a matrix A. If a given dropping tolerance of G_1 is greater than or equal to that of G_2, then

$$L_{1i,j} \le L_{2i,j}, \quad U_{1j,i} \le U_{2j,i}, \quad \text{and} \quad D_{1j,j} \le D_{2j,j},$$

where $G_1 = L_1 D_1 U_1$ and $G_2 = L_2 D_2 U_2$. Also, from Proposition **??**, we know that G_1 and G_2 are nonnegative matrices if A is an M-matrix. Thus, the following proposition is straightforward.

Proposition 3.1 *Let A be an M-matrix and $A^{-1} = LDU$ be the inverse matrix of A. If $G_1 = L_1 D_1 U_1$ is obtained by FAPINV [?] with a static dropping tolerance τ_1, and $G_2 = L_2 D_2 U_2$ is from Algorithm* **??** *or Algorithm* **??** *with a dynamic dropping tolerance τ_2. Then*

$$D_A^{-1} \le G_1 \le G_2 \le A^{-1}, \tag{3.9}$$

where D_A is the diagonal part of A.

Proof. Because of $\tau_1 \ge \tau_2$ and by Proposition **??**,

$$0 \le L_{1i,j} \le L_{2i,j}, \quad 0 \le U_{1j,i} \le U_{2j,i}, \quad \text{and} \quad 0 < D_{1j,j} \le D_{2j,j}.$$

It follows that $G_1 \leq G_2$. ■

As we can see in Proposition ??, the accuracy of FAPINV depends on the value of the dropping tolerance τ. This implies that, with the same amount of storage space, the FAPINV preconditioners with the dynamic sparsity patterns become more accurate than the FAPINV with a static sparsity pattern.

4 Numerical Experiments

We present numerical experiments of FAPINV with the proposed selection strategies of dynamic sparsity patterns on solving a few general sparse matrices. The descriptions of the test matrices are given in Table ??. The matrices[1] were solved as they were, that is, no scalings or permutations were applied.

Matrix	Description	*n*	*nnz*	*nnzdiag*	*condition*
CAVITY03	Driven Cavity Problems	317	7311	243	3.30E+06
CAVITY04	Driven Cavity Problems	317	5923	243	1.40E+07
CAVITY06	Driven Cavity Problems	1182	32747	883	N/A
CAVITY08	Driven Cavity Problems	1182	32747	883	N/A
CAVITY11	Driven Cavity Problems	2597	76367	1923	N/A
CAVITY13	Driven Cavity Problems	2597	76367	1923	N/A
CAVITY15	Driven Cavity Problems	2597	76367	1923	N/A
E05R0300	Driven cavity driven cavity, 5x5 elements, Re= 300	236	5856	162	1.30E+06
E05R0400	Driven cavity driven cavity, 5x5 elements, Re= 400	236	5846	162	2.40E+06
E05R0500	Driven cavity driven cavity, 30x30 elements, Re=500	9661	306002	6962	1.31E+11
FIDAP006	Matrices generated by the FIDAP Package	1651	49479	1180	3.45E+21
FIDAP020	Matrices generated by the FIDAP Package	2203	69579	1603	5.89E+08
FIDAP021	Matrices generated by the FIDAP Package	656	18962	476	9.10E+08
FIDAP025	Matrices generated by the FIDAP Package	848	24261	608	7.90E+07
FIDAP026	Matrices generated by the FIDAP Package	2163	93749	1706	4.66E+18
FIDAPM02	Matrices generated by the FIDAP Package	537	19145	441	1.40E+05
LNS 131	Fluid flow modeling	131	536	112	1.50E+15
NNC261	Nuclear reactor models	261	1500	150	1.20E+15
NNC666	Nuclear reactor models	666	4032	410	1.80E+11

Table 1: Description of the test matrices ; *n*, *nnz*, *nnzdiag*, and *condition* denotes the order, the number of nonzero entries, the number of nonzero entries on the main diagonal, and the condition number of a matrix, respectively. Under "N/A", we report that the condition number was not available from Matrix Market [?].

