Incomplete Cholesky Factorization with Sparsity Pattern Modification

Xiaoge Wang
Department of Computer Science
Indiana University - Bloomington

Kyle Gallivan
Department of Electrical and Computer Engineering
University of Illinois- Urbana

Randall Bramley
Department of Computer Science
Indiana University - Bloomington *

December 21, 1993

Abstract

This paper proposes, analyzes, and numerically tests methods to assure the existence of incomplete Cholesky (IC) factorization preconditioners, based solely on the target sparsity pattern for the triangular factor R. If the sparsity pattern has a simple property (called property C+), then the IC factor exists in exact arithmetic. Two algorithms for modifying the target sparsity pattern to have property C+ are proposed, one based on adding elements into the set of retained elements and the other based on dropping elements. Tests show that the modifications do ensure the numerical existence of the IC factor, and the resulting preconditioners are effective in accelerating the conjugate gradient iteration method.

1 Introduction

The incomplete Cholesky (IC) factorization is one of the most important and commonly used preconditioners for iterative methods of solving large sparse symmetric positive definite linear systems [4]. Its major weakness is a lack of robustness, by which we mean the failure of

*Work supported by NSF grants CDA-9309746 and CCR-9120105

the factorization due to nonpositive pivots. To overcome this difficulty, several modification methods have been suggested [6][5][9]. Those modification methods use information about the values of dropped elements and alter the factorization process to assure the existence of the triangular factor R. In contrast to these *numerical* strategies, a completely different approach is proposed here, which uses only information about the nonzero structure of the matrix to be factored and the structure of the target sparsity pattern for the IC factor. This *structural* strategy was first discovered in a study of the relationship between incomplete modified Gram-Schmidt (IMGS) factorization of a matrix A and incomplete Cholesky (IC) factorization of A^TA. With a certain condition imposed on the sparsity pattern P, IC on A^TA produces the same factor R as applying IMGS on A. Since the IMGS factorization always exists when A is full rank, this guarantees the robustness of IC factorization (proofs of these results can be found in [8, 7]). In this paper we present this strategy independent from the IMGS algorithm and instead base it on an understanding of the relationship between complete and incomplete Cholesky factorization.

In the next section, we define the requisite sparsity pattern characteristics **Property C** and **Property C+**. When the set P of retained elements has properties C or C+, IC applied to a symmetric positive definite matrix will always exist in exact arithmetic. This theoretical result has an important application: it leads to methods of modifying the sparsity pattern for IC to guarantee the existence of the factorization. The modification can be done during the data structure allocation phase of the factorization. This strategy is especially attractive for applications which need to solve multiple linear systems with the same nonzero structure but with different numerical values. It also provides a new direction of research for generally improving the quality of incomplete factorization preconditioning. Combining both numerical and structural information to compute higher quality preconditioners is a promising topic for further research.

Two algorithms for modifying a given position set P to have property C+ are discussed in this paper: **MPADD** and **MPDROP**. **MPADD** modifies the pattern by adding positions into P while **MPDROP** does so by eliminating positions from P. Numerical test results on these modification methods are presented, comparing them in terms of storage, computation cost, and quality of the resulting preconditioner. The effect of matrix orderings on the method **MPADD** is also discussed. The numerical results show that the pattern modification approach does greatly improve the robustness of IC factorization, and produces factors which effectively accelerate the conjugate gradient (CG) algorithm.

2 Property C and C+

We begin by stating the definition of property C. Let $P_n = \{(i,j)|1 \leq i \leq j \leq n\}$ be the set of all the positions in a $n \times n$ upper triangular matrix. Let $P \subseteq P_n$ be the target set of non-zero positions for the triangular IC factor R of a symmetric positive definite matrix A. In order for the IC factorization to complete the positions of the diagonal elements must be in P.

Definition 2.1 *A position set $P \subset P_n$ is said to have* **property C** *if for any $(j,k) \in P$, either of the following is true:*

1. *For any $1 \leq i < j$, $(i,j) \in P$ and $(i,k) \in P$;*
2. *For any $1 \leq i < j$, $(i,j) \notin P$ and $(i,k) \notin P$.*

A trivial example of a position set with property C is the position set of a dense block diagonal matrix. A small nontrivial example is shown in Figure 1.

A set of columns $i_1 < i_2 < ... < i_k$ are said to have the same structure if for any $j < i_1$, positions (j, i_t) where $1 < t < k$ are all in P, or none are in P. If set P has the property C, then we have the following:

1. If $(i,j) \in P$, then columns i and j have the same strucuture.
2. If $(i,j) \in P$ and $(i,k) \in P$, then columns i, j, and k have the same structrue.
3. If $(i,j) \in P$ and $(k,j) \in P$, then columns i, j and k have the same structure.
4. Any subset of P corresponding to a principal submatrix of A also has property C.

$$\begin{pmatrix} \times & \times & \times & \times & \times & \times & \times & \times \\ & \times & & \times & \times & & \times & \\ & & \times & & & \times & & \times \\ & & & \times & \times & & \times & \\ & & & & \times & & \times & \\ & & & & & \times & & \times \\ & & & & & & \times & \\ & & & & & & & \times \end{pmatrix}$$

Figure 1: Position Set P with Property C

Next we show that if the set specifying the sparsity pattern of R has property C, the incomplete Cholesky factor of a symmetric positive definite matrix exists. The proof is based on property C and the relationship between the incomplete Cholesky factorization of a matrix and the Cholesky factorization of a principal submatrix.

Theorem 1 *Let the matrix $A \in \Re^{n \times n}$ be symmetric positive definite and $P \subset P_n$ be a position set with property C. Then the incomplete Cholesky factorization of A using position set P completes.*

Proof: For any $1 \leq i \leq n$ and for $j \leq i$, let $S_i(j) = \{k|(k,i) \in P \quad \text{and} \quad k \leq j\}$ be the set of row indices of positions in column i which have row index less than or equal to j. Let $P_i = \{(j,k) \in P_n | j \in S_i(i) \text{ and } k \in S_i(i)\}$.

By property C, we know that $S_j(j) = S_i(j)$ for all $j \in S_i(i)$ and $P_i \subseteq P$. Now consider an element (k,j) where $j \in S_i(i)$ and $k \notin S_i(i)$. Property C implies that $(k,j) \notin P$, that is, the element indexed by (k,j) is dropped. This means that all of the retained positions in the principal submatrix defined by P_i are in P and so are kept during the incomplete factorization. Therefore, the computation of the incomplete Cholesky factorization of the matrix A with the pattern P on positions in P_i is the same as the operation of (complete) Cholesky factorization on the principal submatrix of A defined by P_i. Since any principal submatrix of a symmetric positive definite matrix is also symmetric positive definite, the Cholesky factor of this principal submatrix exists. So the computation of the incomplete Cholesky factor cannot break down at any step i, allowing completion of the whole process of incomplete Cholesky factorization. □

Since the purpose of this study is to provide an understanding of IC factorization, we are only interested in positions that will affect the computation of the IC factor. For this purpose property C is too restrictive; numerical testing indicates that it tends to create dense submatrices. We now define a new property which includes the notion of restricting our attention to only the non-zeros defined by the complete Cholesky factor for A.

For a given symmetric positive definite matrix A, by symbolic analysis we can find the non–zero positions of the upper triangular factor U, denoted as $P^+(A)$, such that $A = U^T U$. We define the new property using those positions:

Definition 2.2 *The position set $P \subset P^+(A) \subseteq P_n$ has* **property C+** *if for any position $(j,k) \in P$ and for all pairs of positions in columns j and k, $[(i,j),(i,k)]$, such that $i < j$, $(i,j) \in P^+(A)$, and $(i,k) \in P^+(A)$, one of the following is true:*

1. *$(i,j) \in P$ and $(i,k) \in P$;*
2. *$(i,j) \notin P$ and $(i,k) \notin P$;*

Note that property C+ is less restrictive than property C, since it only requres that either of the two listed in the definition hold for pairs of the elements that are both in $P^+(A)$.

For a given symmetic matrix A and a set of positions P, a set of columns $i_1 < i_2 < ... < i_k$ of P are said to have the same structure on $P^+(A)$ if for those rows j where all (k,i_l), $1 \leq l \leq k$, are in $P^+(A)$, all (j,i_l) are in P or all (j,i_l) are not in P. Let $P \subset P^+(A)$ be a set of position with property C+. Then we have:

1. If $(i,j) \in P$, then columns i and j have the same structure on $P^+(A)$.
2. If $(i,j) \in P$ and $(i,k) \in P$, then columns i, j and k have the same structure on $P^+(A)$.
3. If $(i,j) \in P$ and $(k,j) \in P$, then columns i, j and k have the same structure on $P^+(A)$.
4. Let A_1 be a principle submatrix of A and let P_1 be the subset of P consisting of positions that are in $P^+(A_1)$. Then P_1 has property C+.

The next result shows that property C+ also guarantees the existence of the IC factor.

Theorem 2 *Let the matrix $A \in \Re^{n \times n}$ be symmetric positive definite, and suppose that position set $P \subset P_n$ has property C+. The incomplete Cholesky factorization of A using position set P completes successfully.*

Proof: We use the same technique as the proof of Theorem 1, creating a set of principal submatrices for which the pattern guarantees that the portion of the IC factorization on its elements is equivalent to a complete Cholesky on the submatrix. For any $1 \leq i \leq n$, let $V_i = \{j | (j,i) \in P\}$ be the set of row indices of elements of P in column i. Construct a set S_i in the following way: Let $S_i = V_i$, and then repeat the following computation until it converges: $S_i = S_i \bigcup_{j \in S_i} V_j$. This construction assures that for any $j \in S_i$, $V_j \subset S_i$. Let P_i be the set of positions (j,k) such that j and k are both in S_i.

We now show that for any $(j,k) \in P_i$, if $(j,k) \in P^+(A)$ then $(j,k) \in P$. Assume that $(j,k) \notin P$. Then there is $(j,l) \in P$ where $l \in S_i$; otherwise j would not be in S_i. Now consider the position $(\min\{l,k\}, \max\{l,k\})$. Since $\{j,k,l\} \subset S_i$, by the definition of P_i we have $(\min\{l,k\}, \max\{l,k\}) \in P_i$. Since both (j,l) and (j,k) are in $P^+(A)$ it follows that $(\min\{l,k\}, \max\{l,k\}) \in P^+(A)$. Also because (j,l) is in P but (j,k) is not in P, and P has property C+, we have $(\min\{l,k\}, \max\{l,k\}) \notin P$. Now we have a position $(\min\{l,k\}, \max\{l,k\})$ which has the same status as (j,k): in P_i, in $P^+(A)$, but not in P. Note that $\min\{l,k\} > j$ and $\max\{l,k\} \geq k$. Apply this argument repeatedly, at each step we either get $(\min\{l,k\}, \max\{l,k\}) \in P$ and therefore prove by contradiction the assertion, or we repeat. Eventually we reach a row in P_i that has only one off-diagonal element. This last element must be in P or the position would not be in P_i which again proves the assertion by contradiction. So the assumption that $(j,k) \notin P$ is not true and therefore $(j,k) \in P$.

On the other hand, any position (j,k) where $j \notin S_i$ and $k \in S_i$ will not affect the computation of IC on position set P_i. Those $(j,k) \notin P^+(A)$ do not affect the computation: Let $(j,k) \in P^+(A)$. If $(j,k) \in P$ then j will be in S_i by construction and this contradicts the assumption that $j \notin S_i$ and hence $(j,k) \notin P$. Therefore, (j,k) is either not in P or not in $P^+(A)$. Thus (j,k) will not affect the value of the elements in the position P_i during the computation of the incomplete Cholesky factorization.

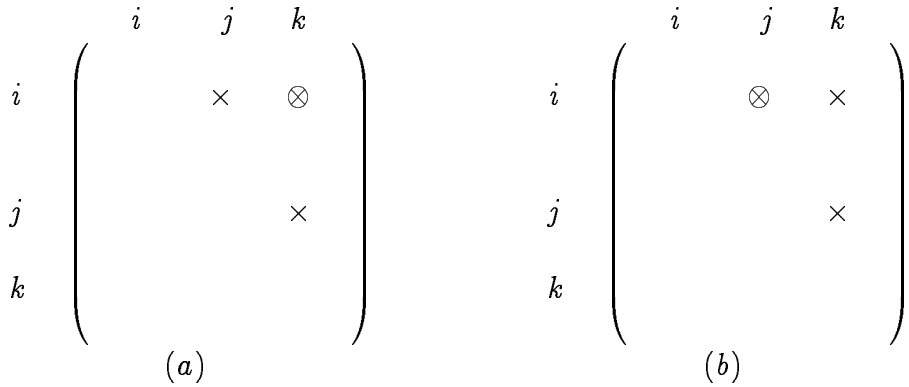
Therefore, the operations of the incomplete Cholesky factorization on A at positions in P_i are equivalent to the operations of Cholesky factorization on the principal submatrix of A defined by P_i. Again because any principal submatrix of a symmetric positive definite matrix is symmetric positive definite, Cholesky factorization of the principal submatrix of A defined by P_i can be completed, and hence the incomplete Cholesky factorization of A with pattern P exists. □

This establishes a sufficient condition on P for the existence of an incomplete Cholesky factorization. In the next section, we discuss algorithms that modify a given position set P so that the modified position set will have property C+. By using property C+ rather than the earlier property C we are able to work within the position set of the complete Cholesky factor rather than P_n – the position set of a dense upper triangular matrix

3 Algorithms for modifying the sparsity pattern for IC

Now we must design algorithms that modify the pattern P to make it have property C+. Considering property C+ more closely, we can see that it basically assures that if $(i,j) \in P$, then columns i and j have the same sparsity structure in those rows $1 \leq k < i$ where both (k,i) and (k,j) are in $P^+(A)$. If there are any patterns of the types shown in Figure 2 in P, property C+ is violated.

