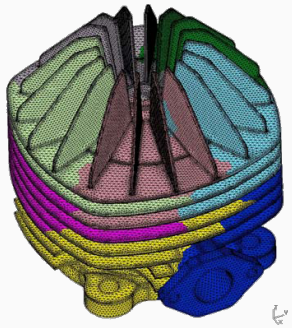
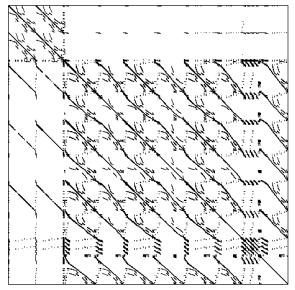
Solving linear systems - Sparse Linear Algebra

P. Berger, M. Daydé, R. Guivarch, E. Simon, D. Ruiz (INP-ENSEEIHT- IRIT),
A. Buttari (CNRS, INP-ENSEEIHT-IRIT),
P. Amestoy, J.-Y. L'Excellent (Mumps Technologies),
A. Guermouche (Univ. Bordeaux-LaBRI),
F.-H. Rouet (Ansys, USA),
Bora Uçar (INRIA-CNRS/LIP-ENS Lyon) and
L. Giraud (INRIA)

2025 - ModIA

Context


Linear System $Ax = b$

At the foundations of many **scientific computing applications** (discretization of PDEs, step of an optimization method, ...).

Large-scale computations...

Up to few **billions (10^9) of unknowns**, applications asking TeraBytes (10^{12}) of memory and Exaflops (10^{18}) of computation.

...require large-scale computers.

Increasingly **large numbers of cores** available, high **heterogeneity in the computation** (CPU, GPU, FPGA, TPU, etc), and high **heterogeneity in data motions** (RAM to cache, out-of-core, node to node transfer, etc).

The sparse case

Matrix sparsity

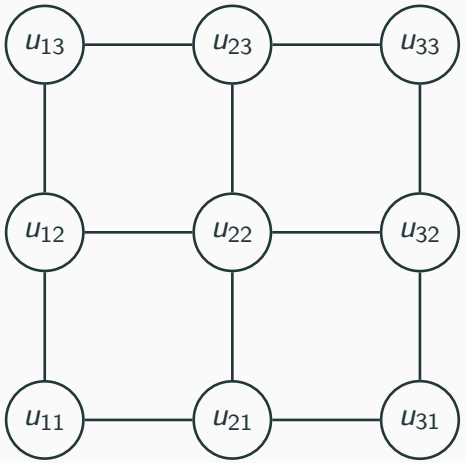
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} & & \\ & A & \\ & & \end{bmatrix} \begin{bmatrix} \\ x \\ \\ \end{bmatrix} = \begin{bmatrix} \\ b \\ \\ \end{bmatrix}$$

The sparse case

Matrix sparsity

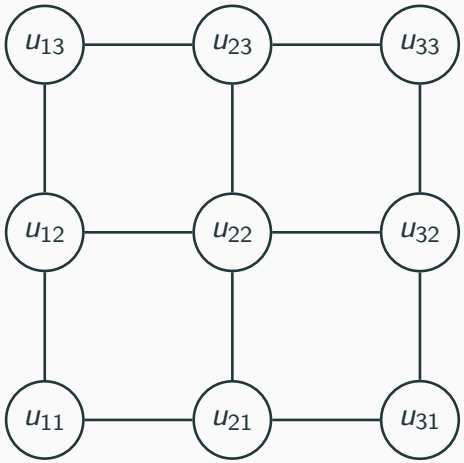
A **sparse matrix** is "any matrix with enough zeros that it pays to take advantage of them" (Wilkinson)

Physical problems are one of the main supplier for sparse linear systems. Let's consider the solution of the Poisson's PDE:

$$-\Delta u(x, y) = f$$

Finite-difference discretization gives the discrete equation:

$$(-\Delta u)_{i,j} \approx \frac{1}{h^2}(-u_{i+1,j} - u_{i-1,j} - u_{i,j+1} - u_{i,j-1} + 4u_{i,j}) = f_{i,j}$$



$$\begin{bmatrix} 4 & -1 & & -1 & & & & & \\ -1 & 4 & -1 & & -1 & & & & \\ & -1 & 4 & & & -1 & & & \\ -1 & & & 4 & -1 & & -1 & & \\ & -1 & & -1 & 4 & -1 & & -1 & \\ & & -1 & & -1 & 4 & & & -1 \\ & & & -1 & & & 4 & -1 & \\ & & & & -1 & & -1 & 4 & -1 \\ & & & & & -1 & & -1 & 4 \end{bmatrix} \begin{bmatrix} u_{11} \\ u_{12} \\ u_{13} \\ u_{21} \\ u_{22} \\ u_{23} \\ u_{31} \\ u_{32} \\ u_{33} \end{bmatrix} = \begin{bmatrix} h^2 f_{11} \\ h^2 f_{12} \\ h^2 f_{13} \\ h^2 f_{21} \\ h^2 f_{22} \\ h^2 f_{23} \\ h^2 f_{31} \\ h^2 f_{32} \\ h^2 f_{33} \end{bmatrix}$$

Solvers

What are the ways to solve a $Ax = b \in \mathbb{R}^n$ on computers ?

Iterative solvers

Compute a sequence of x_k converging towards x.

Examples: Gauss-Seidel, SOR, Steepest Descent, Conjugate Gradient, Krylov subspace methods, etc.

- Low computational cost and memory consumption if the convergence is quick (about $\mathcal{O}(n^2)$ (dense case)) operations per iteration), ...
- BUT convergence depends on the matrix properties.

Direct solvers

Compute a factorization of A followed by forward and backward substitutions.

Examples: LDL^T, LU, QR, etc.

- High computational cost and memory consumption...
- BUT they are robust and easy to use.

Solvers

What are the ways to solve a sparse $Ax = b \in \mathbb{R}^n$ on computers ?

Iterative solvers

Compute a sequence of x_k converging towards x.

Examples: Gauss-Seidel, Steepest Descent, Conjugate Gradient, SOR, Krylov subspace methods, etc.

- Low computational cost and memory consumption if the convergence is quick (about $\mathcal{O}(nnz(A))$ (sparse case)) operations per iteration), ...
- BUT convergence depends on the matrix properties.

Direct solvers: dense/sparse matrix $\Rightarrow$ dense/sparse solver

Compute a factorization of A followed by forward and backward substitutions.

Examples: LDL^T, LU, QR, etc. LL^T Cholesky

- High computational cost and memory consumption...
- BUT they are robust and easy to use.

Outline

Sparse Direct Solvers - Ordering

Sparse Direct Solvers - Multifrontal Method

Sparse Direct Solvers - Ordering

A selection of references

- Books
  - Duff, Erisman and Reid, Direct methods for Sparse Matrices, Clarendon Press, Oxford 1986.
  - Dongarra, Duff, Sorensen and van der Vorst, Solving Linear Systems on Vector and Shared Memory Computers, SIAM, 1991.
  - Davis, Direct methods for sparse linear systems, SIAM, 2006.
  - George, Liu, and Ng, Computer Solution of Sparse Positive Definite Systems.
  - Saad, Iterative methods for sparse linear systems, 2nd edition, SIAM, 2004.
- Articles
  - Gilbert and Liu, Elimination structures for unsymmetric sparse LU factors, SIMAX, 1993.
  - Liu, The role of elimination trees in sparse factorization, SIMAX, 1990.
  - Heath, Ng and Peyton, Parallel Algorithms for Sparse Linear Systems, SIAM review 1991.

Sparse Direct Solvers - Ordering

Sparse matrices

Sparse matrices

Example:

$$\begin{array}{rcrcrcl} 3\,x_1 & + & 2\,x_2 & & & = & 5 \\ & & 2\,x_2 & - & 5\,x_3 & = & 1 \\ 2\,x_1 & & & + & 3\,x_3 & = & 0 \end{array}$$

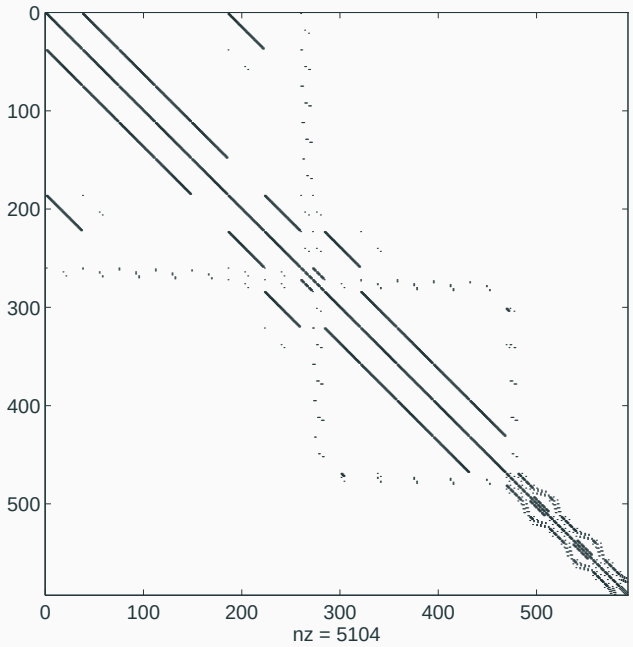
can be represented as

$$\mathbf{A}\mathbf{x} = \mathbf{b},$$

where $\mathbf{A} = \begin{pmatrix} 3 & 2 & 0 \\ 0 & 2 & -5 \\ 2 & 0 & 3 \end{pmatrix}$, $\mathbf{x} = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix}$, and $\mathbf{b} = \begin{pmatrix} 5 \\ 1 \\ 0 \end{pmatrix}$

Sparse matrix: only nonzeros are stored.

Sparse matrix

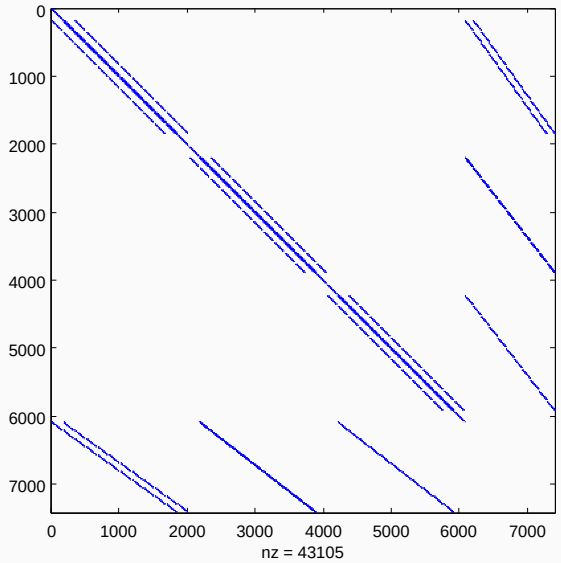


number of non-zero

Matrix dwt_592.rua (n=592, nnz=5104, 1.5%);
Structural analysis of a submarine

Sparse matrix

Matrix from Computational Fluid Dynamics;
(collaboration Univ. Tel Aviv)

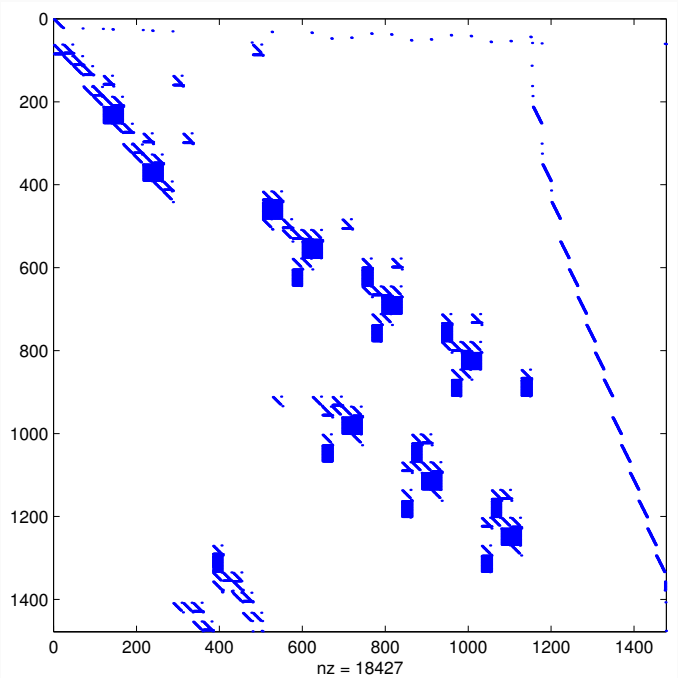


"Saddle-point" problem

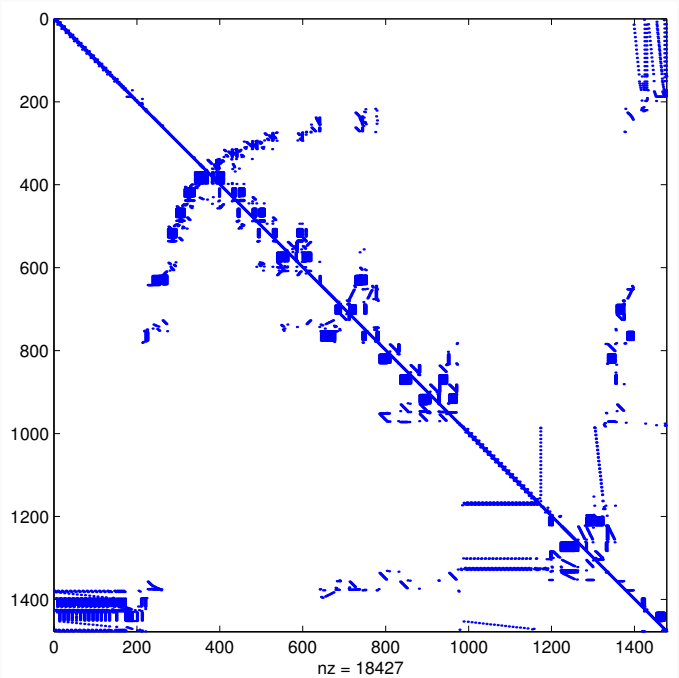
$n \approx 7400,\ nnz = 43105, 0.07\%$

Preprocessing sparse matrices

Original ($A =$LHR01)



Preprocessed matrix (A'(LHR01))



Modified Problem: $A'x' = b'$ with $A' = P_n P D_r A D_c Q P^T Q_n$

Factorization process

Solution of $\mathbf{Ax} = \mathbf{b}$

- $\mathbf{A}$ is unsymmetric :
  - $\mathbf{A}$ is factorized as: $\mathbf{A} = \mathbf{LU}$, where
    $\mathbf{L}$ is a lower triangular matrix, and
    $\mathbf{U}$ is an upper triangular matrix.
  - Forward-backward substitution: $\mathbf{Ly} = \mathbf{b}$ then $\mathbf{Ux} = \mathbf{y}$
- $\mathbf{A}$ is symmetric:
  - general $\mathbf{A} = \mathbf{LDL}^T$
  - positive definite $\mathbf{A} = \mathbf{LL}^T$

Creux

2nnz(L)
2nnz(U)

Difficulties

- Only non-zero values are stored
- Factors **L** and **U** have far more nonzeros than **A**
- Data structures are complex
- Computations are only a small portion of the code (the rest is data manipulation)
- Memory size is a limiting factor ($\rightarrow$ *out-of-core solvers*)

Difficulties

- Only non-zero values are stored
- Factors **L** and **U** have far more nonzeros than **A** $=>$ fill-in
- Data structures are complex
- Computations are only a small portion of the code (the rest is data manipulation)
- Memory size is a limiting factor ($\rightarrow$ *out-of-core solvers*)

Factorization of sparse matrices: problems

The factorization of a sparse matrix is problematic due to the presence of fill-in.