The FAPINV (for factored approximate inverse) [?] preconditioner was used as an approximate inverse right preconditioner in the experiments. The preconditioned iterative solver employed was GMRES(50). For all linear systems, the right-hand side was generated by assuming that the solution is a vector of all ones. The initial guess was a zero vector. The iteration was terminated when the l_2-norm of the initial residual was reduced by at least eight orders of magnitude, or when the number of iterations reached 500. The programs of our approaches were coded in standard Fortran 77 programming language in double precision with 64-bit arithmetic, and the computations were carried out on a Sun-Blade-100 workstation with a 500 MHz UltraSPARC IIi CPU and 1 GB of RAM.

In all tables with numerical results, STATIC denotes an FAPINV [?] preconditioner with a static sparsity pattern. NLD and NND represent FAPINV preconditioners with Norm-Largest-Dynamic and Norm-Norm-Dynamic sparsity patterns described in Algorithm ?? and Algorithm ??,

[1]All of these matrices are available on-line from the Matrix Market of the National Institute of Standards and Technology at http://math.mist.gov/matrixMarket.

respectively; The "iter" refers to the number of GMRES iterations, the "time" represents the CPU time in seconds for computing the preconditioner and for the solution phase, and "spar" denotes the sparsity ratio that is the ratio of the number of nonzero elements of the preconditioner to that of the original matrix; The dropping tolerance "τ" is utilized as both a dropping tolerance of STATIC and initial dropping tolerances of NLD and NND; The value "-1" indicates the failure of convergence within the maximum number of allowed iterations (500).

4.1 Comparison of preconditioners with static and dynamic pattern

Table ?? shows the comparisons between a static and dynamic sparsity pattern for the FAPINV preconditioners with two different settings of the dropping tolerance, $\tau = 0.1$ and $\tau = 0.01$.

	$\tau = 0.1$			$\tau = 0.01$					
	STATIC	NLD	NND	STATIC		NLD		NND	
Matrix	iter	iter	iter	iter	spar	iter	spar	iter	spar
CAVITY03	-1	37	41	-1	8.32	19	9.16	36	8.95
CAVITY04	-1	25	22	-1	6.19	10	7.35	23	8.35
CAVITY06	-1	-1	22	-1	7.07	-1	15.55	15	30.96
CAVITY08	-1	-1	40	-1	18.02	46	25.71	22	31.48
CAVITY11	-1	-1	31	-1	9.10	-1	25.49	22	65.57
CAVITY13	-1	-1	58	-1	21.57	-1	43.55	33	65.63
CAVITY15	-1	-1	92	-1	44.40	-1	52.39	40	67.95
E05R0300	-1	-1	33	197	5.26	28	7.19	20	8.22
E05R0400	-1	-1	46	-1	6.37	18	8.13	23	8.56
E05R0500	-1	100	47	-1	6.98	15	8.72	23	9.03
FIDAP006	-1	94	295	-1	21.72	9	35.71	37	43.52
FIDAP020	-1	-1	14	-1	18.08	-1	27.85	7	62.47
FIDAP021	-1	-1	27	-1	8.31	-1	13.57	19	20.28
FIDAP025	-1	-1	7	-1	16.21	-1	19.83	4	28.82
FIDAP026	-1	452	220	-1	9.033	191	21.83	124	34.28
FIDAPM02	-1	-1	36	-1	6.91	42	10.77	14	13.13
LNS 131	-1	-1	5	-1	2.65	3	5.12	4	9.89
NNC261	-1	15	36	-1	21.23	28	28.52	30	27.06
NNC666	-1	17	44	-1	46.16	17	60.53	28	57.29

Table 2: Comparisons of the number of preconditioned GMRES iterations with different sparsity pattern based FAPINV preconditioners.

As seen in the table, STATIC failed in solving any of the test matrices with the two values of τ except E05R0300 that was solved in 197 iterations when $\tau = 0.01$, whereas NLD and NND solved the matrix in 28 and 20 iterations, respectively. NND solved all the test matrices, while NLD solved 7 and 12 of the test matrices when $\tau = 0.1$ and $\tau = 0.01$, respectively.