There are many ways to modify a position set so that it will have property C+, and we develop two basic methods: adding elements into P or removing elements from P, as shown in Figure 3.



$\times$ denotes the position that in P.

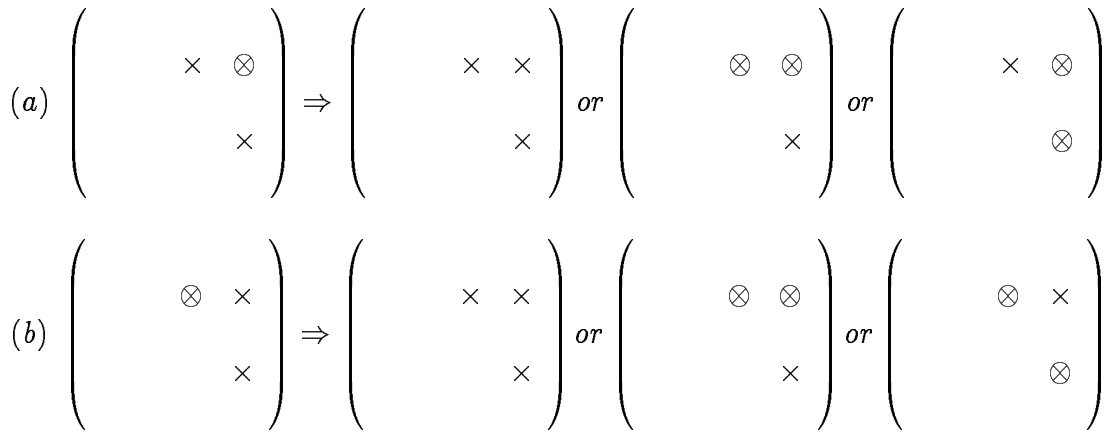
$\otimes$ denotes the position that in Cholesky factor R but not in P.

Figure 2: Patterns that do not have the property C+

Since the relation "has the same sparsity structure" is an equivalence relation, it induces a partitioning of the rows and columns $\{1, 2, \ldots, n\}$. A simple way to proceed is to divide up the rows and columns, and then make all the ones in the same partition have the same structure. The details are described in the following lemma:

Lemma 1 *Let $A \in \Re^{n \times n}$ and $P \subset P_n$ be a non-zero position set. Partition the set $\{1, 2, ..., n\}$ into subsets such that if $(i,j) \in P$ then i and j are in the same subset S_l. Define $\bar{P}$ in the following way. For any $(i,j) \in P^+(A)$, if $j \in S_k$ and $i \in S_k$, then $(i,j) \in \bar{P}$. Then $P \subset \bar{P}$, and $\bar{P}$ has property C+.*

Proof: Let $(i,j) \in P$; then i and j are in the same subset S_k. By the definition, $(i,j) \in \bar{P}$. For any $(i,j) \in \bar{P}$ and any $k < i$ such that (k,i) and (k,j) are in $P^+(A)$, if $(k,i) \in \bar{P}$ then we have k, i and j are in the same set S_l. By the definition of $\bar{P}$, $(k,j) \in \bar{P}$. Similarly we can show that if $(k,j) \in \bar{P}$ then we have $(k,i) \in \bar{P}$. Therefore $\bar{P}$ has property C+. $\square$



$\times$ denotes the position in P.
$\otimes$ denotes the position in Cholesky factor R but not in P.

Figure 3: Modifications of the patterns

Figure 4 shows an example of A, P and $\bar{P}$ as defined in Lemma 1, where A is a full matrix.

This method of modification is simple and easy to understand and implement. Its drawback is that it may add elements into the position set which are not necessary for property C+ to hold. This could unnecessarily increase the density of the preconditioner, which in turn may increase computation and storage requirements without improving the quality of the preconditioner.

Figure 5 shows an example of a situation where positions are added into P unnecessarily when using the method of Lemma 1. For the given initial position set P, the partition of $S = \{1, 2, ..., n\}$ is itself. This makes the modified position set $\bar{P} = P_n$, but there are several positions shown in (b) which are not necessary to assure property C+.

The reason that above algorithm can add too many elements for the modification can be clarified further. When using the equivalence relation, we ignore the fact that the columns in the same partition may not necessarily have the same structure over all the rows. For example, if there are $(i, j) \in P$ and $(i, k) \in P$, then i, j and k have the same structure on $P^+(A)$ only for rows less than i. There is no restriction on columns j and k to be the same in rows greater than i. The algorithm fails to identify this type of situation, and other methods are needed which can overcome this drawback.

To avoid these situations, consider more closely the relationship between columns that are in the same subset of the partition. When $(i, j) \in P$, two facts must hold: first, columns i and j share in part the same structure, and second the part is from 1 to i, while the remainder of column j is unrelated to column i. Using the partition set of Lemma 1 to store the information needed for the modification ignores the second fact. For the example in Figure 5, $(5, 6)$ in P says that columns 5 and 6 have the same structure from row 1 to

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} \times & & \times & & & & \\ & \times & & & \times & & \\ & & \times & \times & & & \\ & & & \times & & \times & \\ & & & & \times & & \times \\ & & & & & \times & \\ & & & & & & \times \end{pmatrix} \end{array} \qquad \bar{P} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} \times & & \times & \times & & \times & \\ & \times & & & \times & & \times \\ & & \times & \times & & \times & \\ & & & \times & & \times & \\ & & & & \times & & \times \\ & & & & & \times & \\ & & & & & & \times \end{pmatrix} \end{array}$$

$\times$ denotes the position that in the set. $S_1 = \{1, 3, 4, 6\}$, $S_2 = \{2, 5, 7\}$.

Figure 4: Example of P and $\bar{P}$

$$P = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} \times & & & \times & & & \times \\ & \times & \times & \times & & & \\ & & \times & & & & \\ & & & \times & & & \\ & & & & \times & \times & \times \\ & & & & & \times & \\ & & & & & & \times \end{pmatrix} \end{array} \qquad MPADD(P) = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \\ 7 \end{array} \begin{array}{c} \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \begin{pmatrix} \times & \star & \star & \times & \star & \star & \times \\ & \times & \times & \times & \otimes & \otimes & \otimes \\ & & \times & \otimes & \otimes & \otimes & \otimes \\ & & & \times & \otimes & \otimes & \otimes \\ & & & & \times & \times & \times \\ & & & & & \times & \otimes \\ & & & & & & \times \end{pmatrix} \end{array}$$

$\times$ denotes the original position that in P.
$\otimes$ and $\star$ denote the positions that added into P.
$\otimes$ denotes the nunecessary position that added into P.

Figure 5: Positions Added Unnecessarily for Property C+

row 5. $(5,7)$ in P says that column 5 and 7 have the same structure from row 1 to row 5. So columns 6 and 7 need have the same structure only from row 1 to row 5, and $(6,7)$ need not be added into P for property C+. Similar reasoning holds for the positions marked by $\otimes$ in Figure 5 (b). $\{2,3,4\}$ and $\{5,6,7\}$ are combined into one subset because of the elements $(1,4)$ and $(1,7)$, but these elements only restrict columns 2–7 to have the same structure in row 1. Simply putting the sets $\{2,3,4\}$ and $\{5,6,7\}$ together will only keep the information of 'the same structure' and lose the information of 'at row 1'. This is the reason why positions (2,5), (2,6), (2,7), (3,5), (3,6), (3,7), (4,5), (4,6), (4,7) are added into $\bar{P}$, even though they are not necessary for property C+ to hold.

As a remedy, trees can be used instead of simple partitions to design a pattern modification algorithm. Here we introduce a special tree called C-tree for this purpose. Given a position set P, there is a tree T that is generated from P by the algorithm **CTREE** shown below. We call this tree T a C-tree of P, and let $tree(i)$ be the subtree of T rooted at i.

Algorithm [T] = CTREE[P]
for $k = n, n-1, \ldots, 1,$
  form a tree with one node k
  for $j = k+1, k+2, \ldots, n$
    if $(k,j) \in P$ *and* $j \in tree(l), l \neq k$ **then**
      connect (k,l) *so that* $tree(l)$ *is a subtree of* $tree(k)$
    endif
  endfor
endfor

For the position set P of the last example, **CTREE(P)** generates a tree in $n = 7$ steps as shown in Figure 6. This tree gives complete information about the relationships between columns. Properties (1) and (2) in the next Lemma follow directly from the algorithm of C-tree.

Lemma 2 *Let $A \in \Re^{n \times n}$ be a symmetric positive definite matrix, $P \subset P^+(A)$ be a position set, and T be the C-tree of P. Let $path(i)$ denote the set the nodes on the path from node i to the root of the tree that node i is in. Then T has the following properties:*

(1) If $j \in tree(i)$ then $j > i$.

(2) If $(i,j) \in P$ then $j \in tree(i)$.

A C-tree provides more information than the partition set does. For the example P in Figure 5, **CTREE(P)** is shown in Figure 6. In general, after constructing the C-tree positions added to guarantee property C+ can be determined as a set of rows over which columns must possess the same structure. This set is given by the the index associated with the nodes on the path from the root to the common ancestor of the column pair that is being considered. For example, columns 6 and 7 are brought into the same tree by node 5 so they have to have the same structure from row 1 to row 5. Columns 2 and 6 are brought

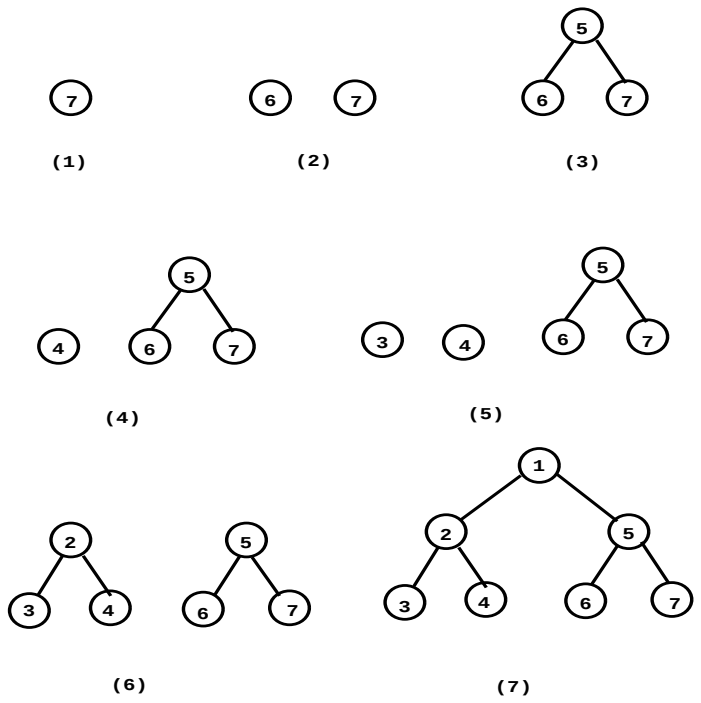


Figure 6: Example of TREE(P)

into the same tree by node 1 so they have to have the same structure at row 1 only. Using this information, we can make modifications that add fewer elements into the given original pattern P.

A position set modification algorithm **MPADD**, based on the C-tree generated by **CTREE(P)**, is now given.

Algorithm $[\bar{P}]$ = **MPADD[P, A]**
1. generate T by **CTREE(P)**.
2. generate $\bar{P}$ in the following way:
for $k = 1, 2, \ldots, n$,
  for $j = k+1, k+2, \ldots, n$,
    if $(k,j) \in P^+(A)$ *and* $j \in tree(k)$ **then**
      (k,j) *in* $\bar{P}$
    endif
  endfor
endfor

Theorem 3 *Let $A \in \Re^{n \times n}$ be a symmetric matrix, $P \subset P_n$ be a non-zero position set, and $\bar{P}$ be the position set generated by algorithm* **MPADD** *above. Then $\bar{P} \supseteq P$, and has property C+.*

Proof: Let $(i,j) \in P$. by Lemma 2, $j \in tree(i)$. Algorithm **MPADD** assures that $(i,j) \in \bar{P}$, so $\bar{P} \supseteq P$.

Next we show that $\bar{P}$ has property C+. Let $(i,j) \in \bar{P}$. For any $k < i < j$, if (k,i) and (k,j) are in $P^+(A)$ and $(k,i) \in \bar{P}$, we will show $(k,j) \in \bar{P}$. Because $(k,i) \in \bar{P}$ and $(i,j) \in \bar{P}$, from algorithm **MPADD**, we have $i \in tree(k)$ and $j \in tree(i)$. Therefore $j \in tree(k)$. By the algorithm **MPADD**, $(k,j) \in \bar{P}$. Similarly we can show that for any $k < i < j$, if (k,i) and (k,j) are in $P^+(A)$ and $(k,j) \in \bar{P}$, then $(k,i) \in \bar{P}$. Thus $\bar{P}$ has property C+. □

Algorithm **MPADD** can modify the position set so that IC is able to complete without breaking down. But experiments have shown that sometimes the resulting factor is too dense for practical use. Next, we present an algorithm based not on adding elements into P, but on eliminating some elements from P. Let $S_P(i,j) = \{k|(k,j) \in P \text{ and } k < i\}$. $S_P(i,j)$ is the set of row indices of positions in P in column j that are smaller than i.