The basic LU step:

$$a_{i,j}^{(k+1)} = a_{i,j}^{(k)} - \frac{a_{i,k}^{(k)} a_{k,j}^{(k)}}{a_{k,k}^{(k)}}$$

Even if $a_{i,j}^{(k)}$ is null, $a_{i,j}^{(k+1)}$ can be a nonzero

$$A = \begin{pmatrix} a & \bullet & & & \bullet & \\ & b & & \bullet & & \bullet \\ \bullet & & c & & \bullet & \\ & & & d & & \bullet\ \bullet \\ & \bullet & & e & & \bullet\ \bullet \\ & & & & f\ \bullet & \bullet \\ & & \bullet & & \bullet\ g & \\ \bullet & & & & \bullet & h\ \bullet\ \bullet \\ & & & \bullet\ \bullet & & \bullet\ i \\ & & \bullet & \bullet\ \bullet & & \bullet & & j \end{pmatrix} \qquad F = \begin{pmatrix} a & \bullet & & & \bullet & \\ & b & & \bullet & & \bullet \\ \bullet & & c & & \bullet & \circ \\ & & & d & & \bullet\ \bullet \\ & \bullet & & e & & \bullet\ \bullet \\ & & & & f\ \bullet & \bullet \\ & & \bullet & & \bullet\ g & \circ \\ \bullet & & \circ & & \bullet & \circ\ h\ \bullet\ \bullet \\ & & & \bullet\ \bullet & & \bullet\ i\ \circ \\ & & \bullet & \bullet\ \bullet & & \bullet & \circ\ j \end{pmatrix}$$

Factorization of sparse matrices: problems

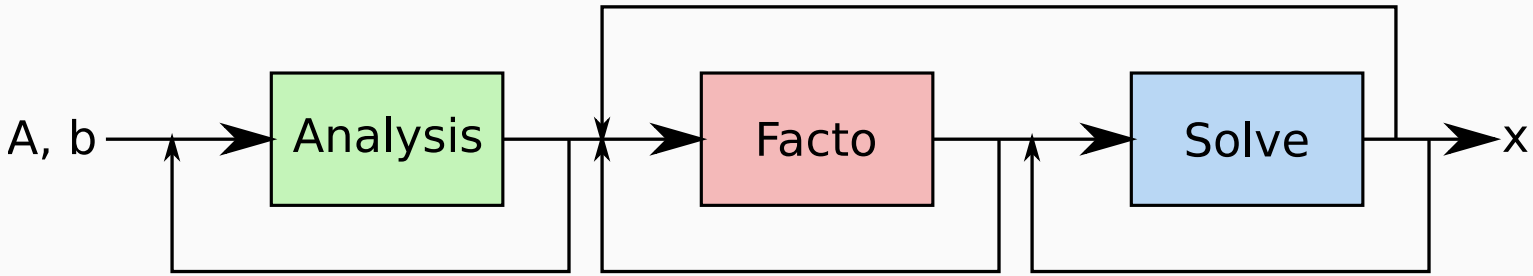
Which kind of problems does fill-in pose?

- more expensive is the factorization
- higher amount of memory required than $\mathcal{O}(nz)$
- more complicated algorithms to achieve the factorization

Factorization of sparse matrices: problems

Because of the fill-in, the matrix factorization is commonly preceded by an **analysis** phase where:

- the fill-in is reduced through **matrix permutations**
- the fill-in is predicted though the use of (e.g.) **elimination graphs**
- computations are structured using **elimination trees**



The analysis must complete much faster than the actual factorization.

The basic tools to achieve all this are GRAPHS.

Data structure for sparse matrices

- Storage scheme depends on the pattern of the matrix and on the type of access required
  - band or variable-band matrices
  - "block bordered" or block tridiagonal matrices
  - general matrix
  - row, column or diagonal access

Data formats for a general sparse matrix A

What needs to be represented

- Assembled matrices: $m \times n$ matrix $\mathbf{A}$ with nnz nonzeros.
- Elemental matrices (unassembled): $m \times n$ matrix $\mathbf{A}$ with NELT elements.
- Symmetric
  $\rightarrow$ store only part of the data.
- Distributed format ?
- Duplicate entries and/or out-of-range values ?

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with nnz=5 nonzeros

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

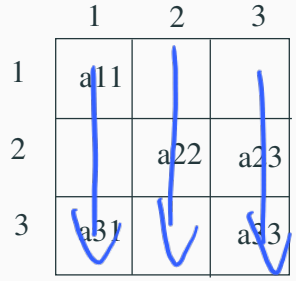
- Coordinate format

i j valeur

IRN	$[1 : nnz]$	$=$	1	3	2	2	3
JCN	$[1 : nnz]$	$=$	1	1	2	3	3
VAL	$[1 : nnz]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with nnz=5 nonzeros



- Coordinate format

IRN	$[1:nnz]$	$=$	1	3	2	2	3
JCN	$[1:nnz]$	$=$	1	1	2	3	3
VAL	$[1:nnz]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}

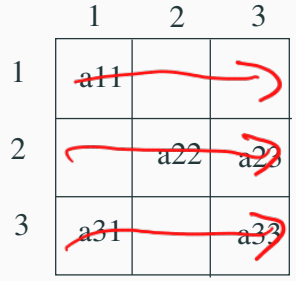
- Compressed Sparse Column (CSC) format

IRN	$[1:nnz]$	$=$	1	3	2	2	3
VAL	$[1:nnz]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}
COLPTR	$[1:N+1]$	$=$	1	3	4	6	

column J is stored in IRN/A locations COLPTR(J)...COLPTR(J+1)-1

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with nnz=5 nonzeros



- Coordinate format

IRN	$[1:nnz]$	$=$	1	3	2	2	3
JCN	$[1:nnz]$	$=$	1	1	2	3	3
VAL	$[1:nnz]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}

- Compressed Sparse Column (CSC) format

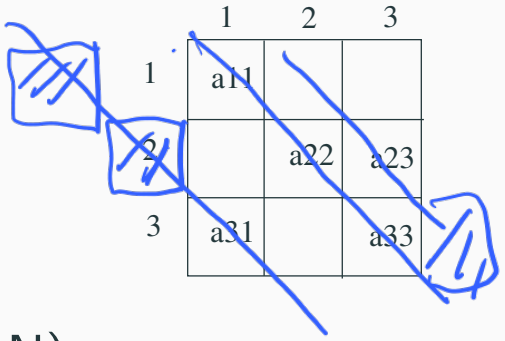
IRN	$[1:nnz]$	$=$	1	3	2	2	3
VAL	$[1:nnz]$	$=$	a_{11}	a_{31}	a_{22}	a_{23}	a_{33}
COLPTR	$[1:N+1]$	$=$	1	3	4	6	

  column J is stored in IRN/A locations COLPTR(J)...COLPTR(J+1)-1

- Compressed Sparse Row (CSR) format:
  Similar to CSC, but row by row

Classical Data Formats for Assembled Matrices

- Example of a 3x3 matrix with nnz=5 nonzeros



- Diagonal format (M=N):

  NDIAG = 3 nb diag

  IDIAG = −2 0 1

  $$\text{VAL} = \begin{bmatrix} na & a_{11} & 0 \\ na & a_{22} & a_{23} \\ a_{31} & a_{33} & na \end{bmatrix}$$ (*na*: not accessed)

  VAL(i,j) corresponds to A(i,i+IDIAG(j)) (for $1 \leq i + \text{IDIAG}(j) \leq N$)

Sparse Matrix-vector products $Y \leftarrow AX$

Algorithm depends on sparse matrix format:

- (Dense format)

```
DO I=1,M
  Yi=0
  DO J=1,N
    Yi = Yi + A(I, J)*X(J)
  END DO
  Y(I)=Yi
END DO
```

- Coordinate format:

- CSC format:

Sparse Matrix-vector products $Y \leftarrow AX$

Algorithm depends on sparse matrix format:

- (Dense format)

```
DO I=1,M
  Yi=0
  DO J=1,N
    Yi = Yi + A(I, J)*X(J)
  END DO
  Y(I)=Yi
END DO
```

- Coordinate format:

```
Y(1:M) = 0
DO k=1,NNZ
  Y(IRN(k)) = Y(IRN(k)) + VAL(k) * X(JCN(k))
ENDDO
```

- CSC format:

Sparse Matrix-vector products $Y \leftarrow AX$

Algorithm depends on sparse matrix format:

- (Dense format)

```
DO I=1,M
  Yi=0
  DO J=1,N
    Yi = Yi + A(I, J)*X(J)
  END DO
  Y(I)=Yi
END DO
```

- Coordinate format:

```
Y(1:M) = 0
DO k=1,NNZ
  Y(IRN(k)) = Y(IRN(k)) + VAL(k) * X(JCN(k))
ENDDO
```

- CSC format:

```
Y(1:M) = 0
DO J=1,N
  Xj=X(J)
  ! SAXPY
  DO k=COLPTR(J),COLPTR(J+1)-1
    Y(IRN(k)) = Y(IRN(k)) + VAL(k)*Xj
  ENDDO
ENDDO
```

File storage: Rutherford-Boeing

- Standard ASCII format for files
- Header + Data (CSC format). key xyz:
  - x=[rcp] (real, complex, pattern)
  - y=[suhzr] (sym., uns., herm., skew sym. ($A = -A^T$), rectang.)
  - z=[ae] (assembled, elemental)
  - ex: M_T1.RSA, SHIP003.RSE
- Supplementary files: right-hand-sides, solution, permutations...
- Canonical format introduced to guarantee a unique representation (order of entries in each column, no duplicates).

File storage: Rutherford-Boeing

```
DNV-Ex 1 : Tubular joint-1999-01-17                                    M_T1
       1733710           9758         492558        1231394              0
rsa                     97578          97578        4925574              0
(10I8)           (10I8)           (3e26.16)
       1      49      96     142     187     231     274     346     417     487
     556     624     691     763     834     904     973    1041    1108    1180
    1251    1321    1390    1458    1525    1573    1620    1666    1711    1755
    1798    1870    1941    2011    2080    2148    2215    2287    2358    2428
    2497    2565    2632    2704    2775    2845    2914    2982    3049    3115
...
       1       2       3       4       5       6       7       8       9      10
      11      12      49      50      51      52      53      54      55      56
      57      58      59      60      67      68      69      70      71      72
     223     224     225     226     227     228     229     230     231     232
     233     234     433     434     435     436     437     438       2       3
       4       5       6       7       8       9      10      11      12      49
      50      51      52      53      54      55      56      57      58      59
...
   -0.2624989288237320E+10    0.6622960540857440E+09    0.2362753266740760E+11
    0.3372081648690030E+08   -0.4851430162799610E+08    0.1573652896140010E+08
    0.1704332388419270E+10   -0.7300763190874110E+09   -0.7113520995891850E+10
    0.1813048723097540E+08    0.2955124446119170E+07   -0.2606931100955540E+07
    0.1606040913919180E+07   -0.2377860366909130E+08   -0.1105180386670390E+09
    0.1610636280324100E+08    0.4230082475435230E+07   -0.1951280618776270E+07
    0.4498200951891750E+08    0.2066239484615530E+09    0.3792237438608430E+08
    0.9819999042370710E+08    0.3881169368090200E+08   -0.4624480572242580E+08
```

File storage: Matrix-market

- Example

```
%%MatrixMarket matrix coordinate real general
% Comments
  5  5  8
    1     1   1.000e+00
    2     2   1.050e+01
    3     3   1.500e-02
    1     4   6.000e+00
    4     2   2.505e+02
    4     4  -2.800e+02
    4     5   3.332e+01
    5     5   1.200e+01
```

Examples of sparse matrix collections

- SuiteSparse Matrix Collection (formerly *The University of Florida Sparse Matrix Collection*) `https://sparse.tamu.edu/`
- Matrix Market `http://math.nist.gov/MatrixMarket/`
- Rutherford-Boeing `http://www.cerfacs.fr/algor/Softs/RB/index.html`

Sparse Direct Solvers - Ordering

Gaussian elimination and sparsity

Gaussian elimination

$\mathbf{A} = \mathbf{A}^{(1)}$, $\mathbf{b} = \mathbf{b}^{(1)}$, $\mathbf{A}^{(1)}\mathbf{x} = \mathbf{b}^{(1)}$:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} \qquad \begin{aligned} 2 &\leftarrow 2 - 1 \times a_{21}/a_{11} \\ 3 &\leftarrow 3 - 1 \times a_{31}/a_{11} \end{aligned}$$

$\mathbf{A}^{(2)}\mathbf{x} = \mathbf{b}^{(2)}$

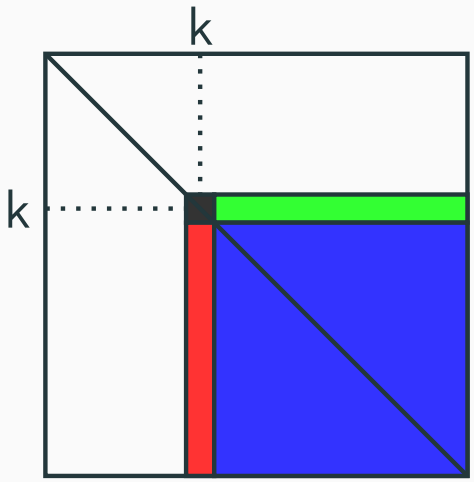
$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & a_{32}^{(2)} & a_{33}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix} \qquad \begin{aligned} b_2^{(2)} &= b_2 - a_{21} b_1 / a_{11} \cdots \\ a_{32}^{(2)} &= a_{32} - a_{31} a_{12} / a_{11} \cdots \end{aligned}$$

Finally $\mathbf{A}^{(3)}\mathbf{x} = \mathbf{b}^{(3)}$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(2)} & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(3)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(2)} \\ b_3^{(3)} \end{pmatrix} \qquad a_{(33)}^{(3)} = a_{(33)}^{(2)} - a_{32}^{(2)} a_{23}^{(2)} / a_{22}^{(2)} \cdots$$

Typical Gaussian elimination step k : $\boxed{a_{ij}^{(k+1)} = a_{ij}^{(k)} - \dfrac{a_{ik}^{(k)} a_{kj}^{(k)}}{a_{kk}^{(k)}}}$

Rappel: factorisation $L.U$



Factorisation $L.U$:

For k from 1 to $n-1$:

1. Compute column k of L:
   $A(k+1:n,k) = A(k+1:n,k)/A(k,k)$
2. Update the sub-block:
   $$\begin{aligned} & A(k+1:n,k+1:n) \\ =& A(k+1:n,k+1:n) \\ -& A(k+1:n,k) \times A(k,k+1:n) \end{aligned}$$

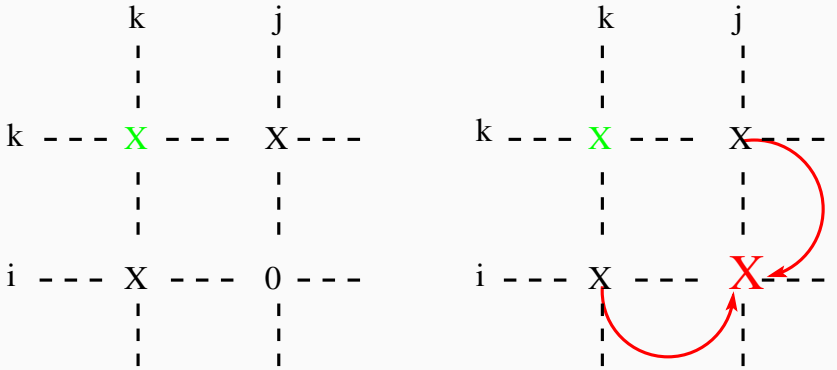
Gaussian elimination and Sparsity

Step k of **LU** factorization (a_{kk} pivot):

