Direct Methods and Algebraic Preconditioners for Solving Large and Sparse Systems of Linear Equations

Miroslav Tůma

Institute of Computer Science

Academy of Sciences of the Czech Republic

tuma@cs.cas.cz

Presentation supported by the project
by the Grant Agency of the Czech Republic
under No. 108/11/095

Liberec, January 23-27, 2012

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

Introductory notes

- Assuming basic knowledge of algebraic iterative (Krylov space) and direct (dense) solvers (elimination/factorization/solve)
- Many techniques can be formulated for both SPD and nonsymmetric cases with only slight algorithmic (but possibly strong theoretical) differences. Orientation in variants of Cholesky and LU decompositions is assumed.
- We will concentrate here on purely algebraic techniques which often serve as building blocks for more complex approaches.
- Some important techniques are not mentioned at all (MG/ML preconditioners, DD techniques, row projection techniques).
- Some ideas and techniques are only mentioned (block algorithms) Only preconditioning of real systems is considered here.

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

The problem

$$Ax = b$$

- Direct methods
- Iterative methods
- Practical boundaries between them more and more fuzzy.
- But they are principially different.

Direct methods and algebraic preconditioners

Direct methods

- Direct methods: the name traditionally used for the approach based on decomposition and subsequent substitutions
- The most simple case: $A \to LL^T$ or LDL^T or LU

  In principal = Gaussian elimination. Modern (decompositional) form based a lot on the work of Householder (end of 1950's)
  - Occasionally other decompositions
  - Most work is in the (Cholesky, indefinite, LU) decomposition.
  - But: It is the computer model (sequential, concurrent processors, multicore, GPU) which decides about the relative complexity of the two steps.
- The algorithms can be made more efficient/stable by the use of additional techniques used before, after or during the decomposition.
- In particular, solution can be made more precise by an auxiliary iterative method.

Direct methods and algebraic preconditioners

Iterative methods

- Iterative method are usually accompanied by a problem transformation based on a direct method called preconditioner.
- Algebraic preconditioners are tools to convert the problem $Ax = b$ into the one which is easier to solve. They are typically expressed in matrix form as a transformation like:

$$MAx = Mb$$

- M can be then used to apply approximation to A^{-1} to vectors used in the iterative method.
- In practice, it can store approximation to A or A^{-1} (approximate inverse).
- The computation is often based on a relaxation of a direct method, but not always.

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

Sparsity

Sparsity: taking into account the structure of matrix nonzeros

- Absolutely crucial for direct methods: complexity for generally dense matrices, sequential case: $O(n^3)$ factorization, $O(n^2)$ substitutions
- Useful for iterative methods as well: repeated multiplications
- sparse matrix: its combinatorial structure of zeros and nonzeros can be exploited
- complexity in the sparse case depends on the decomposition model (implementation, completeness/incompleteness)

Dense matrix

dim	space	dec time (s)
3000	4.5M	5.72
4000	8M	14.1
5000	12.5M	27.5
6000	18M	47.8

Sparse matrix

dim	space	dec time (s)
10000	40k	0.02
90000	0.36M	0.5
1M	4M	16.6
2M	8M	49.8

SPARSITY!

Sparse decompositions

- Exact (direct) decompositions $A = LL^T$, LU (up to the floating-point model) $\rightarrow$ Direct methods

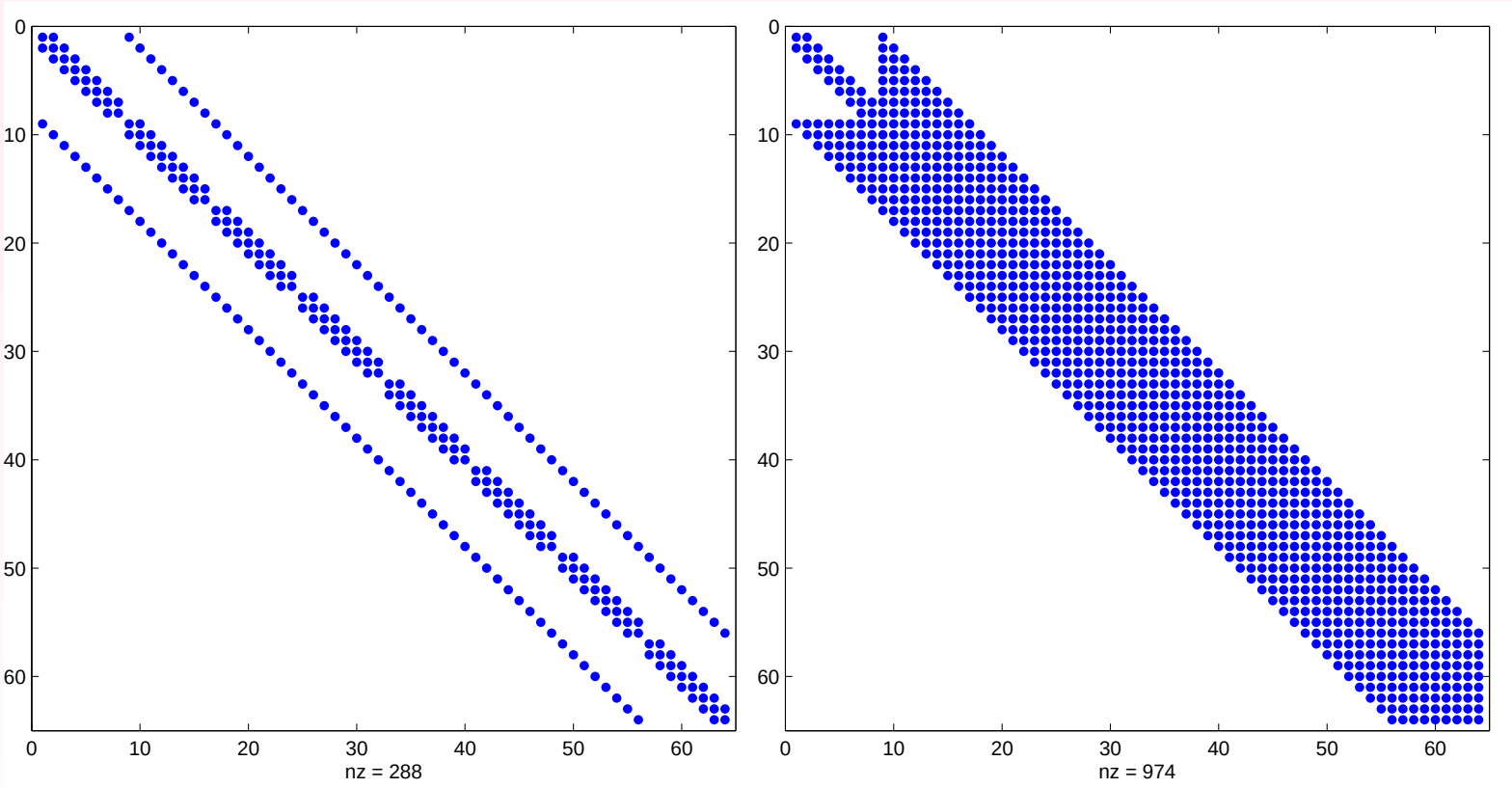
- Inexact processes able to provide approximation to A^{-1}
  - incomplete decompositions ($A \approx LL^T$, LU etc.)
  - incomplete inverse decompositions ($A^{-1} \approx ZZ^T$, WZ^T etc.)
- $\rightarrow$ Preconditioners

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

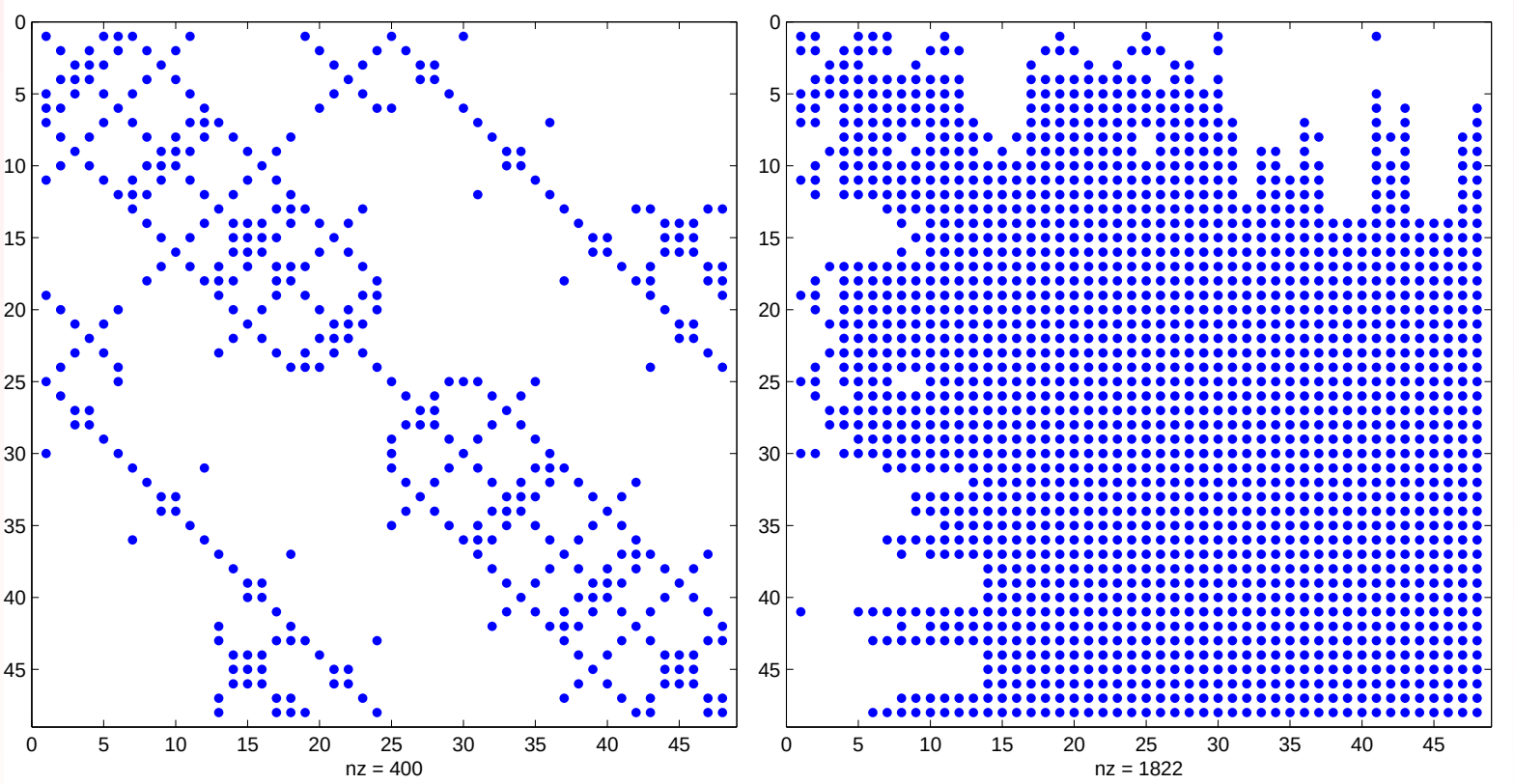
Direct methods

Direct decomposition may fill



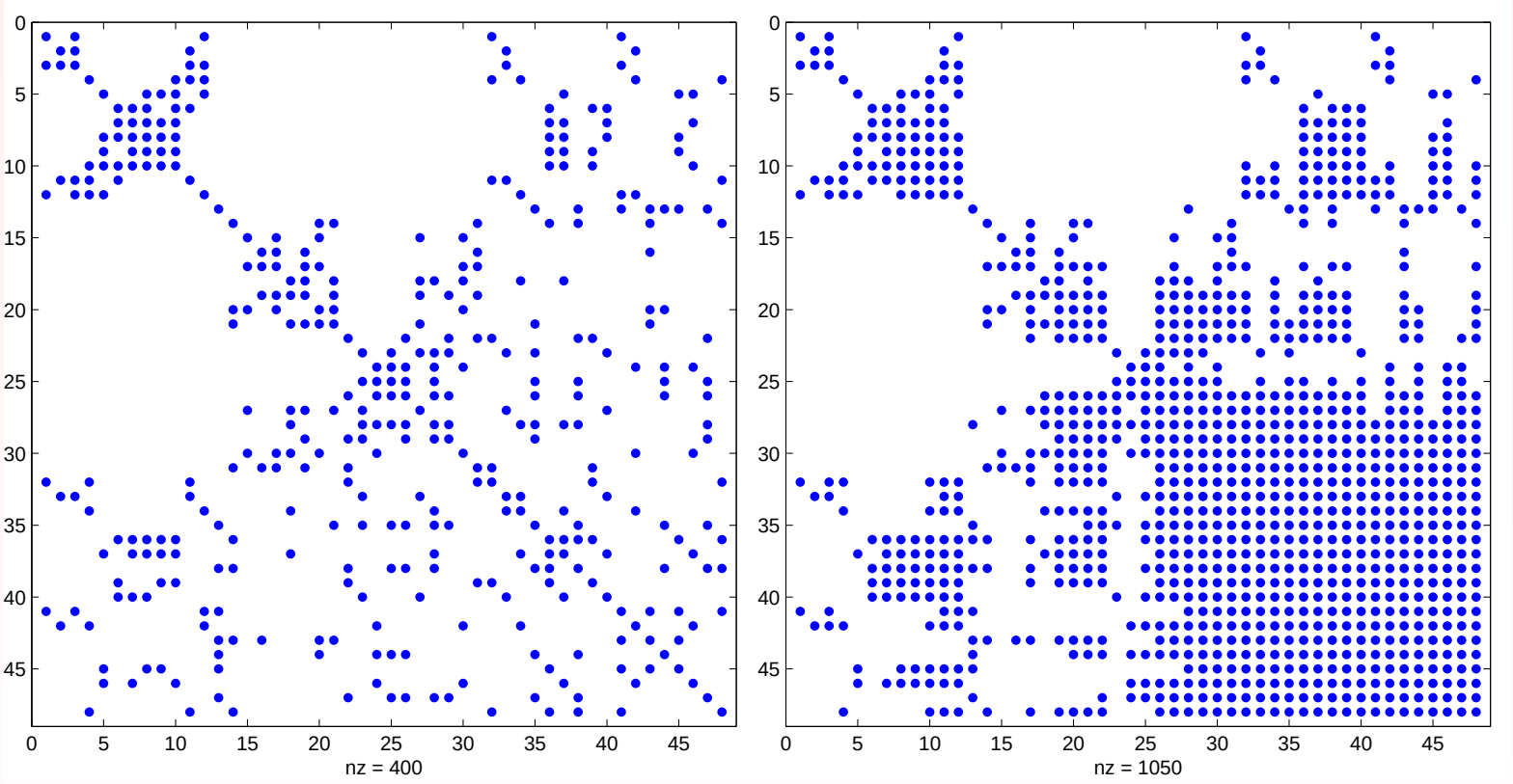
Direct methods

Direct decomposition may fill



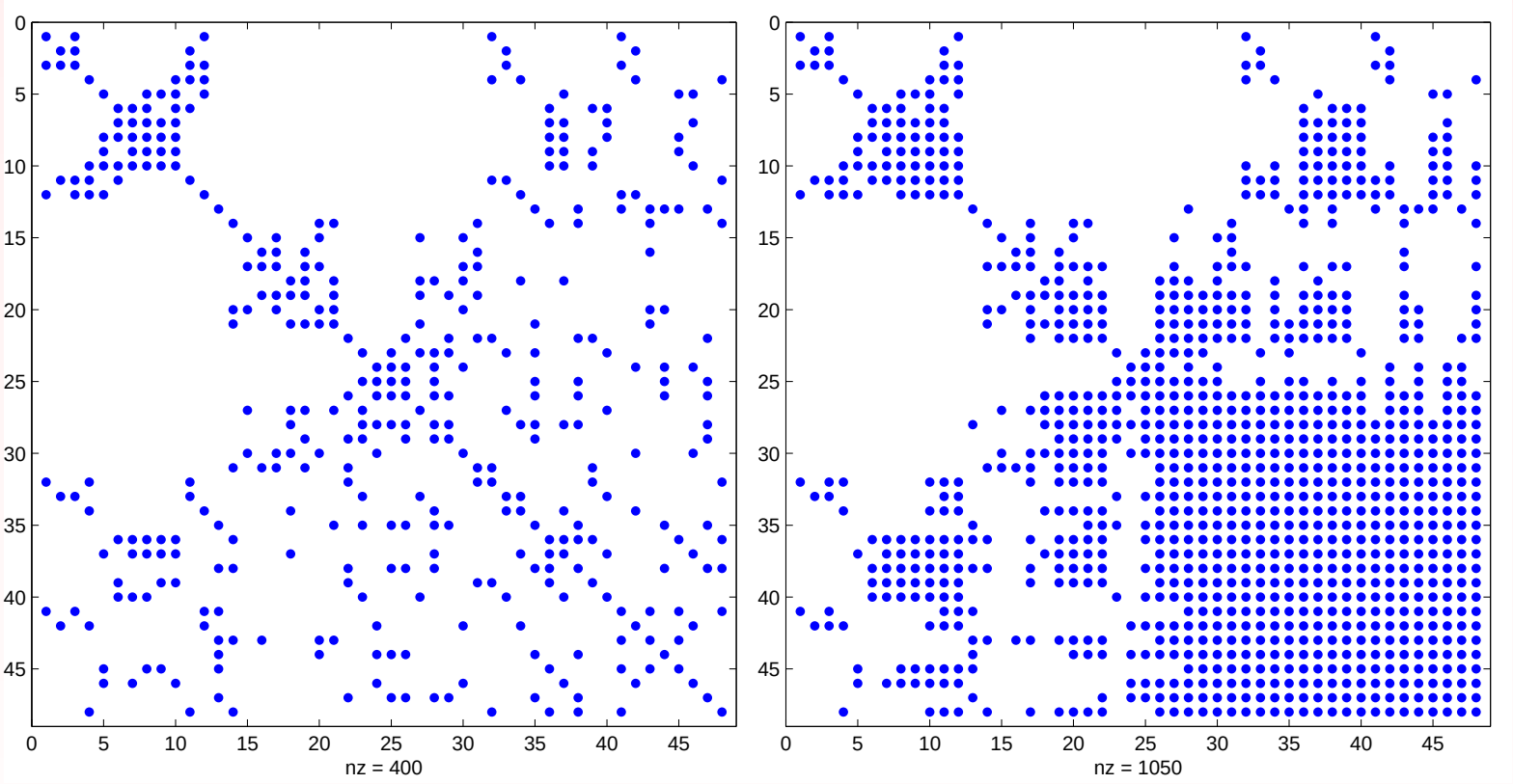
Direct methods

Direct decomposition may fill



Direct methods

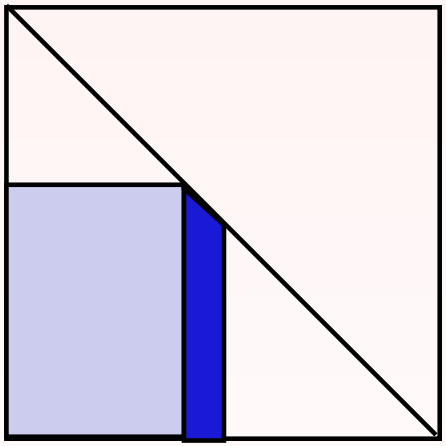
Direct decomposition may fill

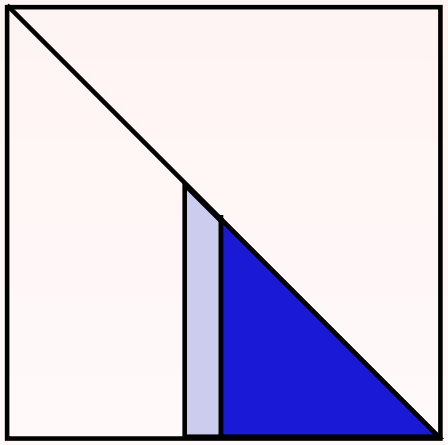


- Need to describe the fill-in: 1) describe it 2) avoid it
- Need to exploit the fill-in structure algorithmically
- Or ... we can cut the fill-in and perform an incomplete process ... later

Direct methods - decomposition schemes

- Left-looking schemes
- Right-looking schemes



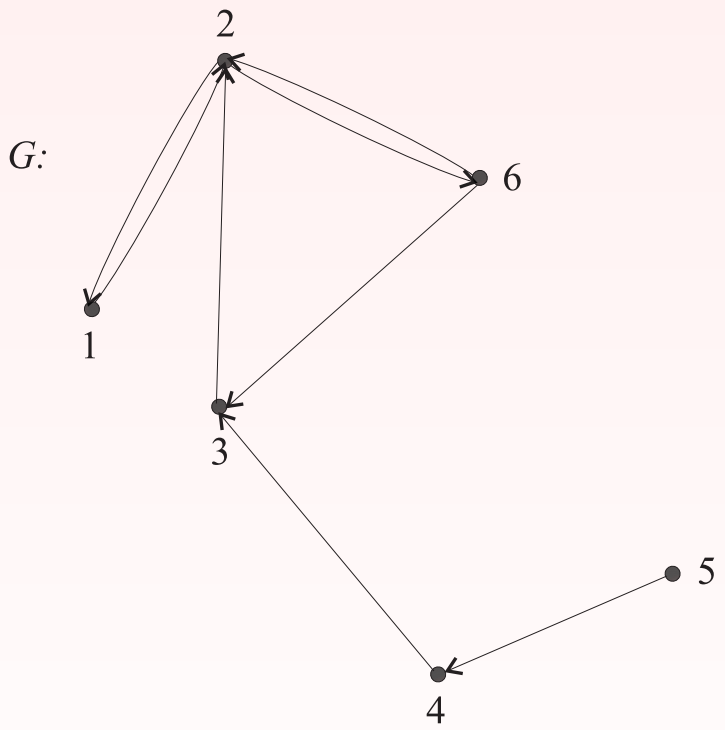


- Although some techniques and theorems are more general we will deal with the SPD systems only

Fill-in description

- Combinatorial structure of zeros and nonzeros $\rightarrow$ graphs

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

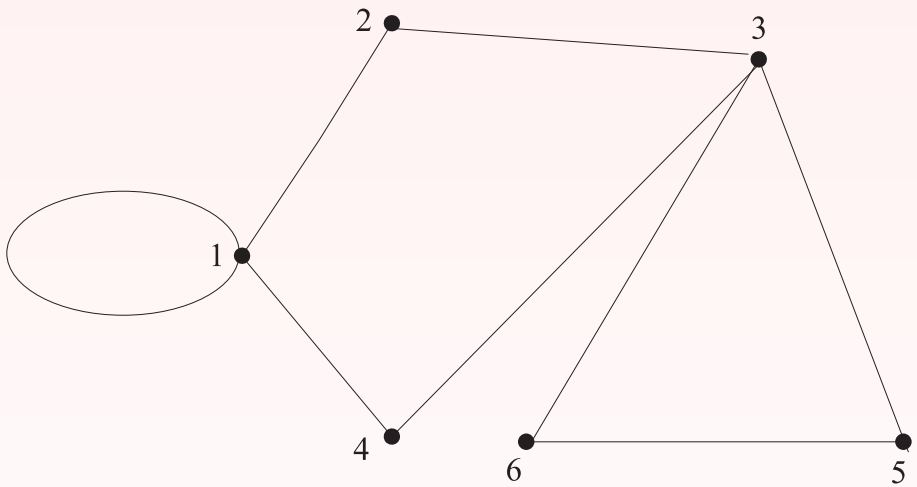


- Fill-in changes during the decomposition: dynamic description
- Data structures, implementation with respect to the architecture

Fill-in description

- Combinatorial structure of zeros and nonzeros $\rightarrow$ graphs

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

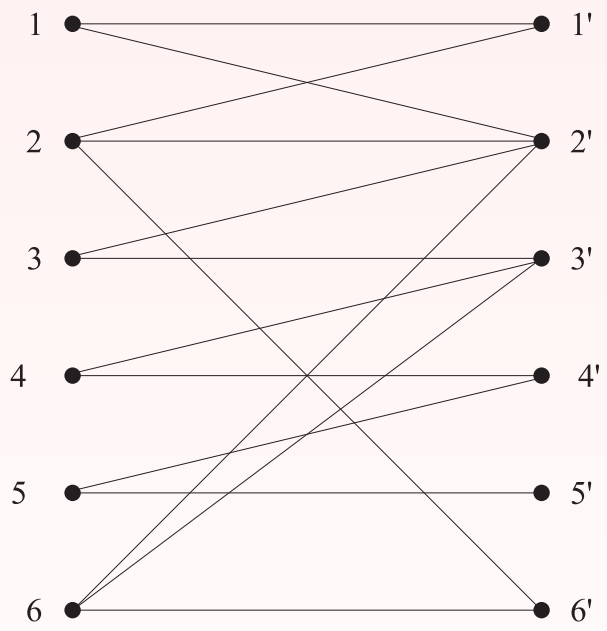


- Fill-in changes during the decomposition: dynamic description
- Data structures, implementation with respect to the architecture

Fill-in description

- Combinatorial structure of zeros and nonzeros $\rightarrow$ graphs

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



- Fill-in changes during the decomposition: dynamic description
- Data structures, implementation with respect to the architecture

Direct methods

The fill-in changes during the decomposition

- Arrow matrix - original matrices

$$\begin{pmatrix} * & * & * & * & * \\ * & * & & & \\ * & & * & & \\ * & & & * & \\ * & & & & * \end{pmatrix} \qquad \begin{pmatrix} * & & & & * \\ & * & & & * \\ & & * & & * \\ & & & * & * \\ * & * & * & * & * \end{pmatrix}$$

Direct methods

The fill-in changes during the decomposition

- Arrow matrix - structure after elimination

$$\begin{pmatrix} * & * & * & * & * \\ * & * & f & f & f \\ * & f & * & f & f \\ * & f & f & * & f \\ * & f & f & f & * \end{pmatrix} \qquad \begin{pmatrix} * & & & & * \\ & * & & & * \\ & & * & & * \\ & & & * & * \\ * & * & * & * & * \end{pmatrix}$$

Direct methods

The fill-in changes during the decomposition

- Arrow matrix - structure after elimination

$$\begin{pmatrix} * & * & * & * & * \\ * & * & f & f & f \\ * & f & * & f & f \\ * & f & f & * & f \\ * & f & f & f & * \end{pmatrix} \qquad \begin{pmatrix} * & & & & * \\ & * & & & * \\ & & * & & * \\ & & & * & * \\ * & * & * & * & * \end{pmatrix}$$

- How to describe and avoid the fill-in dynamically?

Dynamic development of the fill-in

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

Direct methods

Dynamic development of the fill-in

$$\begin{pmatrix} * & * & & & \\ & * & & * & \\ * & & * & & * \\ & * & * & * & \\ & * & & & * \end{pmatrix} \qquad \begin{pmatrix} * & * & & & \\ & * & & * & \\ * & f & * & & * \\ & * & * & * & \\ & * & & & * \end{pmatrix}$$

elimination of the first row and column

Direct methods

Dynamic development of the fill-in

$$\begin{pmatrix} * & * & & & \\ & * & & * & \\ * & & * & & * \\ & * & * & * & \\ & * & & & * \end{pmatrix} \qquad \begin{pmatrix} * & * & & & \\ & * & & * & \\ * & f & * & f & * \\ & * & * & * & \\ & * & & f & * \end{pmatrix}$$

elimination of the second row and column

Dynamic development of the fill-in

$$\begin{pmatrix} * & * & & & \\ & * & & * & \\ * & & * & & * \\ & * & * & * & \\ & * & & & * \end{pmatrix} \qquad \begin{pmatrix} * & * & & & \\ & * & & * & \\ * & f & * & f & * \\ & * & * & * & f \\ & * & & f & * \end{pmatrix}$$

elimination of the third row and column

Direct methods

Dynamic development of the fill-in

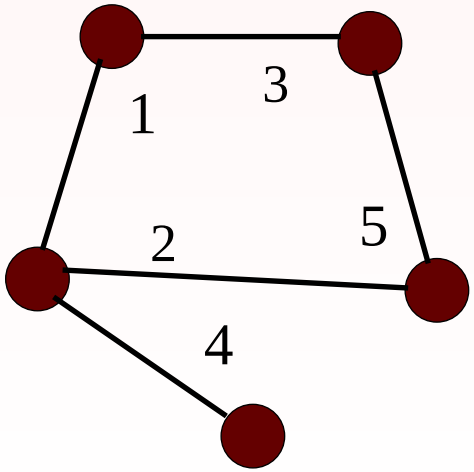
$$\begin{pmatrix} * & * & & & \\ & * & & * & \\ * & & * & & * \\ & * & * & * & \\ & * & & & * \end{pmatrix} \qquad \begin{pmatrix} * & * & & & \\ & * & & * & \\ * & f & * & f & * \\ & * & * & * & f \\ & * & & f & * \end{pmatrix}$$

elimination of the third row and column

- Formal description: sequence $\mathcal{E}$ of elimination matrices
- How should be $\mathcal{E}$ captured in the graph form?
- How should be $\mathcal{E}$ stored in the computer?

Dynamic development of the fill-in: II.