Algorithm [P] = MPDROP[A, P]
for $k = 1, 2, \ldots, n$,
  for $j = k+1, k+2, \ldots, n$
    if $(k,j) \in P$ *and* $S_P(k,k) \neq S_P(k,j)$ *on* $P^+(A)$ **then**
      remove (k,j) *from* P.
    endif
  endfor
endfor

Theorem 4 *Let $P \subset P_n$ be a non-zero position set, and let $\bar{P}$ be the position set generated by algorithm* **MPDROP** *above. Then $\bar{P}$ has property C+.*

Proof: From the description of the algorithm we see that for every $(i,j) \in \bar{P}$, $S_P(i,i) = S_P(i,j)$ on $P^+(A)$. This means that for any $k < i$, if $k \in S(i,i)$, that is $(k,i) \in \bar{P}$, we have also $k \in S_P(i,j)$, that is, $(k,j) \in \bar{P}$. By the definition of property C+, $\bar{P}$ has property C+. □

The preceding two algorithms target arbitrary sparse symmetric positive definite systems. When additional structure is present in the matrix, the pattern modification algorithms can be specialized to take advantage of this structure. As an example, a symmetric matrix is called a bordered block diagonal matrix if it has the following structure.

$$\begin{pmatrix} B_1 & B_2 & B_3 & \cdots & B_k \\ B_2^T & D_2 & & & \\ B_3^T & & D_3 & & \\ \vdots & & & \ddots & \\ B_n^T & & & & D_k \end{pmatrix}$$

where the B_i are called border blocks and the D_i are called diagonal blocks. A matrix is said to be a nested bordered block diagonal matrix if the diagonal blocks are also bordered block diagonal matrices.

It is well known that if a position set P is the position set of a nested bordered block diagonal matrix, many of the factorization and iterative computations can be done in parallel. An important result here is that this structure is preserved by the modification algorithms proposed above.

Theorem 5 *Let $A \in \Re^{n\times n}$ be a symmetric matrix and $P \subset P^+(A)$ be a position set which has nested bordered block diagonal structure. Let $\bar{P} = \mathbf{MPADD}(P, A)$ and $\tilde{P} = \mathbf{MPDROP}(P, A)$. Then $\bar{P}$ and $\tilde{P}$ are of the same structure as P.*

Proof: Because $\tilde{P} \subset P$, it is obvious that **MPDROP** preserves bordered block diagonal structure.

Now we will show that any position (i, j) that is outside a border block B_l or a diagonal block D_l, $1 \leq l \leq k$, will not be added into $\bar{P}$ by **MPADD**.

Let (i, j) be any position that is outside the border block B_l or the diagonal block D_l, $1 \leq l \leq k$. Because of the scan order used and connection rules used in the algorithm **CTREE**, there is no path from i to j, or from j to i. So in the algorithm **MPADD**, (i, j) will not be added into $\bar{P}$. □

This result has an interesting application in parallel processing. By first ordering the matrix so that the given position set P is of the bordered block diagnal strucuture, then the modification algorithms have parallel implementations. For **MPADD**, in creating the C-tree, all the subtrees corresponding to diagonal blocks as well as all the subtrees that correspond to border blocks can be generated in parallel. When generating the position set, we can first determine the positions in the border blocks and then the positions in all the diagonal blocks in parallel. For **MPDROP**, the parallelism is more obvious; we process all the diagonal blocks and their corresponding border blocks in parallel. In this paper we will not go further into details of parallel pattern modification, but it is a current area of research.

4 Numerical experiments

In this section, we numerically test the algorithms **MPADD** and **MPDROP**. Details of the testing results can be found in the Appendix.

The test problems are from the Harwell/Boeing collection RSA of symmetric matrices. We use only those matrices which are positive definite and neither diagonal nor fully dense, leaving 35 matrices. The orders of the test matrices vary from $n = 100$ to $n = 3948$, and the densities vary from 0.0021 to 0.2094. The distribution of the size and densities of the test matrices are shown in Figure 7. The distribution of the average number of elements per row of test matrices is also shown in Figure 7. The algorithms were implemented in standard FORTRAN77 and all experiments were performed on an IBM RS6000 model 520H with 64 Mbytes of memory, running version 3.2 of the AIX operating system. CG was the basic iterative method that the preconditioners are applied to. The iteration was stopped when either the residual norm of the preconditioned system is smaller than 10^{-6}, or the number of iterations reaches a maximum allowed number of iterations, which is set $3n$ for CG without preconditioning and n for CG with preconditioning. CG without preconditioning is allowed more iterations because it requires fewer operations on each iteration than CG with preconditioning.

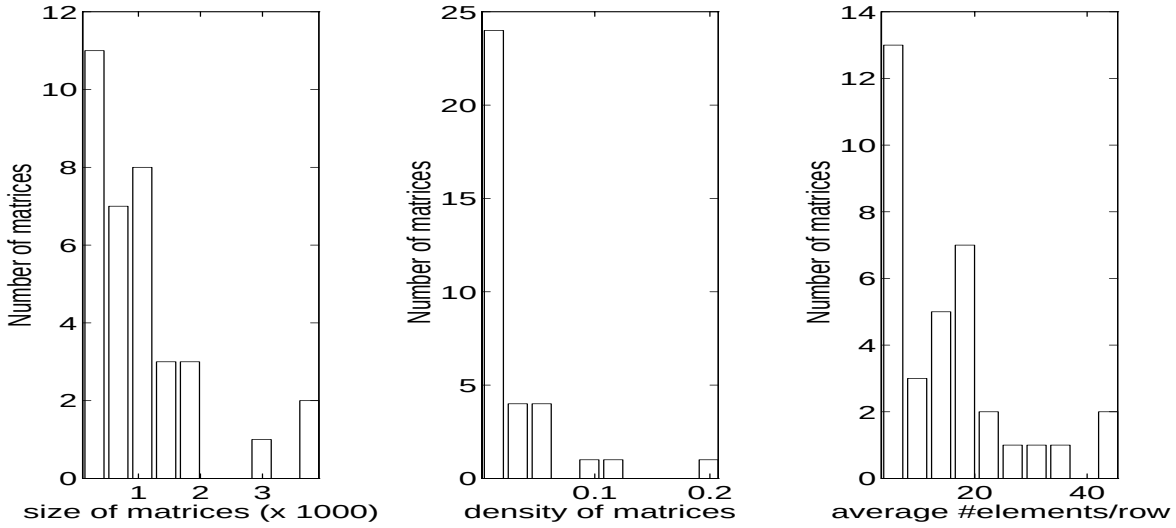


Figure 7: Distribution of sizes, densities, and average number of elements per row for the test matrices.

Our experiments also tested the effect of matrix reorderings on the quality of preconditioning in terms of storage and time. We tested 4 orderings: minimum degree MD, nested dissection ND, and reverse nested dissection RND. We used the SPARSPAK-A package [1] for the reorderings. These orderings can be effective in reducing storage for Cholesky factorization and so also reduce the amount of computation. Testing and analysis by other researchers [2, 3] has shown that orderings which favor parallelism may degrade the quality of IC preconditioner by causing an increase in the number of iterations. Our testing in part is to see if such an ordering degrades the quality of the preconditioner even when the pattern is modified. Reverse nested dissection is studied here because it gives matrices with a nested bordered block diagonal structure, which **MPADD** and **MPDROP** preserve and which provides parallelism both in the factorization and preconditioning phases.

In general, unpreconditioned CG does not work well for this set of test problems. CG fails to converge within $3n$ iterations for 20 out of the 35 test problems, so proper preconditioning is necessary for the test suite.

Before comparing the performance of the pattern modification algorithms, we first verify the theoretical statements that the sparsity pattern modifications do assure existence of the IC factor. Our experiments show that without pattern modification, IC with the original nonzero pattern of matrix A fails to complete the factorization on 17 problems, without matrix ordering. The situation changes slightly with reordering, with 13 failures using RND. But with pattern modification, IC can complete for all the problems. Table 1 shows the number of failures for all combinations of methods and orderings. Note that a problem is counted as a failure if it fails to exist during the factorization phase, can not complete the factorization, or fails in the iterative phase, meaning that the maximum number of iterations

Method:	CG	IC	MPADD	MPDROP
None	20	17	0	2
MD	20	16	1	3
ND	20	14	3	5
RND	20	13	3	5

Table 1: Number of failures.

is reached. The failures counted in **MPADD** and **MPDROP** are all due to a failure of convergence. From the table we can also see that the quality of the MPADD and MPDROP preconditioners are degraded by the orderings. This is consistent with experiments on IC without pattern modification, and suggests that the modified pattern preconditioners also degrade in quality with parallel orderings. On the other hand, the number of failures when reordering for IC without pattern modification decreases. Since the failures in this column are due to a break down of IC factorization, this suggests that reordering can increase the robustness of IC slightly. In addition, we note that pattern modification with nested dissection and reversed nested dissection orderings have similar behavior, with both orderings having the same number of failures. Reverse nested dissection has one less failure than the nested dissection ordering in standard IC preconditioning. The test suggests that reversing the ordering does not reduce the robustness of the preconditioner and may even improve it, and so using the reverse order to allow parallel processing does not cause a loss in effectiveness of the preconditioner.

As discussed in previous sections, pattern modification changes the storage requirements of preconditioners, with **MPADD** requiring more storage while **MPDROP** needs less storage than standard IC. Figures 8 show the ratio of storage required by standard IC over IC with **MPADD**, the ratio of the storage required by IC with **MPDROP** over standard IC and the ratio of storage required by IC with **MPADD** over sparse (complete) Cholesky factorization. These ratios (instead of their inverses) were chosen so that they would lie between 0 and 1. Because the storage requirements can be computed solely from the sparsity pattern of the matrices, those Figures include cases for which the IC factorization fails to exist, with the number of such failures indicated in each group by solid bars. The storage required by **MPADD** is close to that needed by complete Cholesky factorization for most of test problems. This can be seen from the large number of ratios in Figure 8 that are close to 1. There are 24 out of 35 problems with this ratio between 0.9 to 1. Clearly, this can cause difficulty for this modification method if it is used without safeguards.

We can also see that the storage for **MPDROP** can be much less than that for standard IC. This explains why there are more convergence failures using **MPDROP** than when using **MPADD**. One way of reducing the number of added elements for **MPADD** is to adapt the methods for reducing fill-in for complete Cholesky factorization by reordering the matrix. Figures 9, 10 and 11 show the storage ratio under different reorderings. From

these figures we can see that by reordering, the ratio of the storage required by standard IC over IC with **MPADD** and the ratio of the storage required by IC with **MPDROP** over standard IC increases while the ratio of storage required by IC with **MPADD** over sparse (complete) Cholesky factorization decreases. This means that by reordering, we can greatly reduce the amount of additional storage required by **MPADD** relative to that needed by complete Cholesky factorization. This also reduces the amount of computation needed in the incomplete factorization. Comparing Figure 10 and Figure 11 we find that reverse nested dissection ordering reduces the storage requirements even further than nested dissection does. This is not suprising because we have shown that positions outside the bordered block diagonal area will not be added, whereas sparse complete Cholesky factorization can create fill-in outside that area.

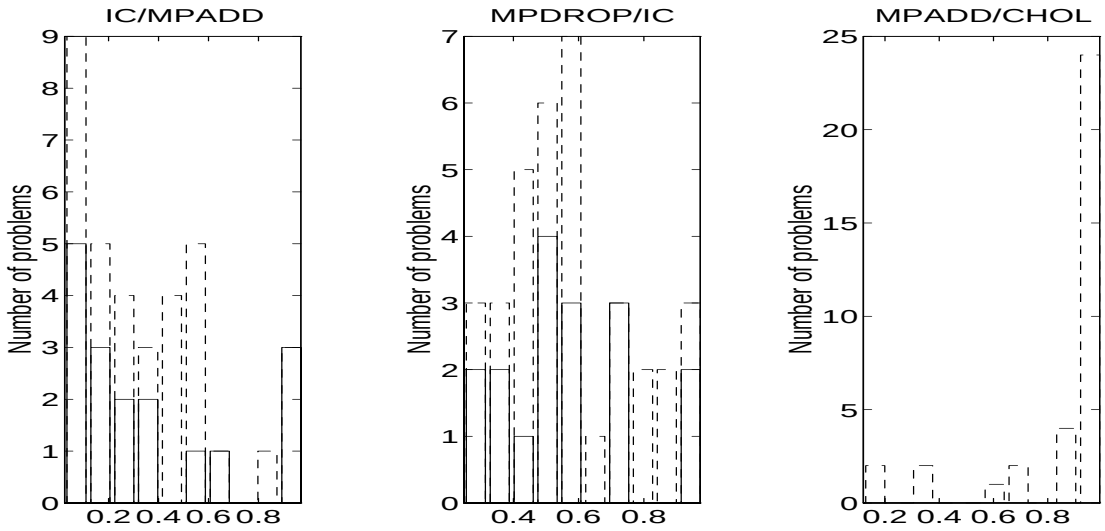


Figure 8: Ratios of storage requirement without ordering.

How does pattern modification affect the performance of IC in terms of execution time? We first measure the time for finding the pattern modification, which does not involve any floating point operations. Note that for applications with matrices that only change their numerical values, the pattern modification can be done once and then used repeatedly. We also measure the total computation time which is the time for IC factorization and the time for the preconditioned CG iterations. The sum of the total computation time and pattern modification time is called the total problem solving time. Since there are many failures for unpreconditioned CG and CG with standard ordering, we first compare performance by counting for each combination of reordering and preconditioning the number of times that that combination provided the fastest solution. Table 2 shows the fastest in terms of total computation time and Table 3 shows the fastest in terms of total problem solving time. The changes in numbers from one table to another is due to the pattern modification time.