- For $i > k$ compute $l_{ik} = a_{ik}/a_{kk}$ $(= a'_{ik})$,
- For $i > k, j > k$ update remaining rows/cols in matrix

$$a'_{ij} = a_{ij} - l_{ik} \times a_{kj} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}}$$

- If $a_{ik} \neq 0$ and $a_{kj} \neq 0$ then $a'_{ij} \neq 0$
- If a_{ij} was zero $\rightarrow$ its non-zero value must be stored



fill-in

- Idem for Cholesky Factorization

Gaussian elimination and sparsity

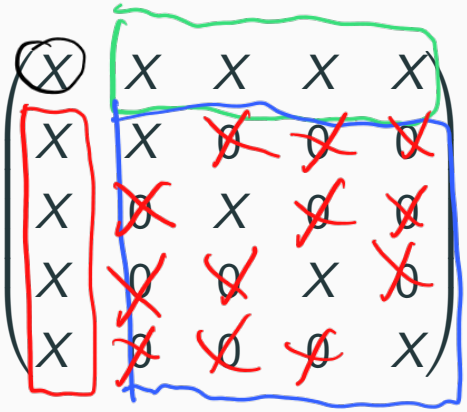
- Interest of permuting a matrix:

$$\begin{pmatrix} X & X & X & X & X \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & X \end{pmatrix} \quad 1 \leftrightarrow 5 \quad \begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix}$$

- Ordering the variables has a strong impact on
  - fill-in
  - number of operations
  - shape of the dependency graph (tree) and parallelism
- Fill reduction is NP-hard in general [Yannakakis 81]

Gaussian elimination and sparsity

- Interest of permuting a matrix:



$1 \leftrightarrow 5$

$$\begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix}$$

- Ordering the variables has a strong impact on
  - fill-in
  - number of operations
  - shape of the dependency graph (tree) and parallelism
- Fill reduction is NP-hard in general [Yannakakis 81]

Gaussian elimination and sparsity

- Interest of permuting a matrix:

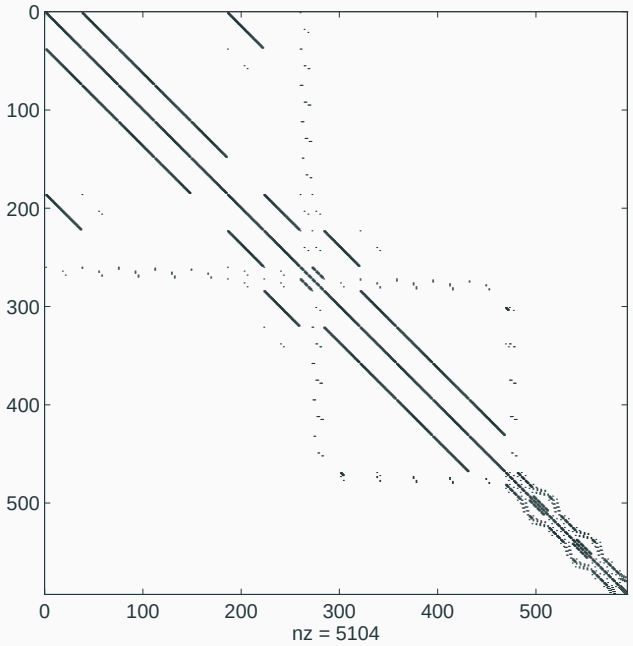
$$\begin{pmatrix} X & X & X & X & X \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & X \end{pmatrix} \quad 1 \leftrightarrow 5 \quad \begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix}$$

- Ordering the variables has a strong impact on
  - fill-in
  - number of operations
  - shape of the dependency graph (tree) and parallelism
- Fill reduction is NP-hard in general [Yannakakis 81]

Illustration: Reverse Cuthill-McKee on matrix dwt_592.rua

Harwell-Boeing matrix: dwt_592.rua, structural computing on a submarine. NNZ(LU factors)=58202

Original matrix



Factorized matrix

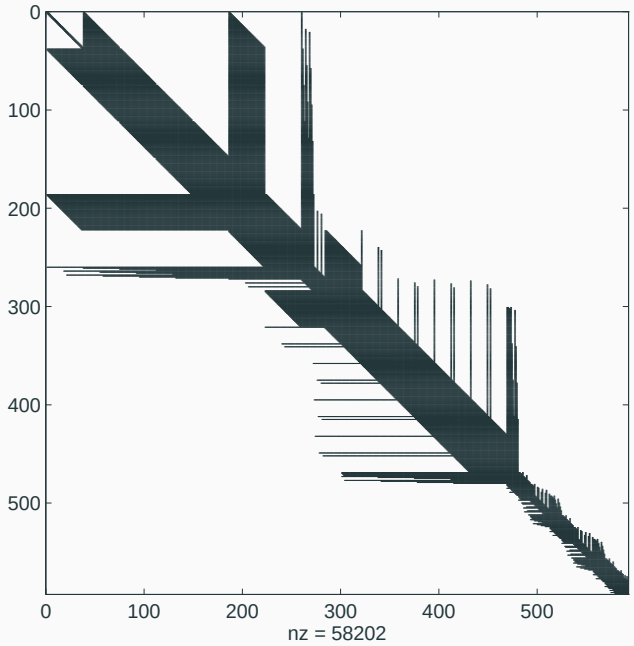
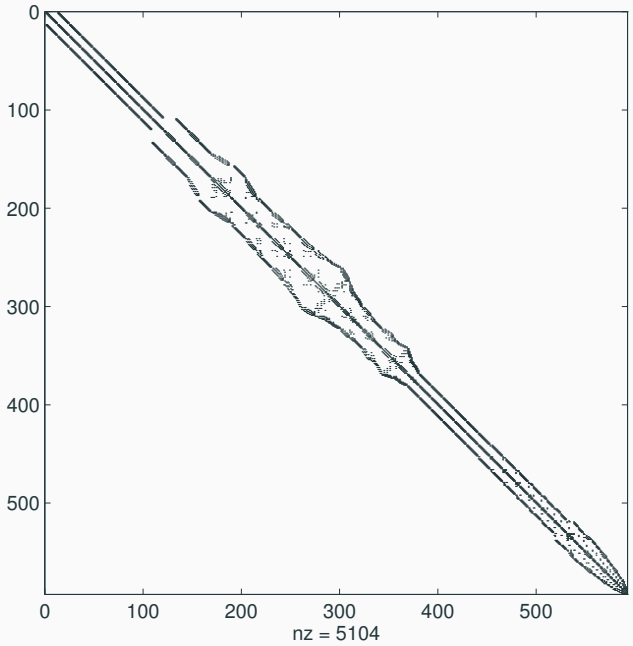


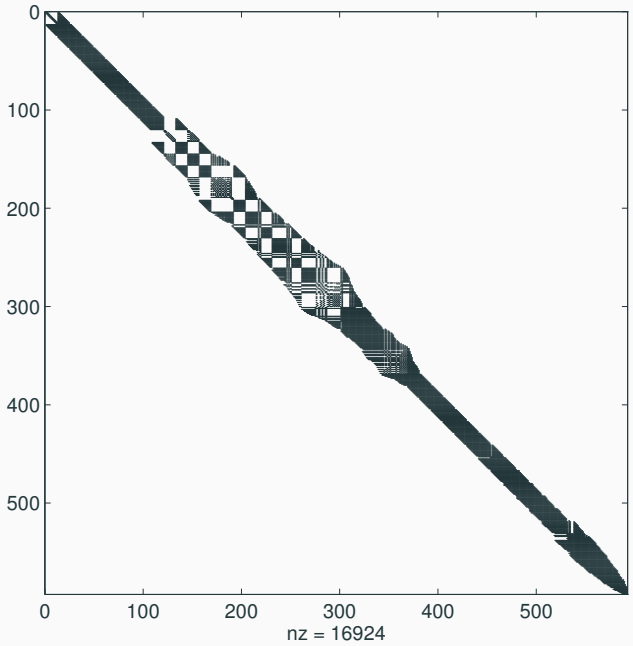
Illustration: Reverse Cuthill-McKee on matrix dwt_592.rua

NNZ(LU factors)=16924

Permuted matrix (RCM)



Factorized permuted matrix



Sparse LU factorization : a (too) simple algorithm

Only non-zeros are stored and operated on

1: Permute matrix A to reduce fill-in and flops (NP complete problem)
2: **for** $k = 1$ **to** n **do**
3: $L(k:k) = 1$; For nonzeros in column k: $L(k+1:n,k) = \frac{A(k+1:n,k)}{A(k,k)}$
4: $U(k,k:n) = A(k,k:n)$
5: **for** $j = k+1$ **to** n limited to nonzeros in row $U(k,:)$ **do**
6: **for** $i = k+1$ **to** n limited to nonzeros in col. $L(:,k)$ **do**
7: $A(i,j) = A(i,j) - L(i,k) \times U(k,j)$
8: **end for**
9: **end for**
10: **end for**

Algorithm 1: Simple sparse **LU** factorization

Questions

Dynamic data structure for A to accommodate fill-in

Data access efficiency; Can we predict position of fill-in ?

$|\mathbf{A}(k,k)|$ too small $\longrightarrow$ numerical permutation needed !!!

Sparse Direct Solvers - Ordering

Permutation matrices

Permutation matrices

A permutation matrix is a square $(0, 1)$-matrix where each row and column has a single 1.

If P is a permutation matrix, $PP^T = I$, i.e., it is an orthogonal matrix.

Permutation matrices

Let,

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} \times & \times & \\ \times & & \times \\ & & \times \end{pmatrix} \end{array}$$

and suppose we want to permute columns as $[2, 1, 3]$.

Define $p_{2,1} = 1$, $p_{1,2} = 1$, $p_{3,3} = 1$ (if column j to be at position i, set $p_{ji} = 1$), and $B = AP$

$$B = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 2 & 1 & 3 \end{array} \\ \begin{pmatrix} \times & \times & \\ & \times & \times \\ & & \times \end{pmatrix} \end{array} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} \times & \times & \\ \times & & \times \\ & & \times \end{pmatrix} \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} & 1 & \\ 1 & & \\ & & 1 \end{pmatrix} \end{array}$$

Permutation matrices - Symmetric case

Let,

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} \times & \times & \times \\ \times & \times & \\ \times & & \times \end{pmatrix} \end{array}$$

$B = A(P, P)$

$P =$

and suppose we want to permute lines and columns as $[3, 1, 2]$ to preserve symmetry.

Define $p_{1,2} = 1$, $p_{2,3} = 1$, $p_{3,1} = 1$, and $B = P^T A P$

$$B = \begin{array}{c} \\ 3 \\ 2 \\ 1 \end{array} \begin{array}{c} \begin{array}{ccc} 3 & 2 & 1 \end{array} \\ \begin{pmatrix} \times & \times & \\ \times & \times & \times \\ & \times & \times \end{pmatrix} \end{array} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} & & 1 \\ 1 & & \\ & 1 & \end{pmatrix} \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} \times & \times & \times \\ \times & \times & \\ \times & & \times \end{pmatrix} \end{array} \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} & 1 & \\ & & 1 \\ 1 & & \end{pmatrix} \end{array}$$

Sparse Direct Solvers - Ordering

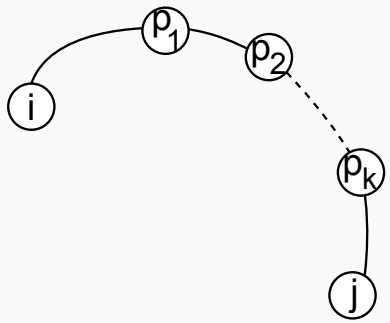
Fill-in characterization (symmetric matrices)

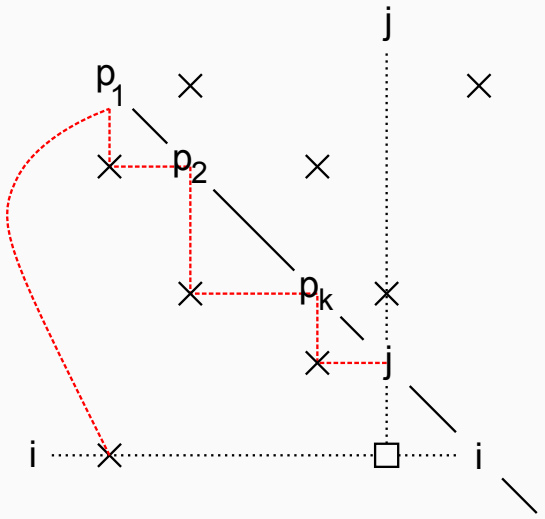
Fill-in characterization

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^T$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.



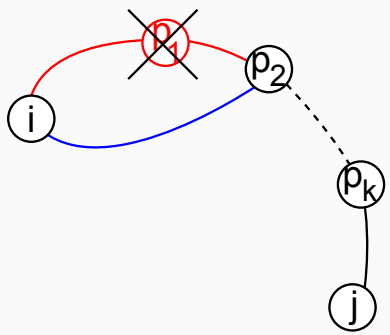


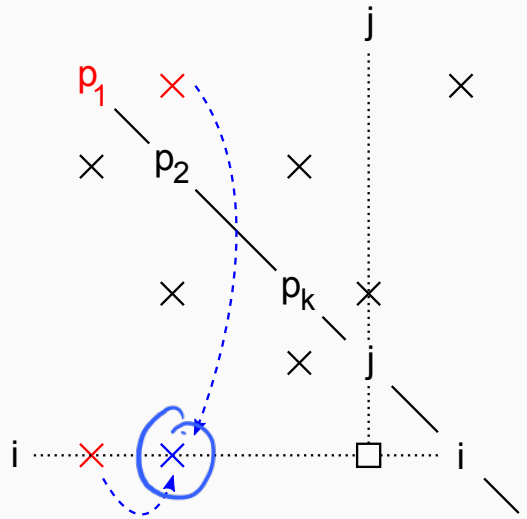
Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^T$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.



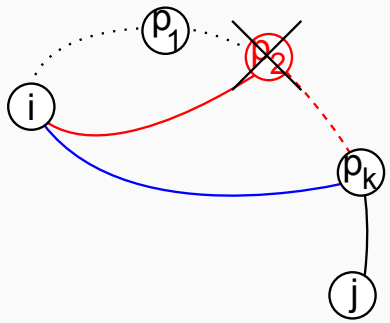


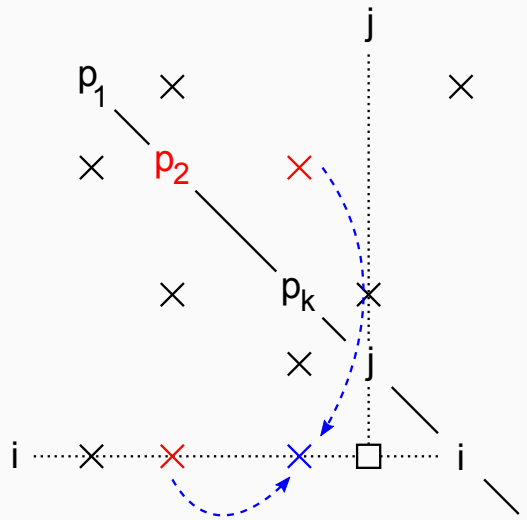
Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^T$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.