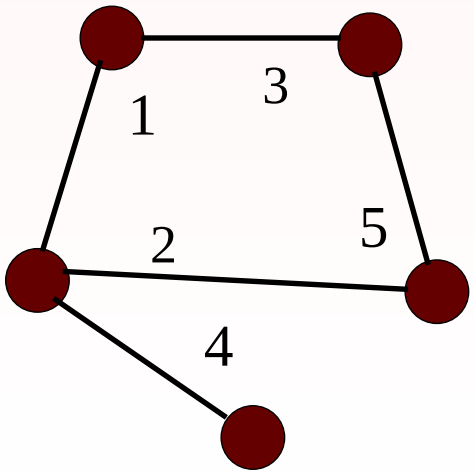
$$\begin{pmatrix} * & * & * & & \\ * & * & & * & * \\ * & & * & & * \\ & * & & * & \\ & * & * & & * \end{pmatrix}$$

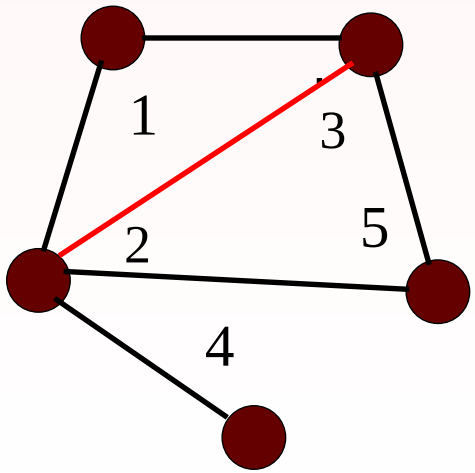


Dynamic development of the fill-in: II.

$$\begin{pmatrix} * & * & * & & \\ * & * & & * & * \\ * & & * & & * \\ & * & & * & \\ & * & * & & * \end{pmatrix} \qquad \begin{pmatrix} * & * & * & & \\ * & * & f & * & * \\ * & f & * & & * \\ & * & & * & \\ & * & * & & * \end{pmatrix}$$

elimination of the first row and column

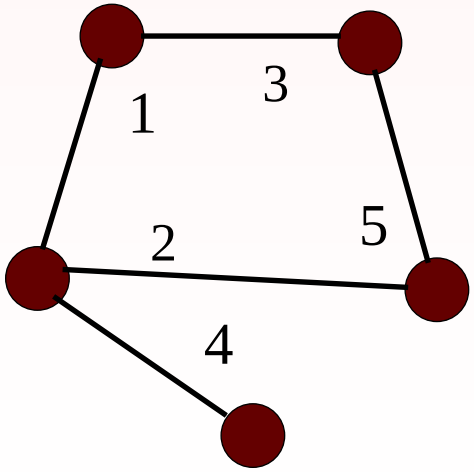


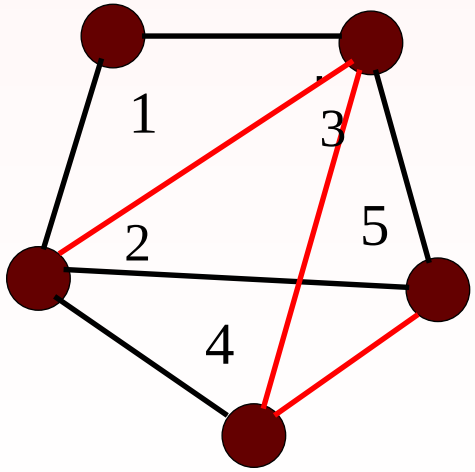


Dynamic development of the fill-in: II.

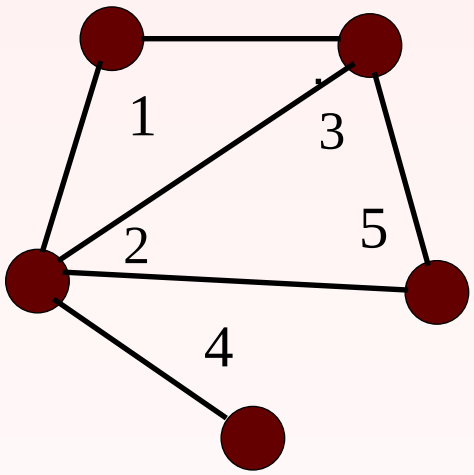
$$\begin{pmatrix} * & * & * & & \\ * & * & & * & * \\ * & & * & & * \\ & * & & * & \\ & * & * & & * \end{pmatrix} \qquad \begin{pmatrix} * & * & * & & \\ * & * & f & * & * \\ * & f & * & f & * \\ & * & f & * & f \\ & * & * & f & * \end{pmatrix}$$

elimination of the second row and column

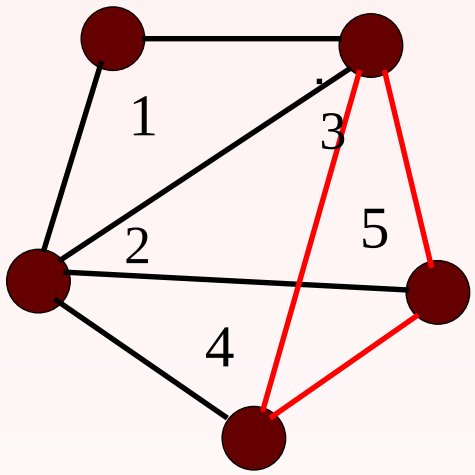




Dynamic development of the fill-in: III.

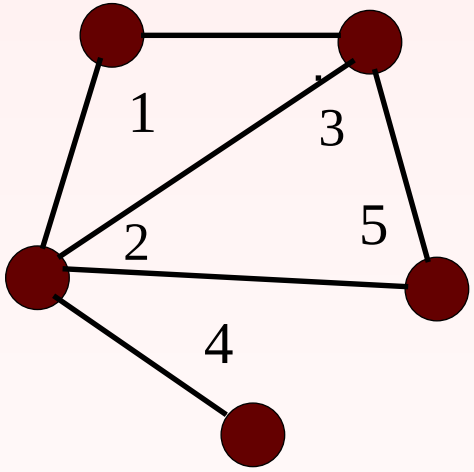


after 1st step

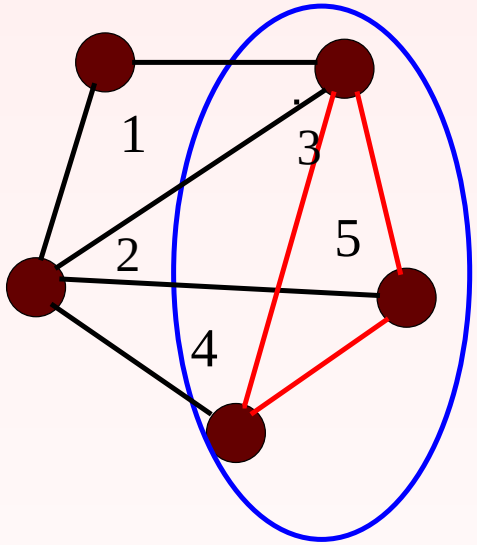


after second step

Dynamic development of the fill-in: III.



after 1st step

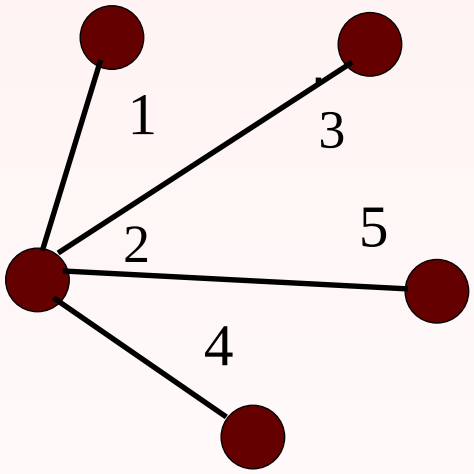


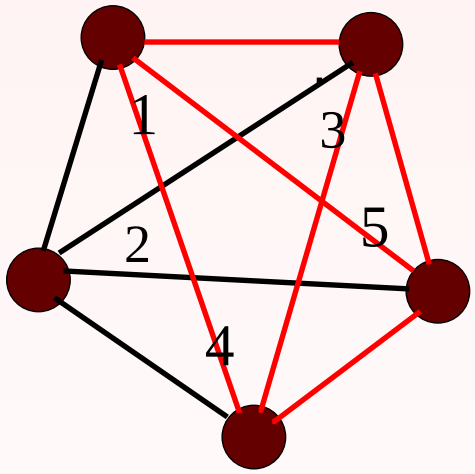
after second step

- The elimination step induces a clique in the graph model
- Storing clique instead of a subgraph $\rightarrow$ complexity?
- A clique can be stored just implicitly - storing entries that caused it!

Memory considerations

- 1st approximation: Data structures for direct methods (here we have just the SPD case): just data structures for recursive storing of the cliques caused by the elimination?





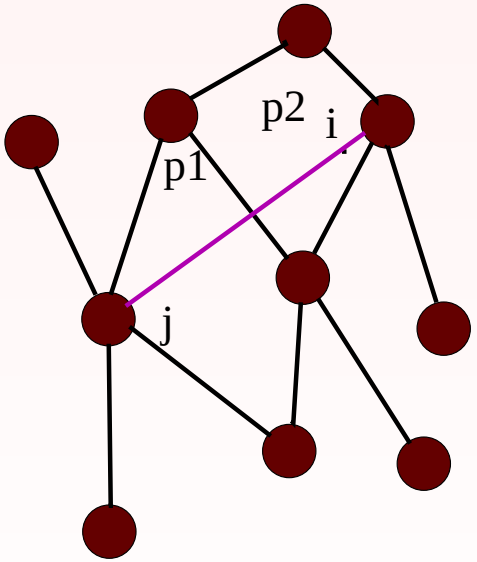
$\{\{1,3\},\{1,4\},\{1,5\},\ \{3,4\},\ \{3,5\},\{4,5\}\} \quad \rightarrow \quad \{1,3,4,5\}$

- too local, no use of the row/column character of the decomposition
- The final "elimination graph" is called the filled graph

Global description of the fill-in

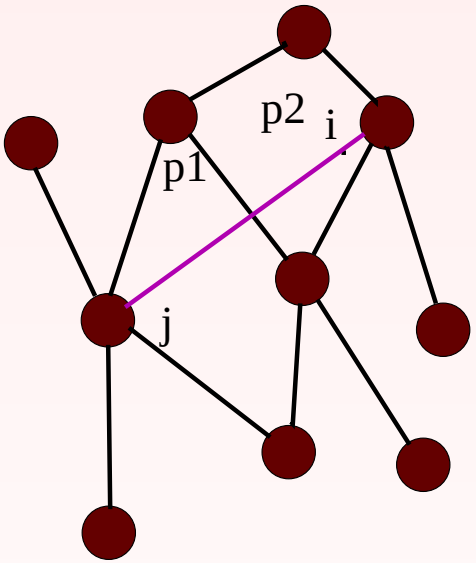
Algorithm

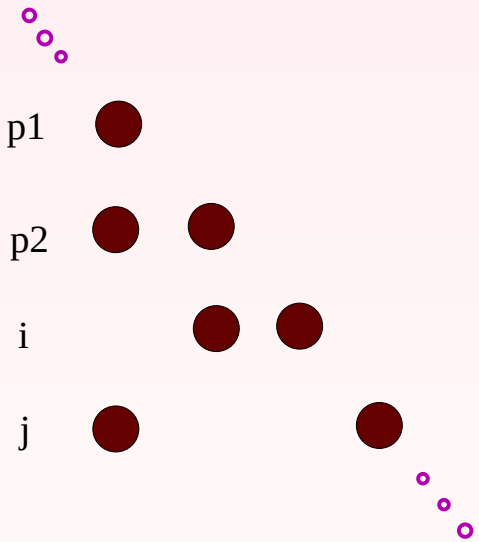
*(**Fill-in path theorem (Rose, Tarjan, Lueker, 1976)**) Let $n > i > j$. Then $l_{ij} \neq 0 \Leftrightarrow \exists$ a path $x_i, x_{p_1}, \ldots, x_{p_t}, x_j$ in $G(A)$ such that $(\forall l \in \hat{t})(p_l < \min(i,j))$.*



- Nice global description, but somewhat implicit as well. Does not seem to be an algorithmic one.

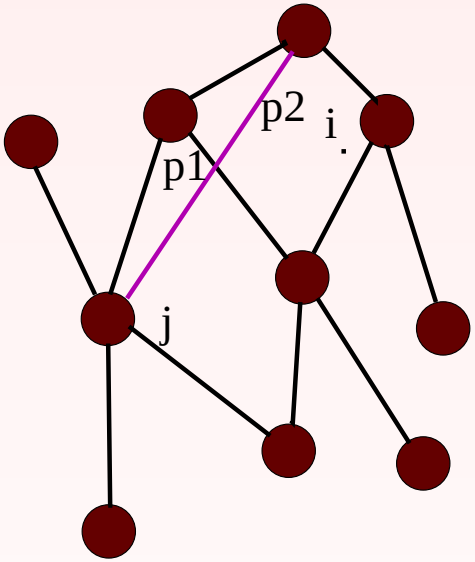
Global description of the fill-in: II.

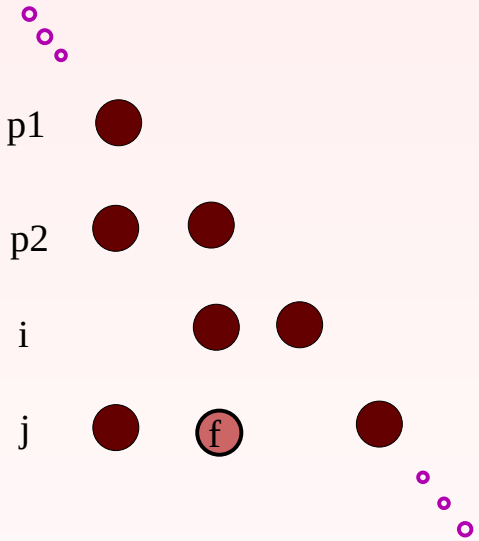




- Graph interpretation of elimination based on the fill-in path theorem

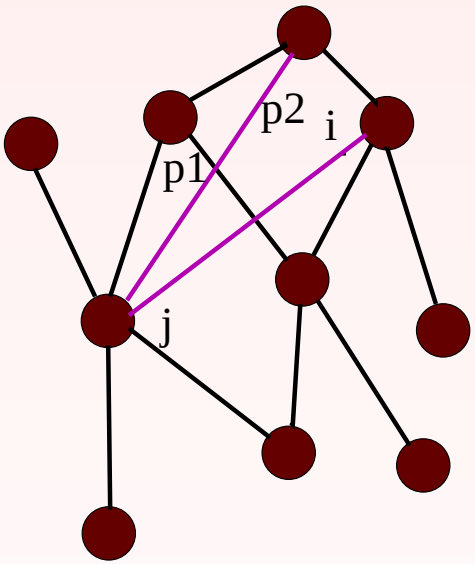
Global description of the fill-in: II.

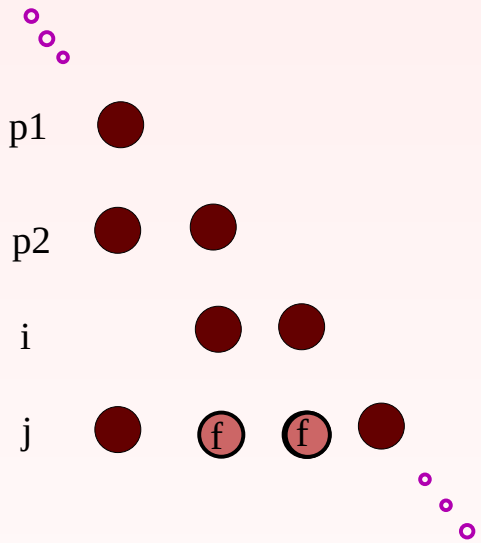




- Graph interpretation of elimination based on the fill-in path theorem

Global description of the fill-in: II.

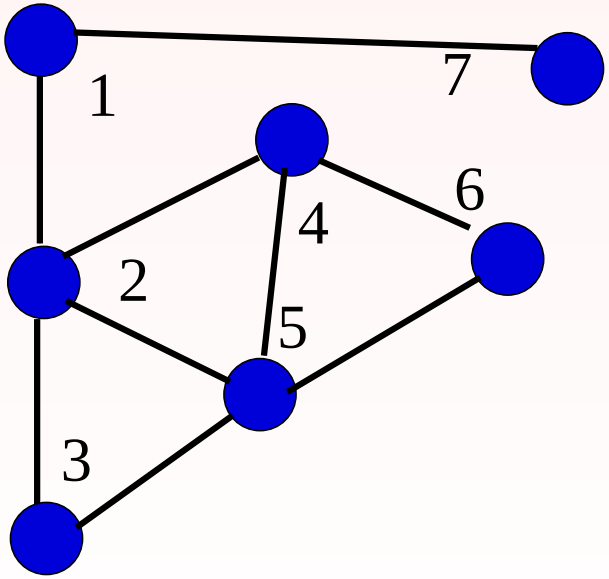




- We need some simple data structure enabling to control the fill-in generation
- We need a symbolic description first for setting up the data structure
- The enabler is called the elimination tree

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?

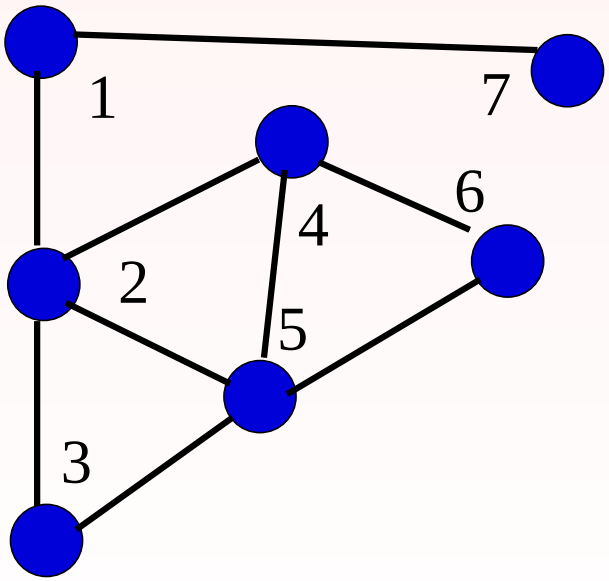


stack

traversal

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?

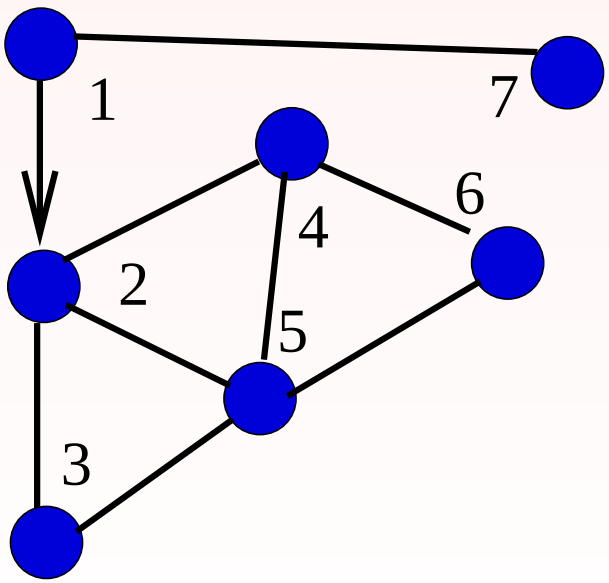


stack $\boxed{1}$

traversal $\boxed{1}$

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

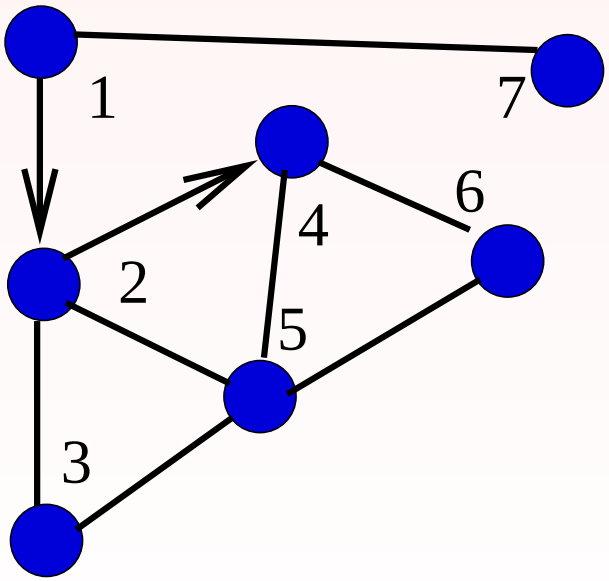
1	2

traversal

1	2

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

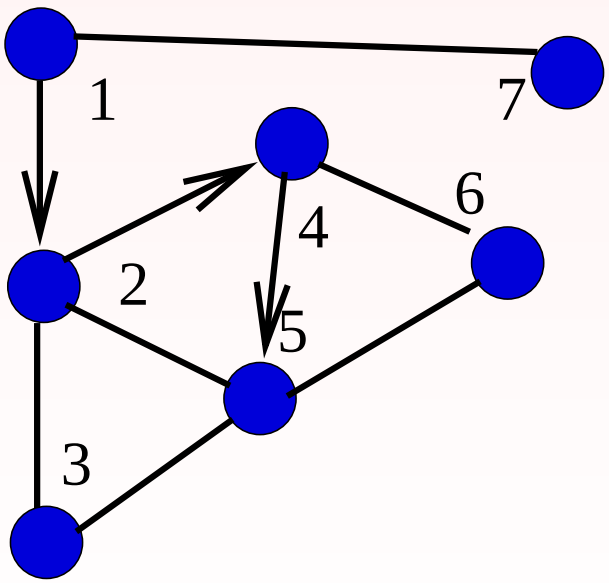
1	2	4

traversal

1	2	4

The elimination tree

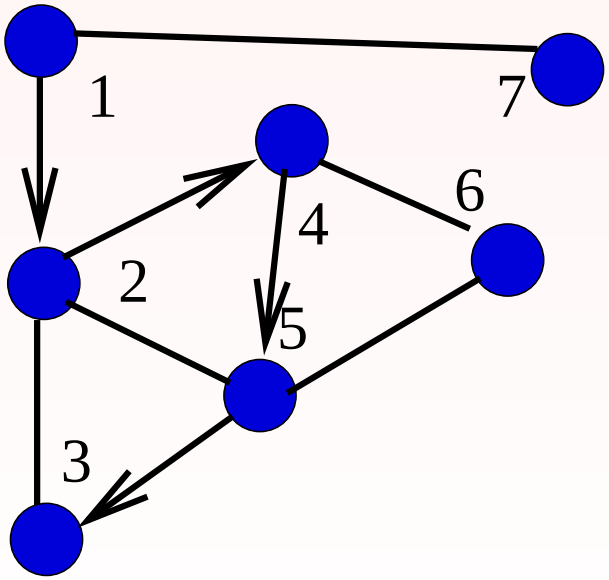
- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack	1	2	4	5
traversal	1	2	4	5