Comparing the numbers in Table 1 2 and 3, we see that if the factorization can complete

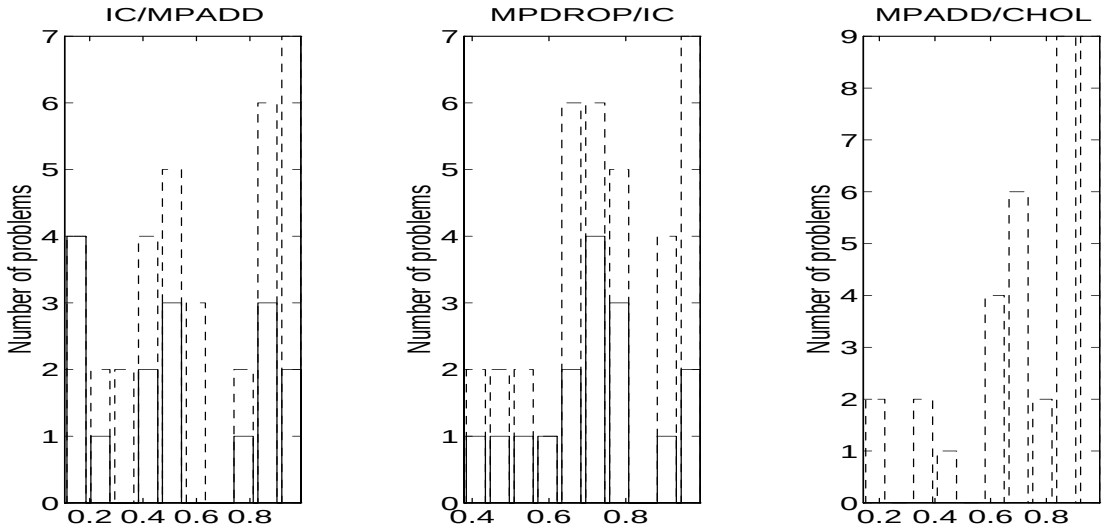


Figure 9: Ratios of storage requirement with minimum degree ordering.

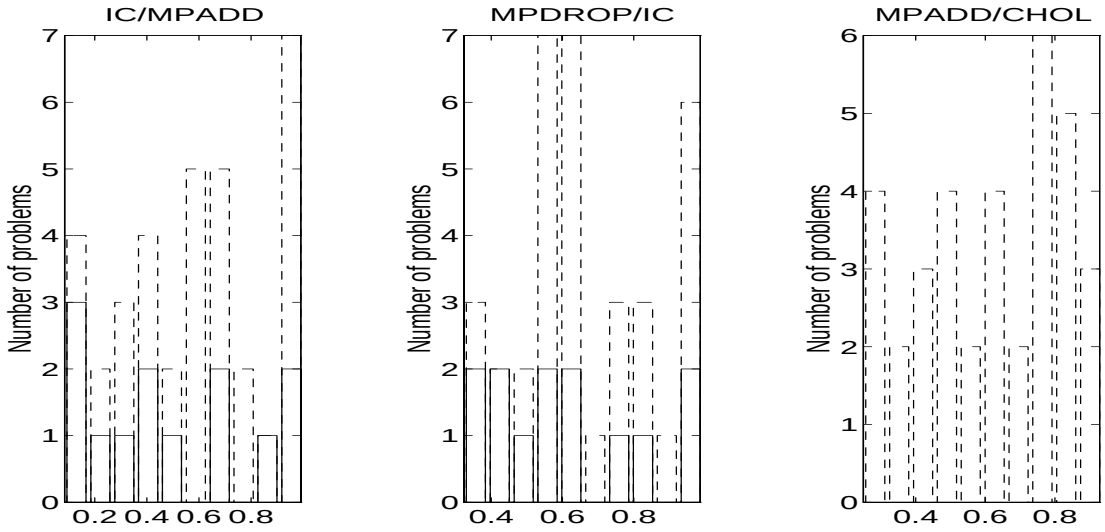


Figure 10: Ratios of storage requirement with nested dissection ordering.

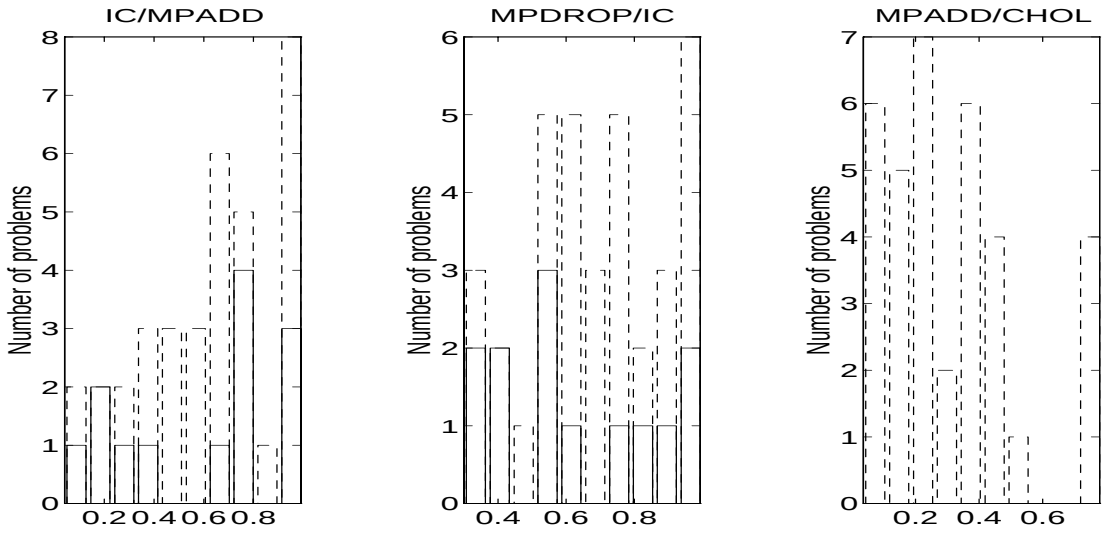


Figure 11: Ratios of storage requirement with reverse nested dissection ordering.

Method:	CG	IC	MPADD	MPDROP
none	1	12	21	1
MD	4	9	21	1
ND	6	15	13	1
RND	6	15	12	2

Table 2: The fastest combination in computation time

Method:	CG	IC	MPADD	MPDROP
none	2	14	18	1
MD	4	13	17	1
ND	6	18	9	2
RND	6	18	9	2

Table 3: The fastest combination in total problem solving time

without pattern modification, IC is fastest for most of the problems. This suggests that only adding or only dropping elements from P may make the preconditioner too dense or too sparse to be optimal. An open research topic is to find algorithms which can mix adding and dropping together to make the minimal modification needed to assure existence via property C+.

Since the comparison of "fastest" counts does not give enough quantitive information to compare the efficiency of the methods, we also compare the time required by standard IC and IC with **MPADD**, the two with the largest number of "fastest" counts. Figure 12 shows the ratio of the total computation time required by standard IC over the total computation time required by IC with **MPADD**, while Figure 13 shows the same ratios for the total problem solving time. Only the fifteen problems for which both IC and MPADD succeed are shown. Although most of the ratios are below 1, which means that standard IC is more efficient, with only a few exceptions the quantities are between 0.5 and 1. This indicates roughly that at most we must double the execution time in exchange for robustness. Also note that the Figures indicate that when using the sparsity pattern of the matrix A, IC without modification is usually the most efficient choice.

The sum of these observations is that using pattern modification does greatly improve the robustness of IC preconditioned CG, and in no cases does the preconditioner fail to exist. The price paid for this is an increase in the execution time, but that increase is reasonable and usually within a factor of 2 of unmodified IC. Combined, these observations suggest that by using the robustness and flexibility of sparsity pattern modification procedures, IC may be made more efficient by selecting the original pattern more carefully.

5 Conclusions and Future Work

A completely new strategy was developed to ensure the existence of the incomplete Cholesky (IC) factorization of symmetric positive definite matrices, based solely on the nonzero structure of the matrices. Theorems backed up by numerical experiments show that if the sparsity pattern P has property C+, IC will exist. This condition greatly improves the robustness of IC preconditioner, reducing the failure rate for IC preconditioned conjugate gradients from 49% to 0% when the matrix is not reordered, and from 37% to 14% in the worst case of reordering. The pattern modification failures all resulted from a loss of effectiveness in the preconditioner, not from the factorization phase. So in general the algorithms **MPADD** and **MPDROP** are effective and robust. The weakness of these methods is that they sometimes can either make the factor R too dense or so sparse that the efficiency of the preconditioner is degraded.

There are several interesting questions arising from this study.

The original pattern P affects the result of pattern modification methods. For example, we can reduce the density of the resulting pattern of **MPADD** by reducing the number of elements in the original pattern. We can also increase the density of the result of **MPDROP** by increase the density of the original pattern. How should we adjust the original pattern so that the output of the modification algorithms is optimal? For example, we can use the

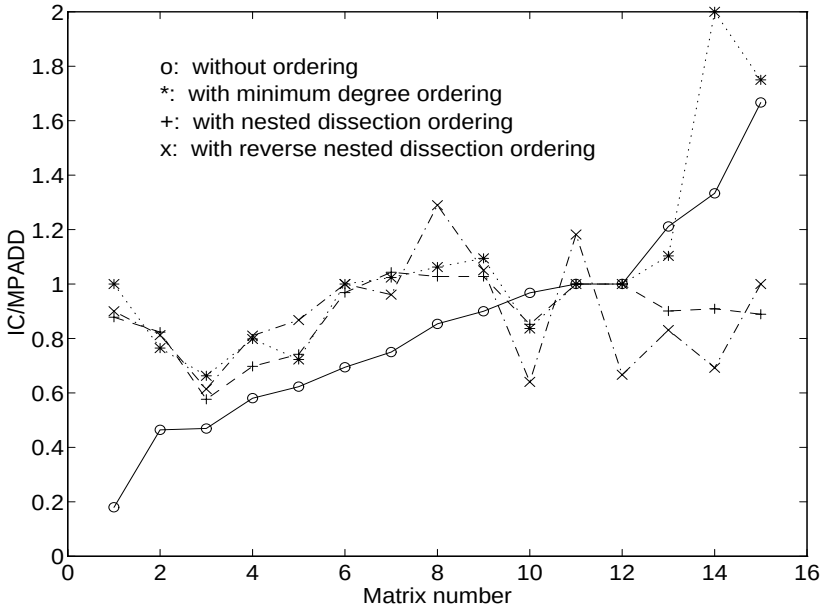


Figure 12: Ratio of total computation times

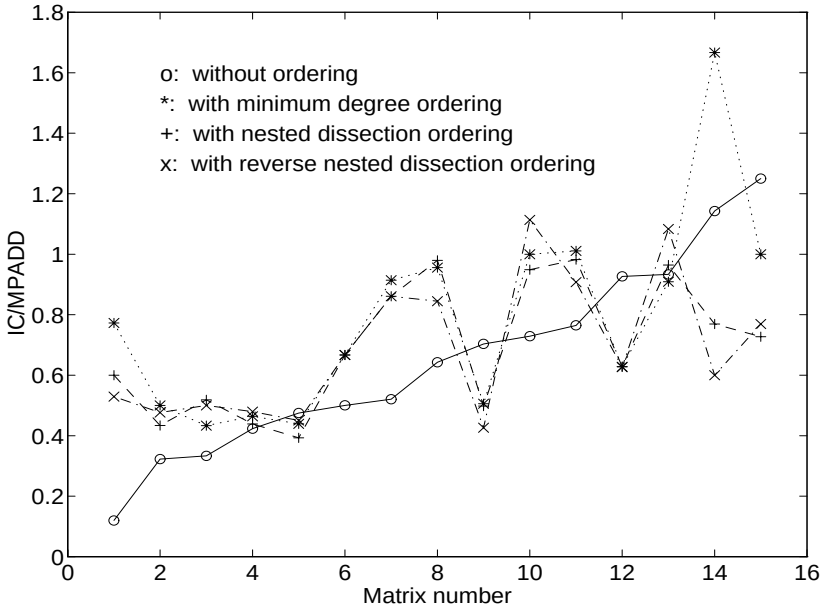


Figure 13: Ratio of total problem solving times

numerical or structural strategies suggested in [7] to reduce the density of the initial target sparsity pattern before applying **MPADD**. We can also use the technique of allowing a given number of levels of fill-in (the so–called IC(s) preconditioners) to increase the density of original pattern before applying **MPDROP**.

Another important goal is to find a method of pattern modification which makes the minimal pattern modifications necessary to assure Property C+. A better understanding of this pattern modification issue will probably come from further delineating the relation between elimination trees and C-trees and the relation between **MPADD** and a symbolic analysis of compressed incomplete modified Gram-Schmidt factorization (see [7]).

The reordering strategies examined here all are derived from efforts to reduce the fill–in during complete factorization. Since the reordering strategies tend to make **MPADD** work more like IC when it exists, and when IC exists it tends to be more efficient, another research direction starting from this work is to find a reordering method whose goal is not the reduction in fill–in of a factorization that won't be computed anyway. The problem is to develop the objective of a reordering in more concrete terms than "provides a more effective preconditioner when **MPADD** is used", and to develop fast heuristics for finding a near optimal reordering with that objective.

A deeper investigation of the influence of bordered block diagonal ordering on pattern modification methods is also needed. The preliminary tests show that the reverse nested dissection ordering behaves similarly to the nested dissection ordering, with slightly better robustness. But reverse nested dissection provides rich parallelism in all phases of problems solving, include pattern modification, incomplete Cholesky factorization and CG iteration. Furthermore, we have not exploited that parallelism in the results here, concentrating instead on basic existence and uniprocessor behaviour. Will the additional parallelism allowed by pattern modification compensate for the slightly worse performance when compared to IC without pattern modification?