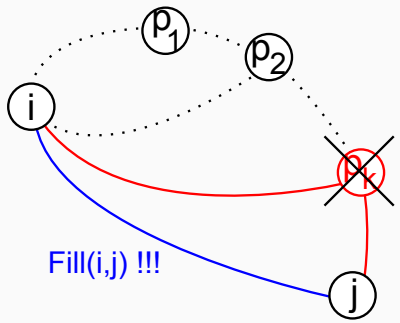


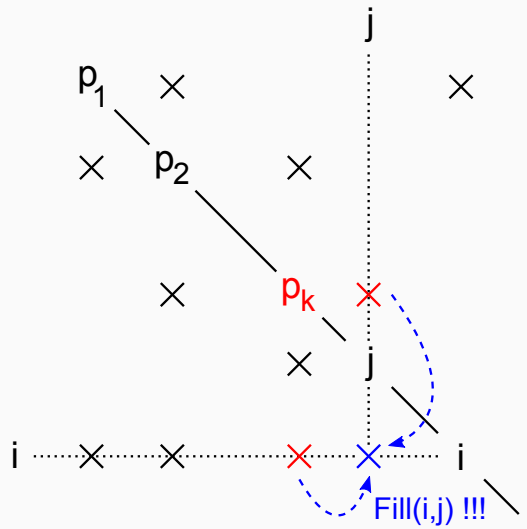
Fill-in characterization (proof intuition)

Let A be a symmetric matrix ($G(A)$ its associated graph), L the matrix of factors $A = LL^T$;

Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ iff there is a path in $G(A)$ between i and j such that all nodes in the path have indices smaller than both i and j.





Sparse Direct Solvers - Ordering

Graph definitions and relations to sparse matrices

Graph notations and definitions

A graph $G = (V, E)$ consists of a finite set V, called the vertex set and a finite, binary relation E on V, called the edge set.

Three standard graph models

Undirected graph: The edges are unordered pair of vertices, i.e., $\{u, v\} \in E$ for some $u, v \in V$.

Directed graph: The edges are ordered pair of vertices, that is, (u, v) and (v, u) are two different edges.

Bipartite graph: $G = (U \cup V, E)$ consists of two disjoint vertex sets U and V such that for each edge $(u, v) \in E$, $u \in U$ and $v \in V$.

An ordering or labelling of $G = (V, E)$ having n vertices, i.e., $|V| = n$, is a mapping of V onto $1, 2, \ldots, n$.

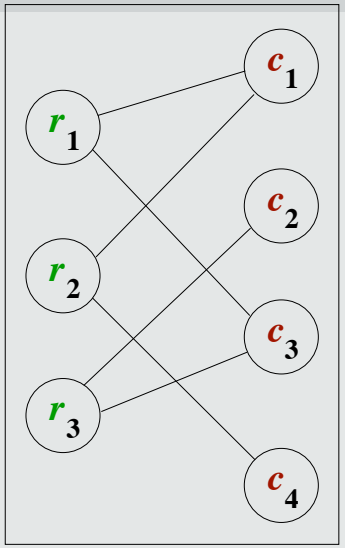
Matrices and graphs: rectangular matrices

The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Rectangular matrices

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} \times & & \times & \\ \times & & & \times \\ & \times & \times & \end{pmatrix} \end{array}$$

Bipartite graph



The set of rows corresponds to one of the vertex set R, the set of columns corresponds to the other vertex set C such that for each $a_{ij} \neq 0$, (r_i, c_j) is an edge.

Matrices and graphs: square unsymmetric pattern

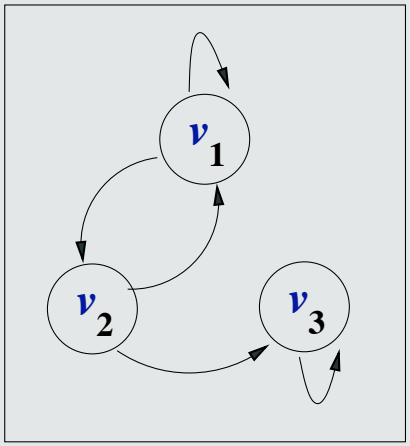
The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Square unsymmetric pattern matrices

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} \times & \times & \\ \times & & \times \\ & & \times \end{pmatrix} \end{array}$$

Graph models

- Bipartite graph as before.
- Directed graph



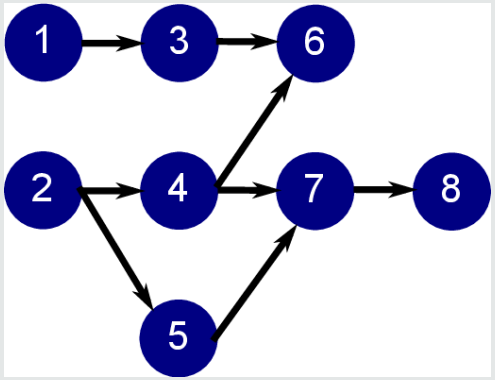
The set of rows/cols corresponds the vertex set V such that for each $a_{ij} \neq 0$, (v_i, v_j) is an edge. Transposed view possible too, i.e., the edge (v_i, v_j) directed from column i to row j. Usually self-loops are omitted.

Matrices and graphs: square unsymmetric pattern

A special subclass

Directed acyclic graphs (DAG):

A directed graphs with no loops (maybe except for self-loops).



DAGs

We can sort the vertices such that if (u, v) is an edge, then u appears before v in the ordering.

Question: What kind of matrices have a DAG structure ?

Matrices and graphs: symmetric pattern

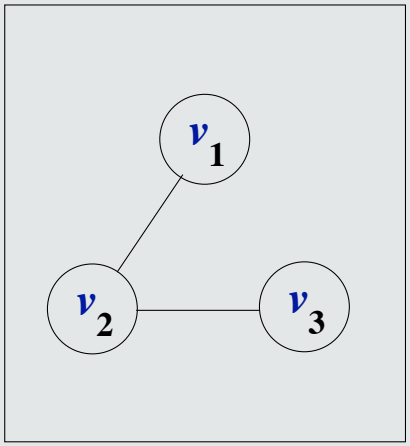
The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Square symmetric pattern matrices

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} & \times & \\ \times & \times & \times \\ & \times & \times \end{pmatrix} \end{array}$$

Graph models

- Bipartite and directed graphs as before.
- Undirected graph



The set of rows/cols corresponds the vertex set V such that for each $a_{ij}, a_{ji} \neq 0$, $\{v_i, v_j\}$ is an edge. No self-loops; usually the main diagonal is assumed to be zero-free.

Definitions: edges, degrees, and paths

Many definitions for directed and undirected graphs are the same. We will use (u, v) to refer to an edge of an undirected or directed graph to avoid repeated definitions.

- An edge (u, v) is said to incident on the vertices u and v.
- For any vertex u, the set of vertices in $\mathrm{adj}(u) = \{v : (u, v) \in E\}$ are called the neighbors of u. The vertices in $\mathrm{adj}(u)$ are said to be adjacent to u.
- The degree of a vertex is the number of edges incident on it.
- A path p of length k is a sequence of vertices $\langle v_0, v_1, \ldots, v_k \rangle$ where $(v_{i-1}, v_i) \in E$ for $i = 1, \ldots, k$. The two end points v_0 and v_k are said to be connected by the path p, and the vertex v_k is said to be reachable from v_0.

Definitions: Components

- An undirected graph is said to be connected if every pair of vertices is connected by a path.
- The connected components of an undirected graph are the equivalence classes of vertices under the "is reachable" from relation.
- A directed graph is said to be strongly connected if every pair of vertices are reachable from each other.
- The strongly connected components of a directed graph are the equivalence classes of vertices under the "are mutually reachable" relation.

Sparse Direct Solvers - Ordering

Trees and spanning trees

Definitions: trees and spanning trees

A tree is a connected, acyclic, undirected graph.

Properties of trees

- Any two vertices are connected by a unique path.
- $|E| = |V| - 1$

A rooted tree is a tree with a distinguished vertex r, called the root.

There is a unique path from the root r to every other vertex v. Any vertex y in that path is called an ancestor of v. If y is an ancestor of v, then v is a descendant of y.

The subtree rooted at v is the tree induced by the descendants of v, rooted at v.

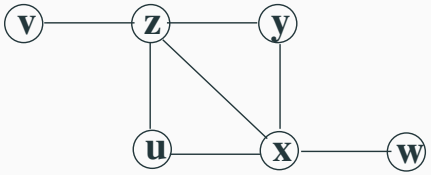
A spanning tree of a connected graph $G = (V, E)$ is a tree $T = (V, F)$, such that $F \subseteq E$.

Sparse Direct Solvers - Ordering

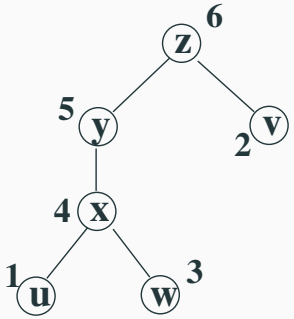
Ordering and tree traversal

Ordering of the vertices of a rooted tree

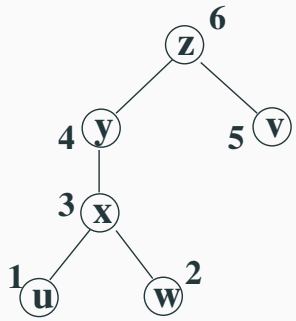
- A topological ordering of a rooted tree is an ordering that numbers children vertices before their parent.
- A postorder is a topological ordering which numbers the vertices in any subtree consecutively.



Connected graph G



Rooted spanning tree
with topological ordering



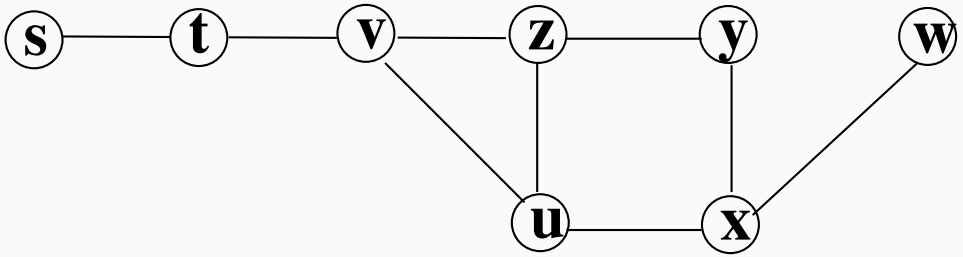
Rooted spanning tre
with postordering

How to explore a graph:

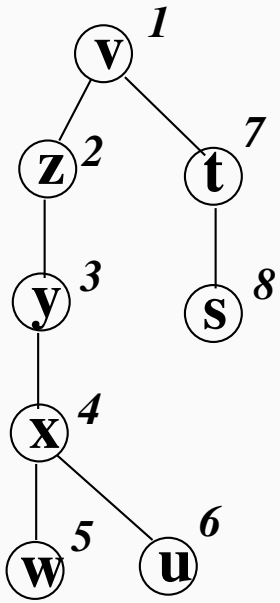
Two algorithms such that each edge is traversed exactly once in the forward and reverse direction and each vertex is visited. (Connected graphs are considered in the description of the algorithms.)

- **Depth first search** : Starting from a given vertex (*mark* it) follow a path (and mark each vertex in the path) until a vertex that has no neighbor or that is adjacent to only marked vertices. Return to previous vertex and continue.
- **Breadth first search** Select a vertex and put it into an initially empty queue of vertices *to be visited*. Repeatedly remove the vertex x at the head of the queue and place all neighbors of x that were not enqueue before into the queue.

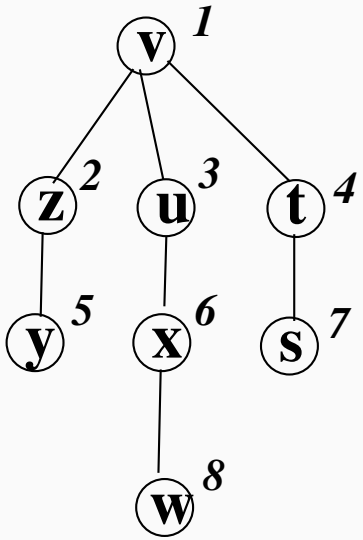
Illustration of DFS and BFS exploration



Depth–first spanning tree (v)



Breadth–first spanning tree (v)



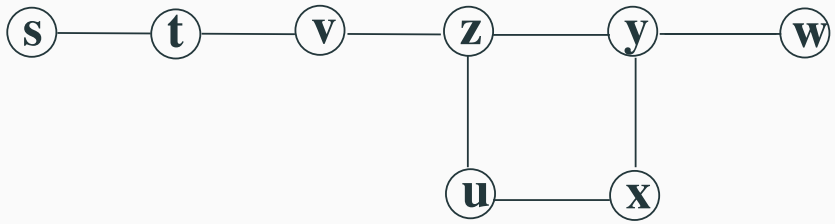
Each vertex is visited once.

Sparse Direct Solvers - Ordering

Peripheral and pseudo-peripheral vertices

Graph shapes and peripheral vertices

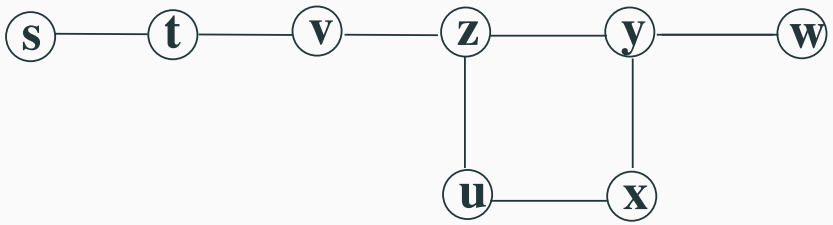
- The distance between two vertices of a graph is the size of the shortest path joining those vertices.
- The eccentricity $l(v) = max\{d(v, w) \mid w \in V\}$
- The diameter $\delta(G) = max\{l(v) \mid v \in V\}$
- v is a peripheral vertex if $l(v) = \delta(G)$
- Example of connected graph with $\delta(G) = 5$ and peripheral vertices s, w, x.



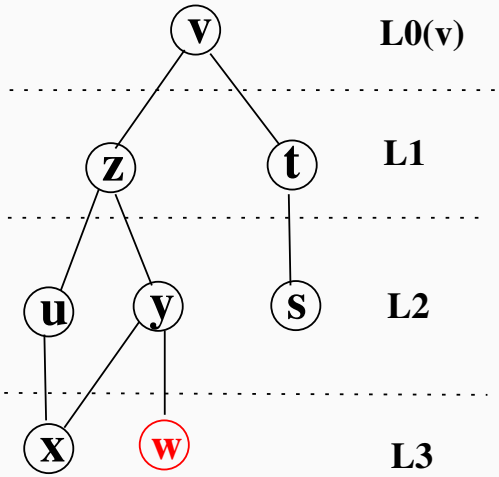
Pseudo-peripheral vertices

- The rooted level structure $\mathcal{L}(v)$ of G is a partitioning of G
  $\mathcal{L}(v) = L_0(v), \ldots, L_{l(v)}(v)$ such that $L_o(v) = v$
  $L_i(v) = Adj(L_{i-1}(v)) \setminus L_{i-2}(v)$ (with $L_{-1} = \emptyset$)
- A pseudo-peripheral is a node with a large eccentricity.
- Algorithm to determine a pseudo-peripheral :
  Let $i = 0$, r_i be an arbitrary node, and $nblevels = l(r_i)$. The algorithm iteratively updates *nblevels* by selecting the vertex r_{i+1} of minimum degree of $L_{l(r_i)}$ and stopping when $l(r_{i+1}) \leq nblevels$.