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

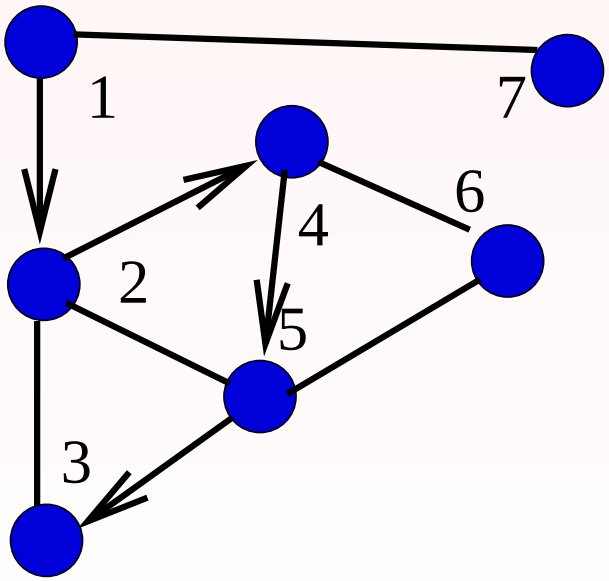
1	2	4	5	3

traversal

1	2	4	5	3

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

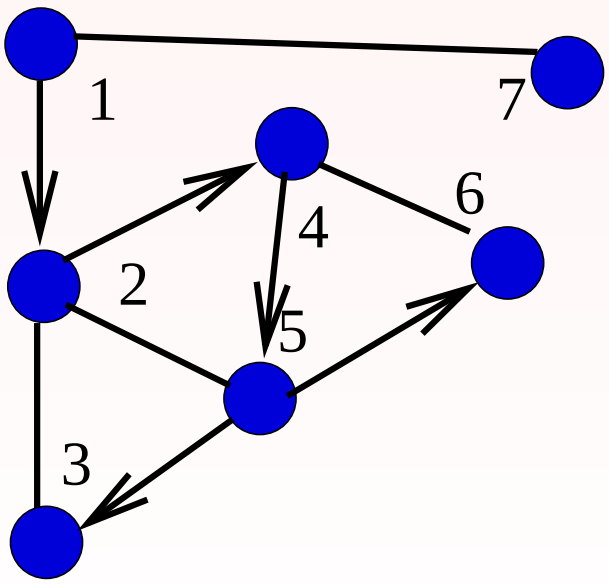
1	2	4	5

traversal

1	2	4	5	3

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

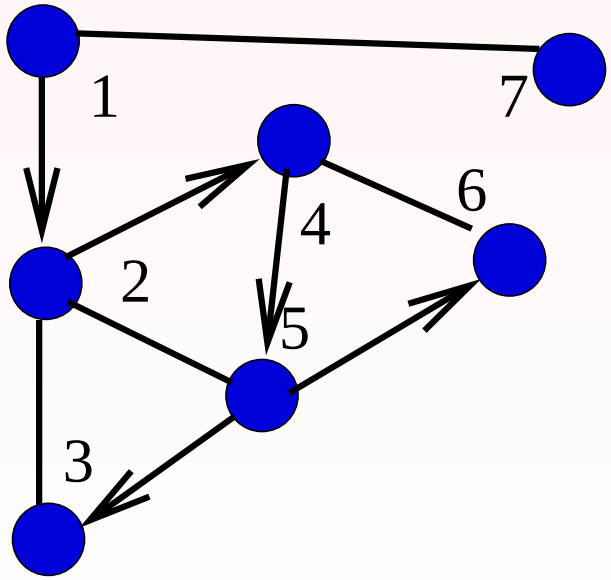
1	2	4	5	6

traversal

1	2	4	5	3	6

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

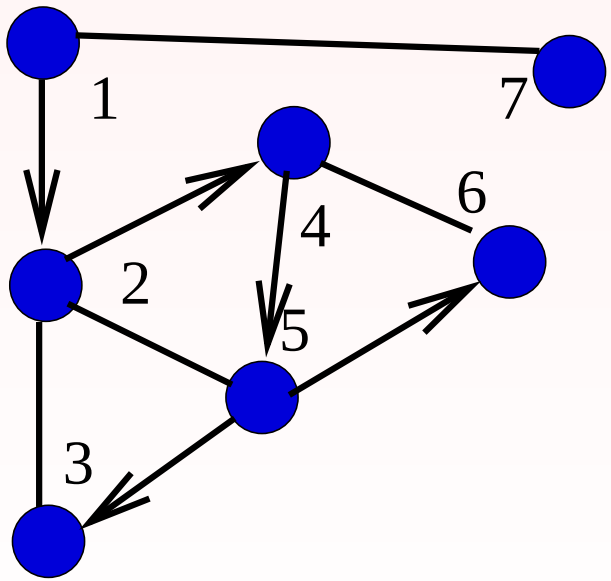
1	2	4	5

traversal

1	2	4	5	3	6

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

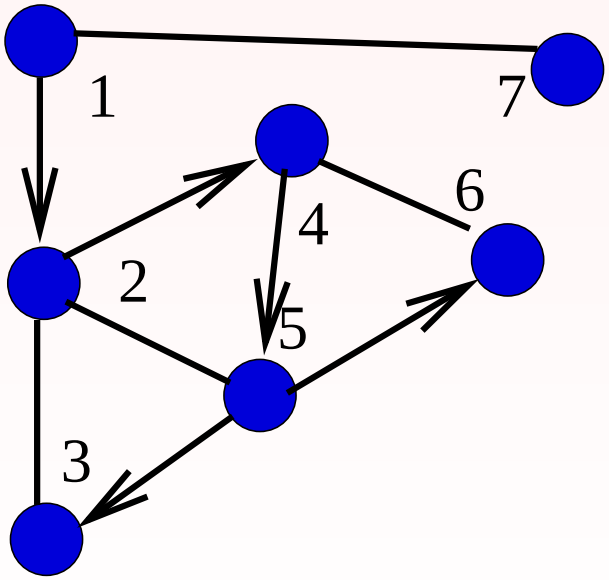
1	2	4

traversal

1	2	4	5	3	6

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

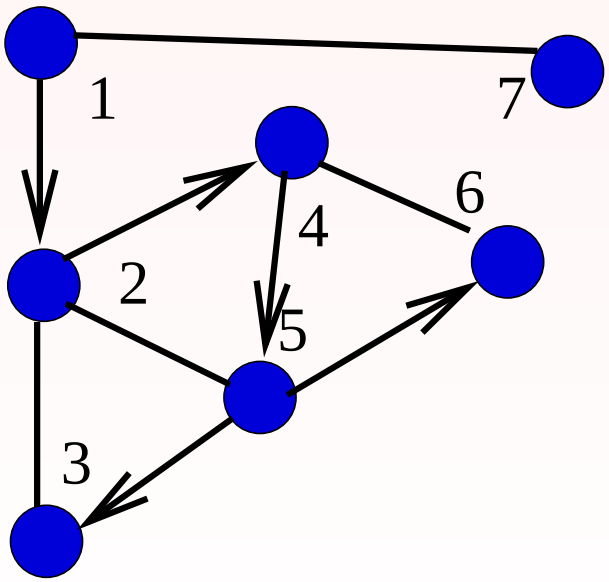
1	2

traversal

1	2	4	5	3	6

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

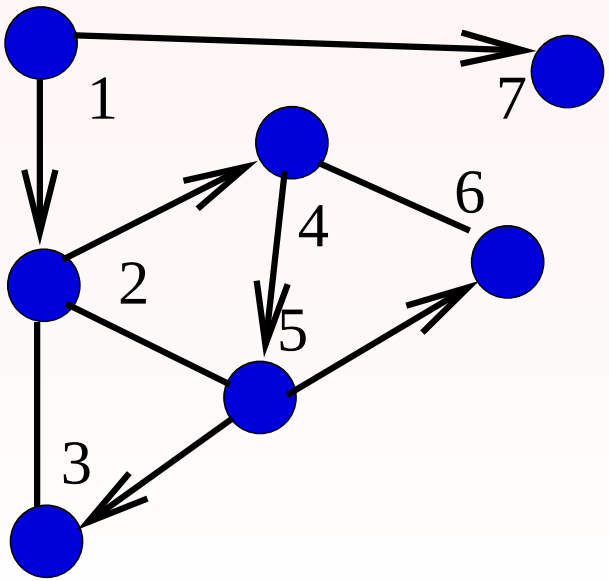
1

traversal

1	2	4	5	3	6

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

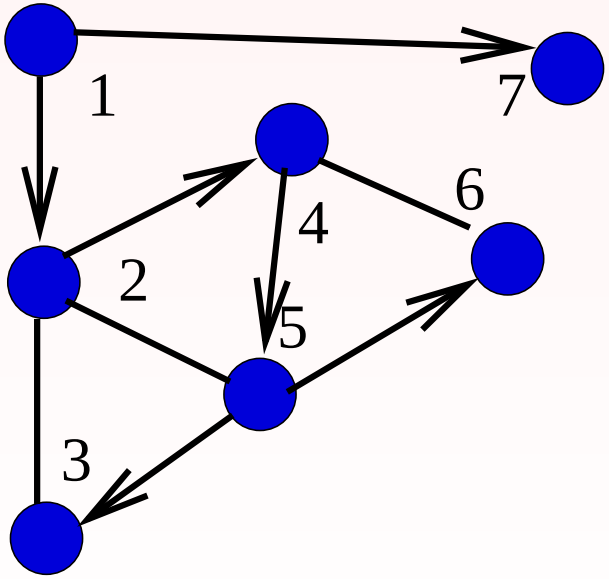
1	7

traversal

1	2	4	5	3	6	7

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

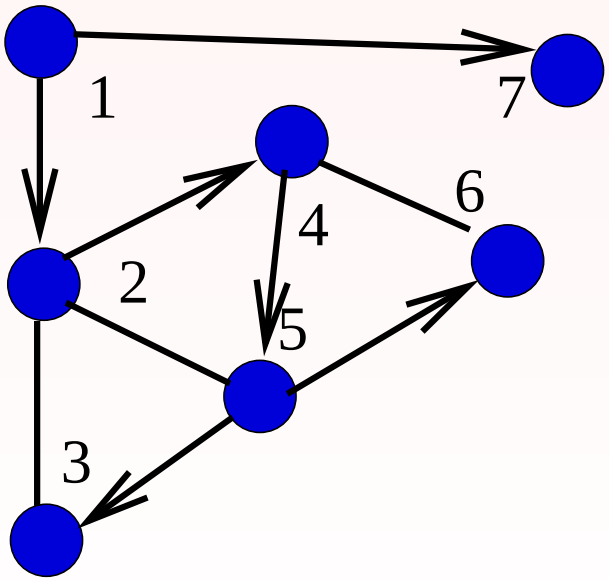
1

traversal

1	2	4	5	3	6	7

The elimination tree

- The elimination tree is a depth-first search tree of the filled graph with the search started at the vertex x_n.
- But, what is the depth-first search tree?



stack

traversal

1	2	4	5	3	6	7

The elimination tree: II

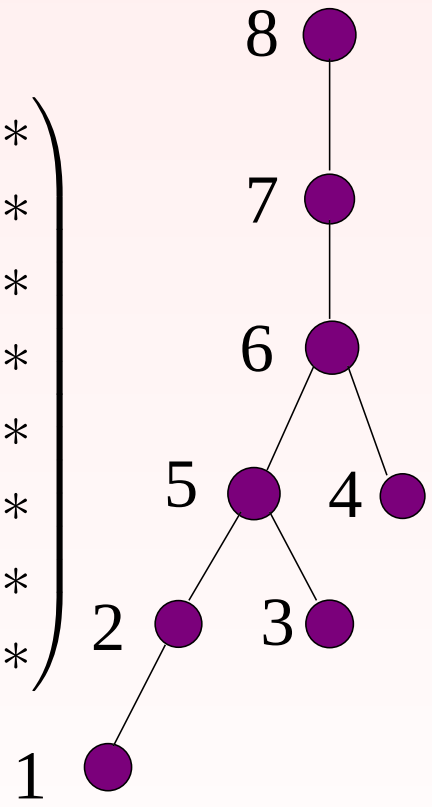
$$\begin{pmatrix} * & * & & & * & & & * \\ * & * & & & & * & & * \\ & & * & & * & & & * \\ & & & * & & * & * & * \\ * & & * & & * & & & * \\ & * & & * & & * & & * \\ & & & * & & & * & * \\ * & * & * & * & * & * & * & * \end{pmatrix}$$

The elimination tree: II

$$\begin{pmatrix} * & * & & & * & & & * \\ * & * & & & & * & & * \\ & & * & & * & & & * \\ & & & * & & * & * & * \\ * & & * & & * & & & * \\ & * & & * & & * & & * \\ & & & * & & & * & * \\ * & * & * & * & * & * & * & * \end{pmatrix} \quad \begin{pmatrix} * & * & & & * & & & * \\ * & * & & & f & * & & * \\ & & * & & * & & & * \\ & & & * & & * & * & * \\ * & f & * & & * & f & & * \\ & * & & * & f & * & f & * \\ & & & * & & f & * & * \\ * & * & * & * & * & * & * & * \end{pmatrix}$$

The elimination tree: II

$$\begin{pmatrix} * & * & & & * & & & * \\ * & * & & & & * & & * \\ & & * & & * & & & * \\ & & & * & & * & * & * \\ * & & * & & * & & & * \\ & * & & * & & * & & * \\ & & & * & & & * & * \\ * & * & * & * & * & * & * & * \end{pmatrix} \quad \begin{pmatrix} * & * & & & * & & & * \\ * & * & & & f & * & & * \\ & & * & & * & & & * \\ & & & * & & * & * & * \\ * & f & * & & * & f & & * \\ & * & & * & f & * & f & * \\ & & & * & & f & * & * \\ * & * & * & * & * & * & * & * \end{pmatrix}$$



- storing elimination tree: vector parent

The elimination tree: III

- Elimination tree (or its variations) is the most fundamental tree structure connected with the elimination.
- Elimination tree is defined via the filled graph (the graph with all fill-in)
- But it should be computed from the original matrix A

The elimination tree: IV

The construction

```
for i = 1 to n do
    parent(i) = 0
    for k such that x_k ∈ adj(x_i) ∧ k < i do
        j = k
        while (parent(j) ≠ 0 ∧ parent(j) ≠ i) do
            j = parent(j)
        end while
        if parent(j) = 0 then parent(j) = i
    end k
end i
```

The elimination tree: V

The construction

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

The elimination tree: V

The construction

$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$

The elimination tree: V

The construction

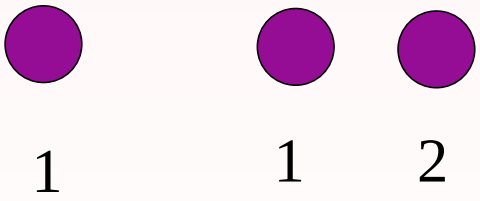
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$

1

The elimination tree: V

The construction

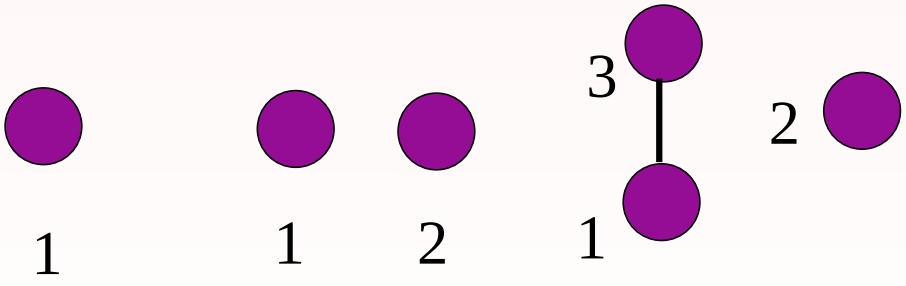
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



The elimination tree: V

The construction

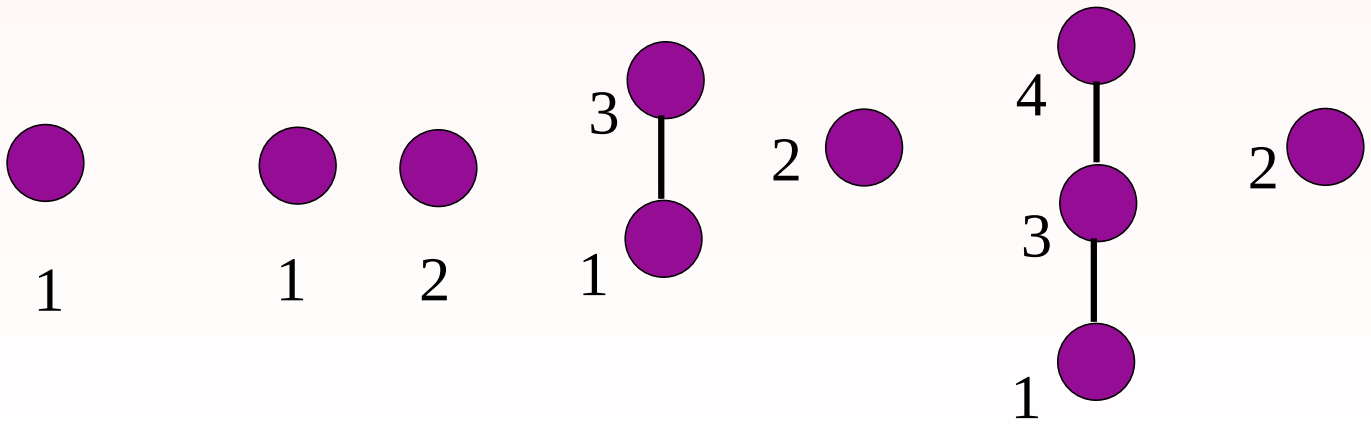
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



The elimination tree: V

The construction

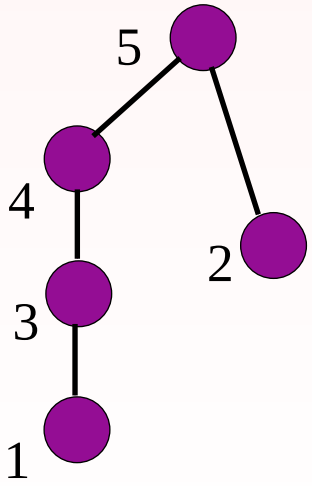
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



The elimination tree: V

The construction

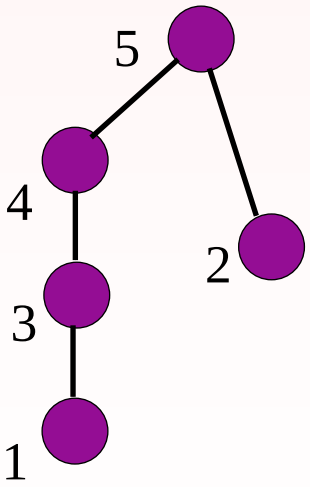
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$

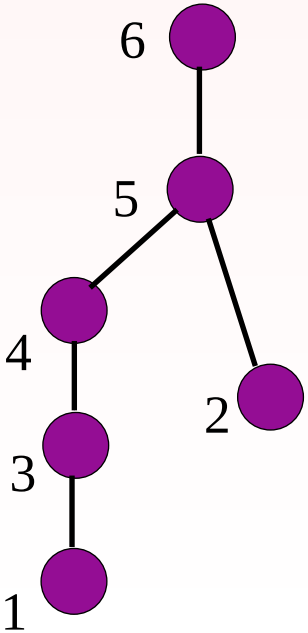


The elimination tree: V

The construction

$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$





The elimination tree: VI
The construction of the elimination tree: improved

- Problem with long dependency chains during the tree traversal

for $i = 1$ to n do
  $parent(i) = 0$; $ancestor(i) = 0$
  for k such that $x_k \in adj(x_i) \wedge k < i$ do
    $j = k$
    while $(ancestor(j) \neq 0 \wedge ancestor(j) \neq i)$ do
      $j = ancestor(j)$; $ancestor(j) = i$; $j = t$
    end while
    if $ancestor(j) = 0$ then $parent(j) = i$; $ancestor(j) = i$
  end k
end i

- Complexity $O(m \log_2 n)$. Can be further reduced by other general tree techniques up to close O(n).

Direct methods

Let us repeat our motivation and goals

- How can be the fill-in described (and avoided ... later)? How should be data structures set up?

Row structure of L: row subtrees of the elimination tree

Lemma

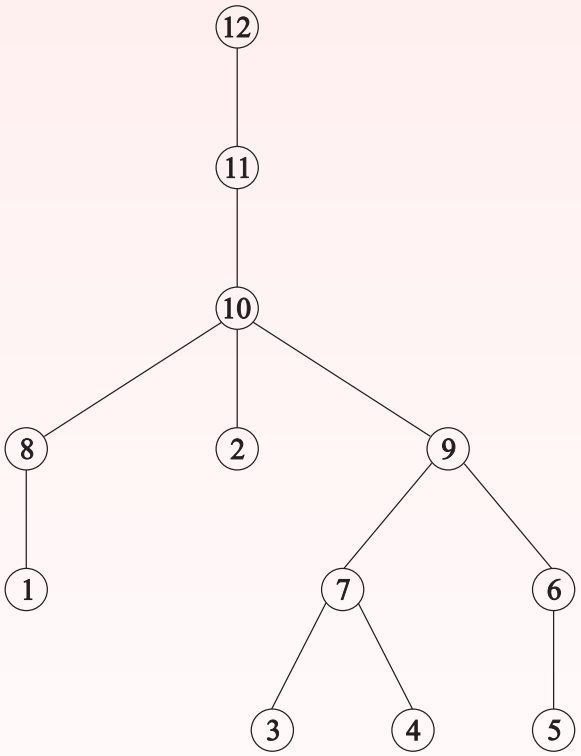
For $j > i$ we have $l_{ji} \neq 0$ if and only if x_i is an ancestor of some x_k in the elimination tree for which $a_{jk} \neq 0$.

- describes fill-in in the j-th row of L, column i
- some x_k must be precede x_i in the elimination tree
- just the vertices in the row subtree rooted at x_j determine nonzeros in the row j of L

Direct methods

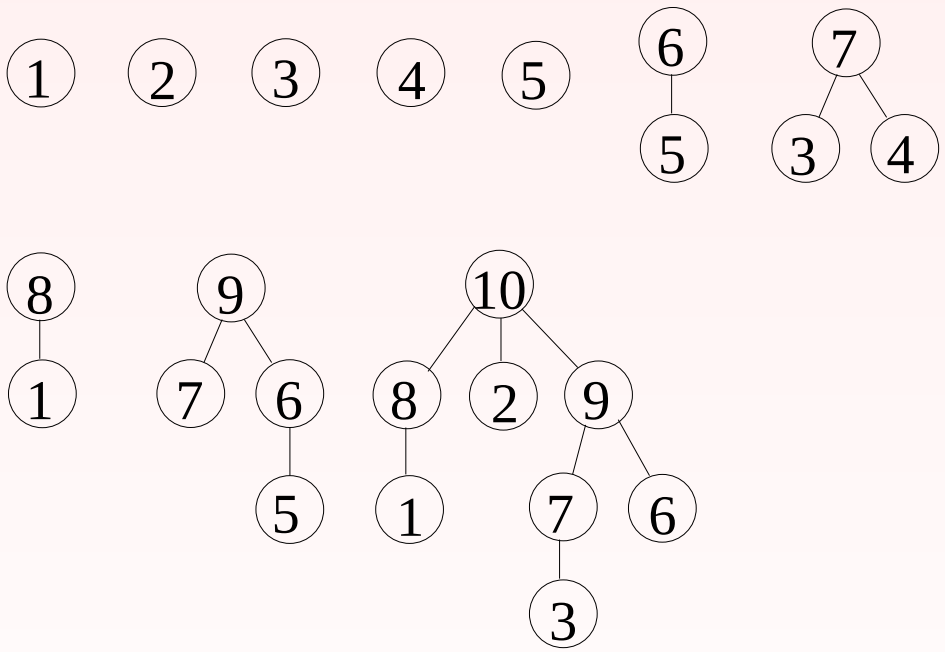
Row subtrees

$$
\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & f & * \\
 & & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
 & & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *
\end{pmatrix}
$$



The elimination tree: VII

$$
\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & f & * \\
 & & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
 & & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *
\end{pmatrix}
$$



Direct methods: postordering

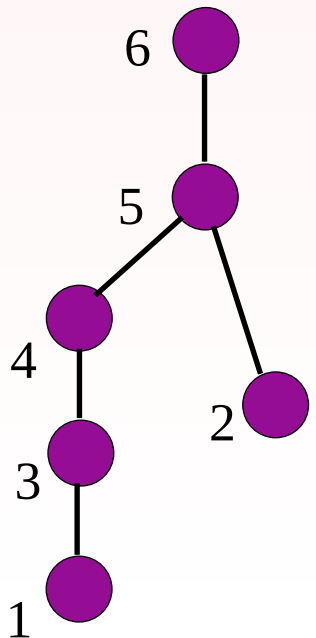
Labels in subtrees form intervals + parents with higher labels

$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$

Direct methods: postordering

Labels in subtrees form intervals + parents with higher labels

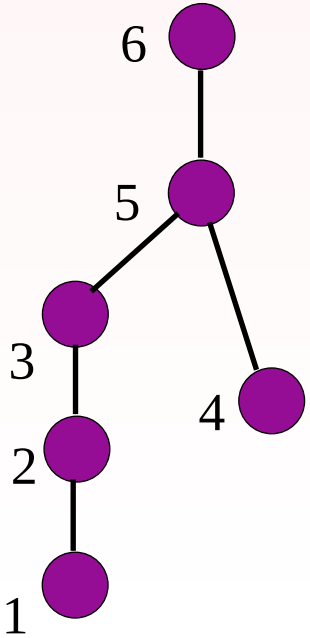
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



Direct methods: postordering

Labels in subtrees form intervals + parents with higher labels

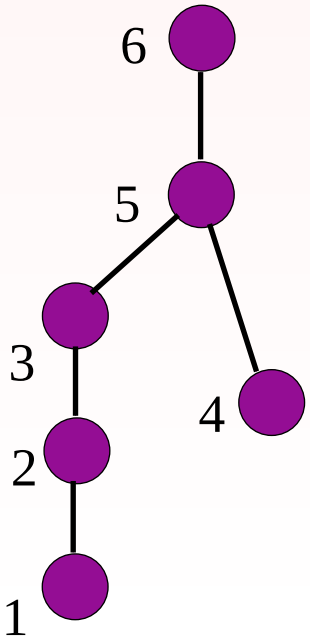
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$



Direct methods: postordering

Labels in subtrees form intervals + parents with higher labels

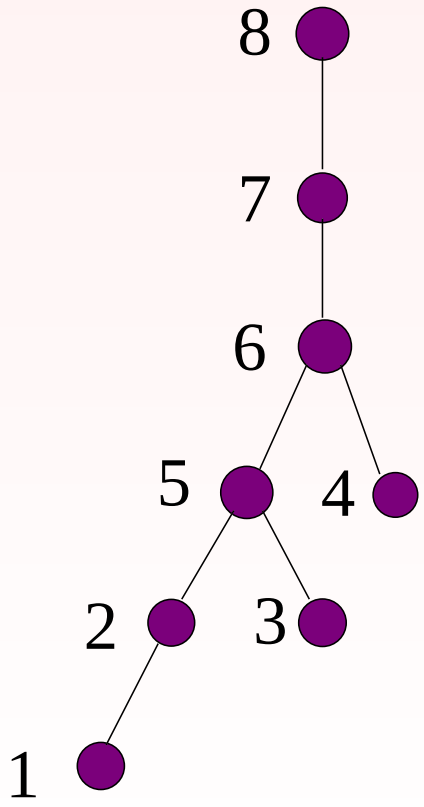
$$\begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & & * \\ * & * & * & & * & \\ & * & & * & & * \end{pmatrix} \quad \begin{pmatrix} * & & * & & * & \\ & * & & & * & * \\ * & & * & * & * & \\ & & * & * & f & * \\ * & * & * & f & * & f \\ & * & & * & f & * \end{pmatrix}$$

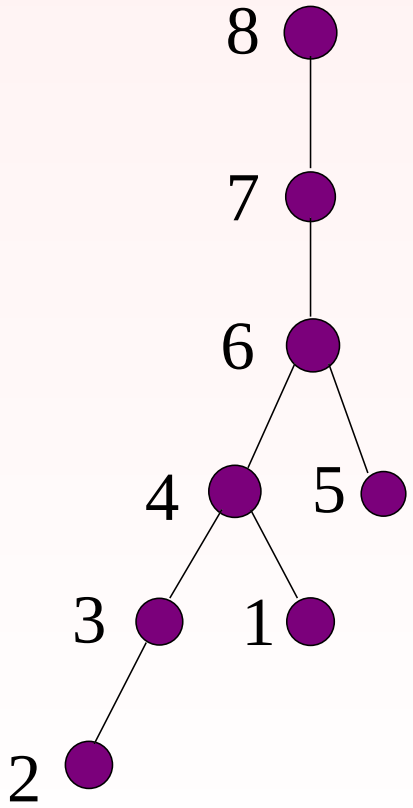


$$\rightarrow \begin{pmatrix} * & * & & & * & \\ * & * & * & & * & \\ & * & * & & f & * \\ & & & * & * & * \\ * & * & f & * & * & f \\ & & * & * & f & * \end{pmatrix}$$

The elimination tree: VIII

- Why do we need a postordering?
- Necessary for efficient exploiting of memory hierarchies, paging environment, crucial for multifrontal methods, efficient computation of factor row counts etc.



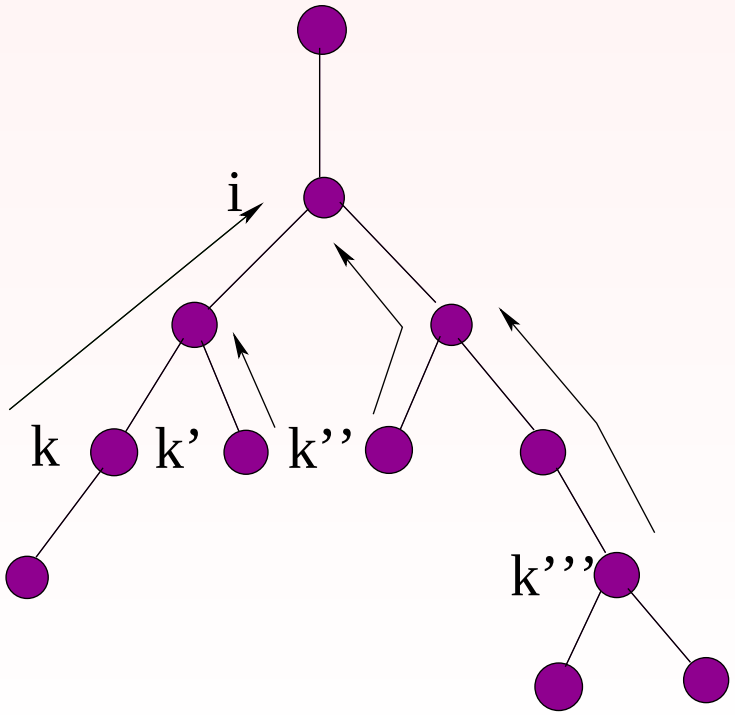


Postordered tree

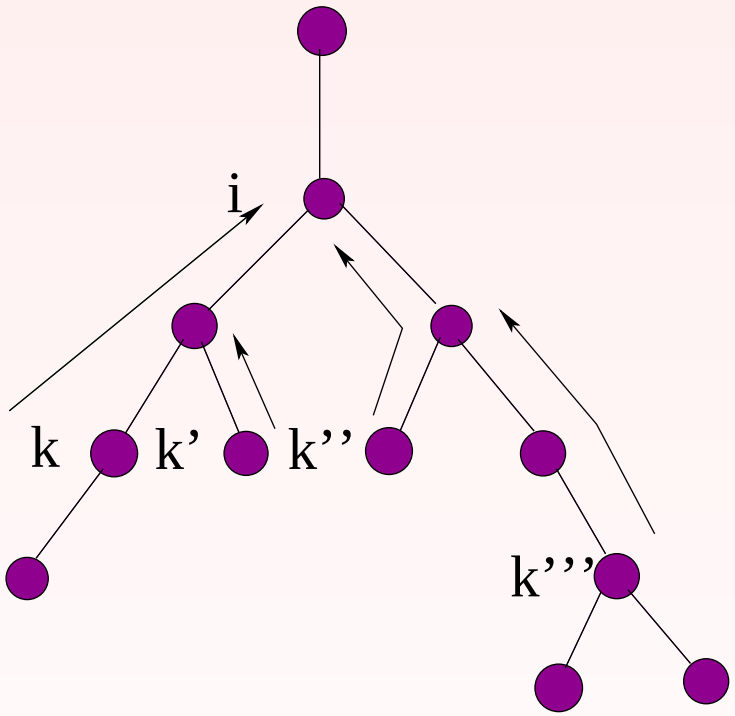
Row counts: simple algorithm

initialize all $colcounts$ to 1
for $i = 1$ to n do
    $rowcount(i) = 1$
    $mark(x_i) = i$
    for k such that $k < i \wedge a_{ik} \neq 0$ do
        $j = k$
        while $mark(x_j) \neq i$ do
            $rowcount(i) = rowcount(i) + 1$
            $colcount(j) = colcount(j) + 1$
            $mark(x_j) = i$
            $j = parent(j)$
        end while
    end k

end i



Row counts: more sophisticated algorithm



- Needed: fast algorithm to determine the "junctions" of branches in the elimination tree,
- and fast algorithm to find leaves of the elimination tree.
- Just by traversing the postordered elimination tree.
- The complexity can be then nearly linear in m.

Motivation and goals again

- How can be the fill-in described (and avoided ... later)? How should be data structures set up?

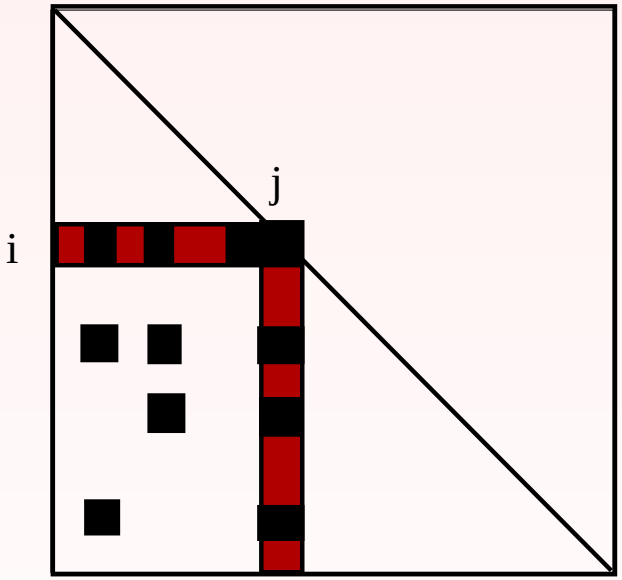
- It could seem that knowing structure of L by rows is enough.
- We then know the size of the factor, we can allocate the final factor structure
- and do just the factorization ...