Combining a numerical modification strategy with pattern modification methods may also provide a signficantly better IC preconditioner while retaining the robustness of pattern modification. Note that pattern modification methods alone are suitable for applications that solving multiple systems with the same structure. When the structure needs to be slightly changed, remodifying the pattern may not worthwhile, so that combining with numerical modification methods may be a better choice.

Ultimately, the best approach would likely combine and interlink all the above issues: developing a strategy based on adding and dropping elements based on both sparsity pattern and numerical properties, developing preliminary reorderings to enhance the effectiveness of the resulting preconditioner, and allowing parallelism at all phases of the solution process. In any case, pattern modification methods provide us with a new choice that greatly improve the robustness of IC preconditioning.

A Test data

All the test data are presented in the form of tables. The contents of each column of the tables are specified in the following way:

1. name: name of test matrix.
2. n: size of test matrix.
3. nz(R): number of non zero elements in R.
4. niter: number of iterations by CG method.
5. d(R): density of matrix R.
6. $\| x^* - x \|_2$: error norm.
7. $\| b - Ax \|_2$: norm of residule.
8. T_1: time of pattern modification.
9. T_2: time of incomplete Cholesky factorization.
10. T_3: time of CG iterations.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	0	2419	.00	.5768D-06	.8686D-06	.00	.00	7.85
bcsstk06	420	0	> 1260	.00	.2425D+01	.8248D+05	.00	.00	3.73
gr_30_30	900	0	40	.00	.1323D-06	.6697D-06	.00	.00	.16
494_bus	494	0	1306	.00	.9365D-06	.7993D-06	.00	.00	1.56
bcsstk07	420	0	> 1260	.00	.2425D+01	.8248D+05	.00	.00	3.73
bcsstk19	817	0	> 2451	.00	.2198D+02	.1704D+10	.00	.00	8.48
lund_a	147	0	370	.00	.1729D-10	.1437D-05	.00	.00	.31
662_bus	662	0	631	.00	.2802D-06	.7383D-06	.00	.00	1.16
bcsstk08	1074	0	> 3222	.00	.5013D-01	.1234D+05	.00	.00	18.19
bcsstk20	485	0	> 1455	.00	.1676D+02	.4347D+10	.00	.00	2.53
nos1	237	0	> 711	.00	.7461D+01	.6579D+05	.00	.00	.45
685_bus	685	0	620	.00	.6717D-07	.9788D-06	.00	.00	1.32
bcsstk09	1083	0	271	.00	.1648D-11	.1021D-05	.00	.00	1.97
bcsstk21	3600	0	>10800	.00	.7682D-05	.3604D-01	.00	.00	169.51
nos2	957	0	> 2871	.00	.2206D+02	.5680D+06	.00	.00	8.38
bcsstk10	1086	0	> 3258	.00	.8732D-04	.4207D+00	.00	.00	26.82
bcsstk22	138	0	> 414	.00	.2776D-09	.7492D-05	.00	.00	.16
nos3	960	0	269	.00	.3873D-07	.7960D-06	.00	.00	1.69
bcsstk11	1473	0	> 4419	.00	.1303D+01	.1554D+04	.00	.00	54.98
bcsstk23	3134	0	> 9402	.00	.3470D+02	.1526D+11	.00	.00	185.51
bcsstm07	420	0	338	.00	.4833D-06	.6492D-06	.00	.00	.94
nos4	100	0	75	.00	.9547D-05	.8347D-06	.00	.00	.02
bcsstk12	1473	0	> 4419	.00	.1303D+01	.1554D+04	.00	.00	54.93
nos5	468	0	494	.00	.5599D-10	.7661D-06	.00	.00	1.15
bcsstk13	2003	0	> 6009	.00	.1625D+02	.2443D+08	.00	.00	164.38
nos6	675	0	1446	.00	.4532D-07	.5863D-06	.00	.00	3.11
bcsstk14	1806	0	> 5418	.00	.6317D+01	.2698D+04	.00	.00	114.85
bcsstk26	1922	0	> 5766	.00	.1996D+01	.8486D+06	.00	.00	72.49
nos7	729	0	> 2187	.00	.3504D-02	.1385D-01	.00	.00	5.89
bcsstk03	112	0	> 336	.00	.7595D-01	.1981D+06	.00	.00	.12
bcsstk15	3948	0	>11844	.00	.2430D+01	.1517D+03	.00	.00	491.40
bcsstk27	1224	0	1520	.00	.4846D-09	.8864D-06	.00	.00	26.96
bcsstk04	132	0	> 396	.00	.1402D-02	.5969D+00	.00	.00	.45
bcsstm12	1473	0	1757	.00	.2858D-02	.8619D-06	.00	.00	14.91
bcsstk05	153	0	315	.00	.5331D-12	.5780D-06	.00	.00	.27

Table 4: Test data of CG without ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	0	2414	.00	.5726D-06	.9063D-06	.00	.00	7.87
bcsstk06	420	0	> 1260	.00	.2430D+01	.5142D+05	.00	.00	3.70
gr_30_30	900	0	40	.00	.1323D-06	.6697D-06	.00	.00	.15
494_bus	494	0	1299	.00	.9397D-06	.7860D-06	.00	.00	1.58
bcsstk07	420	0	> 1260	.00	.2430D+01	.5142D+05	.00	.00	3.74
bcsstk19	817	0	> 2451	.00	.2198D+02	.2557D+09	.00	.00	8.39
lund_a	147	0	370	.00	.1736D-10	.1645D-05	.00	.00	.31
662_bus	662	0	628	.00	.2855D-06	.7727D-06	.00	.00	1.15
bcsstk08	1074	0	> 3222	.00	.4181D-01	.1361D+05	.00	.00	18.24
bcsstk20	485	0	> 1455	.00	.1676D+02	.1044D+10	.00	.00	2.51
nos1	237	0	> 711	.00	.7471D+01	.1598D+06	.00	.00	.46
685_bus	685	0	622	.00	.5946D-07	.9383D-06	.00	.00	1.34
bcsstk09	1083	0	271	.00	.3946D-12	.1064D-05	.00	.00	1.96
bcsstk21	3600	0	>10800	.00	.7667D-05	.3880D-01	.00	.00	169.95
nos2	957	0	> 2871	.00	.2206D+02	.2100D+06	.00	.00	8.51
bcsstk10	1086	0	> 3258	.00	.8820D-04	.3330D+00	.00	.00	27.00
bcsstk22	138	0	> 414	.00	.3602D-09	.4208D-05	.00	.00	.17
nos3	960	0	269	.00	.3867D-07	.8023D-06	.00	.00	1.69
bcsstk11	1473	0	> 4419	.00	.1306D+01	.1663D+04	.00	.00	55.69
bcsstk23	3134	0	> 9402	.00	.3470D+02	.1722D+11	.00	.00	202.45
bcsstm07	420	0	338	.00	.4834D-06	.6976D-06	.00	.00	.93
nos4	100	0	75	.00	.9405D-05	.8020D-06	.00	.00	.02
bcsstk12	1473	0	> 4419	.00	.1306D+01	.1663D+04	.00	.00	55.64
nos5	468	0	494	.00	.5975D-10	.4466D-06	.00	.00	1.12
bcsstk13	2003	0	> 6009	.00	.1628D+02	.3508D+08	.00	.00	170.10
nos6	675	0	1471	.00	.4199D-07	.4533D-06	.00	.00	3.08
bcsstk14	1806	0	> 5418	.00	.6317D+01	.2351D+04	.00	.00	117.81
bcsstk26	1922	0	> 5766	.00	.1996D+01	.4666D+06	.00	.00	73.87
nos7	729	0	> 2187	.00	.3620D-02	.4753D-03	.00	.00	5.58
bcsstk03	112	0	> 336	.00	.8638D-01	.6555D+06	.00	.00	.12
bcsstk15	3948	0	>11844	.00	.2430D+01	.1840D+03	.00	.00	542.08
bcsstk27	1224	0	1518	.00	.4900D-09	.8571D-06	.00	.00	27.39
bcsstk04	132	0	> 396	.00	.1403D-02	.5301D+00	.00	.00	.45
bcsstm12	1473	0	1768	.00	.2831D-02	.8586D-06	.00	.00	15.06
bcsstk05	153	0	314	.00	.5219D-12	.9909D-06	.00	.00	.27

Table 5: Test data of CG with minimum degree ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	0	2431	.00	.5063D-06	.9162D-06	.00	.00	7.91
bcsstk06	420	0	> 1260	.00	.2430D+01	.1074D+06	.00	.00	3.72
gr_30_30	900	0	40	.00	.1323D-06	.6697D-06	.00	.00	.15
494_bus	494	0	1321	.00	.7838D-06	.8931D-06	.00	.00	1.58
bcsstk07	420	0	> 1260	.00	.2430D+01	.1074D+06	.00	.00	3.69
bcsstk19	817	0	> 2451	.00	.2198D+02	.8173D+09	.00	.00	8.44
lund_a	147	0	371	.00	.1197D-10	.1611D-05	.00	.00	.31
662_bus	662	0	632	.00	.2854D-06	.8646D-06	.00	.00	1.17
bcsstk08	1074	0	> 3222	.00	.4149D-01	.3910D+04	.00	.00	18.35
bcsstk20	485	0	> 1455	.00	.1676D+02	.3320D+10	.00	.00	2.50
nos1	237	0	> 711	.00	.7492D+01	.3121D+06	.00	.00	.45
685_bus	685	0	621	.00	.5053D-07	.6951D-06	.00	.00	1.33
bcsstk09	1083	0	271	.00	.1006D-11	.1137D-05	.00	.00	1.97
bcsstk21	3600	0	>10800	.00	.7677D-05	.4571D-01	.00	.00	166.81
nos2	957	0	> 2871	.00	.2206D+02	.2081D+07	.00	.00	8.42
bcsstk10	1086	0	> 3258	.00	.8718D-04	.5508D+00	.00	.00	26.79
bcsstk22	138	0	> 414	.00	.3575D-09	.7374D-05	.00	.00	.17
nos3	960	0	269	.00	.3835D-07	.8201D-06	.00	.00	1.70
bcsstk11	1473	0	> 4419	.00	.1302D+01	.1424D+04	.00	.00	55.03
bcsstk23	3134	0	> 9402	.00	.3469D+02	.5536D+11	.00	.00	188.60
bcsstm07	420	0	338	.00	.4830D-06	.6758D-06	.00	.00	.93
nos4	100	0	75	.00	.9527D-05	.8285D-06	.00	.00	.02
bcsstk12	1473	0	> 4419	.00	.1302D+01	.1424D+04	.00	.00	55.11
nos5	468	0	495	.00	.5400D-10	.8034D-06	.00	.00	1.14
bcsstk13	2003	0	> 6009	.00	.1625D+02	.1586D+08	.00	.00	165.34
nos6	675	0	1444	.00	.4673D-07	.8480D-06	.00	.00	3.05
bcsstk14	1806	0	> 5418	.00	.6317D+01	.2824D+04	.00	.00	115.50
bcsstk26	1922	0	> 5766	.00	.2023D+01	.8116D+06	.00	.00	72.59
nos7	729	0	> 2187	.00	.3622D-02	.1411D-02	.00	.00	5.71
bcsstk03	112	0	> 336	.00	.1214D+00	.1003D+07	.00	.00	.11
bcsstk15	3948	0	>11844	.00	.2430D+01	.1414D+03	.00	.00	498.32
bcsstk27	1224	0	1523	.00	.4773D-09	.8611D-06	.00	.00	27.02
bcsstk04	132	0	> 396	.00	.1403D-02	.1949D+01	.00	.00	.45
bcsstm12	1473	0	1760	.00	.2843D-02	.8535D-06	.00	.00	15.02
bcsstk05	153	0	315	.00	.3259D-12	.3464D-06	.00	.00	.26

Table 6: Test data of CG with nested dissection ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	0	2420	.00	.5782D-06	.9141D-06	.00	.00	7.76
bcsstk06	420	0	> 1260	.00	.2427D+01	.8944D+05	.00	.00	3.70
gr_30_30	900	0	40	.00	.1323D-06	.6697D-06	.00	.00	.16
494_bus	494	0	1317	.00	.9304D-06	.9403D-06	.00	.00	1.59
bcsstk07	420	0	> 1260	.00	.2427D+01	.8944D+05	.00	.00	3.70
bcsstk19	817	0	> 2451	.00	.2198D+02	.4211D+09	.00	.00	8.42
lund_a	147	0	371	.00	.1047D-10	.1468D-05	.00	.00	.31
662_bus	662	0	631	.00	.2793D-06	.9385D-06	.00	.00	1.16
bcsstk08	1074	0	> 3222	.00	.4927D-01	.2159D+05	.00	.00	18.05
bcsstk20	485	0	> 1455	.00	.1676D+02	.2207D+10	.00	.00	2.45
nos1	237	0	> 711	.00	.7465D+01	.8661D+05	.00	.00	.45
685_bus	685	0	618	.00	.6964D-07	.9749D-06	.00	.00	1.31
bcsstk09	1083	0	271	.00	.1352D-11	.1178D-05	.00	.00	1.98
bcsstk21	3600	0	>10800	.00	.7674D-05	.2973D-01	.00	.00	170.09
nos2	957	0	> 2871	.00	.2206D+02	.1671D+06	.00	.00	8.33
bcsstk10	1086	0	> 3258	.00	.8715D-04	.3278D+00	.00	.00	26.76
bcsstk22	138	0	> 414	.00	.3634D-09	.3165D-05	.00	.00	.16
nos3	960	0	269	.00	.3873D-07	.7959D-06	.00	.00	1.67
bcsstk11	1473	0	> 4419	.00	.1300D+01	.2074D+04	.00	.00	54.97
bcsstk23	3134	0	> 9402	.00	.3470D+02	.4109D+11	.00	.00	187.38
bcsstm07	420	0	337	.00	.4841D-06	.8229D-06	.00	.00	.93
nos4	100	0	75	.00	.9606D-05	.8526D-06	.00	.00	.02
bcsstk12	1473	0	> 4419	.00	.1300D+01	.2074D+04	.00	.00	54.95
nos5	468	0	495	.00	.5878D-10	.6916D-06	.00	.00	1.13
bcsstk13	2003	0	> 6009	.00	.1629D+02	.2978D+08	.00	.00	165.11
nos6	675	0	1455	.00	.5510D-07	.1004D-05	.00	.00	3.12
bcsstk14	1806	0	> 5418	.00	.6317D+01	.4085D+04	.00	.00	115.65
bcsstk26	1922	0	> 5766	.00	.2006D+01	.7482D+06	.00	.00	72.76
nos7	729	0	> 2187	.00	.3616D-02	.4041D-03	.00	.00	5.69
bcsstk03	112	0	> 336	.00	.1217D+00	.6587D+06	.00	.00	.11
bcsstk15	3948	0	>11844	.00	.2430D+01	.1079D+03	.00	.00	495.43
bcsstk27	1224	0	1520	.00	.4845D-09	.9480D-06	.00	.00	27.01
bcsstk04	132	0	> 396	.00	.1403D-02	.1455D+01	.00	.00	.44
bcsstm12	1473	0	1760	.00	.2841D-02	.8566D-06	.00	.00	14.83
bcsstk05	153	0	314	.00	.4994D-12	.6974D-06	.00	.00	.26