Pseudo-peripheral vertices

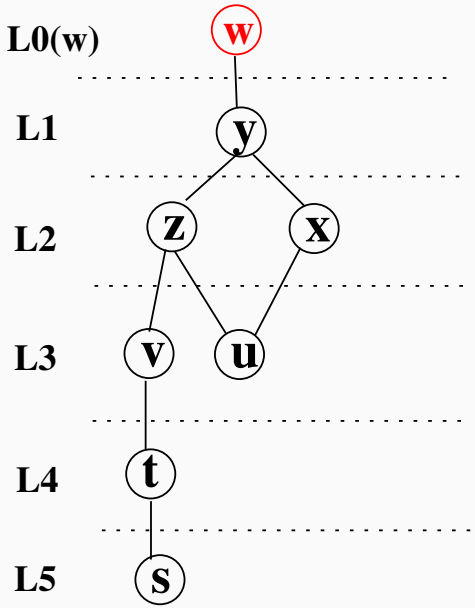


First step, select v



Level structure(v)

2nd step: select W in L3
(min. degree)



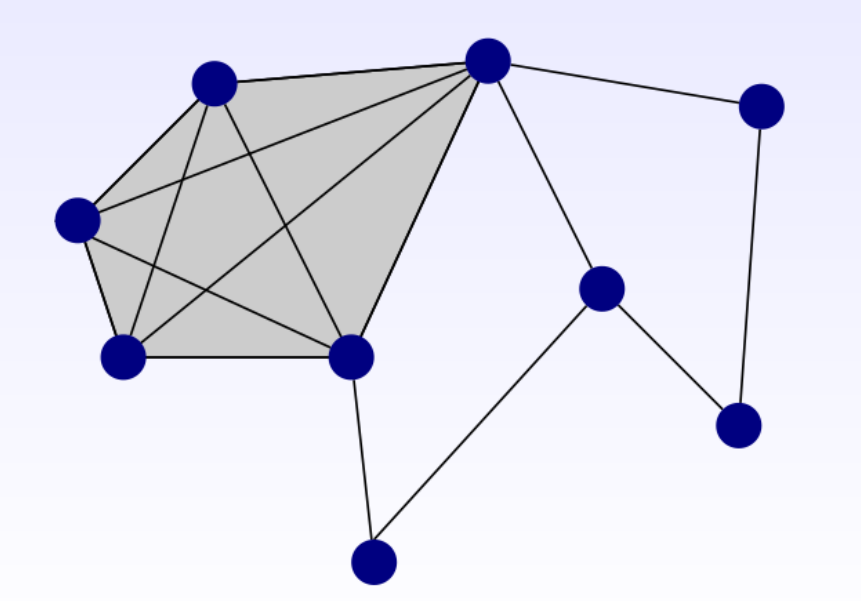
Sparse Direct Solvers - Ordering

Cliques

Cliques

In an undirected graph $G = (V, E)$, a set of vertices $S \subseteq V$ is a clique if for all $s, t \in S$, we have $(s, t) \in E$.

In a symmetric matrix A, a clique corresponds to a subset of rows R and the corresponding columns such that the matrix $A(R, R)$ is full.



Sparse Direct Solvers - Ordering

The elimination process in the graphs

Symmetric matrices and graphs

Predicting structure helps

1. in reducing the memory requirements,
2. in achieving high performance,
3. in simplifying the algorithms.

We will consider the Cholesky factorization $A = LL^T$. In this case, structural and numerical aspects are neatly separated.

For general case (e.g., LU factorization) pivoting is necessary and depends on the actual numerical values. There are combinatorial tools for these, but we will not cover.

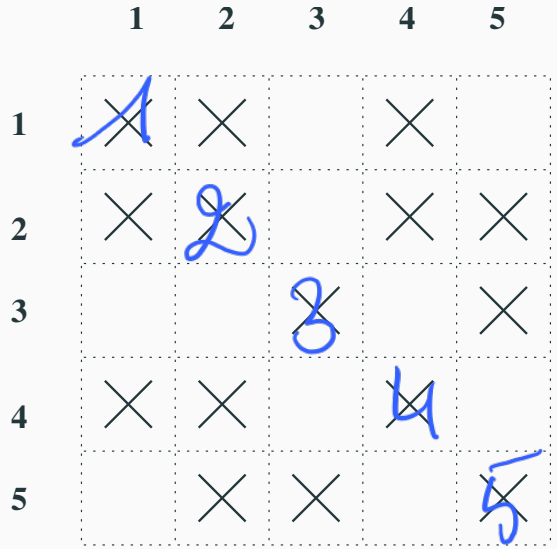
Structure prediction algorithms should run, preferably, faster than the numerical computations that will follow.

Symmetric matrices and graphs

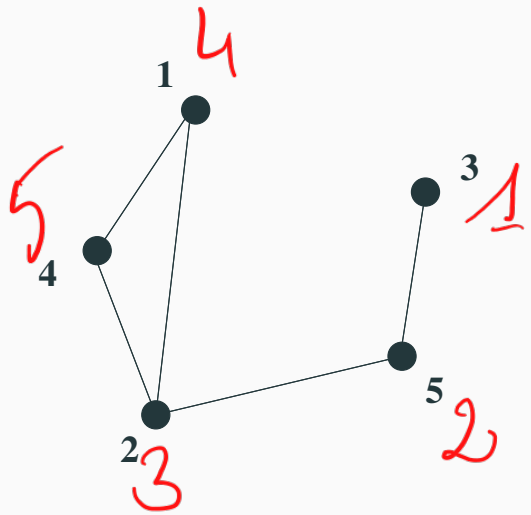
- Assumptions: $\mathbf{A}$ symmetric and pivots are chosen from the diagonal
- Structure of $\mathbf{A}$ symmetric represented by the graph $G = (V, E)$
  - Vertices are associated to columns: $V = \{1, \ldots, n\}$
  - Edges E are defined by: $(i, j) \in E \leftrightarrow a_{ij} \neq 0$
  - G undirected (symmetry of $\mathbf{A}$)

Symmetric matrices and graphs

- Remarks:
  - Number of nonzeros in column $j = |\mathrm{adj}_G(j)|$
  - Symmetric permutation $\equiv$ renumbering the graph



Symmetric matrix



Corresponding graph

The elimination model for symmetric matrices

A symmetric, positive definite matrix can be factorized by means of the Cholesky algorithm

for $k = 1, ..., n$ **do**
  $l_{kk} = \sqrt{a_{kk}^{(k-1)}}$
  for $i = k+1, ..., n$ **do**
    $l_{ik} = a_{ik}^{(k-1)} / l_{kk}$
    for $j = k+1, ..., i$ **do**
      $a_{ij}^{(k)} = a_{ij}^{(k)} - l_{ik} l_{jk}$
    end for
  end for
end for

$$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

The elimination model for symmetric matrices

- Let A be a symmetric positive definite matrix of order n
- The LL^T factorization can be described by the equation:

$$A = A_0 = \begin{pmatrix} d_1 & v_1^T \\ v_1 & \overline{A_1} \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{d_1} & 0 \\ \frac{v_1}{\sqrt{d_1}} & I_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & A_1 \end{pmatrix} \begin{pmatrix} \sqrt{d_1} & \frac{v_1^T}{\sqrt{d_1}} \\ 0 & I_{n-1} \end{pmatrix}$$

$= L_1 A_1 L_1^T$, where

$$A_1 = \overline{A_1} - \frac{v_1 v_1^T}{d_1}$$

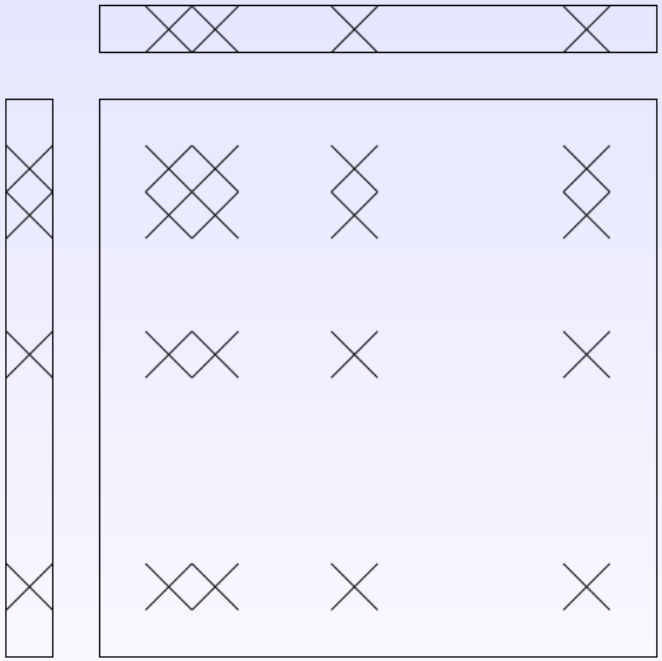
- The basic step is applied on $A_1 A_2 \cdots$ to obtain :

$$A = (L_1 L_2 \cdots L_{n-1})\, I_n \left(L_{n-1}^T \ldots L_2^T L_1^T\right) = LL^T$$

The elimination model for symmetric matrices

The basic step: $A_1 = \overline{A_1} - \frac{v_1 v_1^T}{d_1}$

What is $v_1 v_1^T$ in terms of structure?



v_1 is a column of A, hence the neighbors of the corresponding vertex.

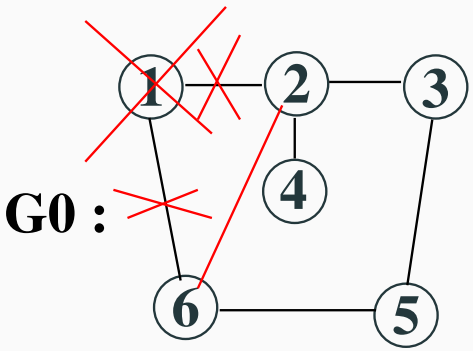
$v_1 v_1^T$ results in a dense sub-block in A_1, i.e., the elimination of a node results in the creation of a clique that connects all the neighbors of the eliminated node.

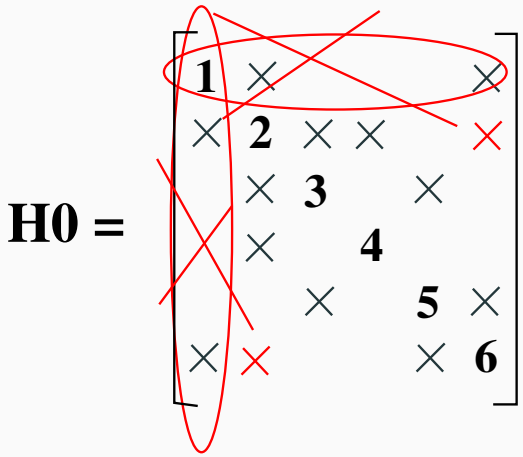
If any of the nonzeros in dense submatrix are not in A, then we have fill-ins.

The elimination process in the graphs

$G_U(V, E) \leftarrow$ undirected graph of **A**
for $k = 1 : n - 1$ **do**
  $V \leftarrow V - \{k\}$ {remove vertex k}
  $E \leftarrow E - \{(k, \ell) : \ell \in \mathrm{adj}(k)\} \cup \{(x, y) : x \in \mathrm{adj}(k) \text{ and } y \in \mathrm{adj}(k)\}$
  $G_k \leftarrow (V, E)$ {for definition}
end for

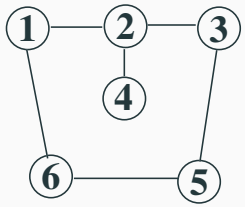
G_k are the so-called elimination graphs (Parter,'61).





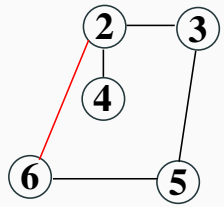
A sequence of elimination graphs

G0 :



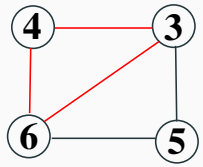
$$H0 = \begin{bmatrix} 1 & \times & & & \times \\ \times & 2 & \times & \times & \\ & \times & 3 & & \times \\ & \times & & 4 & \\ & & \times & & 5 & \times \\ \times & & & & \times & 6 \end{bmatrix}$$

G1 :



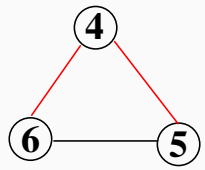
$$H1 = \begin{bmatrix} 2 & \times & \times & & + \\ \times & 3 & & \times & \\ \times & & 4 & & \\ & \times & & 5 & \times \\ + & & & \times & 6 \end{bmatrix}$$

G2 :



$$H2 = \begin{bmatrix} 3 & + & \times & + \\ + & 4 & & + \\ \times & & 5 & \times \\ + & + & \times & 6 \end{bmatrix}$$

G3 :



$$H3 = \begin{bmatrix} 4 & + & + \\ + & 5 & \times \\ + & \times & 6 \end{bmatrix}$$

$$\begin{pmatrix} l_{11} & & & & & \\ l_{21} & l_{22} & & & & \\ & l_{32} & l_{33} & & & \\ & l_{42} & l_{43} & l_{44} & & \\ & & l_{53} & l_{54} & l_{55} & \\ l_{61} & l_{62} & l_{63} & l_{64} & l_{65} & l_{66} \end{pmatrix}$$

Sparse Direct Solvers - Ordering

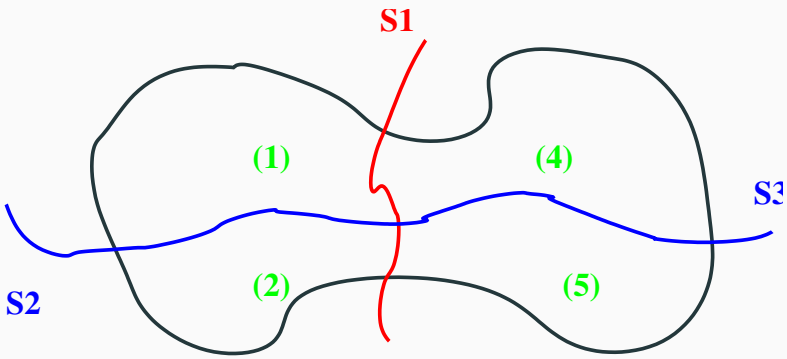
Fill-reducing heuristics

Fill-reducing heuristics

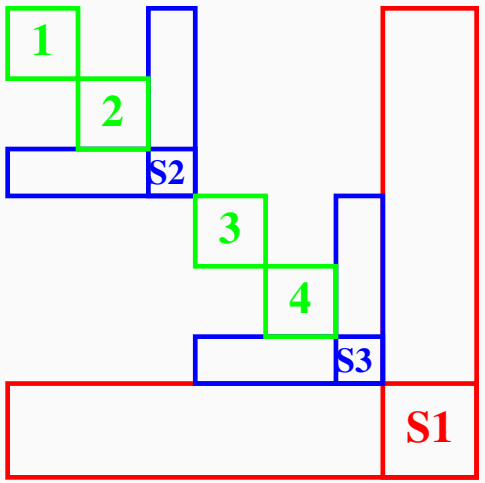
Three main classes of methods for minimizing fill-in during factorization

1. Global approach: The matrix is permuted into a matrix with a given pattern; fill-in is restricted to occur within that structure
   - Cuthill-McKee (block tridiagonal matrix)
     (Remark: BFS traversal of the associated graph)
   - Nested dissections ("block bordered" matrix)
     (Remark: interpretation using the fill-path theorem)

Graph partitioning



Permuted matrix



Fill-reducing heuristics

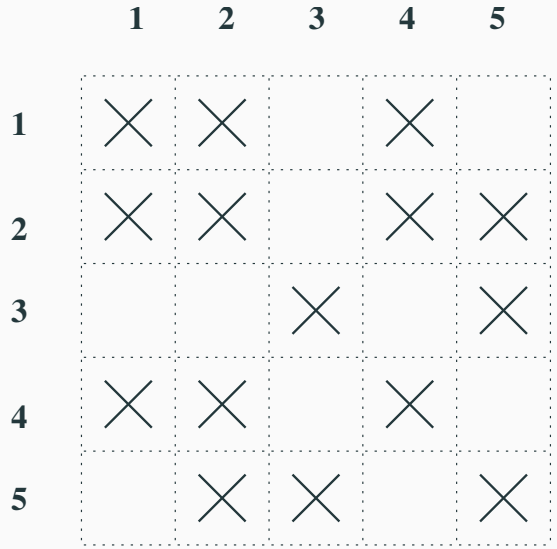
2. Local heuristics: at each step of the factorization, selection of the pivot that is likely to minimize fill-in.
   Method is characterized by the way pivots are selected.
   - Markowitz criterion (for a general matrix).
   - Minimum degree or Minimum fill-in (for symmetric matrices).
3. Hybrid approaches: once the matrix is permuted to block structure, local heuristics are used within the blocks.