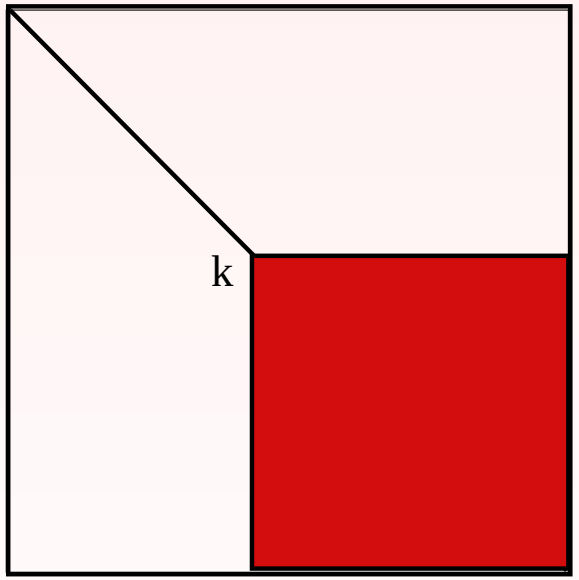
But, what are the ways to do the factorization (repetition)?

Direct methods

The factorization

- Basically, two main ways to factorize a sparse SPD matrix efficiently: 1) Column algorithm, 2) Submatrix algorithm

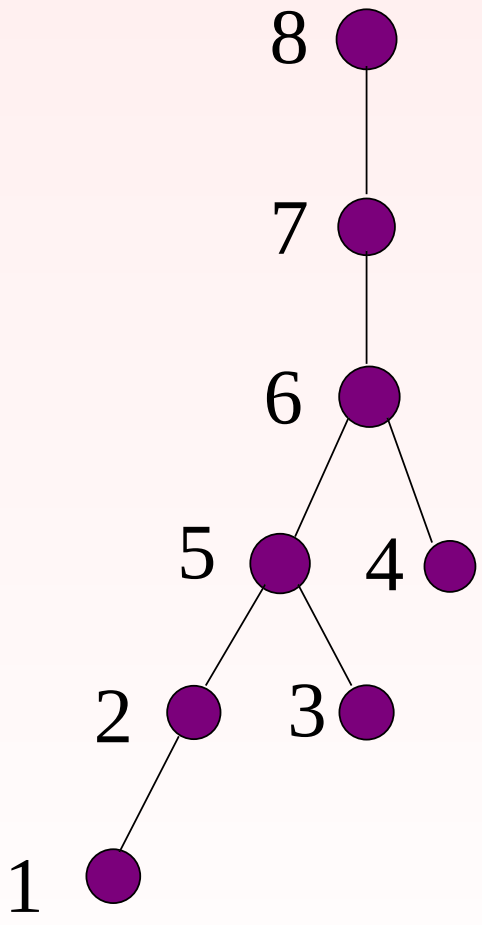




- Indices i, j, k: traditional meaning for 6 possible ways to describe the decomposition.
- In the sparse setting: totally different computational aspects.
- Still very different implementations possible.
- Column structure important as well.

Direct methods

It would be nice to know the column structure of L



row structure

column structure

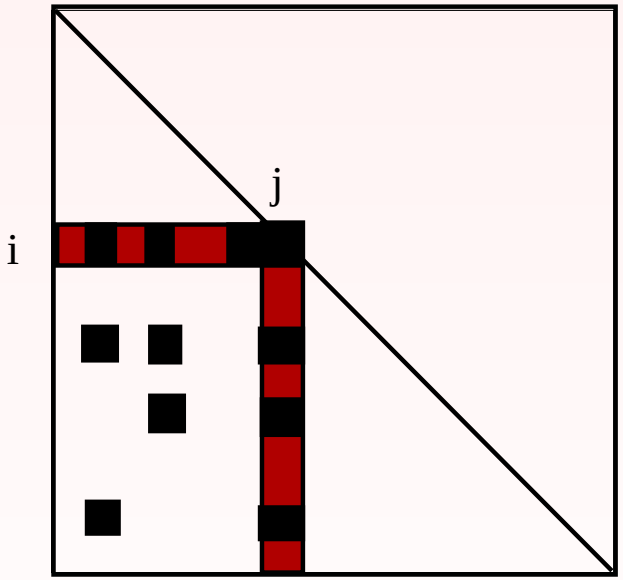
row subtrees

?

Column structure of L

Lemma

Column j is updated by the columns i such that $l_{ij} \neq 0$.



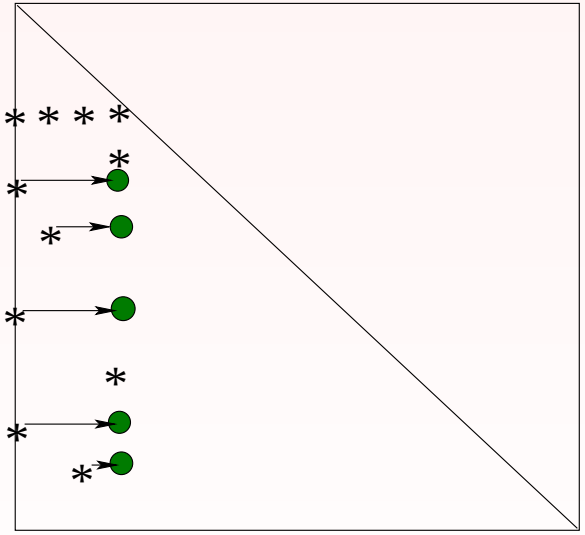
Lemma

$Struct(L_{*j}) = Struct(A_{*j}) \ \cup \ \bigcup_{i, l_{ij} \neq 0} Struct(L_{*i}) \setminus \{1, \ldots, j-1\}.$

Column structure of L

Lemma

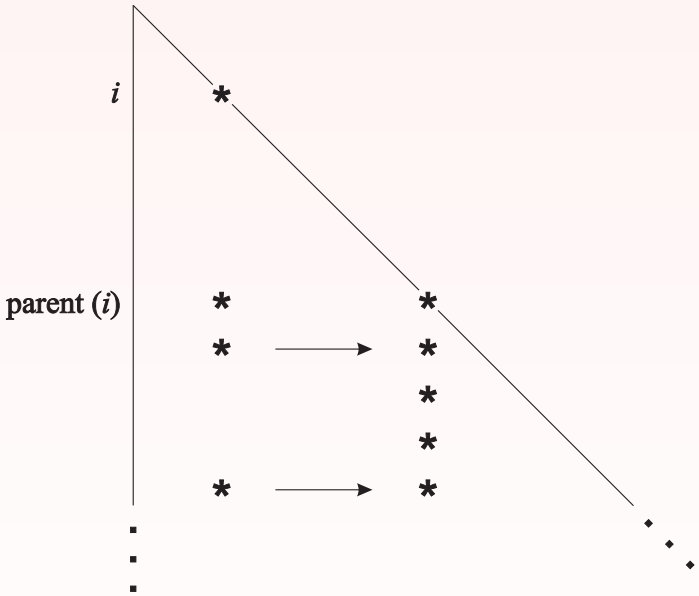
$$Struct(L_{*j}) = Struct(A_{*j}) \ \cup \ \bigcup_{i, l_{ij} \neq 0} Struct(L_{*i}) \setminus \{1, \ldots, j-1\}.$$



Column structure of L

Lemma

$Struct(L_{*j})\ \{j\} \subseteq Struct(L_{*parent(j)})$



$$Struct(L_{*j}) = Struct(A_{*j}) \ \cup \bigcup_{i,j=parent(i)} Struct(L_{*i}) \setminus \{1,\ldots,j-1\}.$$

Column structure of L: algorithm

for $j = 1$ to n do
    $list(x_j) = \emptyset$
 end j
for $j = 1$ to n do
    $col(j) = adj(x_j) \setminus \{x_1, \ldots, x_{j-1}\}$
    for $x_k \in list(x_j)$ do
        $col(j) = col(j) \cup col(k) \setminus \{x_j\}$
        end x_k
        if $col(j) \neq 0$ then
            $p = \min\{i \mid x_i \in col(j)\}$
            $list(x_p) = list(x_p) \cup \{x_j\}$
        end if
    end j
end i

- $list(x)$ is nothing more than the list of the vertices y for which we have $parent(y) = x$

Recapitulation

- Fill-in described both by rows and columns.
- How to avoid it: reorderings: just keep in mind the arrow matrix example

$$\begin{pmatrix} * & * & * & * & * \\ * & * & & & \\ * & & * & & \\ * & & & * & \\ * & & & & * \end{pmatrix} \qquad \begin{pmatrix} * & & & & * \\ & * & & & * \\ & & * & & * \\ & & & * & * \\ * & * & * & * & * \end{pmatrix}$$

But this is not enough for an efficient algorithm: we need also blocks

Avoiding fill-in: reorderings: two basic types

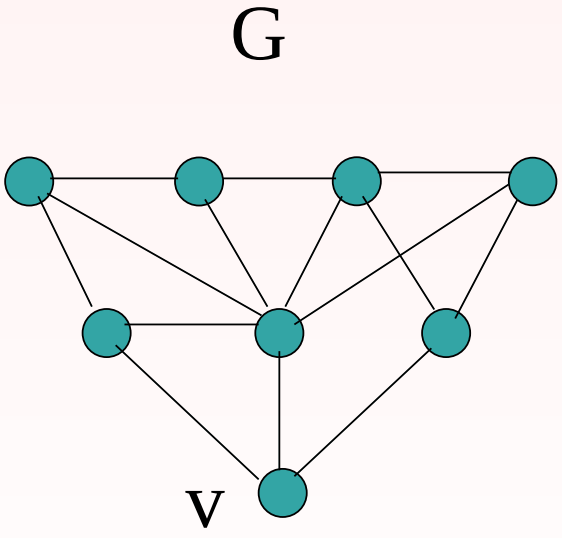
- local reorderings: based on local greedy criterion
- global reorderings: taking into account the whole graph / matrix

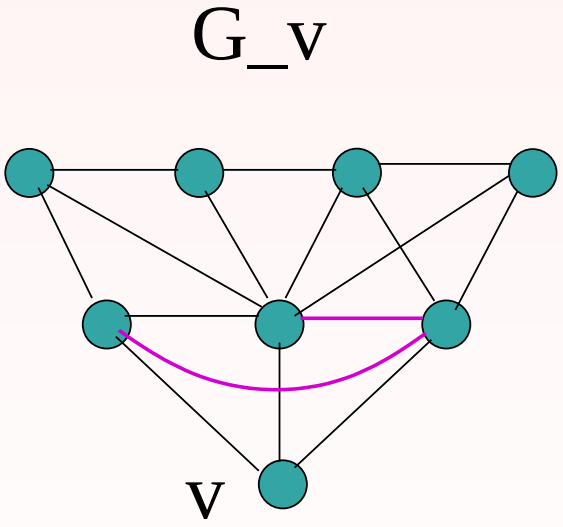
Local reorderings

$G = G(A)$
for $i = 1$ to n do
    find v such that $deg_G(v) = min_{v \in V} deg_G(v)$
    $G = G_v$
end i
The order of found vertices induces their new renumbering

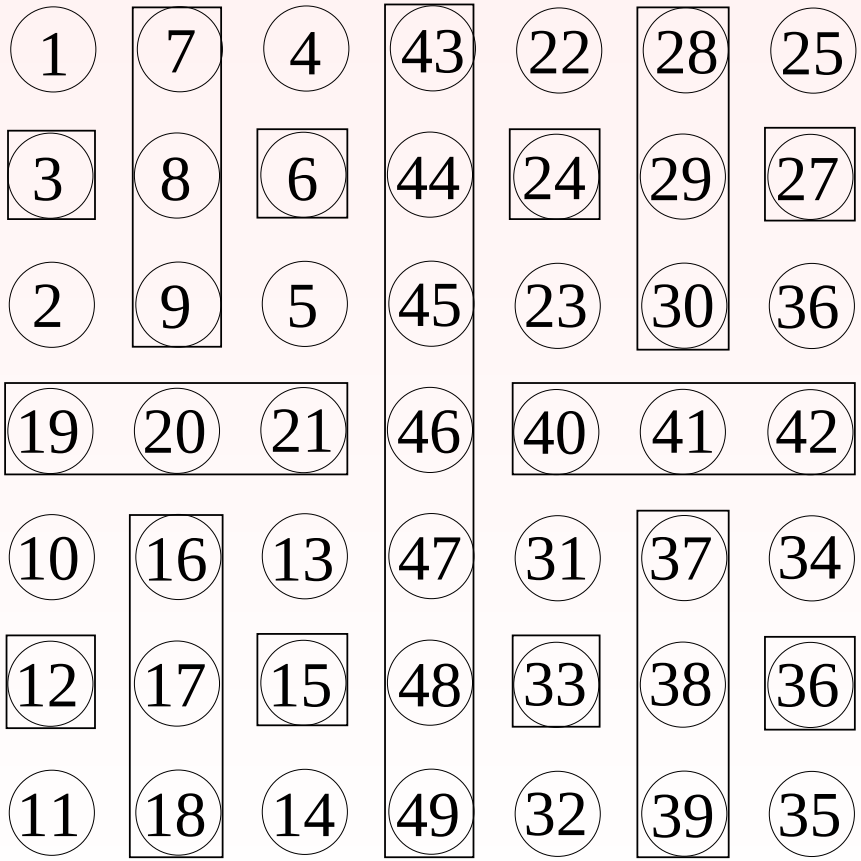
- $deg(v) = |Adj(v)|$; graph G as a superscript determines the current graph

Local reorderings: example

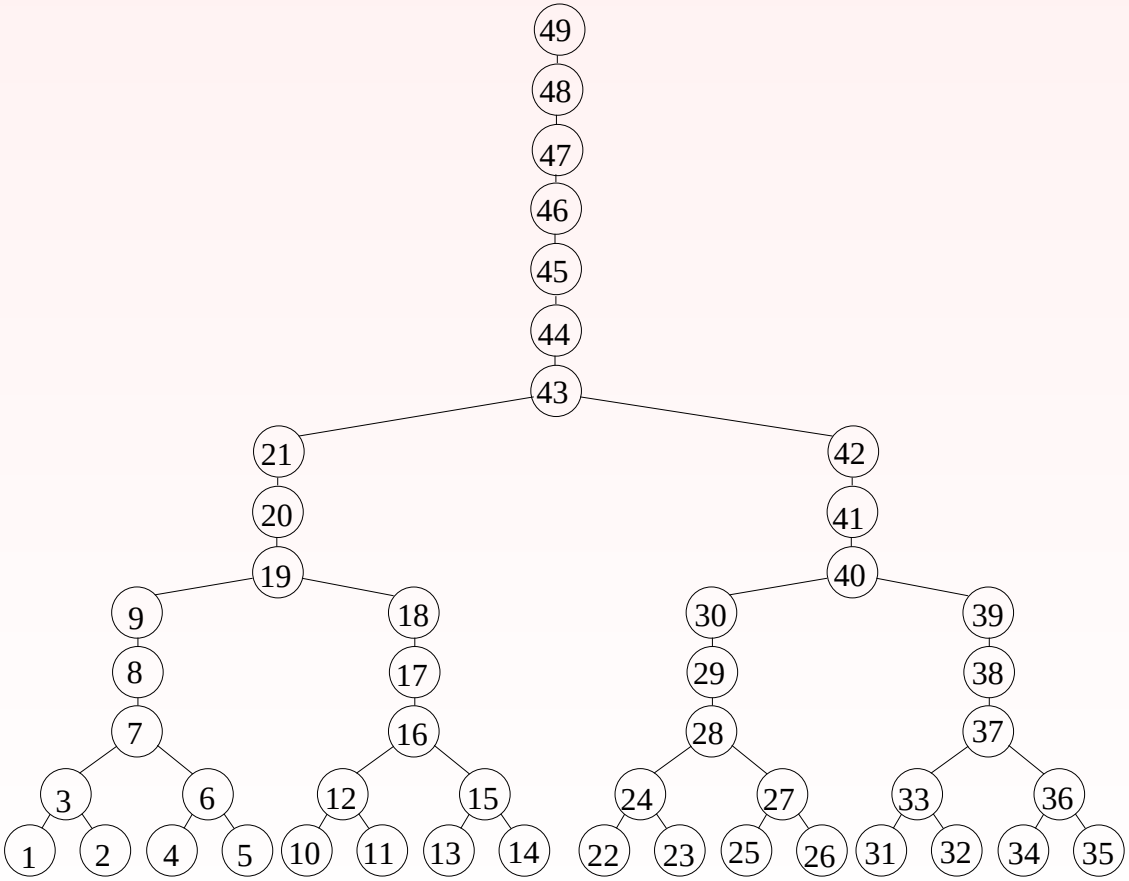




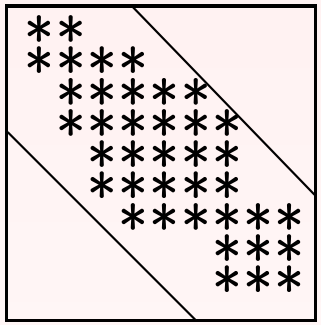
Global reorderings: nested dissection



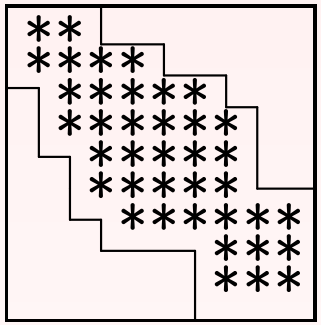
Global reorderings: nested dissection: tree



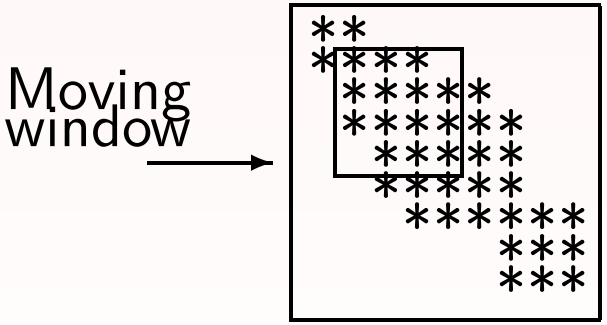
Classical local reorderings: shape pushers



Band



Profile



Frontal method - dynamic band

Classical local reorderings: shape pushers

$$Band(L + L^T) = Band(A)$$

$$Profile(L + L^T) = Profile(A)$$

Blocks

- Blocks are absolutely crucial to compute efficiently on contemporary computers: we need as much data as possible for a unit of data transfer inside memory hierarchy.
- In BLAS terminology:

$$z = x + \alpha y \longrightarrow Z = X + \alpha Y$$

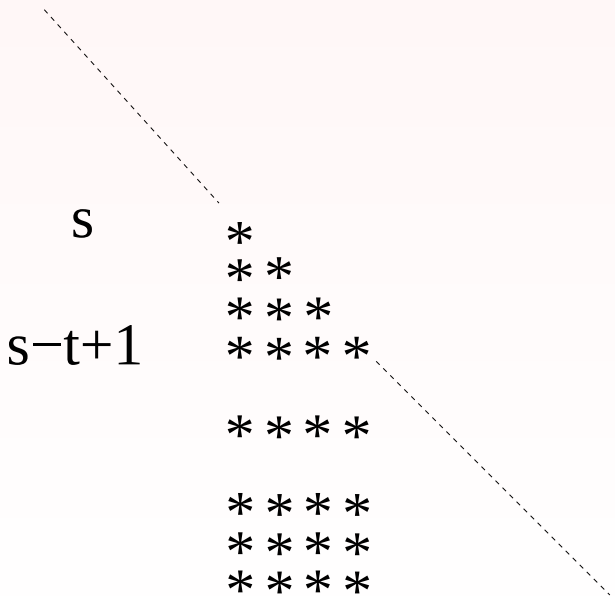
$$\text{saxpy} \longrightarrow \text{dgemm}$$

- But we have sparse matrices. It is not so straightforward to split their nonzeros into blocks.
- In fact, we need to reorder them in order to get blocks.
  - Application-based blocks in discretized systems.
  - Graph-based strategies which can be very fast.
  - But we need to optimize the block structure of L: supernodes.
  - Help: again our good friend, the elimination tree.

Supernodes

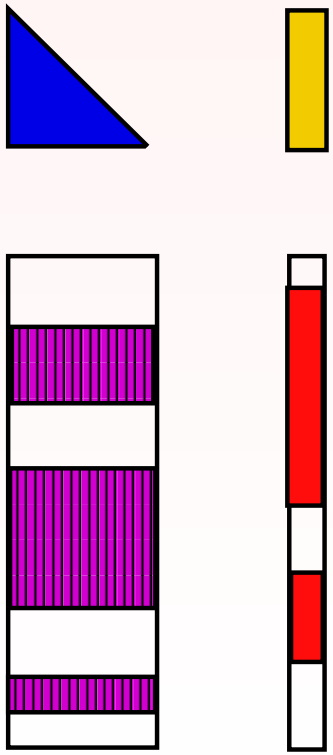
Definition

Let $s, t \in \{1, \ldots, n\}$ such that $s + t - 1 \leq n$. Then the columns with indices $\{s, s+1, \ldots, s+t-1\}$ form a *supernode* if these columns satisfy $Struct(L_{*s}) = Struct(L_{*s+t-1}) \cup \{s, \ldots, s+t-2\}$, and the sequence is maximal.



Supernodes and efficient computation

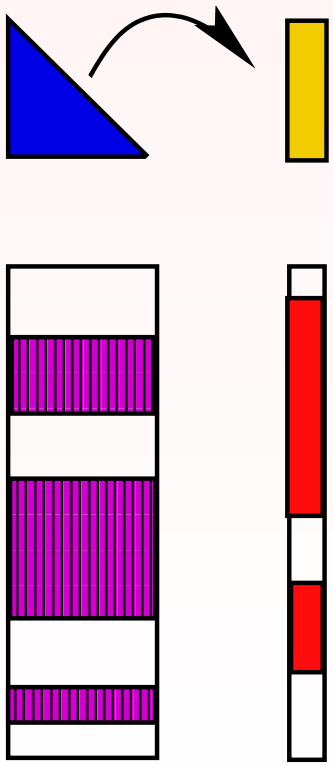
- the loop over rows has no indirect addressing: (dense BLAS1)



Direct methods

Supernodes and efficient computation

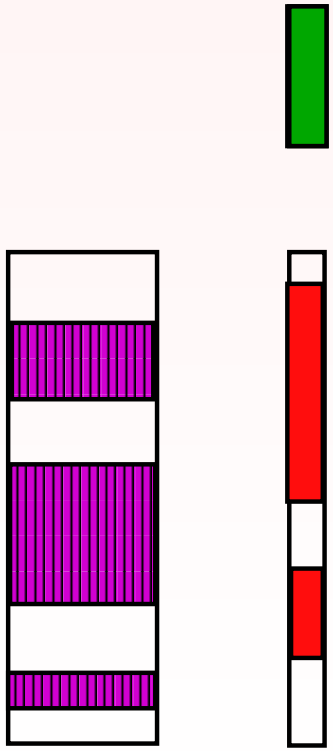
- the loop over rows has no indirect addressing: (dense BLAS1)



Direct methods

Supernodes and efficient computation

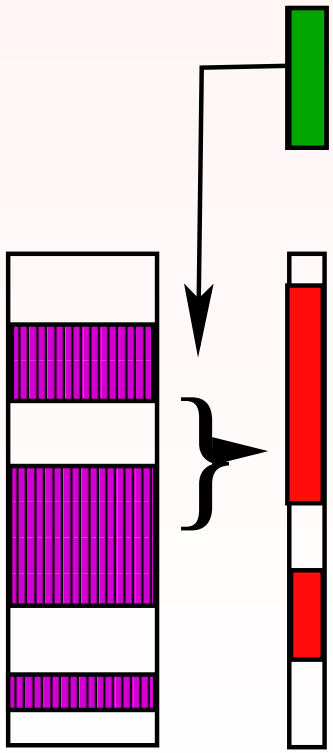
- the loop over rows has no indirect addressing: (dense BLAS1)
- the loop over columns of the updating supernode can be unrolled to save memory references (dense BLAS2)



Direct methods

Supernodes and efficient computation

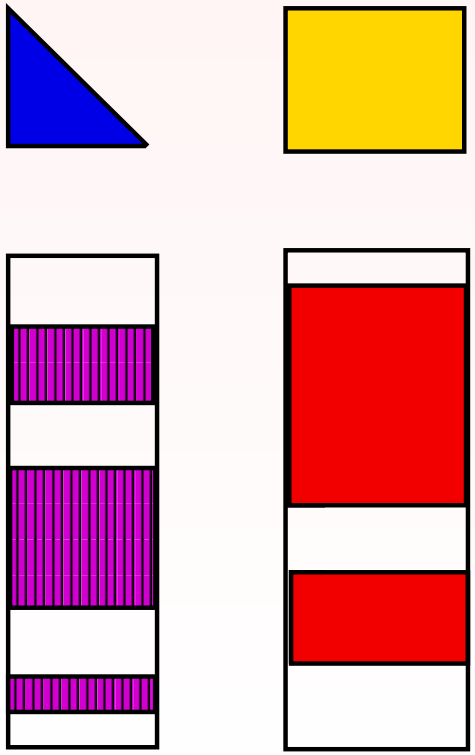
- the loop over rows has no indirect addressing: (dense BLAS1)
- the loop over columns of the updating supernode can be unrolled to save memory references (dense BLAS2)



Direct methods

Supernodes and efficient computation

- the loop over rows has no indirect addressing: (dense BLAS1)
- the loop over columns of the updating supernode can be unrolled to save memory references (dense BLAS2)
- parts of the updating supernode can be used for blocks of updated supernode (dense BLAS3)



Direct methods

Factorization again: general strategy in the SPD case

Preprocessing

– prepares the matrix so that the fill-in would be as small as possible

Symbolic factorization

– elimination tree, determines structures of columns of L. Consequently, L can be allocated and used for the actual decomposition

– the boundary between the first two steps is somewhat blurred due to many possible enhancements

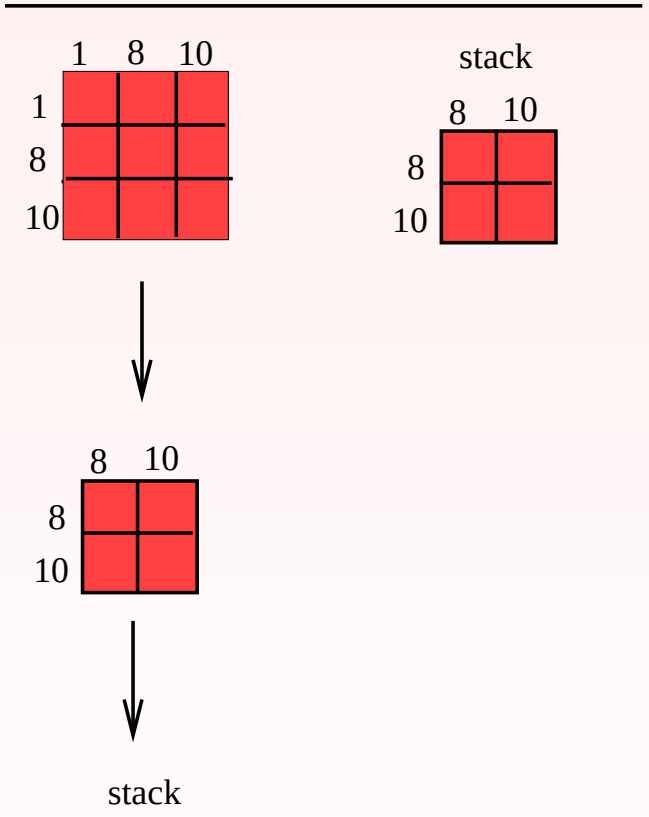
Numeric factorization

– the actual decomposition to obtain numerical values of the factor L

- Multifrontal algorithm
- Block left-looking algorithm

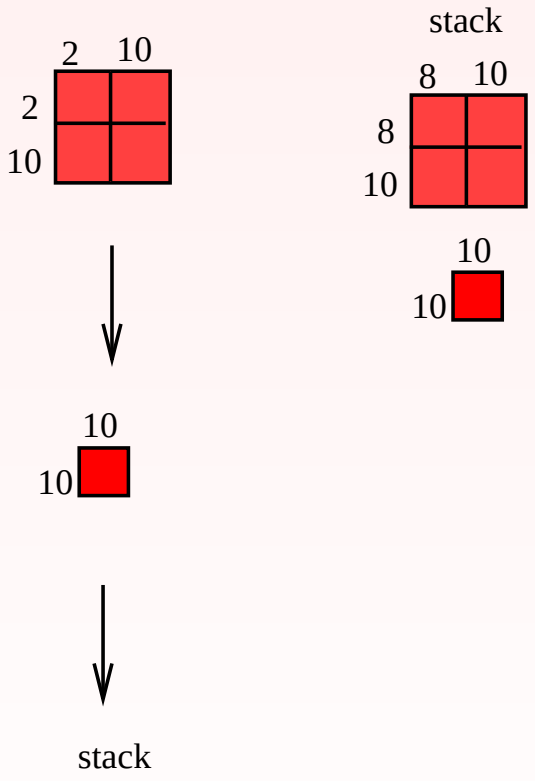
Direct methods: Multifrontal method

$$
\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & \color{red}{f} & * \\
 & & * & * & & & * & & * & \color{red}{f} \\
* & & & & & & & * & & * \\
 & & & & * & \color{red}{f} & * & & * & \color{red}{f} \\
* & * & * & & & * & \color{red}{f} & * & \color{red}{f} & *
\end{pmatrix}
$$