Table 7: Test data of CG with reverse nested dissection ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2596	135	.004	.1111D-06	.9891D-06	.000	.380	.790
bcsstk06	fail								
gr_30_30	900	4322	20	.011	.1375D-05	.8519D-06	.000	.230	.150
494_bus	494	1080	91	.009	.2942D-06	.8824D-06	.000	.070	.210
bcsstk07	fail								
bcsstk19	fail								
lund_a	147	1298	23	.119	.5292D-11	.7667D-06	.000	.010	.040
662_bus	662	1568	69	.007	.2416D-06	.7052D-06	.000	.120	.230
bcsstk08	1074	7017	47	.012	.4377D-11	.2727D-04	.000	.380	.520
bcsstk20	fail								
nos1	fail								
685_bus	685	1967	84	.008	.2841D-07	.6851D-06	.000	.140	.310
bcsstk09	fail								
bcsstk21	fail								
nos2	fail								
bcsstk10	fail								
bcsstk22	138	417	44	.043	.2546D-11	.3643D-06	.000	.010	.030
nos3	960	8402	50	.018	.1471D-07	.5480D-06	.000	.300	.600
bcsstk11	fail								
bcsstk23	fail								
bcsstm07	420	3836	13	.043	.6460D-08	.1707D-06	.000	.060	.070
nos4	100	347	20	.069	.1287D-05	.1618D-06	.000	.000	.010
bcsstk12	fail								
nos5	468	2820	55	.026	.4369D-10	.9667D-06	.000	.070	.240
bcsstk13	fail								
nos6	675	1965	31	.009	.9166D-10	.2883D-06	.000	.130	.120
bcsstk14	fail								
bcsstk26	fail								
nos7	729	2673	25	.010	.3332D-07	.9251D-06	.000	.150	.130
bcsstk03	fail								
bcsstk15	fail								
bcsstk27	1224	28675	30	.038	.2429D-10	.2943D-06	.000	.830	1.060
bcsstk04	132	1890	40	.215	.3386D-10	.6297D-06	.000	.010	.100
bcsstm12	1473	10566	13	.010	.5978D-05	.5895D-06	.000	.720	.220
bcsstk05	153	1288	44	.109	.1230D-11	.2863D-06	.000	.010	.070

Table 8: Test data of CG with IC(0) preconditioner method without ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2596	93	.004	.6818D-07	.7523D-06	.000	.380	.550
bcsstk06	fail								
gr_30_30	900	4322	31	.011	.8656D-06	.5782D-06	.000	.240	.230
494_bus	494	1080	58	.009	.7174D-07	.6139D-06	.000	.070	.130
bcsstk07	fail								
bcsstk19	fail								
lund_a	147	1298	37	.119	.2134D-11	.8288D-06	.000	.010	.060
662_bus	662	1568	63	.007	.1461D-06	.6274D-06	.000	.130	.210
bcsstk08	fail								
bcsstk20	fail								
nos1	fail								
685_bus	685	1967	76	.008	.4363D-07	.8561D-06	.000	.130	.300
bcsstk09	fail								
bcsstk21	3600	15100	403	.002	.2404D-10	.3229D-05	.000	4.500	11.540
nos2	fail								
bcsstk10	fail								
bcsstk22	138	417	42	.043	.3849D-11	.3414D-06	.000	.010	.030
nos3	960	8402	86	.018	.2489D-07	.7500D-06	.000	.300	1.030
bcsstk11	fail								
bcsstk23	fail								
bcsstm07	420	3836	14	.043	.2076D-07	.5879D-06	.000	.060	.070
nos4	100	347	28	.069	.1830D-05	.2338D-06	.000	.000	.020
bcsstk12	fail								
nos5	468	2820	110	.026	.2171D-10	.4644D-06	.000	.080	.470
bcsstk13	fail								
nos6	675	1965	49	.009	.1132D-09	.3791D-06	.000	.130	.190
bcsstk14	1806	32630	120	.020	.6190D-11	.4014D-04	.000	1.610	5.080
bcsstk26	fail								
nos7	729	2673	38	.010	.3697D-07	.5281D-06	.000	.160	.180
bcsstk03	fail								
bcsstk15	fail								
bcsstk27	1224	28675	80	.038	.3624D-10	.9037D-06	.000	.840	2.790
bcsstk04	132	1890	40	.215	.1063D-10	.2268D-06	.000	.020	.090
bcsstm12	1473	10566	12	.010	.9767D-05	.8518D-06	.000	.720	.190
bcsstk05	153	1288	53	.109	.5271D-12	.2025D-06	.000	.010	.090

Table 9: Test data of CG preconditioned by IC(0) with minimum degree ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2596	126	.004	.1343D-06	.8772D-06	.000	.380	.740
bcsstk06	420	4140	175	.047	.6443D-11	.1706D-04	.000	.060	.980
gr_30_30	900	4322	29	.011	.4019D-06	.6136D-06	.000	.240	.220
494_bus	494	1080	83	.009	.1469D-06	.7890D-06	.000	.070	.200
bcsstk07	420	4140	175	.047	.6443D-11	.1706D-04	.000	.070	.970
bcsstk19	fail								
lund_a	147	1298	46	.119	.1495D-10	.6942D-06	.000	.010	.070
662_bus	662	1568	71	.007	.8803D-07	.6967D-06	.000	.140	.230
bcsstk08	1074	7017	74	.012	.1158D-11	.5460D-04	.000	.360	.800
bcsstk20	485	1810	> 485	.015	.2501D-05	.1163D+04	.000	.070	1.520
nos1	fail								
685_bus	685	1967	84	.008	.2059D-07	.7900D-06	.000	.140	.340
bcsstk09	fail								
bcsstk21	3600	15100	524	.002	.1496D-10	.3480D-05	.000	4.310	14.770
nos2	fail								
bcsstk10	fail								
bcsstk22	fail								
nos3	960	8402	88	.018	.2414D-07	.9554D-06	.000	.310	1.070
bcsstk11	fail								
bcsstk23	fail								
bcsstm07	420	3836	14	.043	.6617D-08	.2060D-06	.000	.070	.070
nos4	100	347	30	.069	.5093D-05	.4799D-06	.000	.000	.020
bcsstk12	fail								
nos5	468	2820	97	.026	.4904D-10	.6622D-06	.000	.070	.420
bcsstk13	fail								
nos6	675	1965	46	.009	.1963D-09	.2815D-06	.000	.140	.170
bcsstk14	fail								
bcsstk26	fail								
nos7	729	2673	42	.010	.7841D-08	.4347D-06	.000	.160	.200
bcsstk03	112	376	58	.059	.1179D-10	.1101D-03	.000	.000	.040
bcsstk15	fail								
bcsstk27	1224	28675	86	.038	.3773D-10	.6825D-06	.000	.810	3.010
bcsstk04	132	1890	39	.215	.2691D-10	.7888D-06	.000	.010	.090
bcsstm12	1473	10566	10	.010	.1113D-04	.9325D-06	.000	.730	.170
bcsstk05	153	1288	54	.109	.1261D-11	.3861D-06	.000	.010	.090

Table 10: Test data of CG preconditioned by IC(0) with nested dissection ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2596	211	.004	.1294D-06	.9509D-06	.000	.400	1.270
bcsstk06	fail								
gr_30_30	900	4322	32	.011	.2468D-06	.3919D-06	.000	.310	.280
494_bus	494	1080	115	.009	.1261D-06	.8067D-06	.000	.080	.310
bcsstk07	fail								
bcsstk19	fail								
lund_a	147	1298	48	.119	.5503D-11	.6115D-06	.000	.010	.090
662_bus	662	1568	76	.007	.1785D-06	.6935D-06	.000	.180	.310
bcsstk08	1074	7017	78	.012	.5878D-11	.1666D-03	.000	.470	1.000
bcsstk20	fail								
nos1	fail								
685_bus	685	1967	91	.008	.5696D-07	.9406D-06	.000	.140	.350
bcsstk09	fail								
bcsstk21	fail								
nos2	fail								
bcsstk10	1086	11578	168	.020	.7012D-11	.8415D-06	.000	.450	2.660
bcsstk22	138	417	60	.043	.4401D-11	.6212D-06	.000	.010	.040
nos3	960	8402	91	.018	.2555D-07	.9569D-06	.000	.310	1.080
bcsstk11	1473	17857	345	.016	.5430D-09	.4497D-05	.000	.910	8.710
bcsstk23	fail								
bcsstm07	420	3836	15	.043	.6734D-08	.2137D-06	.000	.050	.080
nos4	100	347	31	.069	.4519D-05	.5289D-06	.000	.010	.010
bcsstk12	1473	17857	345	.016	.5430D-09	.4497D-05	.000	1.020	8.630
nos5	468	2820	100	.026	.2045D-10	.5248D-06	.000	.080	.430
bcsstk13	fail								
nos6	675	1965	46	.009	.1987D-09	.6752D-06	.000	.140	.170
bcsstk14	fail								
bcsstk26	fail								
nos7	729	2673	41	.010	.1374D-07	.2818D-06	.000	.160	.200
bcsstk03	112	376	57	.059	.2571D-10	.9117D-04	.000	.000	.040
bcsstk15	fail								
bcsstk27	1224	28675	88	.038	.2212D-10	.5974D-06	.000	.830	3.150
bcsstk04	132	1890	40	.215	.7039D-11	.1469D-06	.000	.010	.090
bcsstm12	1473	10566	11	.010	.6491D-05	.2762D-06	.000	.740	.200
bcsstk05	153	1288	60	.109	.1089D-11	.2853D-06	.000	.010	.080

Table 11: Test data of CG preconditioned by IC(0) with reverse nested dissection ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	4619	132	.007	.1049D-06	.7768D-06	.230	.380	.920
bcsstk06	420	13274	27	.150	.1315D-09	.8373D-05	.110	.180	.280
gr_30_30	900	27870	1	.069	.5312D-13	.5892D-13	.190	.570	.040
494_bus	494	1240	89	.010	.1566D-06	.6554D-06	.020	.070	.210
bcsstk07	420	13274	27	.150	.1315D-09	.8373D-05	.120	.210	.320
bcsstk19	817	37459	48	.112	.6121D-06	.1092D+01	1.360	.840	1.230
lund_a	147	3017	1	.277	.6556D-11	.4570D-06	.010	.030	.000
662_bus	662	3251	65	.015	.1566D-06	.8252D-06	.070	.140	.270
bcsstk08	1074	162590	36	.282	.7159D-11	.8946D-04	5.820	9.010	3.780
bcsstk20	485	2687	98	.023	.1495D-05	.1527D+02	.020	.070	.370
nos1	237	704	5	.025	.3126D-09	.6725D-05	.000	.020	.010
685_bus	685	5249	74	.022	.1701D-07	.6391D-06	.100	.170	.430
bcsstk09	1083	63700	3	.109	.5194D-12	.1214D-06	.720	1.670	.170
bcsstk21	3600	212323	35	.033	.2123D-11	.1013D-05	2.170	10.220	5.610
nos2	957	2864	6	.006	.1178D-05	.4009D-02	.010	.260	.030
bcsstk10	1086	20771	1	.035	.3035D-11	.5536D-07	.110	.610	.040
bcsstk22	138	995	13	.104	.1077D-10	.1035D-06	.020	.010	.010
nos3	960	40061	1	.087	.3827D-11	.4609D-11	.350	.880	.050
bcsstk11	1473	67372	70	.062	.5925D-08	.2099D-05	1.570	1.940	3.980
bcsstk23	3134	1015951	44	.207	.1271D-06	.3370D+02	52.330	98.930	26.690
bcsstm07	420	13274	11	.150	.4274D-08	.1761D-06	.110	.160	.120
nos4	100	763	9	.151	.6412D-06	.4335D-07	.010	.000	.010
bcsstk12	1473	67372	70	.062	.5925D-08	.2099D-05	1.700	1.900	4.010
nos5	468	25138	14	.229	.1779D-10	.3469D-06	.300	.430	.230
bcsstk13	2003	433292	23	.216	.1509D-09	.1654D-02	31.360	28.200	6.890
nos6	675	16679	4	.073	.4119D-10	.5534D-08	.120	.290	.070
bcsstk14	1806	187710	22	.115	.1688D-11	.2186D-04	4.120	7.860	2.930
bcsstk26	1922	113553	27	.061	.4567D-10	.8937D-04	2.510	4.250	2.140
nos7	729	53873	1	.202	.4400D-07	.1347D-07	.770	1.480	.080
bcsstk03	112	384	2	.061	.7367D-11	.4315D-04	.000	.010	.000
bcsstk15	3948	942562	5	.121	.1753D-09	.1187D-04	38.440	73.860	3.430
bcsstk27	1224	50515	1	.067	.8166D-12	.5065D-08	.480	1.460	.100
bcsstk04	132	3753	5	.428	.9878D-12	.1475D-07	.030	.030	.020
bcsstm12	1473	38362	8	.035	.2568D-04	.3097D-06	.600	1.320	.300
bcsstk05	153	2491	17	.211	.3056D-12	.5731D-08	.010	.020	.040