Sparse Direct Solvers - Ordering

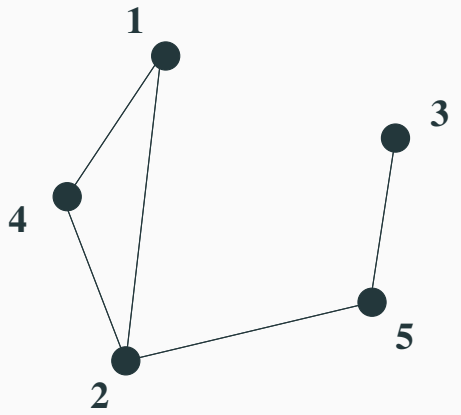
Cuthill-McKee - BFS traversal of the associated graph

Symmetric matrices and graphs (matlab exo1.m)

- Remarks:
  - Number of nonzeros in column $j = |\mathrm{adj}_G(j)|$
  - Symmetric permutation $\equiv$ renumbering the graph



Symmetric matrix



Corresponding graph

Breadth-First Search (BFS) of a graph to permute a matrix

Exercise (BFS traversal to order a matrix)

Given a symmetric matrix $\mathbf{A}$ *and its associated graph* $G = (V, E)$

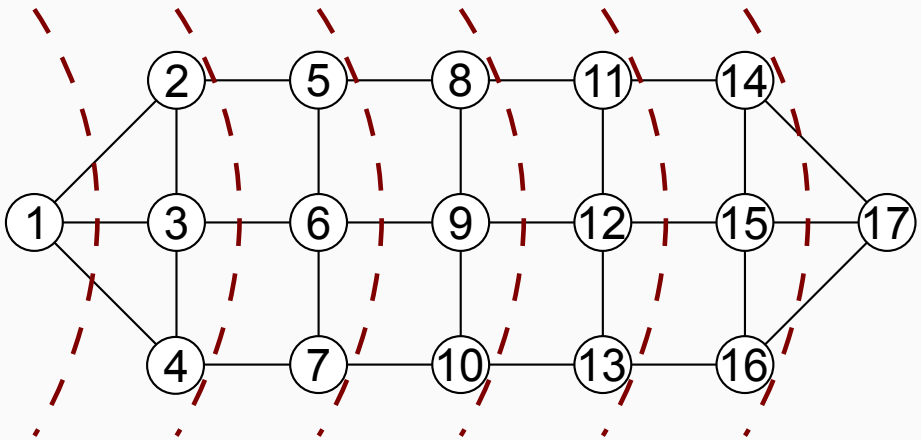
1. *Derive from BFS traversal of* G *an algorithm to reorder matrix* $\mathbf{A}$
2. *Explain the structural property of the permuted matrix?*
3. *Explain why such an ordering can help reducing fill-in during LU factorisation?*

CM: Algorithm

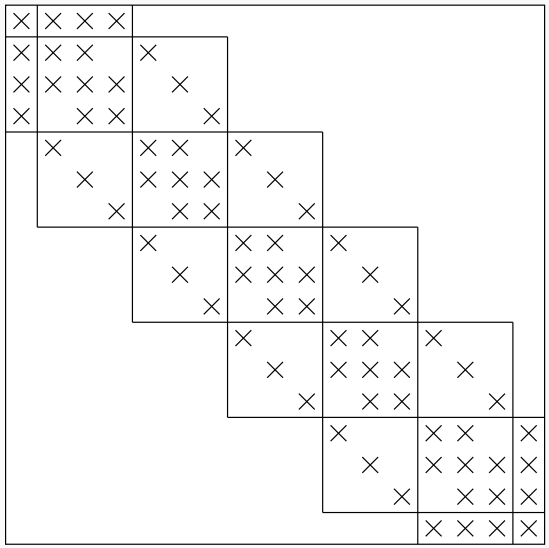
Level sets are built from the vertex of minimum degree. At any level, priority is given to a vertex with smaller number of neighbors.

Cuthill-McKee

pick a vertex v and order it as the first vertex
$S \leftarrow \{v\}$
while $S \neq V$ **do**
  $S' \leftarrow$ all vertices in $V \setminus S$ which are adjacent to S
  order vertices in S' in increasing order of degrees
  $S \leftarrow S \cup S'$
end while



(example from Duff, Erisman and Reid 86)

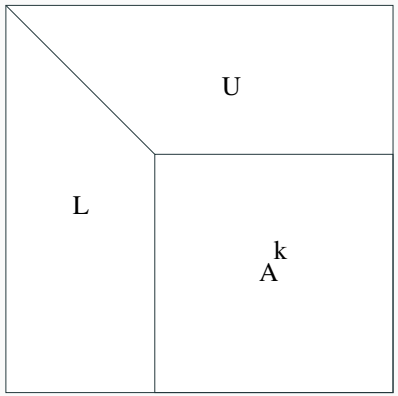


Sparse Direct Solvers - Ordering

Minimum degree

Reordering unsymmetric matrices: Markowitz criterion

- At step k of Gaussian elimination:

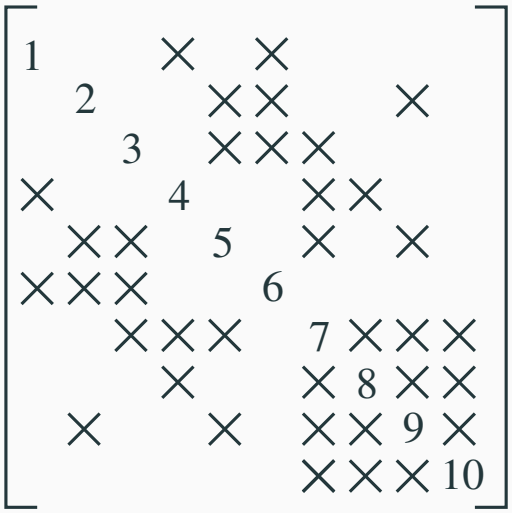


  - r_i^k = number of non-zeros in row i of $\mathbf{A}^k$
  - c_j^k = number of non-zeros in column j of $\mathbf{A}^k$
  - a_{ij} must be large enough and should minimize $(r_i^k - 1) \times (c_j^k - 1) \quad \forall i, j > k$

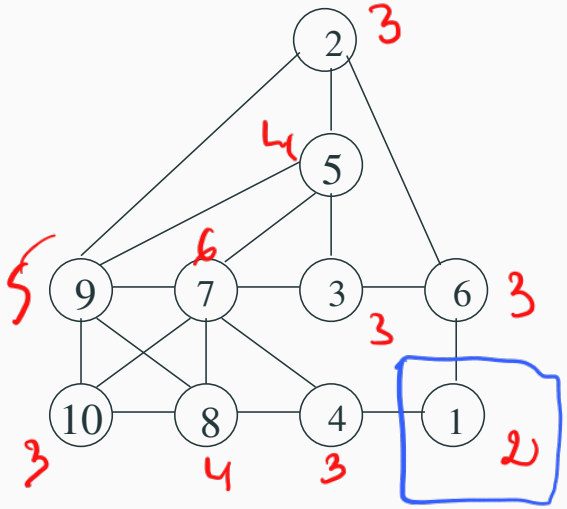
- Minimum degree : Markowitz criterion for symmetric diagonally dominant matrices

Minimum degree algorithm

- **Step 1**:
  Select the vertex that possesses the smallest number of neighbors in G^0.



(a) Sparse symmetric matrix

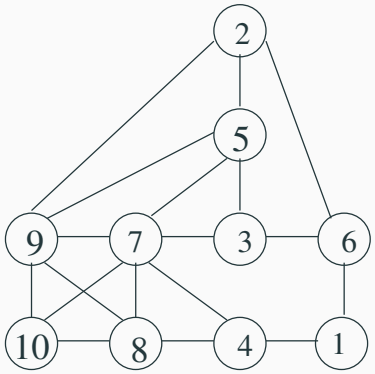


(b) Elimination graph

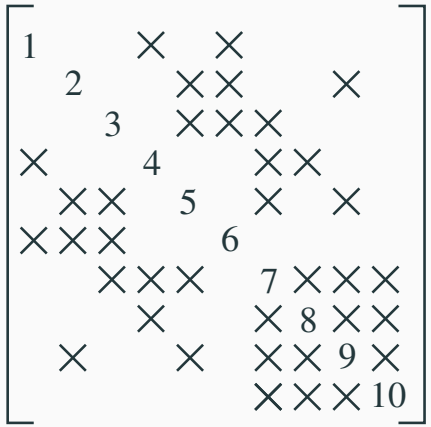
The node/variable selected is 1 of degree 2.

Illustration

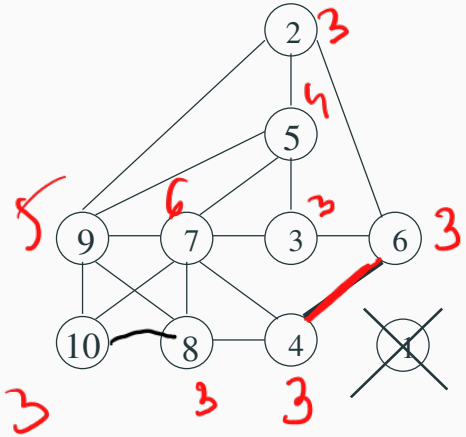
Step 1: elimination of pivot 1

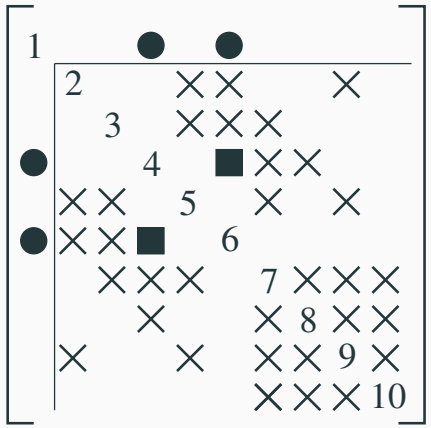


(a) Elimination graph



(b) Factors and active submatrix





X **Initial nonzeros** ● **Nonzeros in factors** ■ **Fill–in**

Minimum degree algorithm

Minimum degree algorithm applied to the graph:

- Step k : Select the node with the smallest number of neighbors
- G^k is built from G^{k-1} by *suppressing the pivot* and *adding edges* corresponding to fill-in.

Minimum degree algorithm based on elimination graphs

$\forall i \in [1 \dots n] \quad t_i = |\text{Adj}_{G^0}(i)|$

For $k = 1$ **to** n **Do**

$p = \min_{i \in V_{k-1}} (t_i)$

For each $i \in \text{Adj}_{G^{k-1}}(p)$ **Do**

$\text{Adj}_{G^k}(i) = (\text{Adj}_{G^{k-1}}(i) \cup \text{Adj}_{G^{k-1}}(p)) \setminus \{i, p\}$

$t_i = |\text{Adj}_{G^k}(i)|$

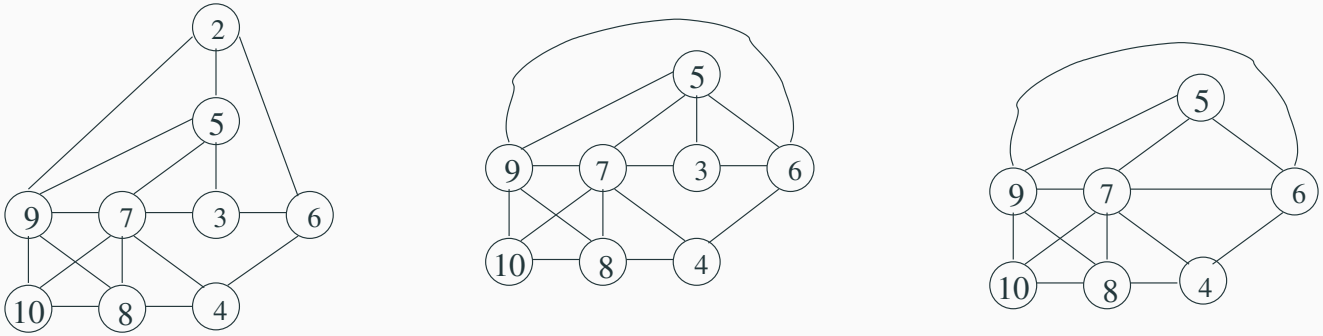
EndFor

$V^k = V^{k-1} \setminus p$

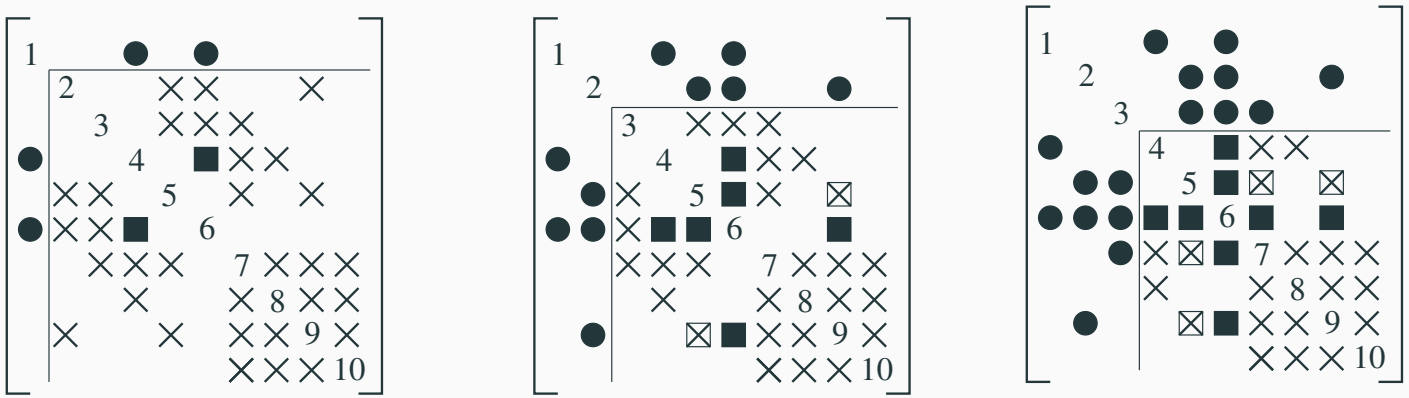
EndFor

Illustration (cont'd)

Graphs G_1, G_2, G_3 and corresponding reduced matrices.