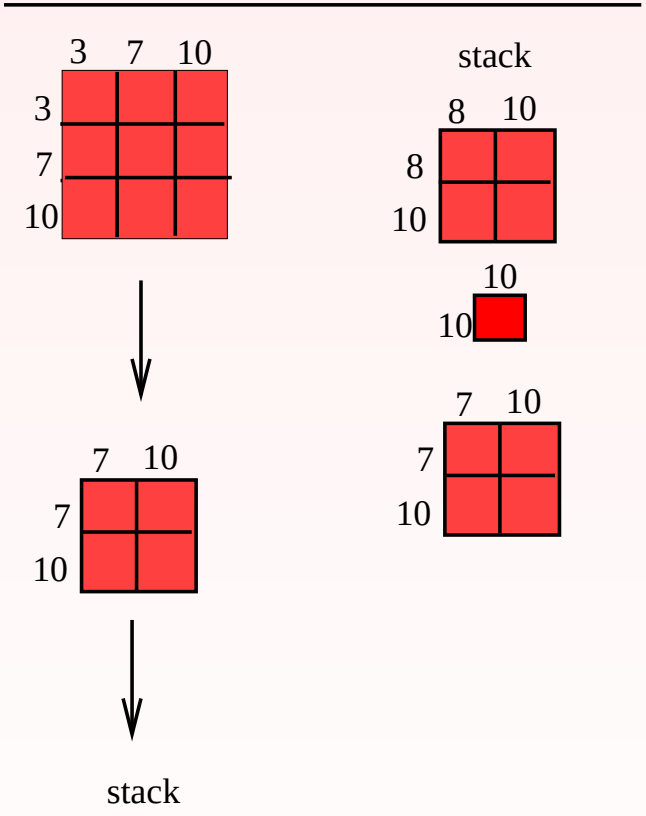
Direct methods: Multifrontal method

$$
\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & f & * \\
 & & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
 & & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *
\end{pmatrix}
$$



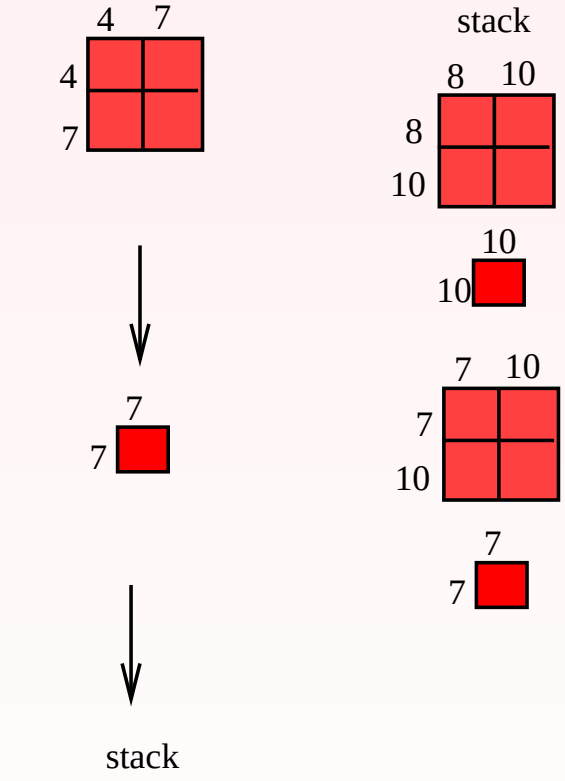
Direct methods: Multifrontal method

$$
\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & f & * \\
 & & * & * & & & * & & * & f \\
* & & & & & & & * & & * \\
 & & & & * & f & * & & * & f \\
* & * & * & & & * & f & * & f & *
\end{pmatrix}
$$



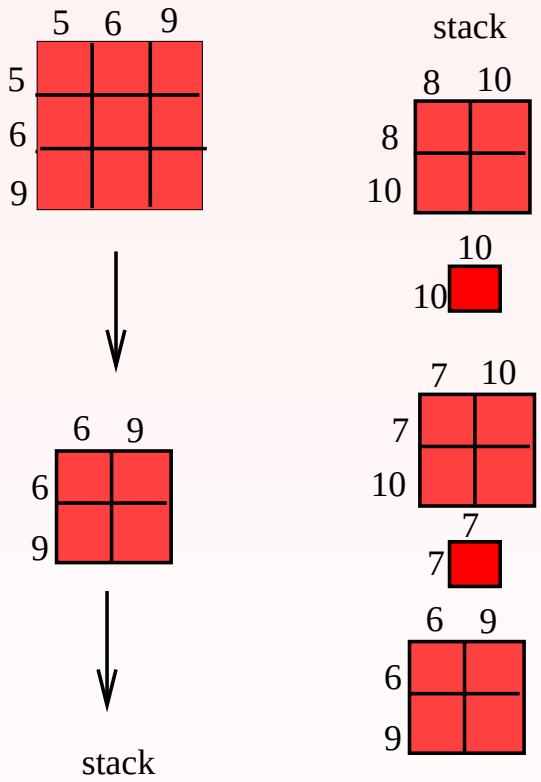
Direct methods: Multifrontal method

$$
\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & \color{red}{f} & * \\
 & & * & * & & & * & & * & \color{red}{f} \\
* & & & & & & & * & & * \\
 & & & & * & \color{red}{f} & * & & * & \color{red}{f} \\
* & * & * & & & * & \color{red}{f} & * & \color{red}{f} & *
\end{pmatrix}
$$



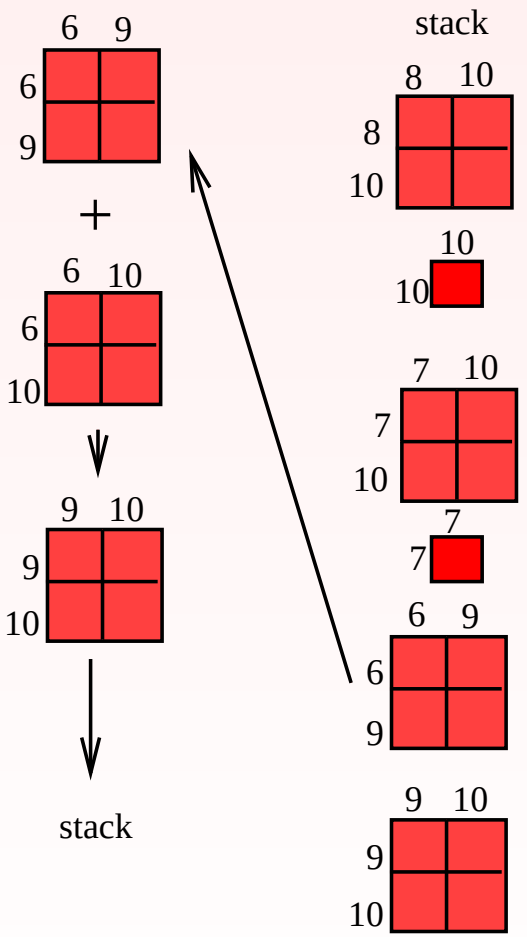
Direct methods: Multifrontal method

$$
\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & \color{red}{f} & * \\
 & & * & * & & & * & & * & \color{red}{f} \\
* & & & & & & & * & & * \\
 & & & & * & \color{red}{f} & * & & * & \color{red}{f} \\
* & * & * & & & * & \color{red}{f} & * & \color{red}{f} & *
\end{pmatrix}
$$



Direct methods: Multifrontal method

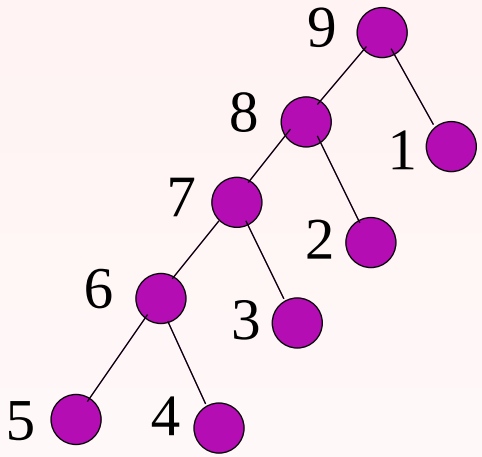
$$
\begin{pmatrix}
* & & & & & & & * & & * \\
 & * & & & & & & & & * \\
 & & * & & & & * & & & * \\
 & & & * & & & * & & & \\
 & & & & * & * & & & * & \\
 & & & & * & * & & & \mathit{f} & * \\
 & & * & * & & & * & & * & \mathit{f} \\
* & & & & & & & * & & * \\
 & & & & * & \mathit{f} & * & & * & \mathit{f} \\
* & * & * & & & * & \mathit{f} & * & \mathit{f} & *
\end{pmatrix}
$$

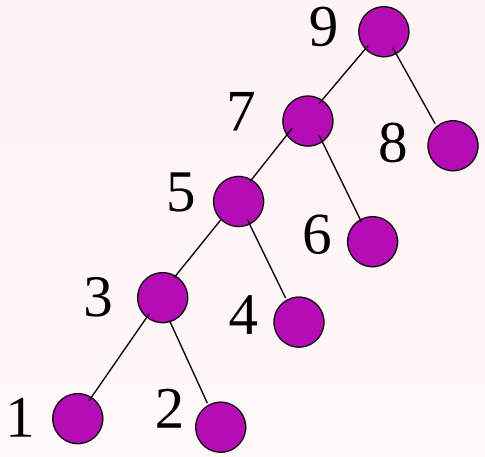


Multifrontal method: Properties

- We do need to have the entries from the stack readily available.
- $\rightarrow$ elimination tree should be postordered
- Arithmetic of dense matrices
- Connection with the frontal method (later) is relatively week.
- One of the most important methods for the sparse direct factorization.

Postorderings and work/memory issues in factorization





- First case: Maximum stack size may be $1 \times 1 + 2 \times 2 + 3 \times 3 + 4 \times 4$
- Second case: Maximum stack size may be 4×4

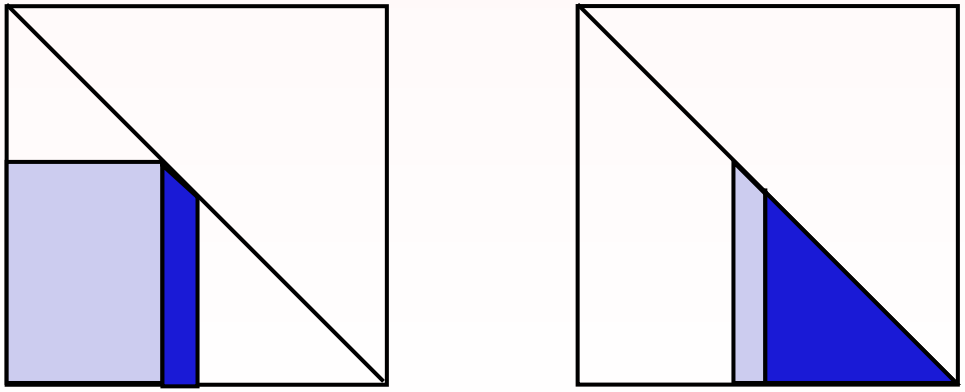
Even postorderings can be very different with respect to particular algorithmic/architecture needs

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

Complexity

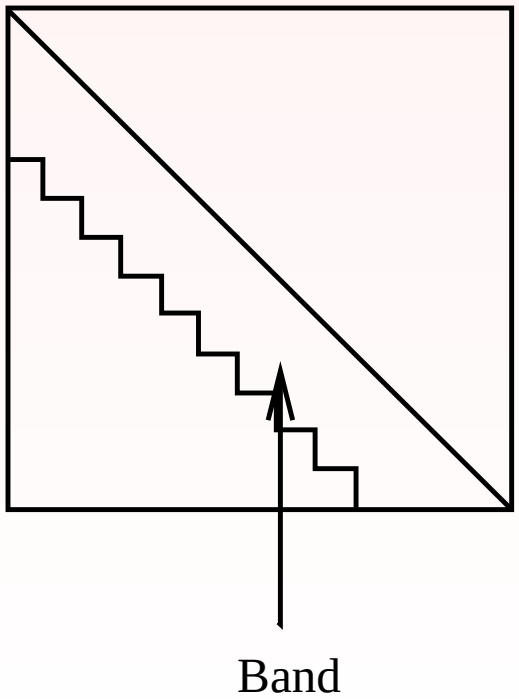
- Time dominated by time for the factorization
- General dense matrices
  - Space: $O(n^2)$
  - Time: $O(n^3)$
- General sparse matrices
  - Space: $\eta(L) = n + \sum_{i=1}^{n-1}(\eta(L_{*i}) - 1)$
  - Time in the i-th step: $\eta(L_{*i}) - 1$ divisions, $1/2(\eta(L_{*i}) - 1)\eta(L_{*i})$ multiple-add pairs
  - Time totally: $1/2\sum_{i=1}^{n-1}(\eta(L_{*i}) - 1)(\eta(L_{*i}) + 2)$



From direct to iterative methods

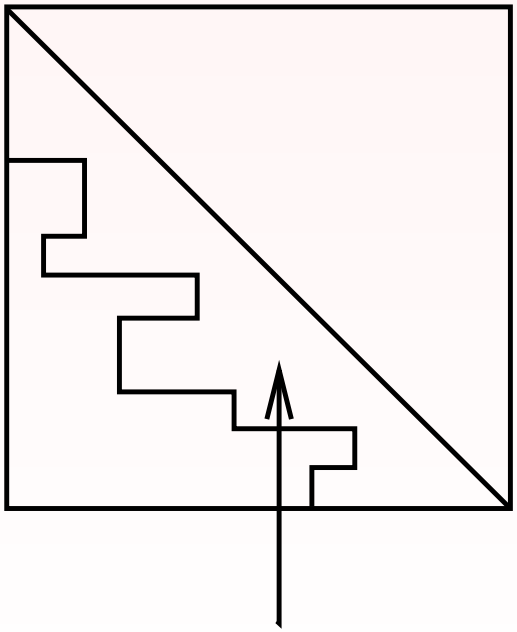
Complexity

- Band schemes ($\beta << n$)
  - Space: $O(\beta n)$
  - Time: $O(\beta^2 n)$



Complexity

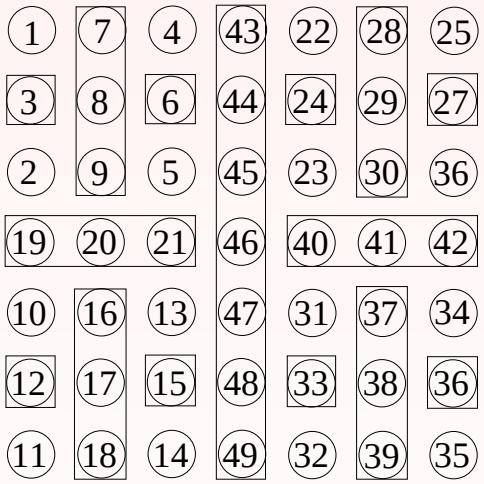
- Profile/envelope schemes
  - Space: $\sum_{i=1}^{n} \beta_i$
  - Frontwidth: $\omega_i(A) = |\{k|k > i \wedge a_{kl} \neq 0 \ for \ some \ l \leq i\}|$
  - Time: $1/2 \sum_{i=1}^{n-1} \omega_i(A)(\omega_i(A) + 3)$



Profile (Envelope)

Complexity

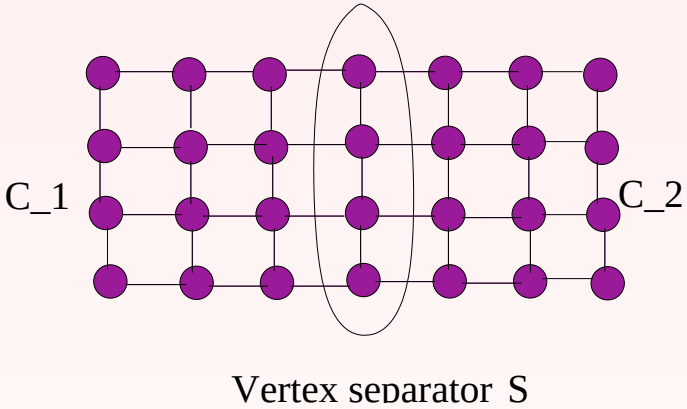
- General sparse schemes can be analyzed in some cases
  - Nested dissection



Definition

(α, σ) separation of a graph with n vertices: each its subgraph can be separated by a vertex separator S such that its size is of the order $O(n^\sigma)$ and the separated subgraphs components have sizes $\leq \alpha n$,$1/2 \leq \alpha < 1$.

Complexity: Generalized nested dissection



- Planar graphs, 2D finite element graphs
  - $\sigma = 1/2$, $\alpha = 2/3$
  - Space: $O(n \log n)$
  - Time: $O(n^3/2)$
- 3D Finite element graphs
  - $\sigma = 2/3$
  - Space: $O(n^{4/3})$
  - Time: $O(n^2)$
- Lipton, Rose, Tarjan (1979), Teng (1997).

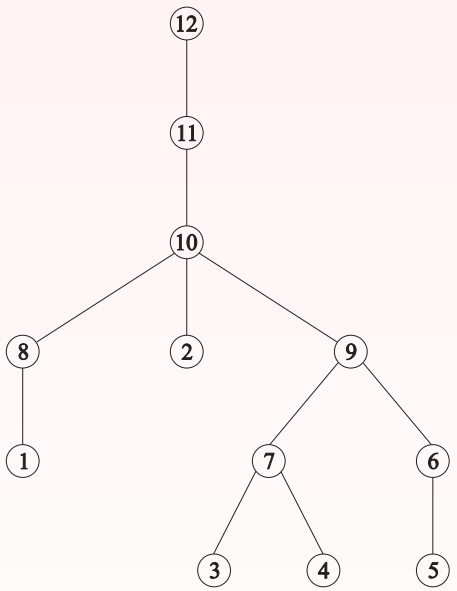
Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

Decomposition and computer architectures: Parallelism

1. Shared memory computers

- 1st level of parallelism: global structure of the decomposition.
- 2nd level of parallelism: local node parallel enhancements.



- Both may/should be coordinated.
- Parallelism in the tree decreases towards its root.
- Dense matrices (e.g. in the multifrontal method) are larger and larger.

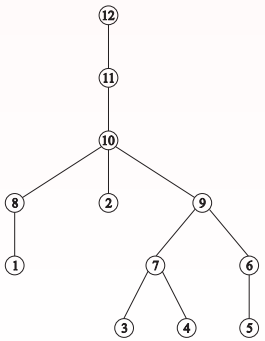
Decomposition and computer architectures: 1st level of parallelism

Two basic possibilities for the 1st level

- Dynamic task scheduling on shared memory computers
- Direct static mapping: subtree to subcube

1. Dynamic task scheduling on shared memory computers

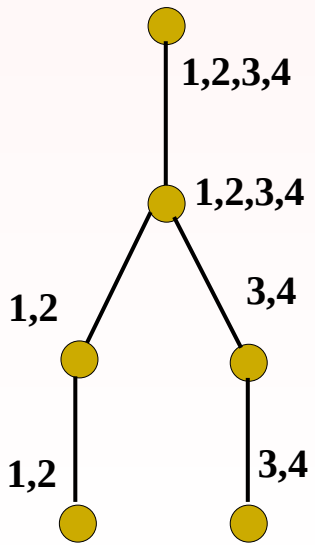
- Dynamic scheduling of the tasks
- Each processor selects a task
- Again, problem of the elimination tree reordering
- Not easy to optimize memory in the multifrontal method



Decomposition and computer architectures: 1st level of parallelism: II

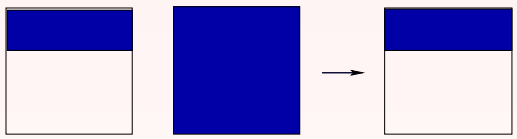
2. Direct static mapping: subtree to subcube

- Recursively map processors to the tree parts from the top
- Various ways of mapping.
- Note: In the SPD (non-pivoting) case we can calculate and consider the arithmetic work
- Good at localizing communication
- More difficult to share the work in more complex models

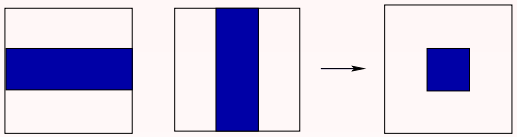


Decomposition and computer architectures: 2nd level of parallelism

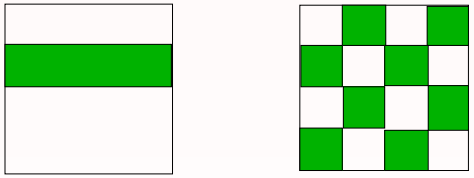
- Block Cholesky/LU factorization
- BLAS / parallel BLAS operations



1D partitioning



2D partitioning



1D and 2D block cyclic distribution

(Only illustrative figures for the talk!)

Decomposition and computer architectures: Distributed memory parallelism

Basic approaches

- Fan-in
  - Demand-driven column-based algorithm
  - Required data are aggregated updates asked from previous columns
- Fan-out
  - Data-driven column-based algorithm
  - Updates are broadcasted once computed and aggregated
  - Historically the first approach; greater interprocessor communication than fan-in
- Multifrontal approach
  - Example: MUMPS

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

From direct to iterative methods: Iterative + Direct

Why complement a direct by an iterative procedure?

- Improving solution accuracy after solving with easier (single) arithmetic.
- Improving solution after solver relaxation (e.g., in parallel computational environment, cf. SuperLU)
- Simple iterative procedure: iterative improvement.

- B is a matrix factorization, $Ax^* = b$, x is a current solution
- $Bx^* = (B - A)x^* - b$
- Iterative procedure: $x^+ = (I - B^{-1}A)x + B^{-1}b$
- $\rho(I - B^{-1}A) < 1$ sufficient for the convergence

Theorem

One step of single precision iterative refinement enough for obtaining componentwise relative backward error to the order of $O(\epsilon)$ under weak conditions.
Strong result for the error using double precision iterative refinement.

From direct to iterative methods: Algebraic preconditioners

What we do not treat here

- Incomplete factorizations
  - By pattern (simple, level-based)
  - By value
  - Compensations
- Incomplete inverse factorizations
  - Factorized
  - Non-factorized
- Polynomial preconditioners
- Algebraic multigrid

- Detailed overviews, citations, etc.: see previous SNA proceedings
- Here we try to see simple analyzable cases

Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

Algebraic preconditioners and complexity: Introduction

Only simple algebraic preconditioners can be analyzed by the standard way via condition number estimation

- We present two examples of the analysis.
- 1) Classical modified incomplete Cholesky MIC for a simple matrix.
  - In particular, MIC(0).
  - Modification consists of adding the neglected fill-in to the diagonal.
- 2) Combinatorial preconditioners.

Power of more complex approaches needed

Algebraic preconditioners and complexity: MIC

The matrix

$$A = \begin{pmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{pmatrix}$$

Algebraic preconditioners and complexity: MIC: II

$$A = \begin{pmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & * & -1 & & & & \\ & -1 & 4 & & * & -1 & & & \\ -1 & * & & 4 & -1 & & -1 & & \\ & -1 & * & -1 & 4 & -1 & * & -1 & \\ & & -1 & & -1 & 4 & & * & -1 \\ & & & -1 & * & & 4 & -1 & \\ & & & & -1 & * & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{pmatrix}$$

$$M = LD^{-1}L^T = A + R$$

$$r_{i,i} = -r_{i,i-m+1} - r_{i,i+m-1}$$

- Here, $m = 3$ (number of grid points in one dimension of 2D grid)

Algebraic preconditioners and complexity: MIC: III

- Idea: Compensate the entries of fill to sum up them to the diagonal of M, perturb diagonal entries
- Origins of MIC: Varga (1960), Dupont, Kendall, Rachford (1968)
- Here: 5-point stencil.
- Generalized by Gustafsson (1978 and later), analysis with similar results for SSOR given by Axelsson (1972)

$$M = LD^{-1}L^T = A + R + \delta\bar{D}$$

$$r_{i,i} = -r_{i,i-m+1} - r_{i,i+m-1}, \delta = ch^2$$

$$d_i = (1+\delta)a_{ii} - r_{i,i-m+1} - r_{i,i+m-1} - a_{i,i-1}^2/d_{i-1} - a_{i,i-m}^2/d_{i-m}$$

Algebraic preconditioners and complexity: MIC: IV

Lemma

$$d_i \geq 2(1 + c_1 h), \alpha_i = 4, \beta_i \leq 1, \gamma_i \leq 1$$

no modification: $\delta = 0$

- $\delta = 0 \Rightarrow d_i \geq 2$
- $d_i = 4 - 2/d_{i-1} - 2/d_{i-m}$ q.e.d.

nonzero modification $\delta \neq 0$

- $d_i \geq 4(1 + ch^2) - 2/(1 + c_1 h) \equiv 4(1 + ch^2) - 2(1 - c_1 h) + O(h^2)$
- $d_i \geq 2(1 + c_1 h) + O(h^2)$ q.e.d.

Corollary

$r_{i+m-1} = a_{i+m-1,i-1} a_{i,i-1}/d_{i-1} \leq 1/d_{i-1} \leq 1/2(1 + c_1 h)$

Algebraic preconditioners and complexity: MIC: V

Lemma

$(Rx, x) = -\sum_i r_{i,i+m-1}(x_{i+m-1} - x_i)^2$

- $(Rx, x) = -\sum_i r_{i,i}x_i^2 + 2\sum_i r_{i,i+m-1}x_{i+m-1}x_i =$
  $-\sum_i (r_{i,i-m+1} + r_{i,i+m-1})x_i^2 + 2\sum_i r_{i,i+m-1}x_{i+m-1}x_i$
- (symmetrically transformed sum + zero row sums)
- Since $2x_{i+m-1}x_i = x_{i+m-1}^2 + x_i^2 - (x_{i+m-1} - x_i)^2$ we get
- $(Rx, x) = -\sum_i (r_{i,i+m-1} + r_{i,i-m+1})x_i^2 + \sum_i (r_{i,i+m-1}x_i^2 +$
  $r_{i,i+m-1}x_{i+m-1}^2) - \sum_i r_{i,i+m-1}(x_{i+m-1} - x_i)^2$
- First entries of the first two sums sum up to zero.
- Second entries of the first two sums give zero (formally by transforming sum indices). q.e.d.

Algebraic preconditioners and complexity: MIC: VI

Lemma

$-(Rx, x) \leq 1/(1 + c_1 h)(Ax, x)$

- $-(Rx, x) = \sum_i r_{i,i+m-1}(x_{i+m-1} - x_i)^2 \leq \sum_i 1/d_{i-1}(x_{i+m-1} - x_i)^2$
- $-(Rx, x) \leq \sum_i 1/2(1 + c_1 h)(x_{i+m-1} - x_i)^2$
  - We have $(a - b)^2 \leq 2(a - e)^2 + 2(e - b)^2$ (can be easily shown by considering various cases of the involved reals)
- $-(Rx, x) \leq \sum_i 1/(1 + c_1 h)[(x_{i+m-1} - x_{i-1})^2 + (x_{i-1} - x_i)^2] \leq 1/(1 + c_1 h)(Ax, x)$

Algebraic preconditioners and complexity: MIC: VII

Corollary

$\kappa(M^{-1}A) = O(h^{-1})$

- $(Ax,x)/(Mx,x) = \frac{(Ax,x)}{(Ax,x)+(Rx,x)+\delta(\bar{D}x,x)} \leq \frac{1}{1+(Rx,x)/(Ax,x)} \leq \frac{1}{1-\frac{1}{(1+c_1 h)}} = 1 + \frac{1}{c_1 h}$
- Note that the smallest eigenvalue of A can be written as $c_0 h^2$.
- $(Ax,x)/(Mx,x) \geq \frac{1/(Ax,x)}{1+\delta(\bar{D}x,x)/(Ax,x)} = \frac{1}{1+ch^2(x,x)/(Ax,x)} \geq \frac{1}{1+\frac{c}{c_0}}$ q.e.d.

Combinatorial preconditioners

Lemma

Let A be symmetric, B SPD. If $\tau B - A$ is positive semidefinite then $\lambda_{max}(B^{-1}A) \leq \tau$ for a real τ.