Table 12: Test data of MPADD without ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2742	82	.004	.2369D-06	.8616D-06	.070	.370	.480
bcsstk06	420	8355	82	.095	.3864D-10	.1083D-04	.150	.110	.640
gr_30_30	900	11128	31	.027	.4037D-06	.7471D-06	.420	.310	.340
494_bus	494	1141	53	.009	.1122D-06	.7698D-06	.020	.070	.130
bcsstk07	420	8355	82	.095	.3864D-10	.1083D-04	.160	.110	.640
bcsstk19	817	4614	> 817	.014	.3244D-05	.1289D+03	.060	.200	5.510
lund_a	147	2261	10	.208	.6394D-11	.3922D-06	.030	.010	.030
662_bus	662	1657	58	.008	.6563D-07	.7211D-06	.020	.120	.200
bcsstk08	1074	27851	39	.048	.7546D-11	.3077D-04	1.030	.750	.860
bcsstk20	485	1832	175	.016	.2000D-05	.3253D+02	.010	.070	.560
nos1	237	703	2	.025	.7729D-10	.9052D-06	.000	.020	.000
685_bus	685	2336	68	.010	.1149D-07	.4371D-06	.030	.140	.280
bcsstk09	1083	54755	40	.093	.1755D-12	.4342D-06	2.430	1.510	1.580
bcsstk21	3600	16643	392	.003	.3489D-10	.3065D-05	.920	4.560	11.540
nos2	957	2863	2	.006	.4260D-07	.1102D-03	.010	.270	.010
bcsstk10	1086	22378	34	.038	.1028D-10	.4791D-06	.470	.620	.760
bcsstk22	138	473	37	.049	.1334D-11	.9109D-07	.000	.000	.030
nos3	960	26571	44	.058	.3214D-07	.9094D-06	1.040	.610	.980
bcsstk11	1473	44919	118	.041	.2240D-08	.2566D-05	2.230	1.380	4.630
bcsstk23	3134	159240	1735	.032	.6209D-06	.1672D+03	14.160	9.240	201.380
bcsstm07	420	6555	12	.074	.2841D-07	.9794D-06	.130	.090	.080
nos4	100	451	26	.089	.8210D-05	.9661D-06	.010	.000	.020
bcsstk12	1473	44919	118	.041	.2240D-08	.2566D-05	2.210	1.390	4.640
nos5	468	12369	66	.113	.9947D-11	.3939D-06	.270	.200	.630
bcsstk13	2003	269712	27	.134	.3458D-10	.2101D-02	21.790	16.030	5.080
nos6	675	2063	49	.009	.1008D-09	.6141D-06	.030	.130	.190
bcsstk14	1806	100012	52	.061	.8098D-11	.3303D-04	6.500	3.430	4.230
bcsstk26	1922	20464	324	.011	.1968D-09	.3027D-03	.780	1.510	8.580
nos7	729	2673	38	.010	.3697D-07	.5281D-06	.100	.160	.180
bcsstk03	112	384	2	.061	.2333D-10	.3392D-04	.000	.010	.000
bcsstk15	3948	572228	92	.073	.1654D-09	.2835D-04	83.860	39.810	34.700
bcsstk27	1224	56123	37	.075	.2044D-10	.4799D-06	2.490	1.380	1.910
bcsstk04	132	3297	11	.376	.4285D-11	.1120D-06	.030	.020	.040
bcsstm12	1473	25431	6	.023	.2305D-04	.5186D-06	.820	.980	.160
bcsstk05	153	2420	17	.205	.3278D-11	.2909D-06	.010	.010	.040

Table 13: Test Data of MPADD with minimum degree ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2734	120	.004	.8947D-07	.7286D-06	.050	.370	.720
bcsstk06	420	6122	150	.069	.1086D-10	.1449D-04	.100	.080	1.000
gr_30_30	900	14444	19	.036	.2098D-06	.4453D-06	.550	.370	.250
494_bus	494	1113	82	.009	.1120D-06	.6760D-06	.010	.080	.190
bcsstk07	420	6122	150	.069	.1086D-10	.1449D-04	.110	.080	1.010
bcsstk19	817	5636	> 817	.017	.7612D-06	.1465D+01	.110	.210	6.020
lund_a	147	2351	36	.216	.1459D-11	.8905D-06	.020	.010	.080
662_bus	662	1661	68	.008	.9252D-07	.6879D-06	.030	.130	.230
bcsstk08	1074	83117	29	.144	.6091D-11	.5818D-04	3.800	3.830	1.610
bcsstk20	485	1840	> 485	.016	.7773D-05	.4426D+04	.020	.070	1.520
nos1	237	642	232	.023	.1198D-09	.7011D-05	.010	.010	.270
685_bus	685	2564	78	.011	.1341D-07	.3802D-06	.030	.130	.330
bcsstk09	1083	38263	51	.065	.5386D-12	.8790D-06	1.920	.960	1.530
bcsstk21	3600	27711	326	.004	.2037D-10	.2863D-05	.940	4.950	11.510
nos2	957	2610	> 957	.006	.2309D+01	.3392D+06	.020	.270	5.020
bcsstk10	1086	22987	62	.039	.3366D-10	.7460D-06	.420	.640	1.380
bcsstk22	138	461	54	.048	.8261D-11	.8535D-06	.010	.010	.040
nos3	960	25594	47	.055	.1446D-07	.4624D-06	1.150	.600	1.020
bcsstk11	1473	46337	147	.043	.2073D-08	.3423D-05	2.160	1.450	5.880
bcsstk23	3134	143063	1758	.029	.2143D-06	.2552D+03	18.400	7.890	183.560
bcsstm07	420	5899	13	.067	.6905D-08	.2210D-06	.100	.080	.090
nos4	100	575	26	.114	.5149D-05	.5353D-06	.010	.000	.020
bcsstk12	1473	46337	147	.043	.2073D-08	.3423D-05	2.190	1.440	5.900
nos5	468	11096	78	.101	.8789D-10	.4493D-06	.280	.160	.690
bcsstk13	2003	282737	150	.141	.6850D-10	.5459D-02	31.850	18.190	28.320
nos6	675	2547	44	.011	.1148D-09	.4667D-06	.040	.140	.180
bcsstk14	1806	121806	60	.075	.3543D-11	.2759D-04	9.970	4.290	5.550
bcsstk26	1922	23912	436	.013	.3055D-09	.3296D-03	.780	1.600	12.260
nos7	729	5321	39	.020	.5524D-07	.7254D-06	.190	.170	.240
bcsstk03	112	376	58	.059	.1179D-10	.1101D-03	.000	.000	.040
bcsstk15	3948	521497	101	.067	.1628D-09	.3273D-04	87.910	32.930	34.450
bcsstk27	1224	52061	60	.069	.4211D-10	.9530D-06	1.840	1.330	2.910
bcsstk04	132	4430	21	.505	.1036D-10	.1729D-07	.050	.040	.080
bcsstm12	1473	25198	10	.023	.3274D-05	.2574D-06	.760	1.040	.250
bcsstk05	153	2038	47	.173	.5828D-12	.1194D-06	.020	.010	.100

Table 14: Test Data of MPADD with nested dissection ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2739	211	.004	.1041D-06	.6897D-06	.250	.370	1.220
bcsstk06	420	5699	154	.064	.1331D-10	.1140D-04	.100	.070	.980
gr_30_30	900	12021	31	.030	.9614D-06	.9082D-06	.630	.320	.360
494_bus	494	1101	115	.009	.1386D-06	.8316D-06	.030	.070	.260
bcsstk07	420	5699	154	.064	.1331D-10	.1140D-04	.110	.070	.990
bcsstk19	817	5225	> 817	.016	.2486D-06	.2666D+01	.200	.200	5.840
lund_a	147	1973	46	.181	.3670D-12	.7568D-06	.030	.010	.090
662_bus	662	1642	75	.007	.9240D-07	.7632D-06	.060	.130	.250
bcsstk08	1074	173532	70	.301	.3726D-11	.9106D-04	7.610	14.550	7.340
bcsstk20	485	1841	> 485	.016	.1306D-03	.1563D+06	.020	.070	1.520
nos1	237	642	235	.023	.7162D-10	.6574D-05	.000	.020	.260
685_bus	685	2487	89	.011	.6015D-07	.6194D-06	.070	.140	.370
bcsstk09	1083	32058	103	.055	.2527D-12	.6233D-06	1.670	.810	2.710
bcsstk21	3600	23150	329	.004	.1457D-10	.2729D-05	1.070	4.690	10.890
nos2	957	2610	> 957	.006	.1408D+01	.7134D+06	.010	.270	5.040
bcsstk10	1086	18973	130	.032	.1015D-09	.6182D-06	.390	.540	2.590
bcsstk22	138	446	58	.047	.3045D-11	.6459D-06	.010	.000	.040
nos3	960	21371	86	.046	.1853D-07	.6800D-06	1.090	.510	1.660
bcsstk11	1473	38420	312	.035	.7622D-09	.4301D-05	2.030	1.250	11.070
bcsstk23	3134	108323	2402	.022	.4080D-06	.3285D+03	14.630	6.370	202.900
bcsstm07	420	5453	14	.062	.1021D-07	.3733D-06	.100	.070	.090
nos4	100	527	31	.104	.4506D-05	.5486D-06	.000	.010	.020
bcsstk12	1473	38420	312	.035	.7622D-09	.4301D-05	2.050	1.240	11.050
nos5	468	8791	93	.080	.7064D-10	.5149D-06	.240	.130	.700
bcsstk13	2003	280098	459	.140	.8313D-10	.1141D-01	24.340	21.020	85.580
nos6	675	2320	45	.010	.1745D-09	.7795D-06	.050	.130	.180
bcsstk14	1806	96968	105	.059	.7051D-11	.4236D-04	7.620	3.410	8.160
bcsstk26	1922	22287	599	.012	.5974D-10	.5041D-03	.840	1.520	16.300
nos7	729	4431	40	.017	.4306D-07	.3674D-06	.280	.170	.230
bcsstk03	112	376	57	.059	.2571D-10	.9117D-04	.000	.010	.030
bcsstk15	3948	459920	203	.059	.1506D-09	.4669D-04	70.830	30.070	61.870
bcsstk27	1224	45301	81	.060	.3650D-10	.6214D-06	1.560	1.180	3.610
bcsstk04	132	3615	32	.412	.2600D-11	.1707D-06	.050	.030	.100
bcsstm12	1473	21390	10	.020	.5618D-05	.2999D-06	.800	.930	.230
bcsstk05	153	1834	57	.156	.9594D-12	.2055D-06	.020	.020	.110

Table 15: Test data of MPADD with reverse nested dissection ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2242	301	.003	.8148D-07	.8292D-06	.540	.390	1.720
bcsstk06	420	2296	406	.026	.9112D-11	.1891D-04	.420	.060	1.850
gr_30_30	900	1941	60	.005	.1739D-05	.9622D-06	.880	.250	.380
494_bus	494	984	170	.008	.3706D-07	.5564D-06	.080	.080	.380
bcsstk07	420	2296	406	.026	.9112D-11	.1891D-04	.420	.060	1.850
bcsstk19	817	2710	> 817	.008	.2081D+03	.5620D+08	.730	.210	4.860
lund_a	147	718	78	.066	.7824D-11	.1036D-05	.060	.010	.100
662_bus	662	1364	102	.006	.7326D-07	.6132D-06	.200	.130	.390
bcsstk08	1074	2869	155	.005	.1290D-10	.1993D-03	2.660	.360	1.330
bcsstk20	485	1774	235	.015	.1228D-05	.3478D+02	.180	.080	.760
nos1	237	473	159	.017	.7601D-10	.5924D-05	.020	.020	.170
685_bus	685	1535	147	.007	.2824D-07	.5436D-06	.270	.140	.530
bcsstk09	1083	3307	368	.006	.5232D-12	.1137D-05	2.500	.330	3.740
bcsstk21	3600	7278	624	.001	.1459D-10	.3765D-05	10.640	3.710	15.040
nos2	957	1913	784	.004	.1110D-06	.2273D-02	.380	.270	3.910
bcsstk10	1086	6090	694	.010	.8831D-11	.8261D-06	2.820	.350	8.840
bcsstk22	138	332	50	.035	.2905D-11	.1745D-06	.020	.010	.030
nos3	960	3653	186	.008	.3102D-07	.9722D-06	1.900	.270	1.740
bcsstk11	1473	8451	761	.008	.5011D-09	.7658D-05	6.310	.690	15.590
bcsstk23	3134	9624	> 3134	.002	.4950D+02	.1375D+11	20.270	2.960	91.080
bcsstm07	420	2190	50	.025	.3018D-07	.6549D-06	.380	.050	.220
nos4	100	207	40	.041	.5770D-05	.4773D-06	.020	.000	.020
bcsstk12	1473	8451	761	.008	.5011D-09	.7658D-05	5.940	.660	14.270
nos5	468	1066	212	.010	.6443D-10	.5437D-06	.360	.060	.690
bcsstk13	2003	10680	1497	.005	.1070D-09	.2162D-01	26.920	1.280	53.210
nos6	675	1349	73	.006	.6357D-09	.9618D-06	.290	.130	.260
bcsstk14	1806	11519	495	.007	.3448D-11	.7784D-04	14.010	1.080	14.880
bcsstk26	1922	9050	1401	.005	.3219D-10	.7124D-03	7.390	1.150	27.740
nos7	729	1457	77	.005	.4858D-08	.9325D-06	.560	.150	.320
bcsstk03	112	360	18	.057	.6433D-11	.7401D-04	.010	.000	.010
bcsstk15	3948	16399	678	.002	.1565D-09	.7943D-04	59.330	5.250	38.030
bcsstk27	1224	13660	166	.018	.7694D-10	.9038D-06	8.390	.590	4.440
bcsstk04	132	545	77	.062	.4758D-11	.5453D-06	.060	.000	.120
bcsstm12	1473	5416	24	.005	.1629D-04	.5098D-06	3.270	.630	.320
bcsstk05	153	570	113	.048	.3982D-11	.6418D-06	.060	.010	.140