In terms of space complexity, NLD and NND demanded more memory space than STATIC on solving the test matrices, but it would seem relatively minor compared to the number of iterations with convergence. For example, NND and NLD solved the CAVITY03, CAVITY04, E05R0500, and NNC666 matrices in 36, 10, 15, and 28 iterations with 1.075, 1.187, 1.249, and 1.241 times more memory storages than STATIC with no convergence, respectively.

4.2 Comparison with strategies of dynamic sparsity pattern

We compare our algorithms, NLD and NND, with the one by Bollhöfer [?] in Table ??. Under

	$\tau = 0.01$								
	NND			Bollhöfer's			NLD		
Matrix	iter	time	spar	iter	time	spar	iter	time	spar
CAVITY03	36	0.53	8.95	47	0.57	8.63	19	0.45	9.16
CAVITY04	23	0.43	8.35	28	0.38	7.24	10	0.32	7.35
CAVITY06	15	16.85	30.96	-1	N/A	14.72	-1	N/A	15.55
CAVITY08	22	17.79	31.48	355	28.73	21.89	46	15.09	25.71
CAVITY11	22	137.27	65.57	-1	N/A	21.21	-1	N/A	25.49
CAVITY13	33	141.67	65.63	-1	N/A	35.02	-1	N/A	43.55
CAVITY15	40	149.86	67.95	-1	N/A	44.92	-1	N/A	52.39
E05R0300	20	0.28	8.22	33	0.26	6.85	28	0.26	7.19
E05R0400	23	0.30	8.56	42	0.33	7.63	18	0.27	8.13
E05R0500	23	0.32	9.03	42	0.35	8.36	15	0.28	8.72
FIDAP006	37	24.17	43.52	46	25.87	43.68	9	15.55	35.71
FIDAP020	7	56.33	62.47	-1	N/A	28.95	-1	N/A	27.85
FIDAP021	19	3.14	20.28	-1	N/A	13.75	-1	N/A	13.57
FIDAP025	4	5.44	28.82	404	19.71	21.76	-1	N/A	19.83
FIDAP026	124	89.99	34.28	-1	N/A	17.84	191	67.39	21.83
FIDAPM02	14	1.73	13.13	46	1.98	11.05	42	1.85	10.77
LNS 131	4	0.01	9.89	-1	N/A	4.91	3	0.01	5.12
NNC261	30	0.15	27.06	27	0.14	27.03	28	0.16	28.52
NNC666	28	1.17	57.29	18	0.99	57.56	17	1.05	60.53

Table 3: Comparisons with different dynamic sparsity pattern strategies.

"N/A," we report that the value is not available due to no convergence, and "Bollhöfer's" refers to the preconditioner with a dynamic sparsity pattern proposed in [?].

Based on the numerical results in Table ??, we reported the comparisons in three aspects of accuracy, efficiency, and memory storage. In terms of accuracy, NND performed better than Bollhöfer's in most cases, and NLD was comparable to Bollhöfer's. For example, NND solved all of the test matrices, while Bollhöfer's and NLD solved 11 and 12 of the test matrices, respectively. In addition, the number of iterations of NND is much smaller than that of Bollhöfer's and NLD. Secondly, NND generally costs less CPU time in solving the test matrices than Bollhöfer's. Finally, as we can see from the results, NLD and NND might need more memory storage than Bollhöfer's, but NND still better performed than Bollhöfer's because of its convergence rate. For example, NLD solved the CAVITY08 matrix in 46 iterations with 1.174 times more memory storage than Bollhöfer's in 355 iterations.