Proof: Let u be an eigenvector of $\lambda \equiv \lambda_{max}(B^{-1}A)$: $Au = \lambda Bu$. If $\tau B - A$ is positive semidefinite then

$$0 \leq u^t(\tau B - A)u = (\tau - \lambda)u^T Bu.$$

$$B \text{ is } SPD \Rightarrow \tau - \lambda \geq 0.$$

Combinatorial preconditioners: II

Definition

Support $\sigma(A, B)$ of B for A define as

$$\min\{\tau \mid \tau B - A \text{ is positive semidefinite}\}.$$

Generalized support $\bar{\sigma}(A, B)$ of B for A define as

$$\min\{\tau | x^T(\tau B - A)x \geq 0 \text{ for all } x, Ax \neq 0, Bx \neq 0.\}$$

- B SPD $\Rightarrow \lambda_{max}(B^{-1}A) \leq \sigma(A, B)$
- A, B SPD $\Rightarrow \kappa(B^{-1}A) = \lambda_{max}(B^{-1}A)/\lambda_{min}(B^{-1}A) = \lambda_{max}(B^{-1}A)\lambda_{max}(A^{-1}B) \leq \sigma(A, B)\sigma(B, A)$

Combinatorial preconditioners: III

Example of the support

$$\tau \begin{pmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{pmatrix} - \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

to be positive semidefinite

$$\Downarrow$$

Combinatorial preconditioners: III

Example of the support

$$\tau \begin{pmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{pmatrix} - \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

to be positive semidefinite

$$\Downarrow$$

$$\tau \geq 2.$$

Combinatorial preconditioners: III

Example of the support

$$\tau \begin{pmatrix} 0.5 & -0.5 \\ -0.5 & 0.5 \end{pmatrix} - \begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$$

to be positive semidefinite

$$\Downarrow$$

$$\tau \geq 2.$$

- $A = A_1 \cup \ldots A_k$, $B = B_1 \cup \ldots B_k$
- Let $\tau_i B_i - A_i$ be positive semidefinite for all i, $\tau^* = \max_i \tau_i$. Then $\tau^* B - A$ is positive semidefinite as well.
- $\sigma(A, B) \leq \max_i \sigma(A_i, B_i)$
- Pairs of symmetric diagonally dominant matrices $\rightarrow$ transformed to pairs of matrices with zero row sums with equivalent support numbers.

Combinatorial preconditioners: IV

Congestion - dilation: more automatic tools for the splitting transformation

Theorem

Let $A = \begin{pmatrix} a & 0 & \dots & & -a \\ 0 & 0 & & 0 & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & & 0 & 0 \\ -a & 0 & \dots & 0 & a \end{pmatrix}$ $\quad B = \begin{pmatrix} a & -a & & & \\ -a & a & -a & & \\ & & \dots & & \\ & & -a & a & -a \\ & & & -a & a \end{pmatrix}$

with dimensions $k+1$, $a > 0$. *Then* $kB - A$ *is positive semidefinite.*

Theorem

Let $A = \begin{pmatrix} a & 0 & \dots & & -a \\ 0 & 0 & & 0 & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & & 0 & 0 \\ -a & 0 & \dots & 0 & a \end{pmatrix}$ $\quad B = \begin{pmatrix} b & -b & & & \\ -b & 2b & -b & & \\ & & \dots & & \\ & & -b & 2b & -b \\ & & & -b & b \end{pmatrix}$

with dimensions $k+1$, $a, b > 0$. *Then* $(ka/b)B - A$ *is positive semidefinite.*

Combinatorial preconditioners: V

Congestion - dilation: more automatic tools for the splitting transformation

Theorem

Let

$$A = \begin{pmatrix} a & 0 & \dots & & -a \\ 0 & 0 & & 0 & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & & 0 & 0 \\ -a & 0 & \dots & 0 & a \end{pmatrix} \qquad B = \begin{pmatrix} d_1 & -b_1 & & & \\ -b_1 & d_2 & -b_2 & & \\ & & \dots & & \\ & & -b_{k-1} & d_k & -b_k \\ & & & -b_k & d_{k+1} \end{pmatrix}$$

with dimensions $k+1$, $d_i > 0, b_i > 0$ *for all* i. *Then* $(ka/min(b_i))B - A$ *is positive semidefinite.*

- $a/min(b_i)$ is called here the congestion
- k is the dilation

Combinatorial preconditioners: VI

clique - star tool for the splitting transformation

Theorem

$$\textit{Let } A = \begin{pmatrix} 0 & 0 & & \dots & 0 & \\ 0 & (k-1)a & -a & \dots & -a & \\ 0 & -a & (k-1)a & & \dots & -a \\ \vdots & & \vdots & & \vdots & \\ 0 & -a & -a & \dots & (k-1)a & \end{pmatrix}$$

$$B = \begin{pmatrix} kb & -b & -b & \dots & -b \\ -b & b & 0 & \dots & 0 \\ -b & 0 & b & \dots & 0 \\ \vdots & & \vdots & & \vdots \\ -b & 0 & 0 & \dots & b \end{pmatrix}$$

with dimensions $k+1$, $a, b > 0$. *Then* $(ka/b)B - A$ *is positive semidefinite.*

Combinatorial preconditioners: VII

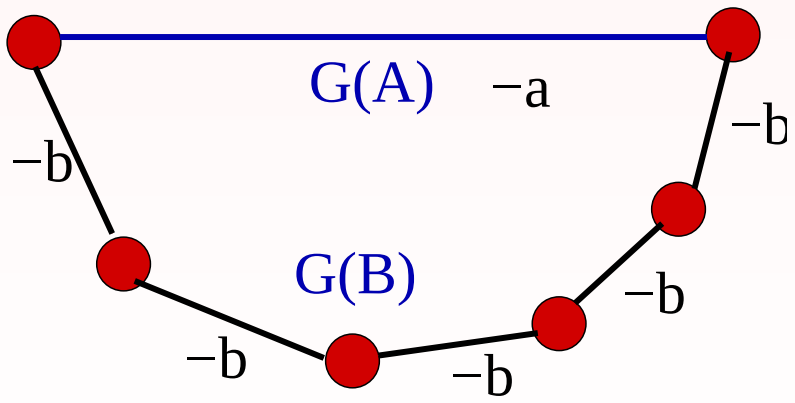
Let $A = \begin{pmatrix} a & 0 & \dots & & -a \\ 0 & 0 & & 0 & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & & 0 & 0 \\ -a & 0 & \dots & 0 & a \end{pmatrix}$ $\quad B = \begin{pmatrix} b & -b & & & \\ -b & 2b & -b & & \\ & & \dots & & \\ & & -b & 2b & -b \\ & & & -b & b \end{pmatrix}$

- Computation of the support numbers can be visualized via graph embeddings.
- Matrix is a generalized Laplacian for the derived graph.

Combinatorial preconditioners: VII

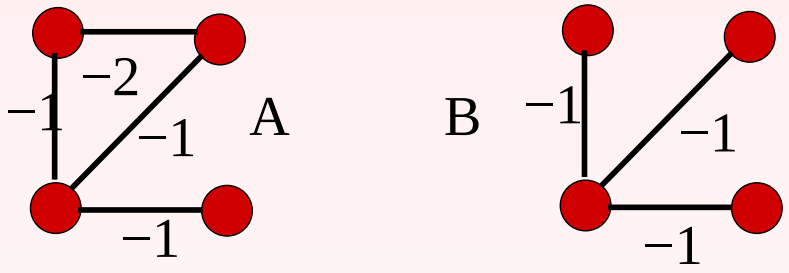
Let $A = \begin{pmatrix} a & 0 & \dots & & -a \\ 0 & 0 & & 0 & 0 \\ \vdots & & \vdots & & \vdots \\ 0 & 0 & & 0 & 0 \\ -a & 0 & \dots & 0 & a \end{pmatrix}$ $\quad B = \begin{pmatrix} b & -b & & & \\ -b & 2b & -b & & \\ & & \dots & & \\ & & -b & 2b & -b \\ & & & -b & b \end{pmatrix}$

- Computation of the support numbers can be visualized via graph embeddings.
- Matrix is a generalized Laplacian for the derived graph.



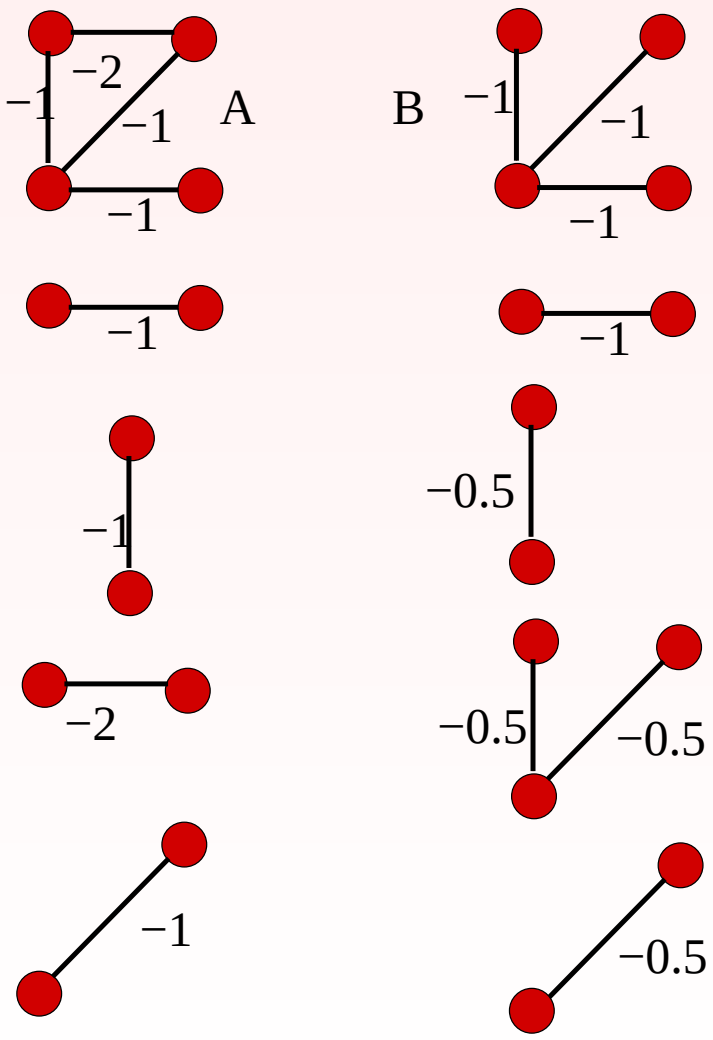
Combinatorial preconditioners: VIII

Example of decomposition



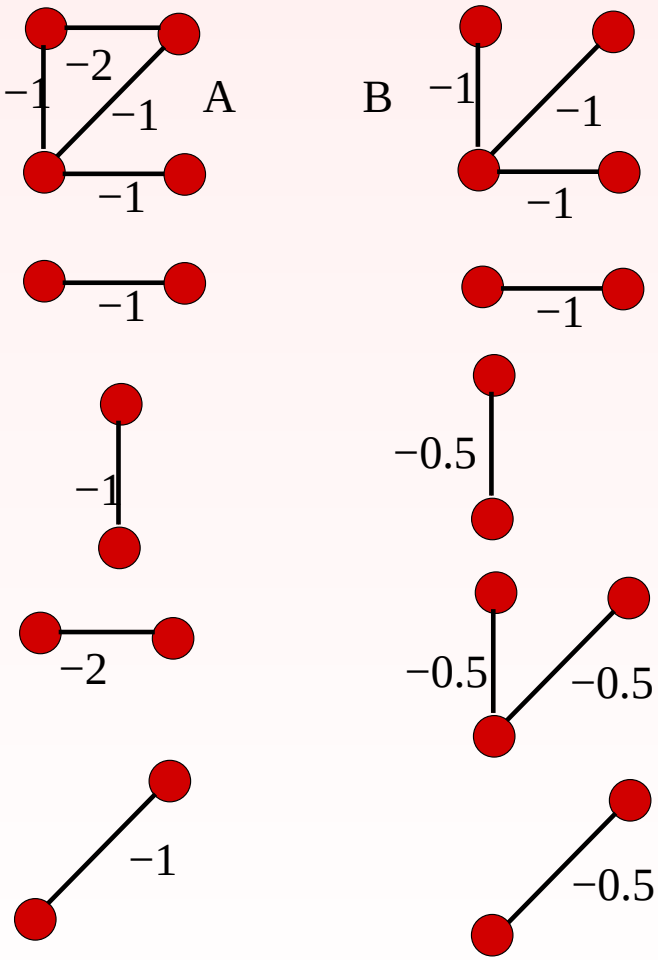
Combinatorial preconditioners: VIII

Example of decomposition



Combinatorial preconditioners: VIII

Example of decomposition



sigma(A1,B1) <= 1

sigma(A2,B2) <= 2

sigma(A3,B3) <= (2/0.5)*2=8

sigma(A4,B4) <= 2

Combinatorial preconditioners: IX

Positive off-diagonals

Problem of edges with positive weights

- Positive edges $B_{m+k+1}, ...B_{m+2k}$
- Negative edges of B should support positive edges of B as well

$\tau B - A = (\tau B_1 - A_1) + \ldots + (\tau B_m - A_m) + (\tau B_{m+1} + \tau B_{m+k+1}) + \ldots + (\tau B_{m+k} + \tau B_{m+2k})$ should be positive semidefinite

Combinatorial preconditioners: X

Simpler application of the support

Lemma

Let $B = A - R$ such that A and B and R are positive semidefinite. If $\bar{\sigma}(R, A) = \tau' < 1$ then $\bar{\sigma}(B, A)\bar{\sigma}(A, B) \leq 1/(1 - \tau')$.

- Let $\tau = 1/(1 - \tau')$
- The matrix $\tau B - A = \tau A - \tau R - A = (\tau - 1)A - \tau R$ is positive semidefinite since $\bar{\sigma}(R, A) = \tau'$
- Then $\bar{\sigma}(A, B) \leq \tau$
- Also $\bar{\sigma}(B, A) \leq 1$.

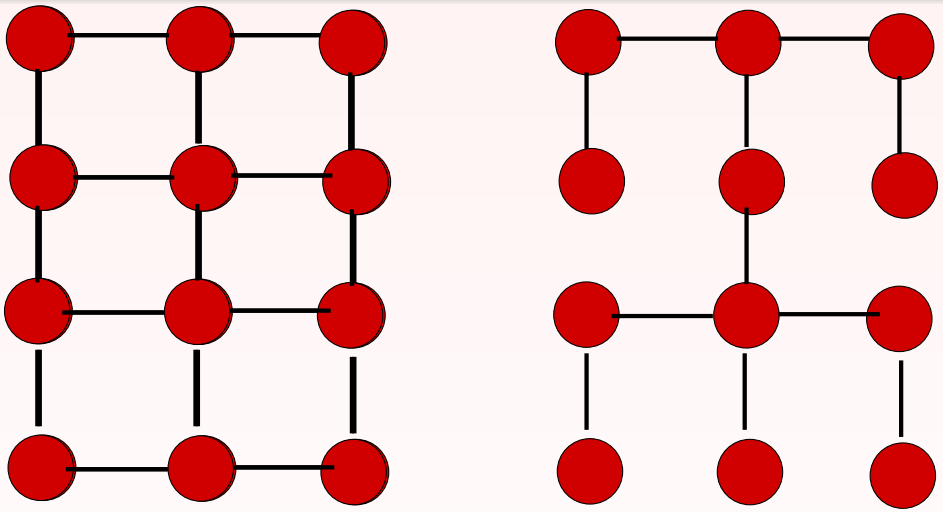
Cholesky decomposition of an M-matrix satisfies this assumption
$(A = LL^T - R, R \text{ is positive semidefinite})$

Combinatorial preconditioners: XI

Vaidya preconditioner 1

Algorithm

Construct a maximum-weight spanning tree of A and use its matrix as a preconditioner



graph of A graph of B

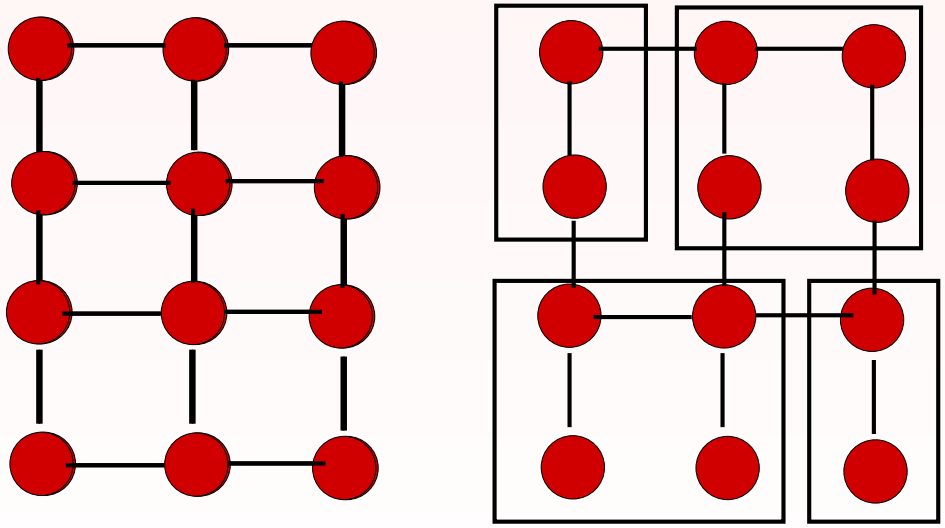
- m nonzeros in $A \Rightarrow$ at most $m/2$ edges in $G(A)$
- $2/m$ fraction of an edge to each path
- path of the maximum lengths of $n-1$
- $\sigma(A,B) \leq O(mn)$, $\sigma(B,A) \leq 1$

Combinatorial preconditioners: XII

Vaidya preconditioner 2

Algorithm

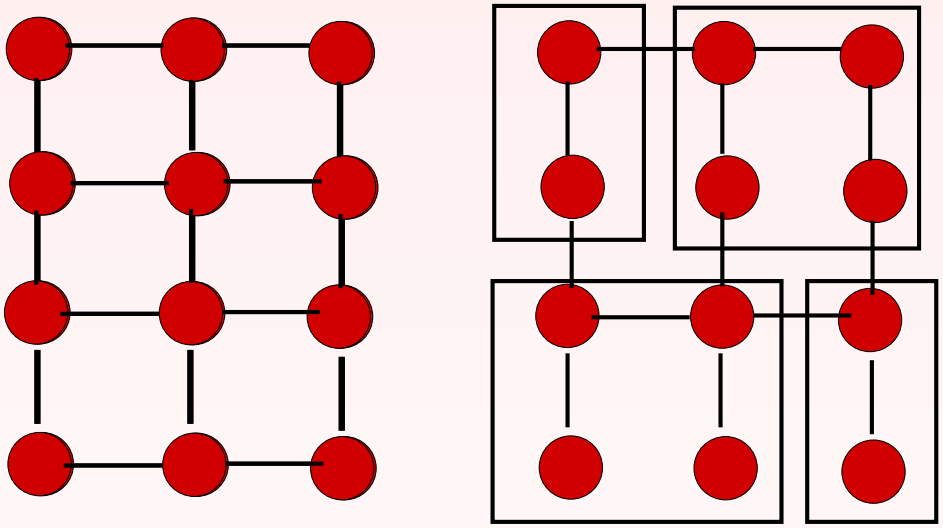
- *Split the matrix graph into t components $V_i, i = 1, \ldots, t$*
- *Construct maximum-weight spanning trees of the components*
- *Connect them pairwise by the edges with the heaviest weights*



graph of A graph of B

Combinatorial preconditioners: XIII

Vaidya preconditioner 2: conditioning

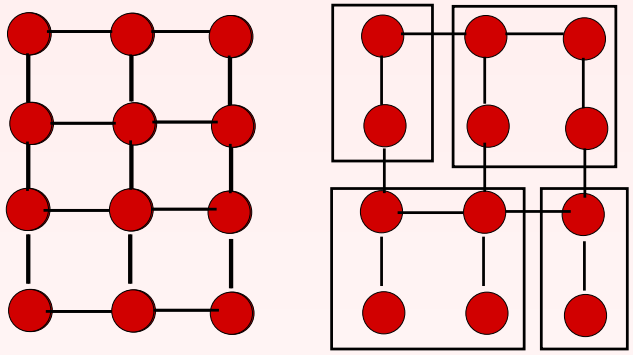


graph of A graph of B

- assume $m \leq dn$ for some d (d maximum degree)
- paths of lengths at most $1 + 2dn/t$
- each edge involved in at most $d \times dn/t \equiv d^2 n/t$ paths
- $\kappa(B^{-1}A)$ bounded by $O(n^2/t^2)$

Combinatorial preconditioners: XIV

Vaidya preconditioner 2: complexity

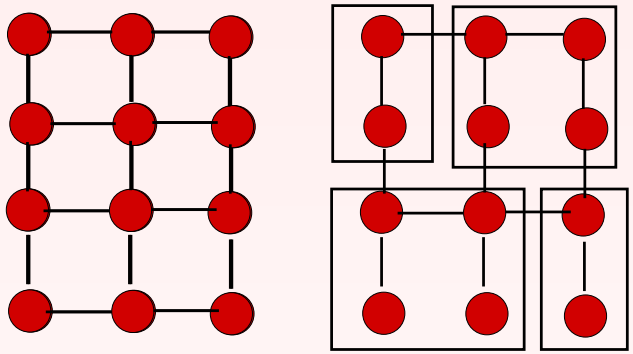


graph of A graph of B

- contraction: eliminate all nodes of degrees 1 and 2: O(n) fill and work
- contracted graph C: number of its internal vertices is at most number of its (componental) leaves
- total number of vertices in C: at most O(number of leaves of all V_i)
- contraction+factoring of C with at most $O(t)$ vertices $\rightarrow$ O(t^6) work, O(n+t^4) nonzeros
- iteration count bounded by $O(\sqrt{n^2/t^2}) = O(n/t)$
- $t = \Theta(n^{0.25}) \Rightarrow$ total work bounded by $O(n/t)O(n + t^4) = O(n^{1.75})$

Combinatorial preconditioners: XV

Vaidya preconditioner 2: complexity in planar case

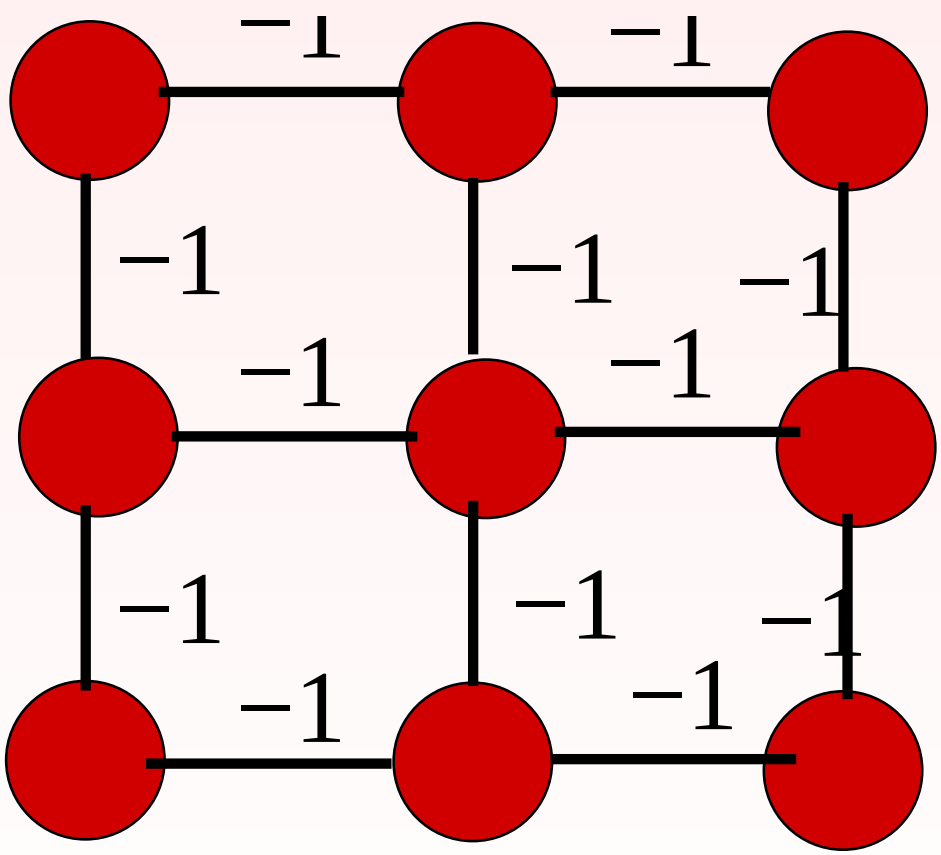


graph of A graph of B

- contracted graph C: O(t) vertices
- number of edges in the planar graph: O(t) (Euler formula; degrees at most 5)
- O(t) edges altogether O($t^{1.5}$) work, O($t \log t$) nonzeros using nested dissection
- iteration count bounded again by $O(\sqrt{n^2/t^2}) = O(n/t)$
- $t = \Theta(n^{0.8}) \Rightarrow$ total work bounded by $O(n/t)O(n + t \log t) = O(n^{1.2})$

Combinatorial preconditioners: XVI

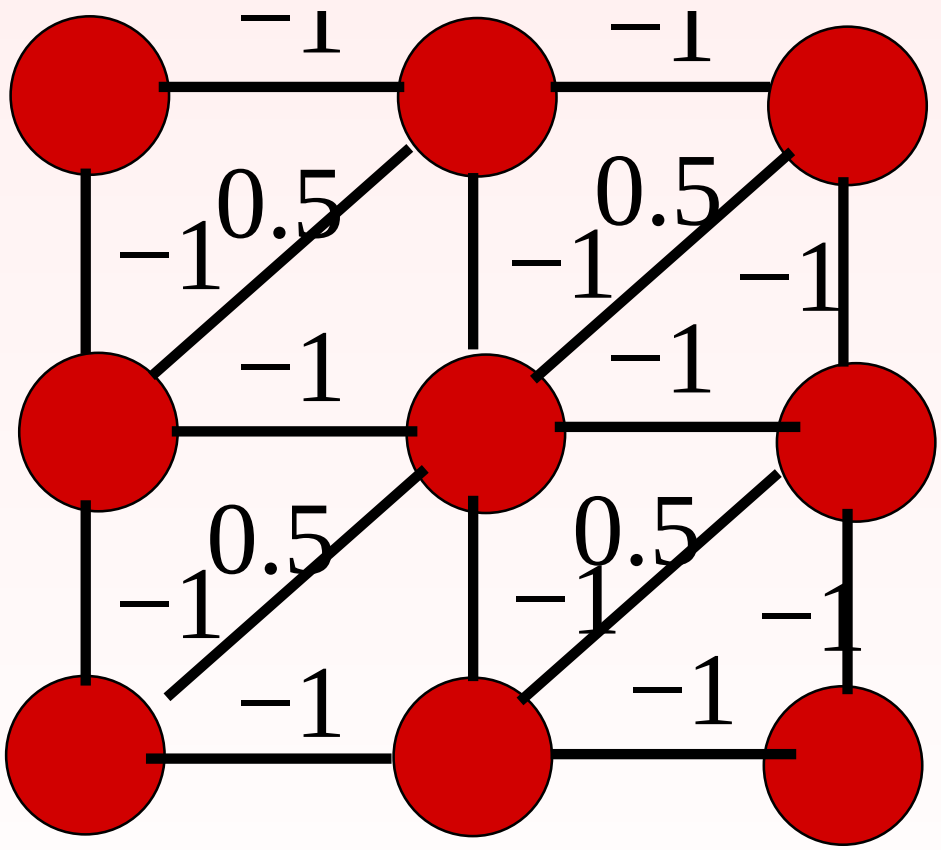
Modified incomplete factorization



2D grid - 5-point stencil

Combinatorial preconditioners: XVII

Modified incomplete factorization: MIC



2D grid - 5-point stencil - fill-in with MIC

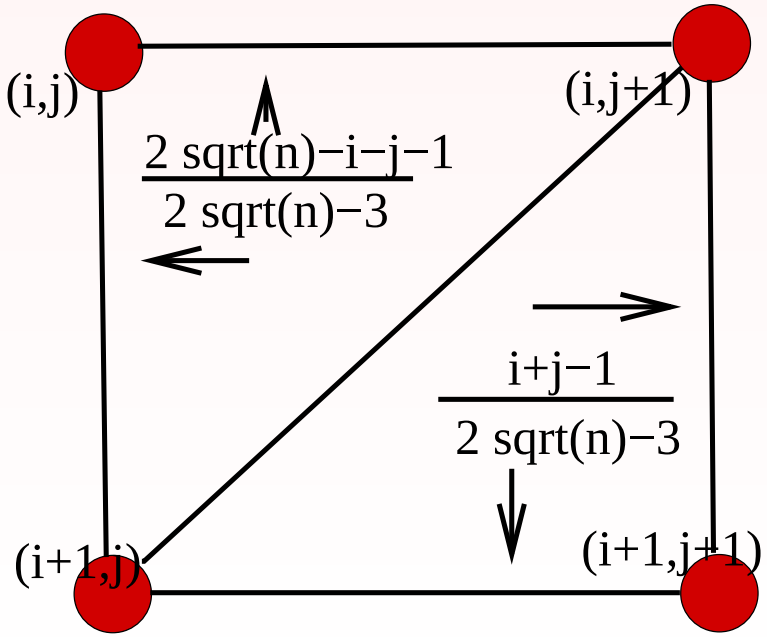
Combinatorial preconditioners: XVIII

Modified incomplete factorization: MIC: Repeat the lemma

Lemma

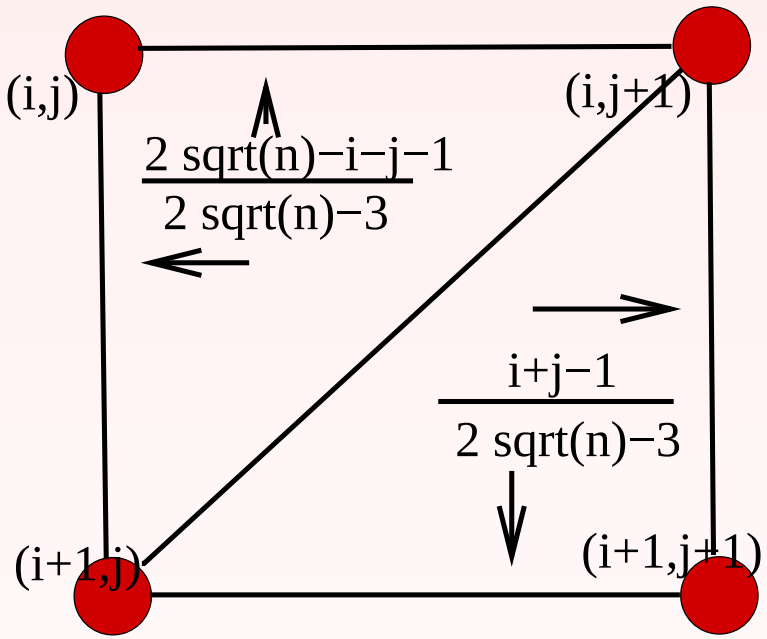
Let $B = A - R$ such that A and and B and R are positive semidefinite. If $\bar{\sigma}(R, A) = \tau' < 1$ then $\bar{\sigma}(B, A)\bar{\sigma}(A, B) \leq 1/(1 - \tau')$.