Table 16: Test data of MPDROP without ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2491	107	.004	.6895D-07	.7606D-06	.440	.370	.630
bcsstk06	420	2931	294	.033	.6740D-11	.1883D-04	.360	.060	1.440
gr_30_30	900	3461	49	.009	.1667D-05	.7938D-06	.550	.230	.330
494_bus	494	1046	75	.009	.1121D-06	.9050D-06	.090	.070	.170
bcsstk07	420	2931	294	.033	.6740D-11	.1883D-04	.380	.060	1.430
bcsstk19	817	3421	> 817	.010	.1422D+02	.9475D+07	.530	.200	4.980
lund_a	147	874	78	.080	.5098D-11	.1281D-05	.060	.010	.120
662_bus	662	1510	74	.007	.2334D-06	.8446D-06	.150	.140	.260
bcsstk08	1074	4153	146	.007	.1834D-11	.4022D-04	1.490	.340	1.340
bcsstk20	485	1788	234	.015	.2551D-05	.2623D+02	.200	.080	.740
nos1	237	475	165	.017	.8415D-10	.4886D-05	.020	.020	.180
685_bus	685	1819	97	.008	.2197D-07	.6383D-06	.220	.130	.350
bcsstk09	1083	5255	359	.009	.5997D-12	.1238D-05	1.570	.340	3.980
bcsstk21	3600	7445	789	.001	.1884D-10	.4222D-05	4.130	3.770	19.110
nos2	957	1915	> 957	.004	.3596D-07	.1678D-02	.430	.260	4.650
bcsstk10	1086	7606	602	.013	.1009D-10	.9382D-06	2.700	.350	8.150
bcsstk22	138	387	45	.040	.3665D-11	.4334D-06	.020	.000	.030
nos3	960	5746	148	.012	.3605D-07	.9642D-06	1.290	.270	1.600
bcsstk11	1473	13221	710	.012	.5752D-09	.7987D-05	5.300	.840	15.590
bcsstk23	3134	15365	> 3134	.003	.6446D+02	.6305D+09	13.760	3.600	112.500
bcsstm07	420	2856	36	.032	.3854D-07	.8007D-06	.330	.060	.180
nos4	100	305	31	.060	.2168D-05	.3579D-06	.000	.010	.010
bcsstk12	1473	13221	710	.012	.5752D-09	.7987D-05	5.200	.840	16.370
nos5	468	1888	176	.017	.1041D-10	.9084D-06	.250	.060	.650
bcsstk13	2003	16269	1410	.008	.4797D-10	.9429D-02	15.380	1.780	60.190
nos6	675	1904	53	.008	.1921D-09	.5909D-06	.150	.140	.200
bcsstk14	1806	17405	291	.011	.3215D-11	.6576D-04	10.350	1.320	9.840
bcsstk26	1922	12923	893	.007	.4802D-10	.4717D-03	6.110	1.340	20.340
nos7	729	2673	38	.010	.3697D-07	.5281D-06	.310	.200	.210
bcsstk03	112	356	24	.056	.1240D-10	.9611D-04	.010	.000	.010
bcsstk15	3948	27926	584	.004	.1643D-09	.7584D-04	30.600	6.150	41.350
bcsstk27	1224	20485	134	.027	.2860D-10	.7106D-06	6.950	.760	4.140
bcsstk04	132	762	69	.087	.3117D-11	.2580D-06	.050	.010	.110
bcsstm12	1473	8322	17	.008	.2153D-04	.4481D-06	3.330	.730	.260
bcsstk05	153	874	93	.074	.7246D-12	.4455D-06	.050	.010	.140

Table 17: Test data of MPDROP with minimum degree ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2557	130	.004	.1502D-06	.9137D-06	.460	.410	.850
bcsstk06	420	2249	362	.025	.1987D-10	.1985D-04	.390	.060	1.630
gr_30_30	900	2717	57	.007	.8477D-06	.8158D-06	.740	.240	.350
494_bus	494	1073	85	.009	.1320D-06	.9840D-06	.080	.070	.230
bcsstk07	420	2249	362	.025	.1987D-10	.1985D-04	.410	.060	1.770
bcsstk19	817	2950	> 817	.009	.6431D+02	.3293D+09	.540	.200	4.660
lund_a	147	977	75	.090	.4831D-11	.7568D-06	.040	.010	.120
662_bus	662	1550	76	.007	.9253D-07	.5749D-06	.130	.130	.240
bcsstk08	1074	3265	146	.006	.2155D-11	.7445D-04	1.470	.340	1.250
bcsstk20	485	1550	> 485	.013	.3516D+02	.1055D+10	.150	.070	1.450
nos1	237	597	> 237	.021	.3025D-05	.6111D+00	.020	.020	.270
685_bus	685	1849	95	.008	.1315D-07	.5064D-06	.210	.130	.370
bcsstk09	1083	4149	360	.007	.3715D-12	.1054D-05	1.990	.350	3.800
bcsstk21	3600	9781	665	.002	.1857D-10	.3886D-05	8.520	3.920	16.940
nos2	957	2421	> 957	.005	.1655D+02	.3036D+06	.310	.250	4.920
bcsstk10	1086	7176	580	.012	.1796D-10	.9845D-06	2.990	.420	8.870
bcsstk22	138	356	77	.037	.7148D-11	.7454D-06	.010	.010	.050
nos3	960	4853	158	.011	.2418D-07	.8555D-06	1.760	.280	1.670
bcsstk11	1473	10373	726	.010	.2746D-09	.5818D-05	6.130	.790	15.890
bcsstk23	3134	12513	> 3134	.003	.7332D+02	.1089D+11	13.780	3.620	102.940
bcsstm07	420	2141	45	.024	.3912D-07	.8168D-06	.370	.060	.230
nos4	100	258	43	.051	.9017D-05	.7131D-06	.010	.000	.030
bcsstk12	1473	10373	726	.010	.2746D-09	.5818D-05	5.910	.820	15.930
nos5	468	1517	206	.014	.6438D-11	.6726D-06	.300	.080	.770
bcsstk13	2003	16083	1406	.008	.5543D-10	.1088D-01	14.120	1.720	59.180
nos6	675	1737	63	.008	.2254D-09	.8562D-06	.220	.130	.240
bcsstk14	1806	12940	350	.008	.3359D-11	.5822D-04	12.160	1.140	10.990
bcsstk26	1922	10263	1352	.006	.3899D-10	.6259D-03	5.990	1.210	28.130
nos7	729	1831	67	.007	.8705D-08	.5711D-06	.300	.130	.320
bcsstk03	112	320	76	.051	.1264D-10	.5848D-04	.010	.010	.040
bcsstk15	3948	20179	665	.003	.1711D-09	.7713D-04	45.110	5.520	39.370
bcsstk27	1224	17962	145	.024	.5302D-10	.9221D-06	8.240	.650	4.300
bcsstk04	132	612	79	.070	.3845D-11	.5440D-06	.060	.010	.120
bcsstm12	1473	6716	21	.006	.2570D-04	.5503D-06	2.970	.650	.290
bcsstk05	153	836	102	.071	.1262D-11	.4915D-06	.050	.010	.150

Table 18: Test data of MPDROP with nested dissection ordering.

name	n	nz(R)	niter	d(R)	$\| x^* - x \|_2$	$\| b - Ax \|_2$	T_1	T_2	T_3
1138_bus	1138	2543	217	.004	.1875D-06	.8850D-06	.670	.380	1.250
bcsstk06	420	2137	369	.024	.3889D-11	.2109D-04	.360	.050	1.610
gr_30_30	900	2971	52	.007	.4248D-06	.6552D-06	1.110	.250	.330
494_bus	494	1069	115	.009	.1734D-06	.8838D-06	.110	.070	.260
bcsstk07	420	2137	369	.024	.3889D-11	.2109D-04	.370	.060	1.630
bcsstk19	817	3048	> 817	.009	.5096D+02	.4942D+09	.710	.200	4.800
lund_a	147	952	78	.088	.3018D-11	.1044D-05	.050	.000	.120
662_bus	662	1538	85	.007	.1739D-06	.9486D-06	.180	.130	.280
bcsstk08	1074	3107	208	.005	.1434D-10	.1335D-03	3.290	.360	1.750
bcsstk20	485	1632	> 485	.014	.3897D+02	.1013D+10	.170	.070	1.470
nos1	237	597	> 237	.021	.1392D-09	.8838D-05	.020	.020	.260
685_bus	685	1813	119	.008	.2403D-07	.6766D-06	.260	.130	.450
bcsstk09	1083	3931	331	.007	.4177D-12	.1178D-05	2.810	.360	3.670
bcsstk21	3600	11232	574	.002	.1218D-10	.3659D-05	9.160	4.080	15.130
nos2	957	2421	> 957	.005	.3348D+01	.4756D+06	.380	.280	5.100
bcsstk10	1086	8541	586	.014	.1159D-10	.9467D-06	2.750	.360	8.660
bcsstk22	138	356	74	.037	.3531D-11	.3761D-06	.010	.010	.070
nos3	960	5343	164	.012	.2279D-07	.8571D-06	2.230	.310	1.640
bcsstk11	1473	11295	661	.010	.7256D-09	.6477D-05	6.600	.750	13.510
bcsstk23	3134	12724	> 3134	.003	.3551D+02	.4341D+10	18.710	3.440	98.390
bcsstm07	420	1995	53	.023	.3577D-07	.7994D-06	.340	.050	.210
nos4	100	236	48	.047	.3594D-05	.6078D-06	.010	.000	.030
bcsstk12	1473	11295	661	.010	.7256D-09	.6477D-05	6.550	.750	13.750
nos5	468	1456	233	.013	.3277D-10	.8218D-06	.320	.070	.810
bcsstk13	2003	15701	1532	.008	.6375D-10	.1353D-01	28.430	1.510	59.010
nos6	675	1765	61	.008	.1942D-09	.5047D-06	.230	.130	.230
bcsstk14	1806	14056	450	.009	.4764D-11	.6461D-04	16.400	1.160	14.020
bcsstk26	1922	10451	1584	.006	.4501D-10	.7936D-03	5.920	1.180	33.200
nos7	729	2070	65	.008	.1023D-07	.4019D-06	.500	.160	.290
bcsstk03	112	376	57	.059	.2571D-10	.9117D-04	.010	.010	.030
bcsstk15	3948	18428	685	.002	.1782D-09	.7336D-04	75.090	5.450	38.950
bcsstk27	1224	22102	130	.029	.4958D-10	.8429D-06	8.200	.690	4.100
bcsstk04	132	568	83	.065	.6425D-11	.7833D-06	.060	.000	.130
bcsstm12	1473	7108	24	.007	.1806D-04	.5856D-06	3.520	.700	.350
bcsstk05	153	808	101	.069	.3117D-11	.6286D-06	.060	.010	.140

Table 19: Test data of MPDROP with reverse nested dissection ordering.

References

[1] E. Chu, A. George, J. Liu, and E. Ng, *SPARSPAK: Waterloo sparse matrix package, user's guide for SPARSPAK-A*, Tech. Rep. CS-84-36, Department of Computer Science, University of Waterloo, 1984.

[2] I. S. Duff and G. A. Meurand, *The effect of ordering on preconditioned conjugate gradients*, BIT, 29 (1989), pp. 635–657.

[3] V. Eijkhout, *Beware of unperturbed modified incomplete factorizations*, Tech. Rep. CSRD Report No. 1109, CSRD, University of Illinois at Urbana-Champaign, 1991.

[4] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins, 2nd ed., 1989.

[5] A. Jennings and G. M. Malik, *Partial elimination*, Journal of the Institute of Mathematics and Its Applications, 20 (1977), pp. 307–316.

[6] T. A. Manteuffel, *Shifted incomplete Choleski factorization*, in Sparse Matrix Proceedings 1978, I. S. Duff and G. Stewart, eds., Philadelphia,PA, 1979, SIAM Publications.

[7] X. Wang, *Incomplete factorization preconditioning for linear least squares problems*, PhD thesis, University of Illinois Urbana-Champaign, 1993. Also available as Technical Report # UIUCDCS-R-93-1834, Department of Computer Science, University of Illinois at Urbana – Champaign.

[8] X. Wang, K. A. Gallivan, and R. Bramley, *CIMGS: A incomplete orthogonalization preconditioner*, Tech. Rep. 393, Department of Computer Science, Indiana University, Bloomington, IN 47405, 1993.

[9] G. Wittum and F. Liebau, *On truncated incomplete decompositions*, BIT, 29 (1989), pp. 719–740.