5 Concluding remarks

We have proposed two algorithms that determine dynamic sparsity patterns for the FAPINV preconditioners on solving general sparse matrices. In the computation phase, the dropping tolerance has been adaptively determined by the norm of the inverse factors and either the norm of the original factors or the largest value of the original factors. Numerical experiments showed that FAPINV with the proposed dynamic sparsity pattern generates more accurate and robust preconditioner than FAPINV with not only a static sparsity pattern but also other dynamic sparsity pattern (Bollhöfer's) preconditioners do.

References

[1] M. Benzi, C.D. Meyer, and M. Tuma, *A sparse approximate inverse preconditioner for the conjugate gradient method*, SIAM J. Sci. Comput., **17-5** (1996), 1135–1149.

[2] M. Benzi and M. Tuma, *A sparse approximate inverse preconditioner for nonsymmetric linear systems*, SIAM J. Sci. Comput., **19-3** (1998), 968–994.

[3] M. Benzi and D. Bertaccini, *Approximate inverse preconditioning for shifted linear systems*, BIT Numerical Mathematics, **43** (2003), 231–244.

[4] M. Bollhöfer, *A robust ILU with pivoting based on monitoring the growth of the inverse factors*, Linear Algebra and its Applications, **338** (2001), 201–218.

[5] M. Bollhöfer, *A robust and efficient ILU that incorporates the growth of the inverse triangular factors*, SIAM J. Sci. Comput., **25-1** (2003), 86–103.

[6] T.F. Chan, W.P. Tang, and W.L. Wan, *Wavelet sparse approximate inverse preconditioners*, BIT, **37-3** (1997), 644–660.

[7] T.F. Chan, and H.A. van der Vorst, *Approximate and incomplete factorizations*, Parallel Numerical Algorithms, ICASE/LaRC Interdisciplinary Series in Science and Engineering **4** (1997), 91–118. (http://www.math.uu.nl/people/vorst/publ.html#publ94)

[8] E. Chow and Y. Saad, *Approximate inverse preconditioners for general sparse matrices*, Technical Report UMSI **94/101** (1994), Minnesota Supercomputer Institute, University of Minnesota, Minneapolis, MN.

[9] E. Chow and Y. Saad, *Experimental study of ILU preconditioners for indefinite matrices*, J. Comput. Appl. Math., **86** (1997), 387–414.

[10] E. Chow and Y. Saad, *Approximate inverse preconditioners via sparse-sparse iterations*, SIAM J. Sci. Comput., **19** (1998), 995–1023.

[11] N.I.M. Gould and J.A. Scott, *Sparse approximate-inverse preconditioners using norm-minimization techniques*, SIAM J. Sci. Comput., **19-2** (1998), 605–625.

[12] G.A. Gravvanis, *An approximate inverse matrix technique for arrowhead matrices*, Intern. J. Computer Math., **70** (1998), 35–45.

[13] M.J. Grote and T. Huckle, *Parallel preconditioning with sparse approximate inverses*, SIAM J. Sci. Comput., **18** (1997), 838–853.

[14] J.-G. Luo, *An incomplete inverse as a preconditioner for the conjugate gradient method*, Computer Math. Applic., **25-2** (1993), 73–79.

[15] The Matrix Market of the National Institute of Standards and Technology, *http://math.mist.gov/matrixMarket.*

[16] J.A. Meijerink and H.A. Van Der Vorst, *An iterative solution method for linear systems of which the coefficient matrix is asymmetric M-matrix*, Math. Comp., **31** (1977), 148–162.

[17] Y. Saad, *Iterative methods for sparse linear systems*, PWS, New York, 1996.

[18] K. Wang and J. Zhang, *MSP: A class of parallel multistep successive sparse approximate inverse preconditioning strategies*, SIAM J. Sci. Comput., **24-4** (2003), 1141–1156.

[19] K. Wang, S. Kim, and J. Zhang, *A comparative study on dynamic and static sparsity patterns in parallel sparse approximate inverse preconditioning*, Journal of Mathematical Modeling and Algorithms, **3-2** (2003), 203–215.

[20] J. Zhang, *A sparse approximate inverse technique for parallel preconditioning of general sparse matrices*, Appl. Math. Comput., **130-1** (2002), 63–85.