(a) Elimination graphs



(b) Factors and active submatrices

× **Original nonzero**

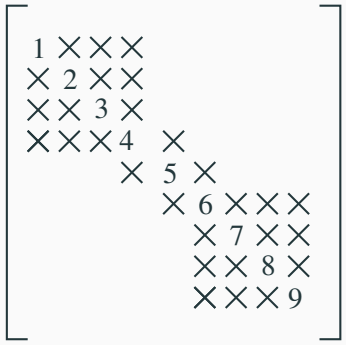
⊠ **Original nonzero modified**

■ **Fill–in**

● **Nonzeros in factors**

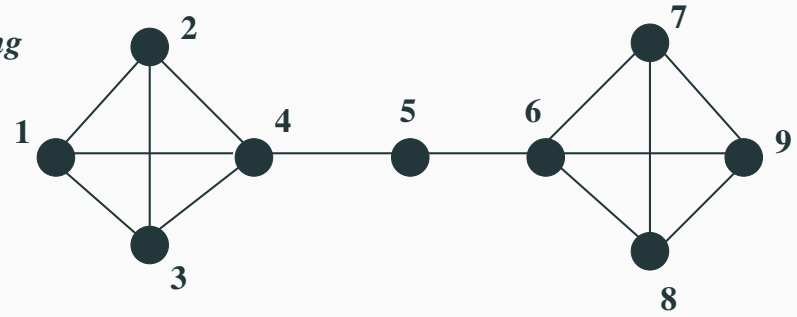
Minimum Degree does not always minimize fill-in !!!

Consider the following matrix



Remark: Using initial ordering

No fill–in

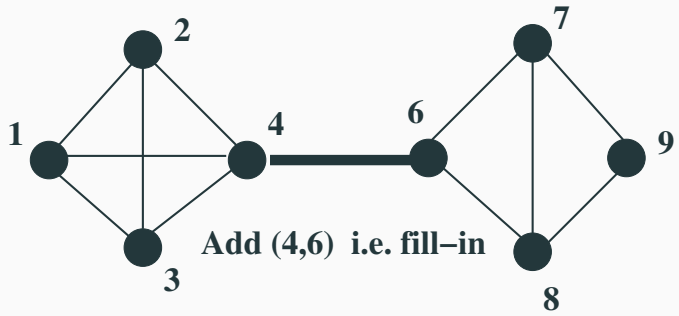


Corresponding elimination graph

Step 1 of Minimum Degree:

Select pivot 5 (minimum degree = 2)

Updated graph



Exercice

	1	2	3	4	5	6	7	8	9
1	X	X				X			
2	X	X		X	X		X		
3			X			X		X	X
4		X		X	X		X		
5		X		X	X		X		
6	X		X			X		X	X
7		X		X	X		X		
8			X			X		X	X
9			X			X		X	X

mat2

1. ordre naturel
2. minimum degree
3. CMK en partant de 1

Exercices

CMK1 et CMK2

Sparse Direct Solvers - Multifrontal Method

Sparse Direct Solvers - Multifrontal Method

The elimination tree

The elimination tree for symmetric matrices

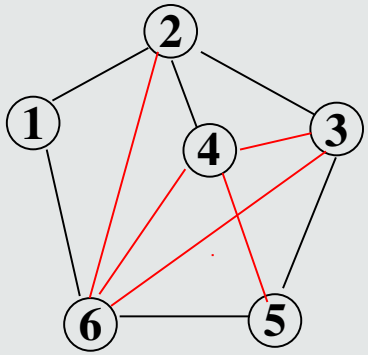
The elimination tree provides structural information relevant to describing both Gaussian elimination of symmetric matrices and the orthogonal factorization.

Its extension to unsymmetric matrices is possible.

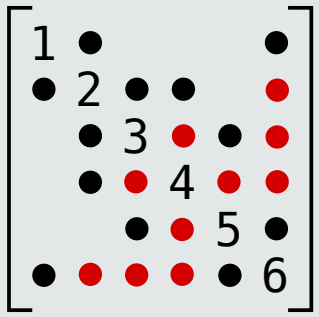
Set-up

Reminder

- A spanning tree of a connected graph $G = (V, E)$ is a tree $T = (V, F)$, such that $F \subseteq E$.
- A topological ordering of a rooted tree is an ordering that numbers children vertices before their parent.
- A postorder is a topological ordering which numbers the vertices in any subtree consecutively.
- Let $\mathbf{F} = \mathbf{L} + \mathbf{L}^{\mathsf{T}}$ be the filled matrix, and $G(\mathbf{F})$ the *filled graph* of $\mathbf{A}$ denoted by $G^{+}(\mathbf{A})$.



$\mathbf{G^{+}(A) = G(F)}$



$\mathbf{F = L + L^{T}}$

Set-up

Let A be an $n \times n$ symmetric positive-definite and irreducible matrix, $A = LL^T$ its Cholesky factorization, and $G^+(A)$ its filled graph (graph of $F = L + L^T$).

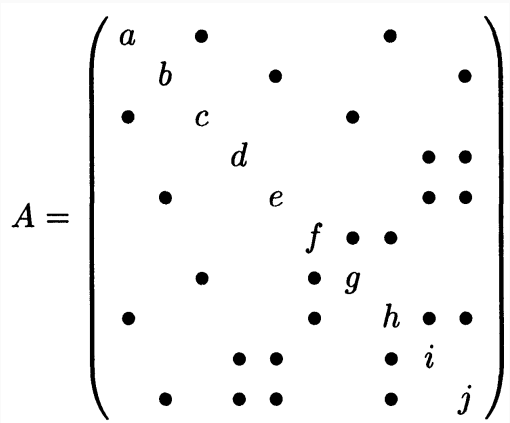
A first definition

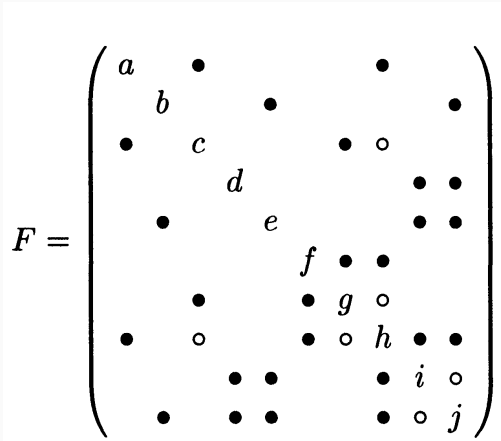
Since A is irreducible, each of the first $n-1$ columns of L has at least one off-diagonal nonzero.

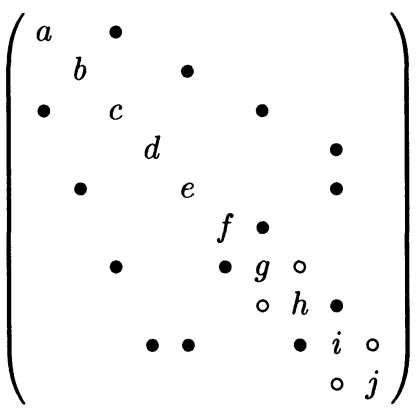
For each column $j < n$ of L, remove all the nonzeros in the column j except the first one below the diagonal.

Let L_t denote the remaining structure and consider the matrix $F_t = L_t + L_t^T$.

The graph $G(F_t)$ is a tree called the elimination tree of A, also denoted $T(A)$.

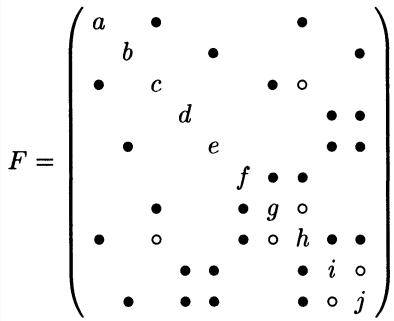
$A =$ 

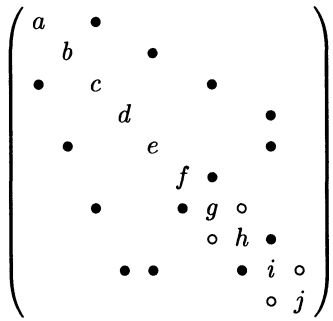
$F =$ 

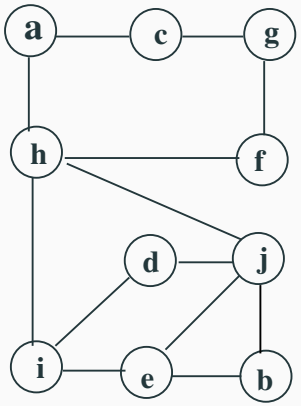
$F_t =$ 

A first definition

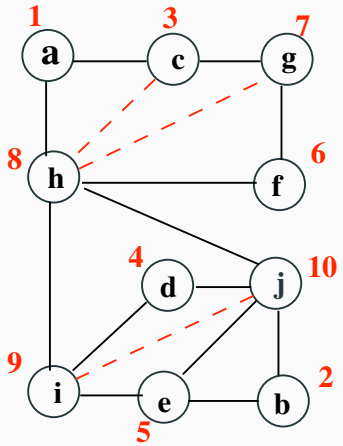
$T(A)$, the elimination tree of A is a spanning tree of $G^+(A)$ satisfying the relation $PARENT[j] = \min\{i > j : \ell_{ij} \neq 0\}$.

$F =$ 

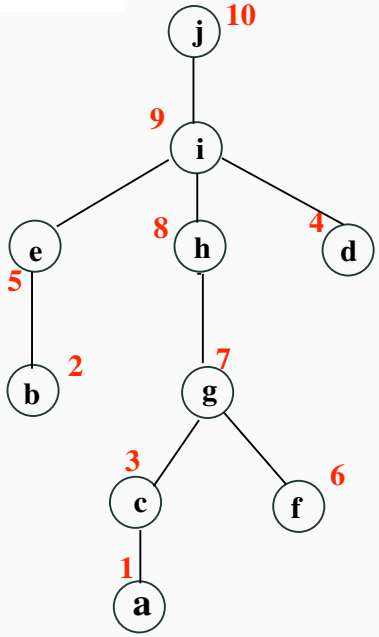
$F_t =$ 



G(A)



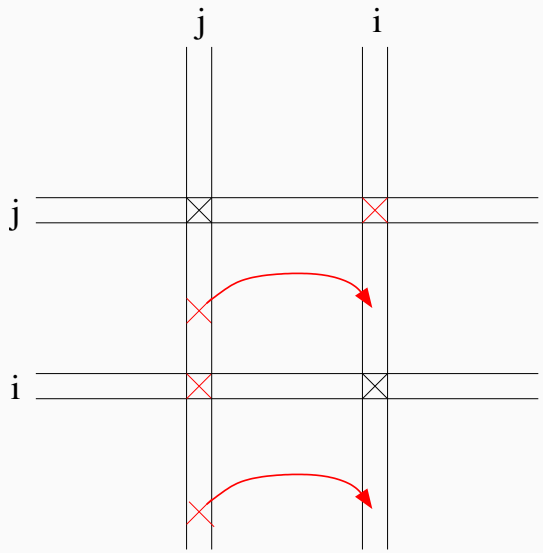
$G^+(A) = G(F)$



T(A)

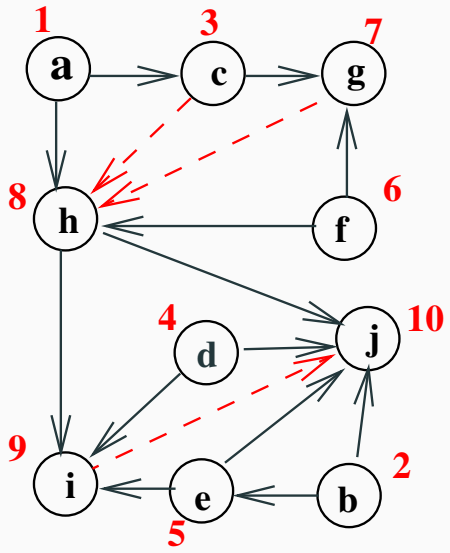
A second definition: Represents column dependencies

- Dependency between columns of L:
  - Column $i > j$ depends on column j iff $\ell_{ij} \neq 0$
  - Use a directed graph to express this dependency (edge from j to i, if column i depends on column j)
  - Simplify redundant dependencies (transitive reduction)

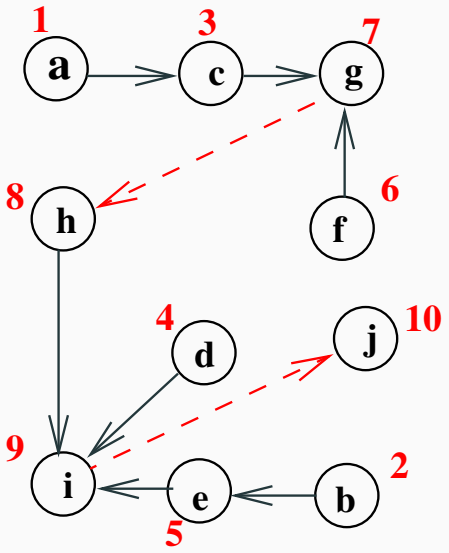


- The transitive reduction of the directed filled graph gives the elimination tree structure. Remove a directed edge (j, i) if there is a path of length greater than one from j to i.

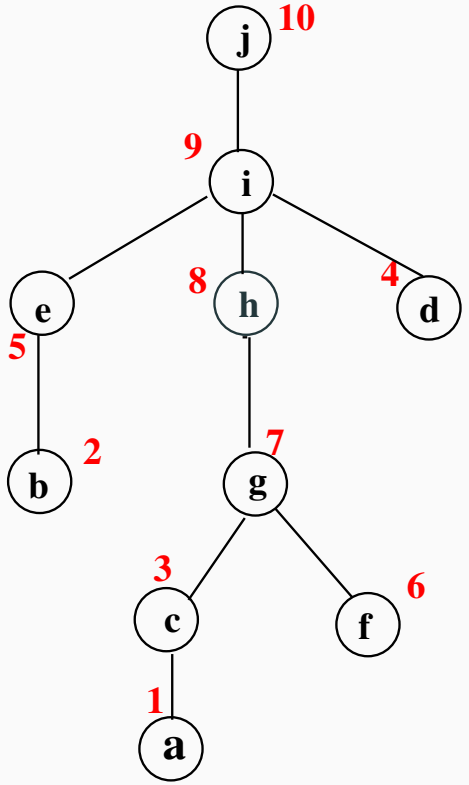
Directed filled graph and its transitive reduction



Directed filled graph



Transitive reduction



T(A)

The elimination tree: a major tool to model factorization

The elimination tree and its generalisation to unsymmetric matrices are compact structures of major importance during sparse factorization

- Express the order in which variables can be eliminated: because the elimination of a variable only affects the elimination of its ancestors, any topological order of the elimination tree will lead to a correct result and to the same fill-in
- Express concurrency: because variables in separate subtrees do not affect each other, they can be eliminated in parallel
- Efficient to characterize the structure of the factors more efficiently than with elimination graphs

Sparse Direct Solvers - Multifrontal Method

Elimination tree and Multifrontal approach

Elimination tree and Multifrontal approach

We recall that:

The elimination tree is the dependency graph of the factorization.

Building the elimination tree

- Permute matrix (to reduce fill-in) $\mathbf{PAP}^T$.
- Build filled matrix $\mathbf{A}_F = \mathbf{L} + \mathbf{L}^T$ where $\mathbf{PAP}^T = \mathbf{LL}^T$
- Transitive reduction of associated filled graph

Multifrontal approach

$\rightarrow$ Each column corresponds to a node of the tree

$\rightarrow$ (multifrontal) Each node k of the tree corresponds to the partial factorization of a frontal matrix whose row and column structure is that of column k of $\mathbf{A}_F$.