- Enough to support edges of R such that $\bar{\sigma}(R, A) = \tau' < 1$
- Sophisticated splitting of edges into paths to support R



Combinatorial preconditioners: XIX

Modified incomplete factorization: MIC



- internal weight splittings $\frac{2\sqrt{n}-i-j-1}{2\sqrt{n}-3} + \frac{i+j-2}{2\sqrt{n}-3} = 1$
- support of fill edges: $\frac{2\sqrt{n}-i-j-1}{2\sqrt{n}-3} + \frac{i+j-1}{2\sqrt{n}-3} = \frac{2\sqrt{n}-2}{2\sqrt{n}-3} = 1/\tau'$
- path length is 2, fill-in edge weight is 0.5
- overall $\kappa(B^{-1}A) = 1/(1-\tau') = 2n^{0.5} - 2$

Preconditioners analyzable in this way and reality

- Matrix pwtk.rsa: stiffness matrix, pressurized wind tunnel
- n=217918, nz=5926171 (a triangular part)

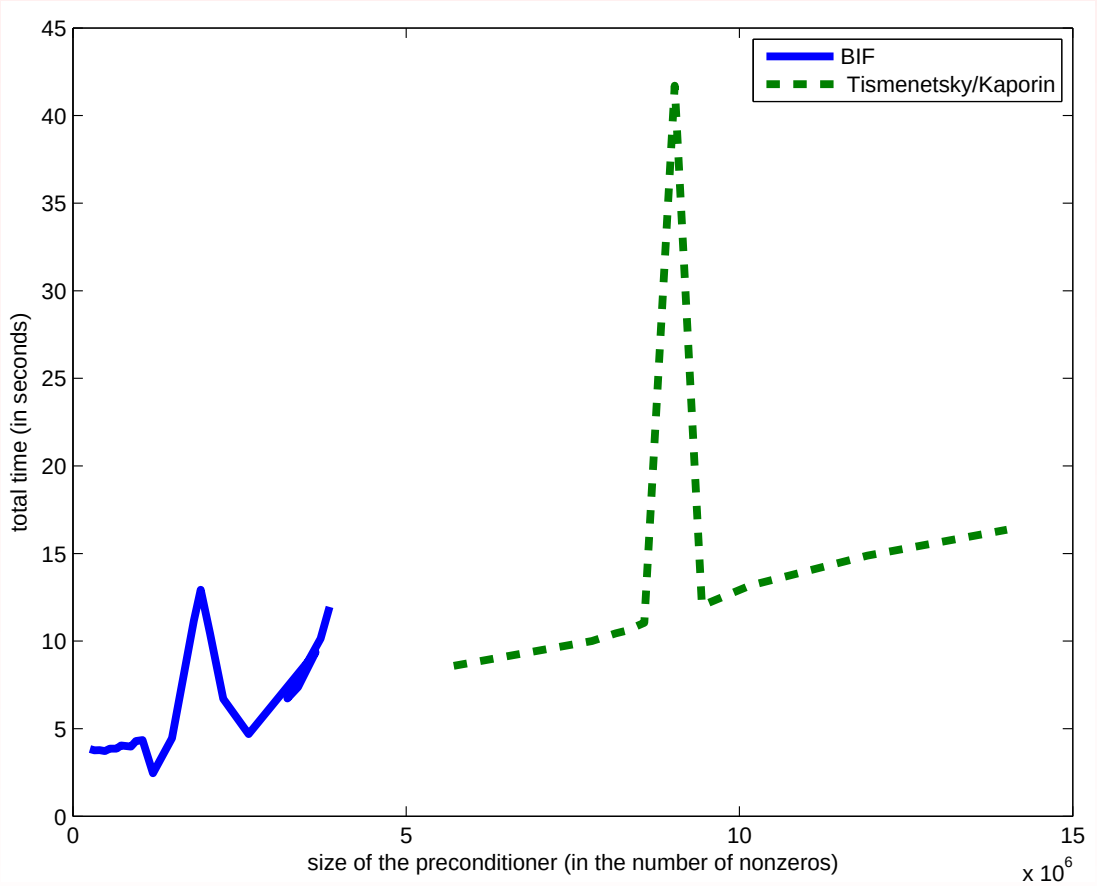
Tested with two preconditioners

- 1) IC with positive semidefinite modification of the Schur complement (Tismenetsky, 1991)
  - Often considered as one of the most robust approaches
  - Suffers from extensive memory demands

$$\begin{pmatrix} d & l^T & s^T \\ l & & \\ s & & B \end{pmatrix} + \begin{pmatrix} 0 & & 0 \\ 0 & \begin{pmatrix} 0 \\ s \end{pmatrix} & \begin{pmatrix} 0 & s^T \end{pmatrix} \end{pmatrix} \rightarrow \begin{pmatrix} d & & l^T & s^T \\ l & \bar{B} \equiv B + \begin{pmatrix} 0 \\ s \end{pmatrix} & \begin{pmatrix} 0 & s^T \end{pmatrix} \\ s & & \end{pmatrix}$$

- 2) IC based on computing both direct and inverse factors (Bru et al., 2008)

Preconditioners analyzable in this way and reality: II



Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

Direct and Inverse Factors

Basic inverse decomposition is in principle very simple

- Inverse decompositions are in principle based on the generalized concept of **QR** decomposition:

$$I = ZU$$

  - U is upper triangular
  - Z is A-orthogonal $Z^T AZ = I$

Consequently

  - U is the Cholesky factor of A
  - Z is its inverse

Direct and Inverse Factors: II

$$\mathbf{I} = \mathbf{ZU},\ \mathbf{I} = \mathbf{WL^T}$$

:

Two generalized Gram-Schmidt recursions

$$z_i^{(j)} = z_i^{(j-1)} - z_j^{(j-1)} \frac{a^j z_i^{(j-1)}}{a^j z_j^{(j-1)}}$$

$$w_i^{(j)} = w_i^{(j-1)} - w_j^{(j-1)} \frac{a_j^T w_i^{(j-1)}}{a_j^T w_j^{(j-1)}}$$

- Fully sparse operations - no relaxations like fixing band or pattern necessary
- Generalized **MGS** in the SPD case

Direct and Inverse Factors: III

- New recursions for Z and V^T:

$$\mathbf{I} = \mathbf{Z}(\mathbf{W^T} + \mathbf{V^T}), \quad \mathbf{I} = \mathbf{W}(\mathbf{Z^T} + \tilde{\mathbf{V}}^\mathbf{T})$$

$$z_i = se_i - \sum_{j=1}^{i-1} \frac{v_j^T e_i}{d_j} z_j$$

$$v_i = (a^i - se^i)^T - \sum_{j=1}^{i-1} \frac{z_j^T (a^i - se^i)}{d_j} v_j$$

Direct and Inverse Factors: III

- New recursions for Z and V^T:

$$\mathbf{I} = \mathbf{Z}(\mathbf{W^T} + \mathbf{V^T}), \quad \mathbf{I} = \mathbf{W}(\mathbf{Z^T} + \tilde{\mathbf{V}}^\mathbf{T})$$

$$z_i = se_i - \sum_{j=1}^{i-1} \frac{v_j^T e_i}{d_j} z_j$$

$$v_i = (a^i - se^i)^T - \sum_{j=1}^{i-1} \frac{z_j^T (a^i - se^i)}{d_j} v_j$$

- Both U and Z **contained** in V^T, similarly for L

Direct and Inverse Factors: III

- New recursions for Z and V^T:

$$\mathbf{I} = \mathbf{Z}(\mathbf{W^T} + \mathbf{V^T}), \quad \mathbf{I} = \mathbf{W}(\mathbf{Z^T} + \tilde{\mathbf{V}}^\mathbf{T})$$

$$z_i = se_i - \sum_{j=1}^{i-1} \frac{v_j^T e_i}{d_j} z_j$$

$$v_i = (a^i - se^i)^T - \sum_{j=1}^{i-1} \frac{z_j^T (a^i - se^i)}{d_j} v_j$$

- Both U and Z **contained** in V^T, similarly for L

(Bru, Cerdán, Marín, Mas, SISC, 2006; Bru, Marín, Mas, T., SISC, 2008; Bru, Marín, Mas, T., SIMAX, 2010);

Direct and Inverse Factors: III

- New recursions for Z and V^T:

$$\mathbf{I} = \mathbf{Z}(\mathbf{W^T} + \mathbf{V^T}), \quad \mathbf{I} = \mathbf{W}(\mathbf{Z^T} + \tilde{\mathbf{V}}^\mathbf{T})$$

$$z_i = se_i - \sum_{j=1}^{i-1} \frac{v_j^T e_i}{d_j} z_j$$

$$v_i = (a^i - se^i)^T - \sum_{j=1}^{i-1} \frac{z_j^T (a^i - se^i)}{d_j} v_j$$

- Both U and Z **contained** in V^T, similarly for L

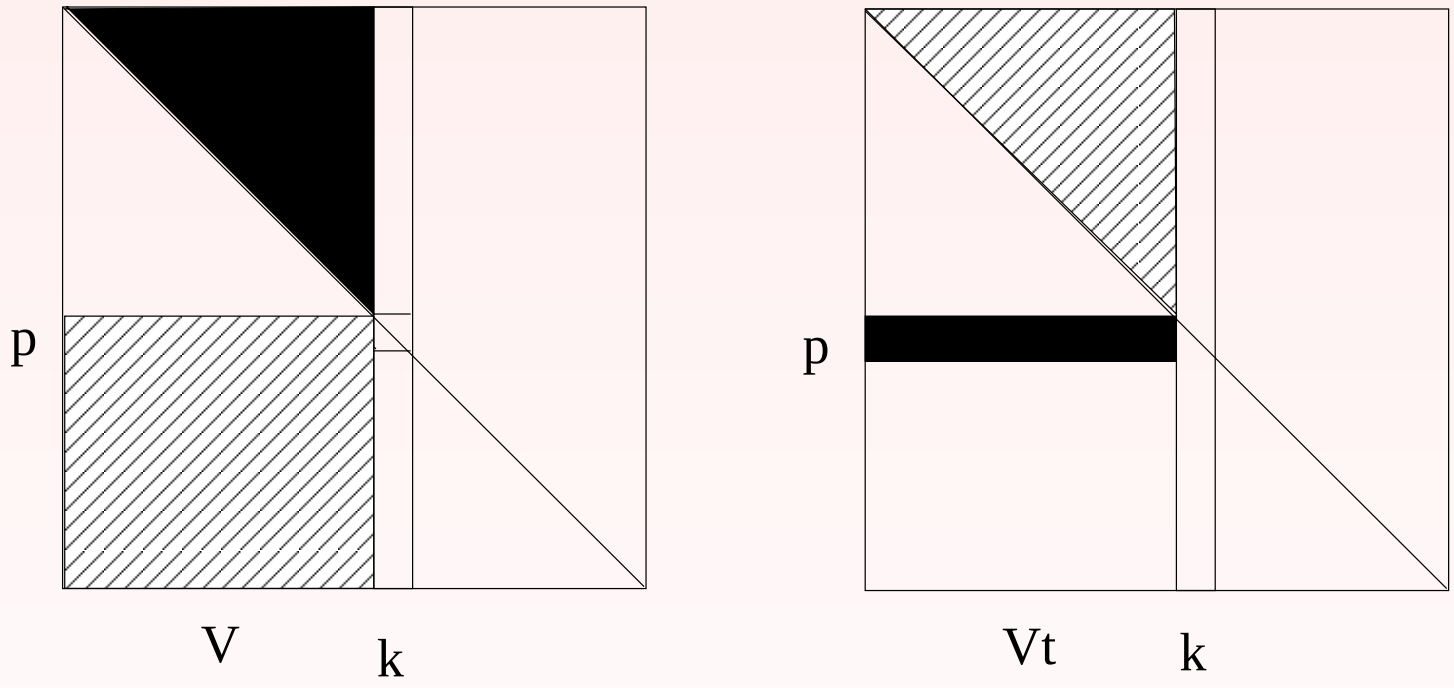
(Bru, Cerdán, Marín, Mas, SISC, 2006; Bru, Marín, Mas, T., SISC, 2008; Bru, Marín, Mas, T., SIMAX, 2010);
A lot of other work, e.g., Bollhöfer, Saad; 2002; Bollhöfer, 2003

Direct and Inverse Factors: IV

$$\mathbf{I} = \mathbf{Z}(\mathbf{W}^{\mathbf{T}} + \mathbf{V}^{\mathbf{T}}), \quad \mathbf{I} = \mathbf{W}(\mathbf{Z}^{\mathbf{T}} + \tilde{\mathbf{V}}^{\mathbf{T}})$$

- Computation of L^{-T}, U, U^{-1}, L is interleaved.
- It uses each other's intermediate data
- Straightforward sparse, column-based algorithms
- Explicit data interconnection of the recursions
  - connected by dropping
  - full interconnection by data exchange between direct and inverse factors possible as well
  - ill-conditioning in inverse factors directly detected.
- Some **practical limitations as well**

Direct and Inverse Factors: V



- $v_{1:p-1}$ computed using fully filled areas
- $v_{p+1:n}$ computed using dashed areas
- direct and inverse factors influence each other

Direct and Inverse Factors: VI

Example: matrix PWTK, n=217,918, nnz=5,926,171

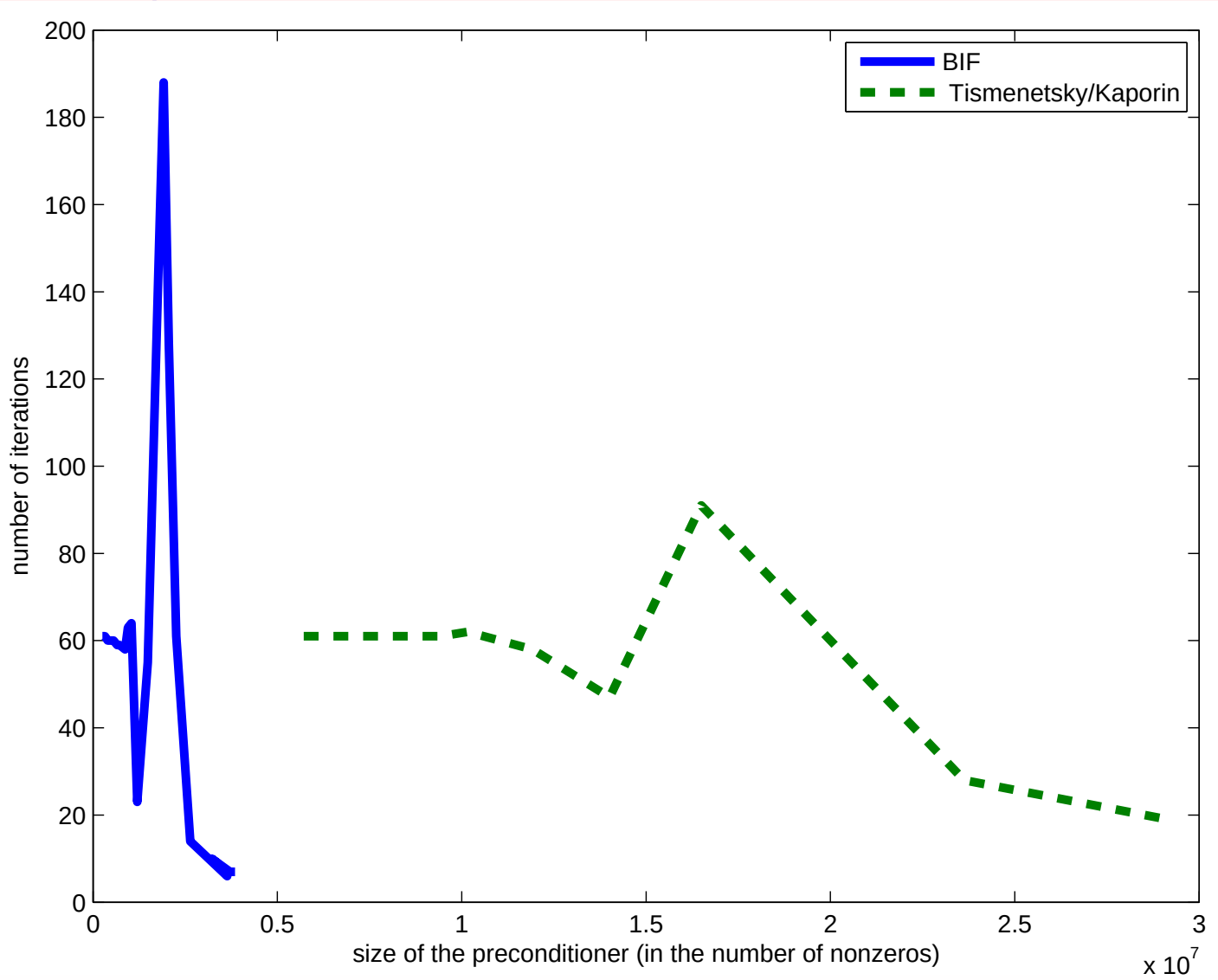


Figure: Iteration counts for CG preconditioned by BIF and Tismenetsky/Kaporin

Direct and Inverse Factors: VII

Example: matrix PWTK, n=217,918, nnz=5,926,171

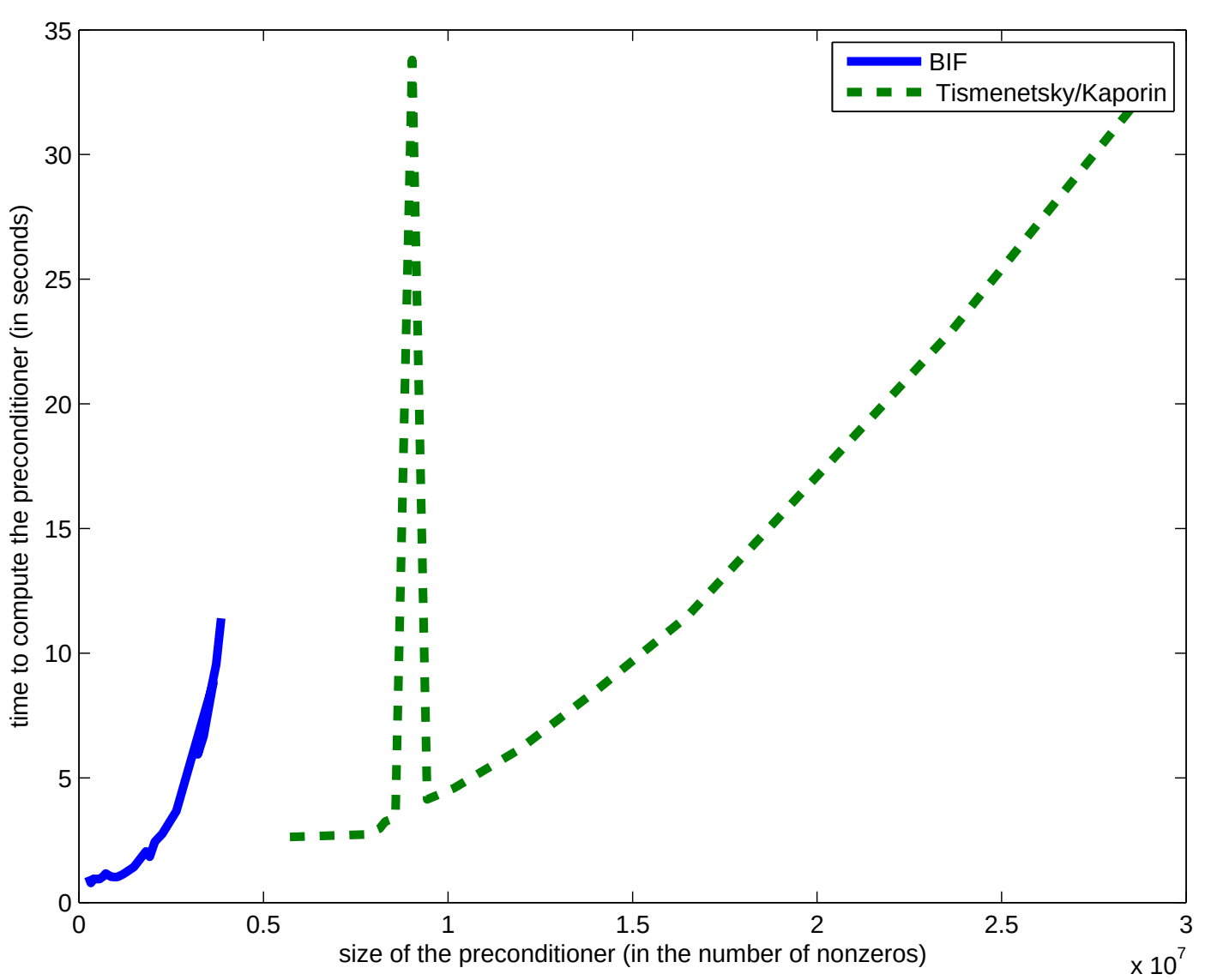


Figure: Preconditioner construction time for CG preconditioned by BIF and

Direct and Inverse Factors: VIII

Example: matrix PWTK, n=217,918, nnz=5,926,171

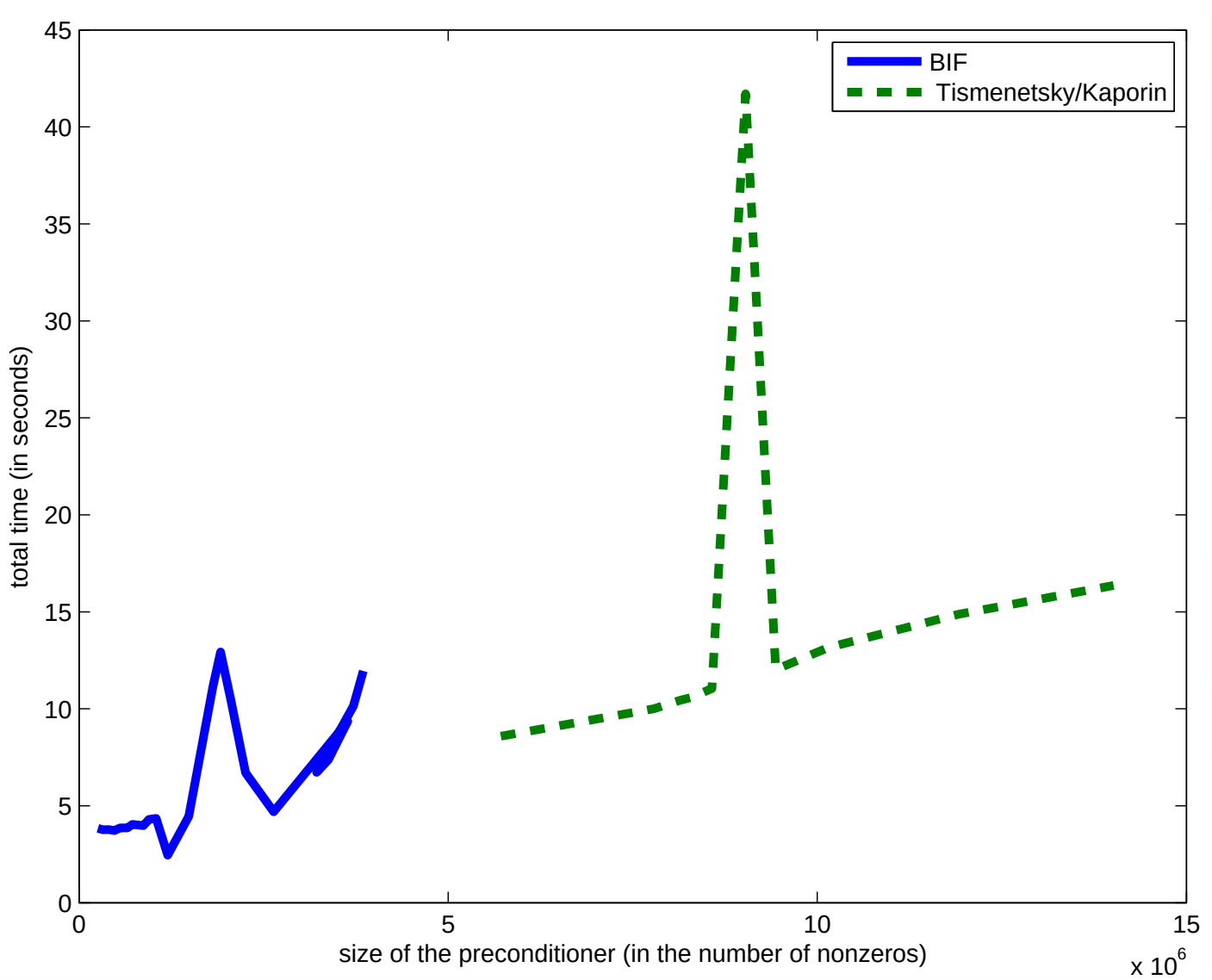


Figure: Preconditioner construction time for CG preconditioned by BIF and

Direct and Inverse Factors: IX

Example: matrix CFD2, n=123,440, nnz=1,605,669

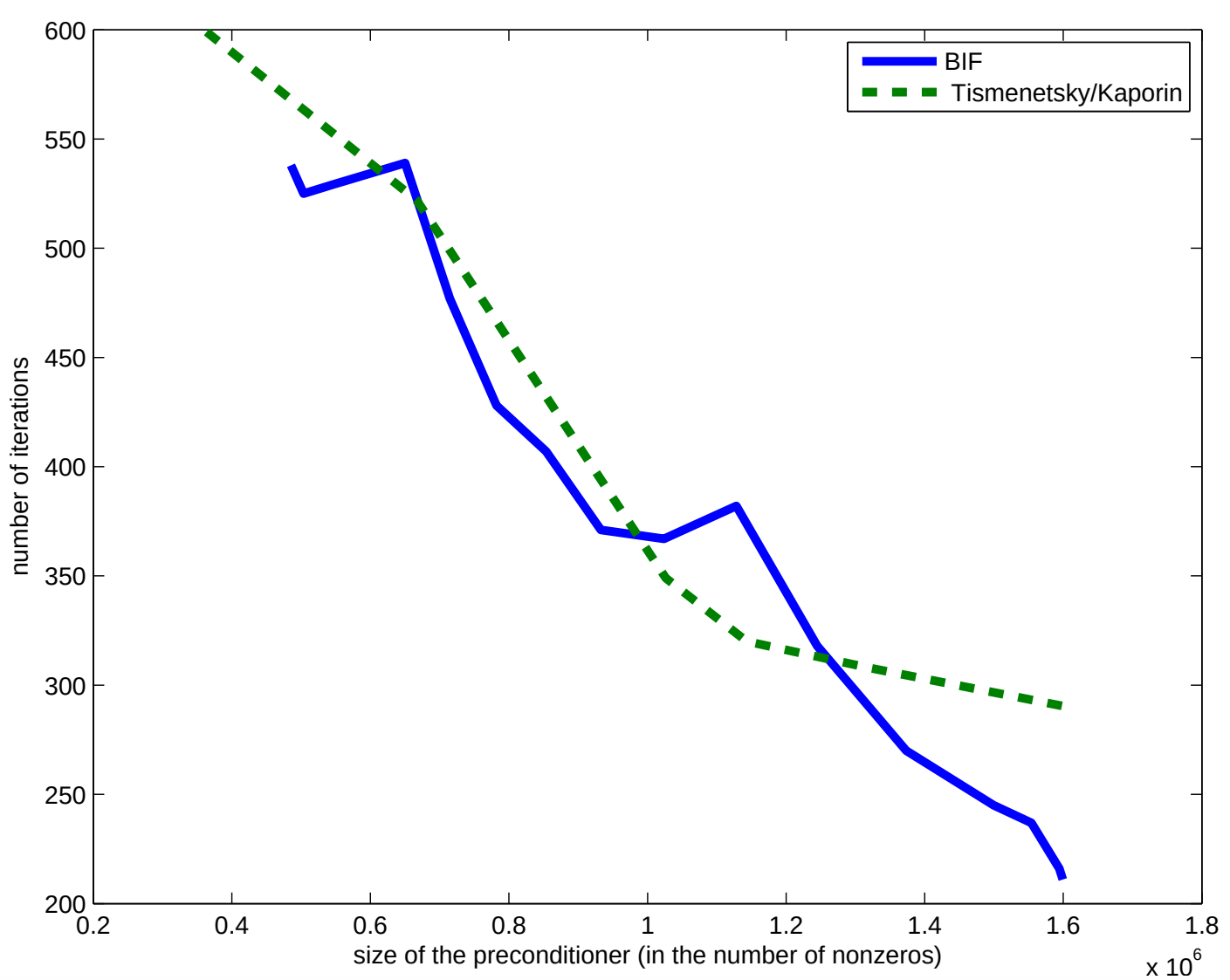


Figure: Iteration counts for CG preconditioned by BIF and Tismenetsky/Kaporin

Direct and Inverse Factors: X

Example: matrix CFD2, n=123,440, nnz=1,605,669

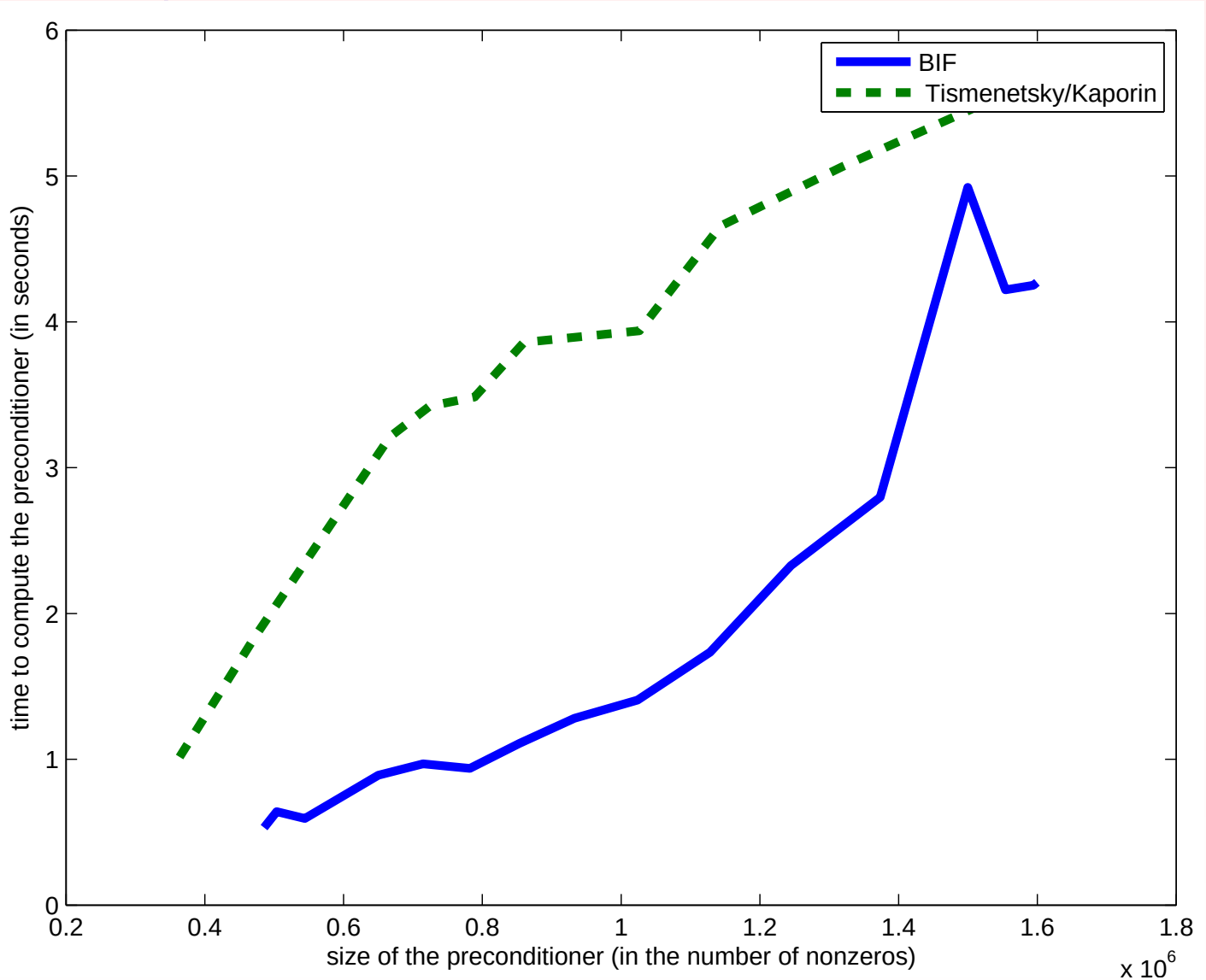


Figure: Preconditioner construction time for CG preconditioned by BIF and

Direct and Inverse Factors: XI

Example: matrix CFD2, n=123,440, nnz=1,605,669

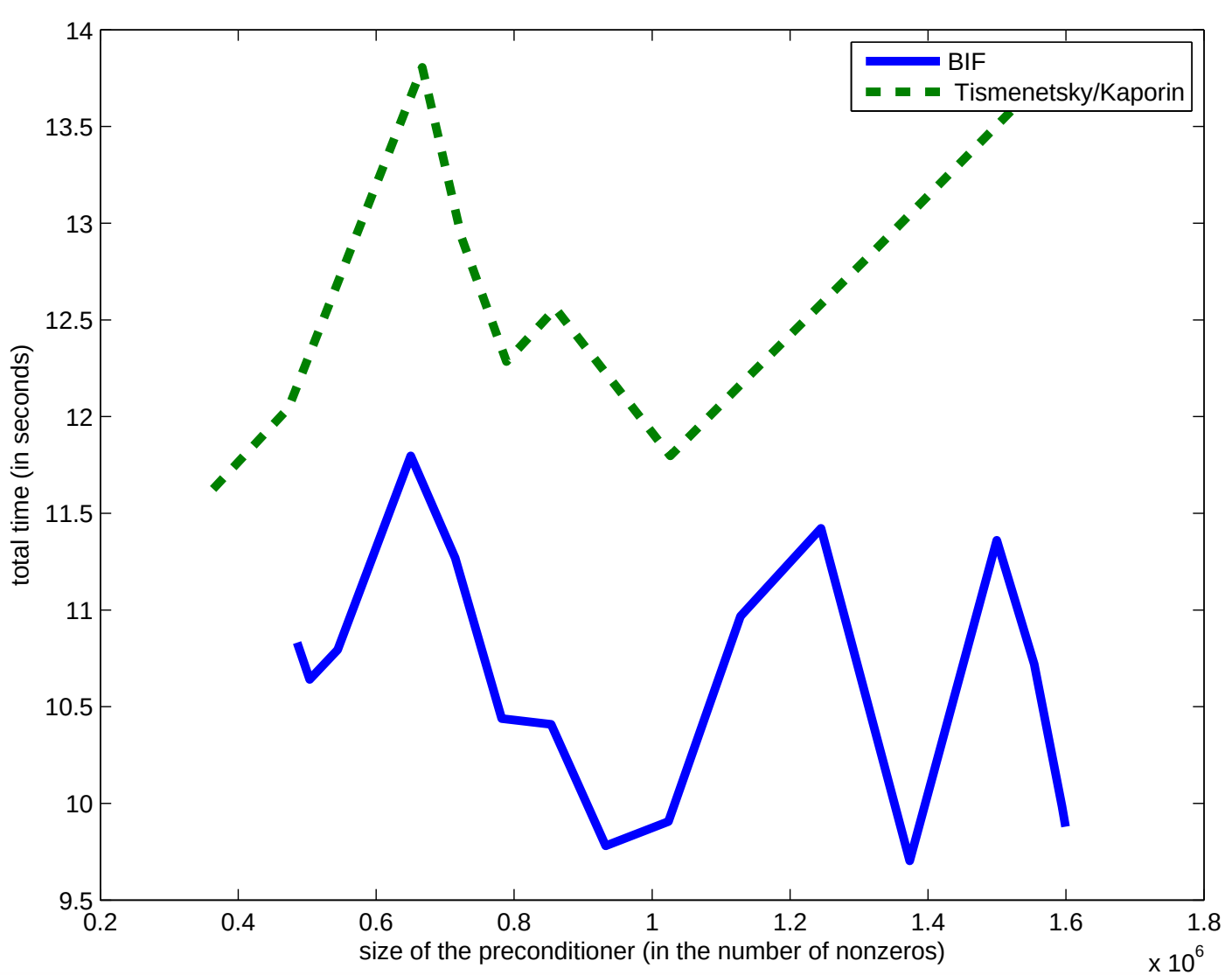
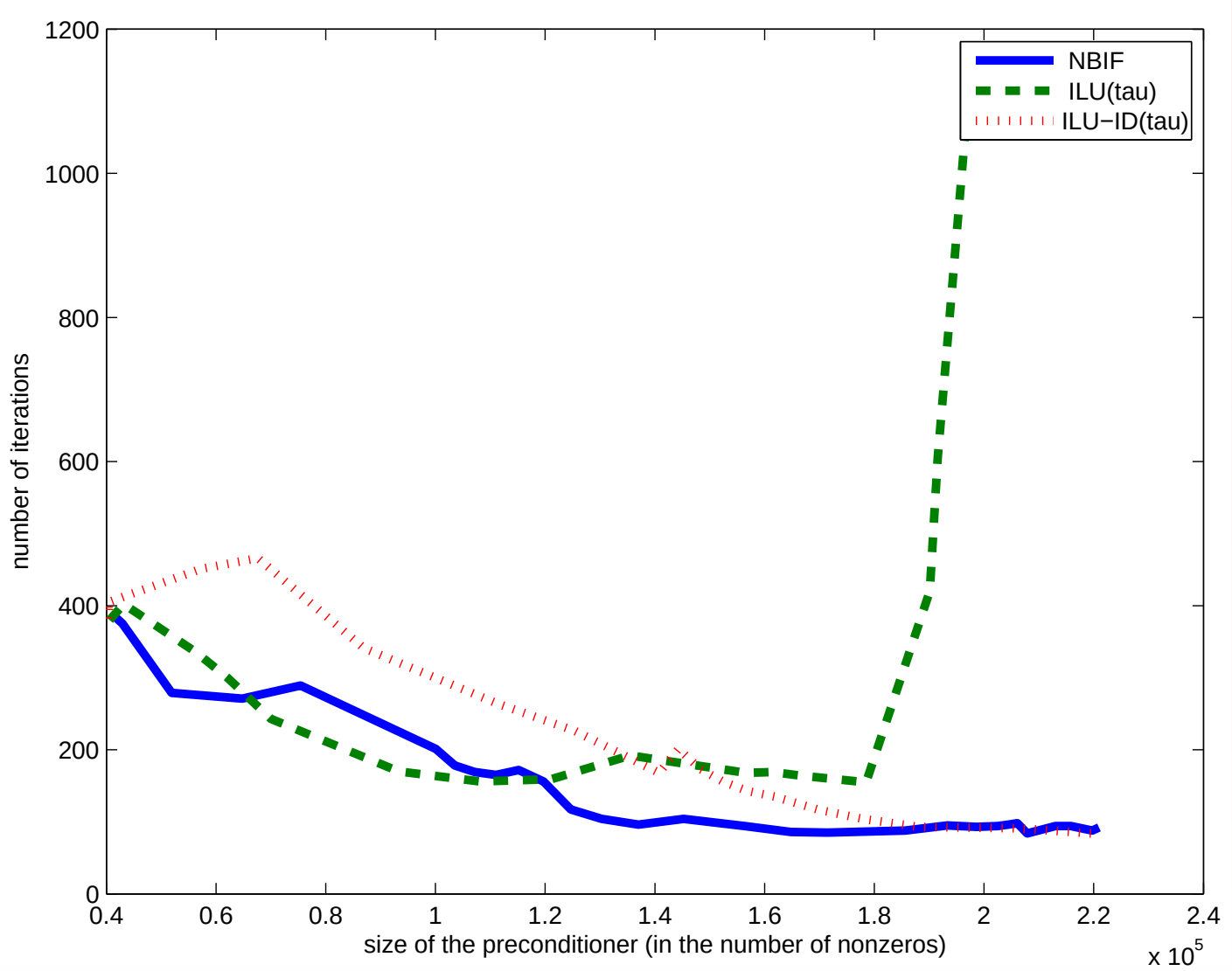


Figure: Preconditioner construction time for CG preconditioned by BIF and

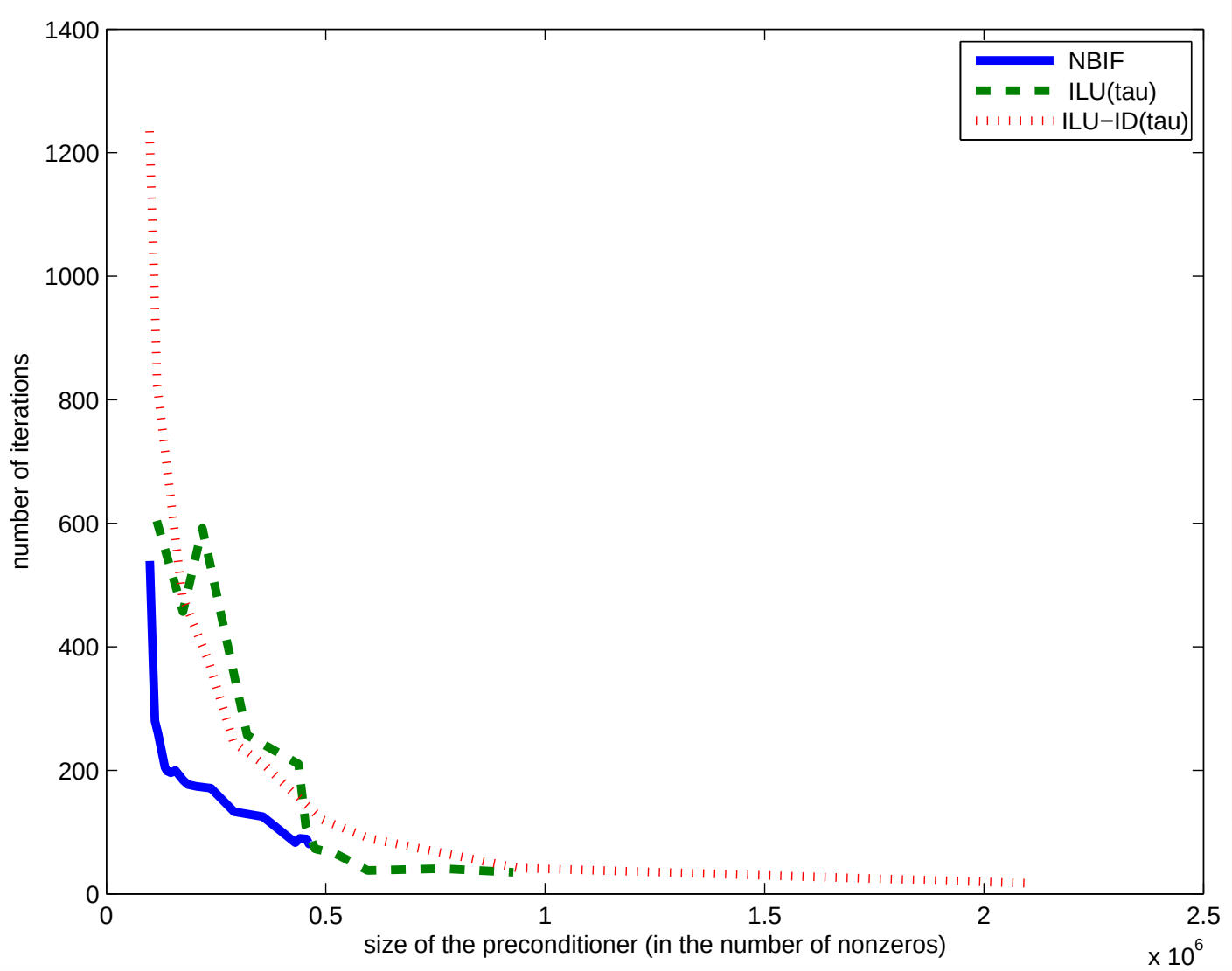
Direct and Inverse Factors: XII

Example: matrix CHEM_MASTER, n=40,401, nnz=201,201



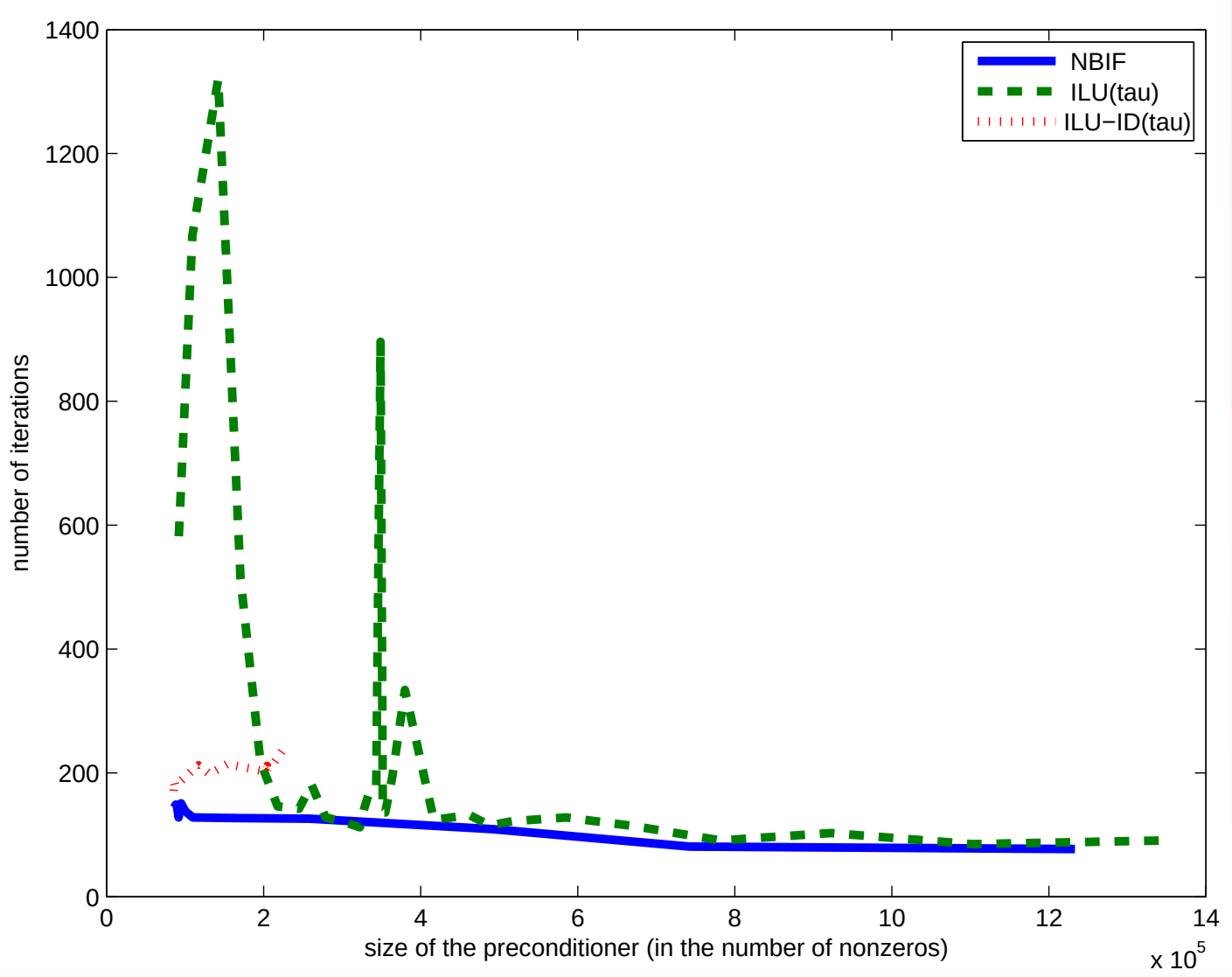
Direct and Inverse Factors: XIII

Example: matrix EPB3, n=84,617, nnz=463,625



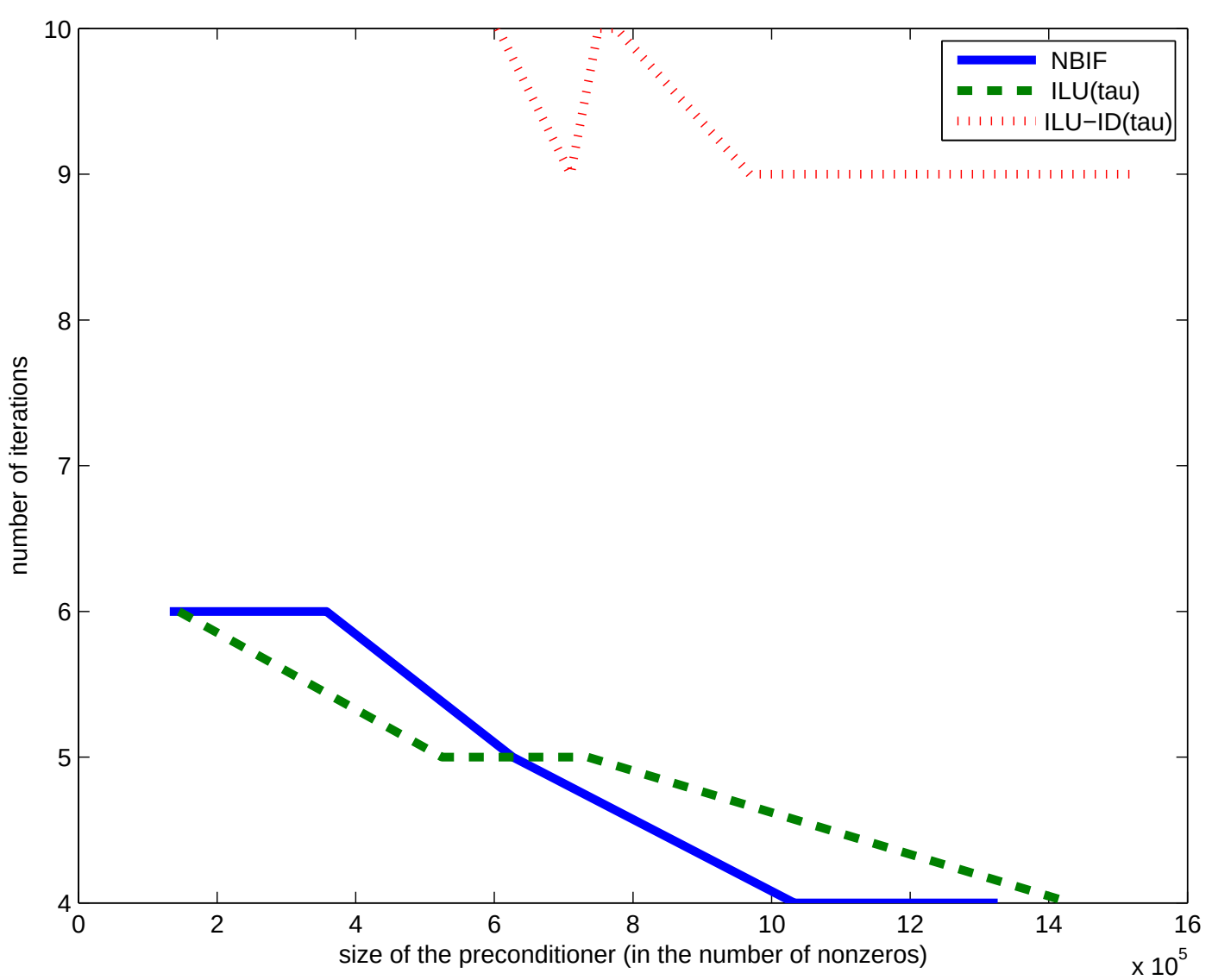
Direct and Inverse Factors: XIV

Example: matrix POISSON3DB, n=85,623, nnz=2,374,949



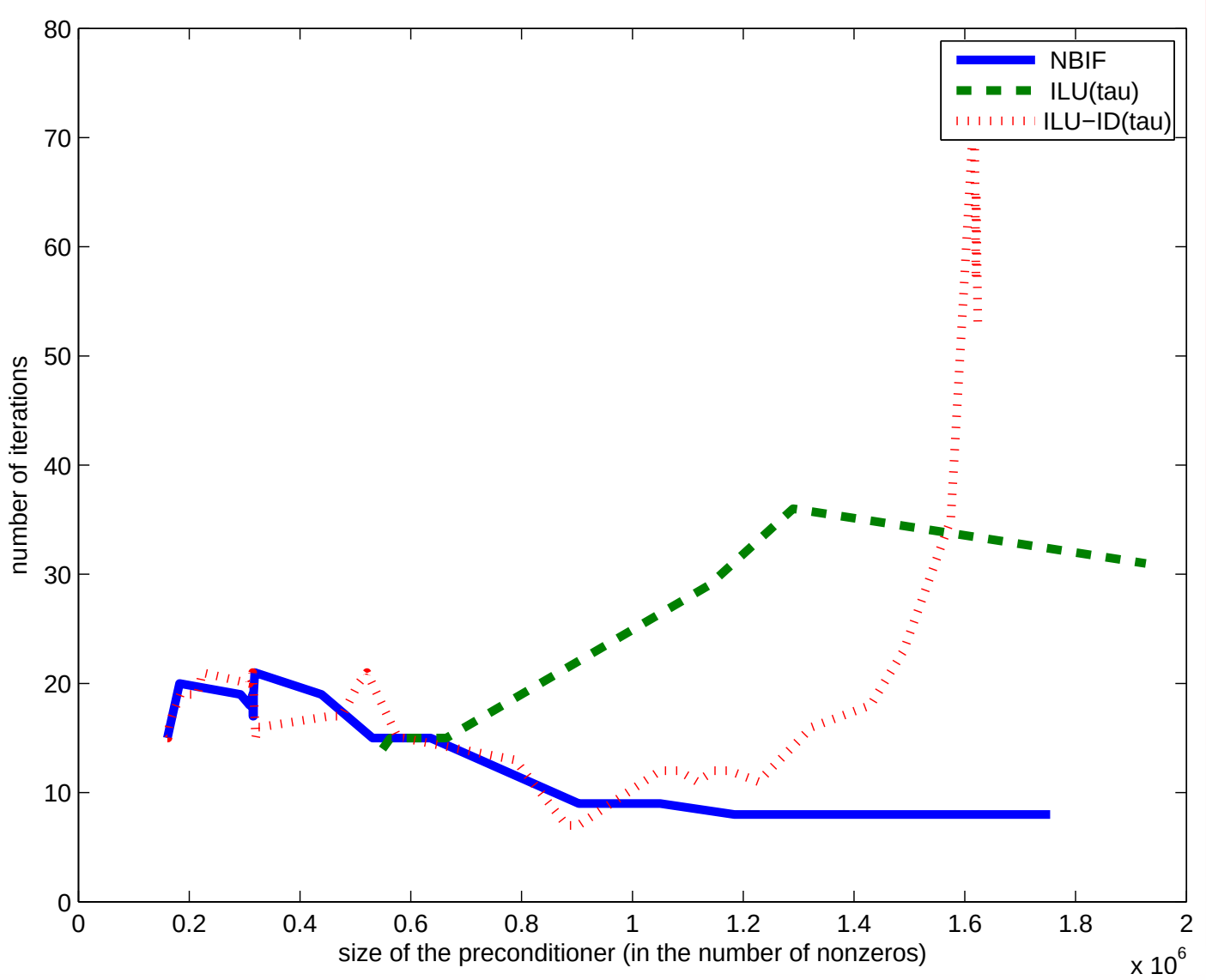
Direct and Inverse Factors: XV

Example: matrix CAGE12, n=130,228, nnz=2,032,536



Direct and Inverse Factors: XVI

Example: matrix MAJOR, n=160,000, nnz=1,750,416



Outline

1. Foreword
2. Direct methods and algebraic preconditioners
3. Sparsity
4. Direct Methods
5. Notes on direct methods: models, complexity, parallelism
6. Decomposition and computer architectures
7. From direct to iterative methods
8. Algebraic preconditioners and complexity
9. Combining direct and inverse decompositions
10. Conclusions

Conclusions

- Direct methods still strongly developing as stand-alone approaches. A lot of open algorithmic/implementational questions.

Conclusions

- Direct methods still strongly developing as stand-alone approaches. A lot of open algorithmic/implementational questions.
- Direct and iterative methods coexist together sharing some algorithms and techniques.

Conclusions

- Direct methods still strongly developing as stand-alone approaches. A lot of open algorithmic/implementational questions.
- Direct and iterative methods coexist together sharing some algorithms and techniques.
- Borrowing from each other may be the way for more robust solvers.

Last but not least

Thank you for your attention!

Last but not least

Thank you for your attention!

Last but not least

Thank you for your attention!

Last but not least

Thank you for your attention!