The elimination model for symmetric matrices

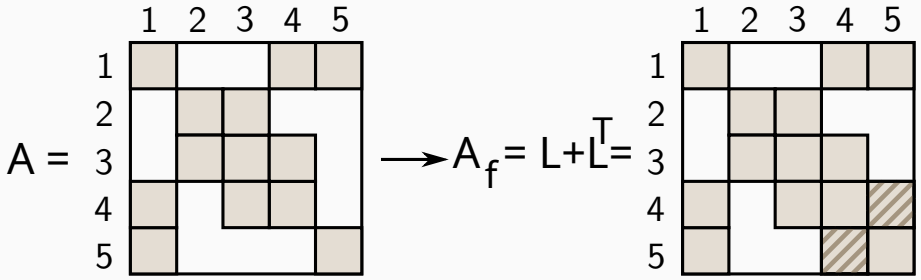
A symmetric, positive definite matrix can be factorized by means of the Cholesky algorithm

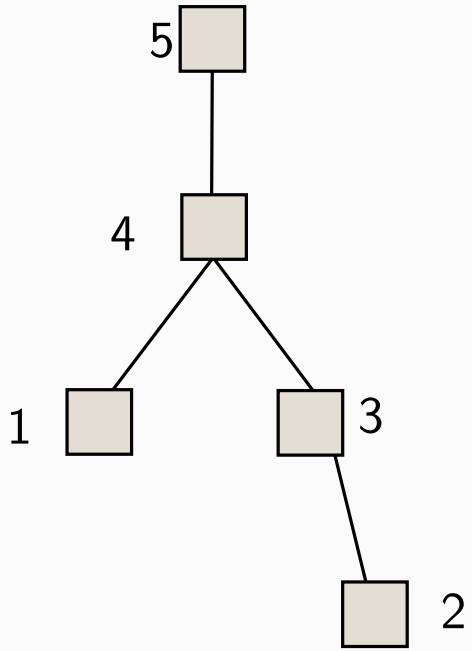
for $k = 1, ..., n$ **do**
  $l_{kk} = \sqrt{a_{kk}^{(k-1)}}$
  for $i = k+1, ..., n$ **do**
    $l_{ik} = a_{ik}^{(k-1)} / l_{kk}$
    for $j = k+1, ..., i$ **do**
      $a_{ij}^{(k)} = a_{ij}^{(k)} - l_{ik} l_{jk}$
    end for
  end for
end for

$$\begin{pmatrix} a_{11} & & & \\ a_{21} & a_{22} & & \\ a_{31} & a_{32} & a_{33} & \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix}$$

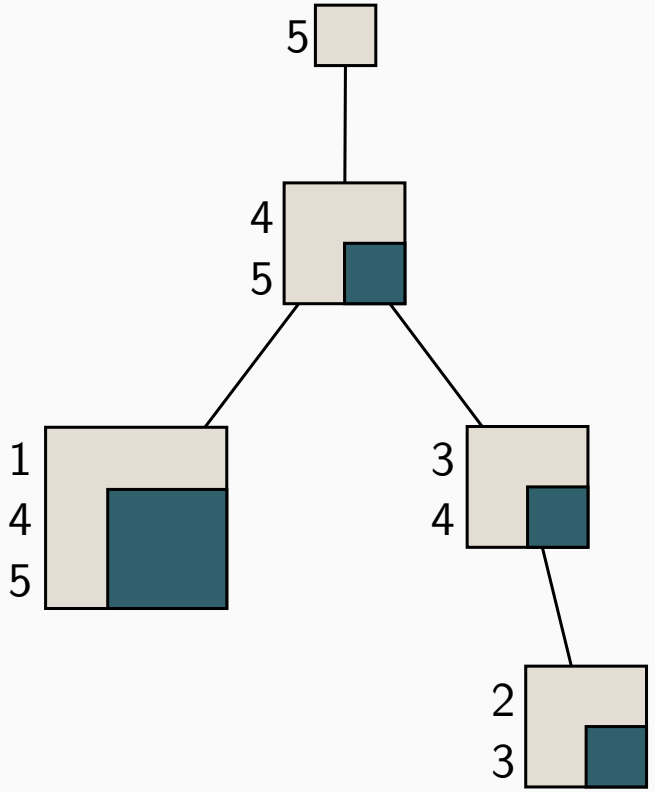
The multifrontal method [Duff & Reid '83]

Each node k of the tree corresponds to the partial factorization of a frontal matrix whose row and column structure is that of column k of $\mathbf{A}_F$.

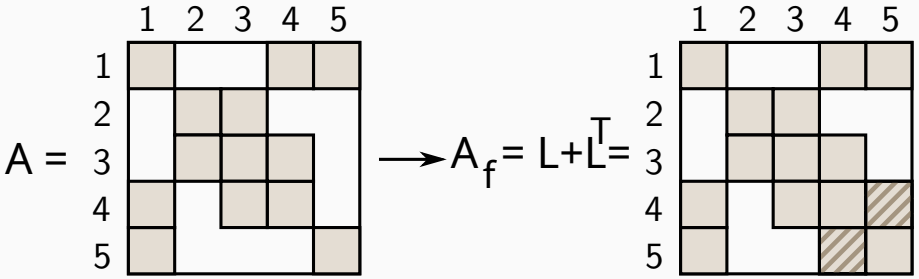




Elimination Tree

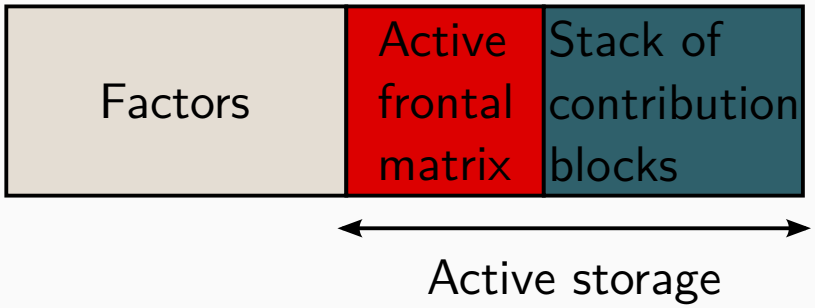


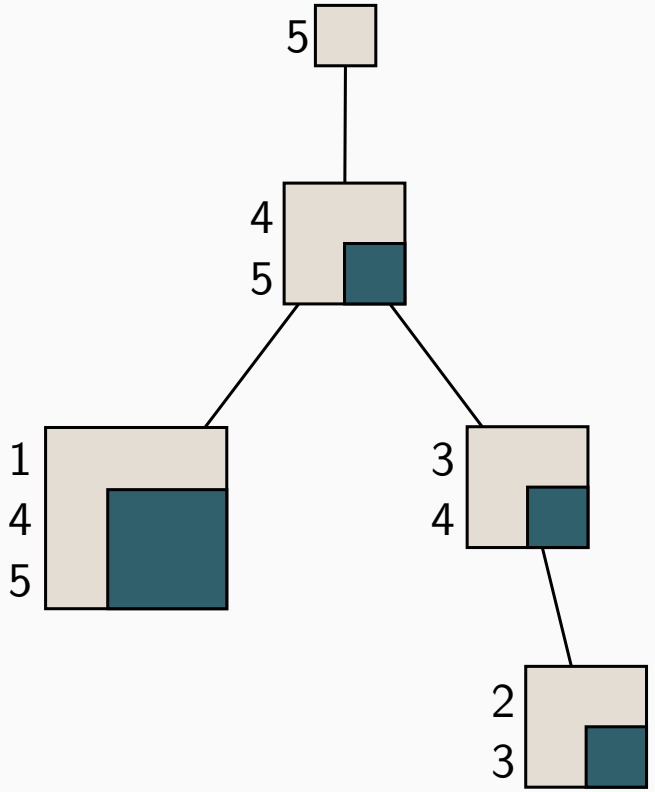
The multifrontal method [Duff & Reid '83]



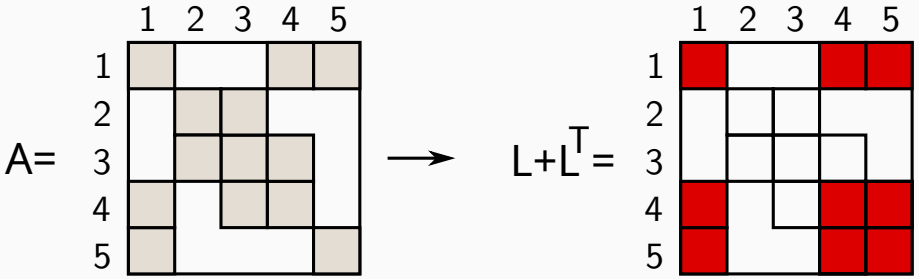
Storage is divided into two parts:

- Factors
- Active memory



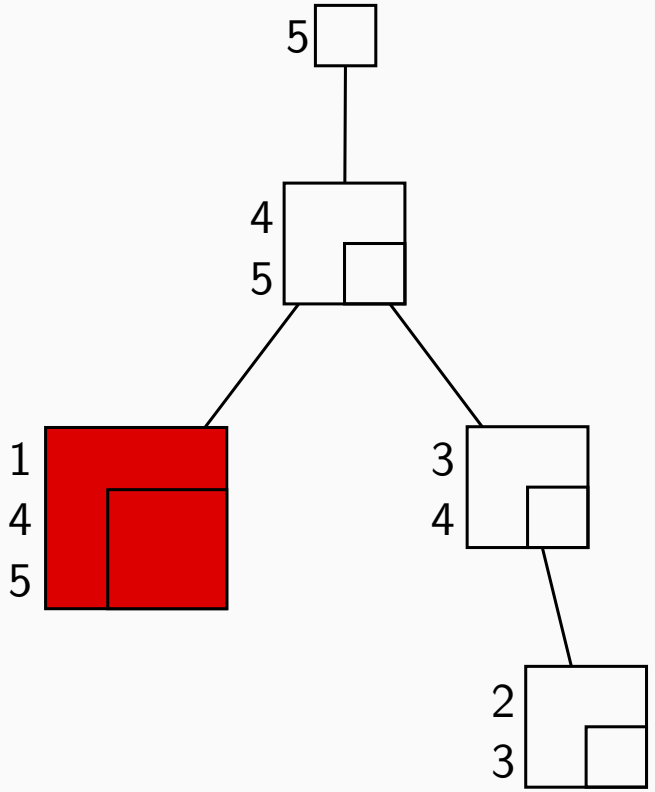


The multifrontal method [Duff & Reid '83]



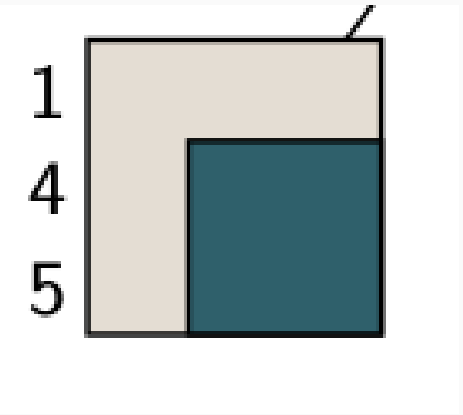
Storage is divided into two parts:

- Factors
- Active memory



Elimination of variable 1

1. Take the symmetry of the matrix into account
2. Assembly of the frontal matrix:
   - $\ell_{11} = \sqrt{a_{11}}$,
   - $\ell_{41} = \frac{a_{14}}{\ell_{11}}$
   - $\ell_{51} = \frac{a_{15}}{\ell_{11}}$

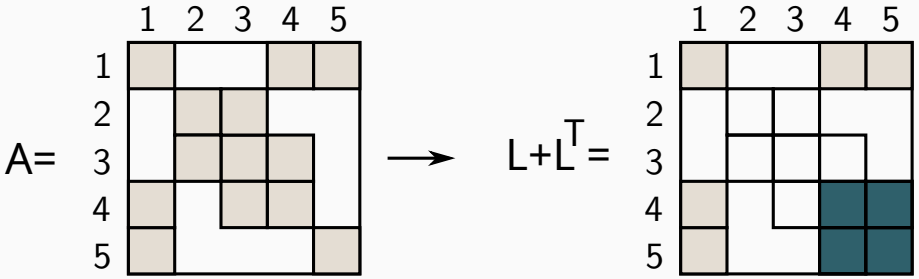


   The terms of the factors (grey) are stored and **will not changed** until the end of the factorization.
3. Compute the contribution matrix:

$$\begin{aligned} a_{44}^{(1)} &= a_{44} - \ell_{41} \times \ell_{41} \\ a_{54}^{(1)} &= a_{54} - \ell_{51} \times \ell_{41} \\ a_{55}^{(1)} &= a_{55} - \ell_{51} \times \ell_{51} \end{aligned}$$

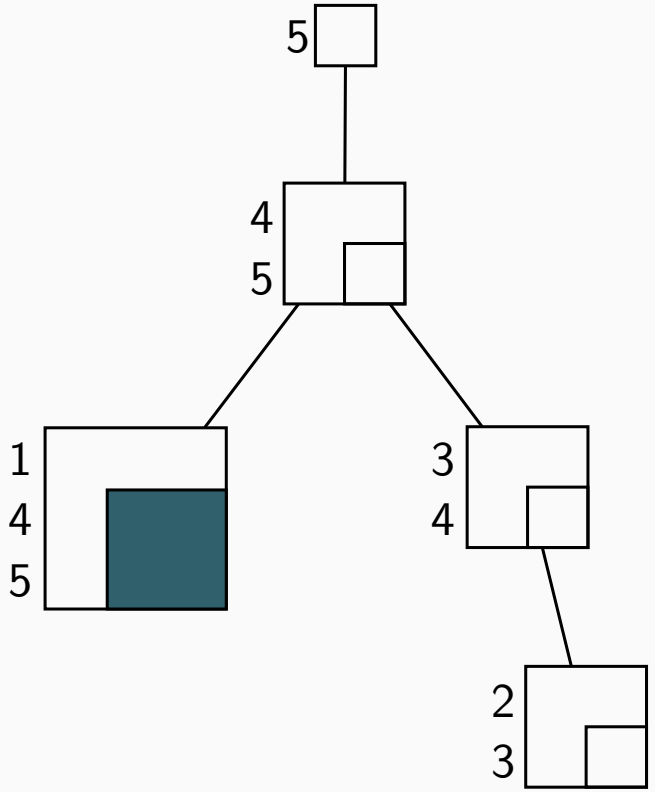
   The terms of the contribution matrix (teal) are stored for later updates when the parent node is eliminated.

The multifrontal method [Duff & Reid '83]

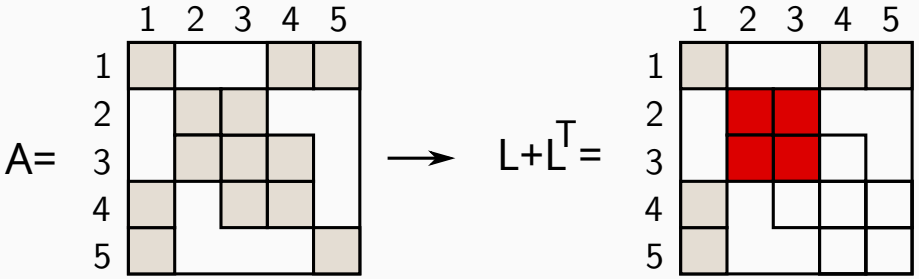


Storage is divided into two parts:

- Factors
- Active memory

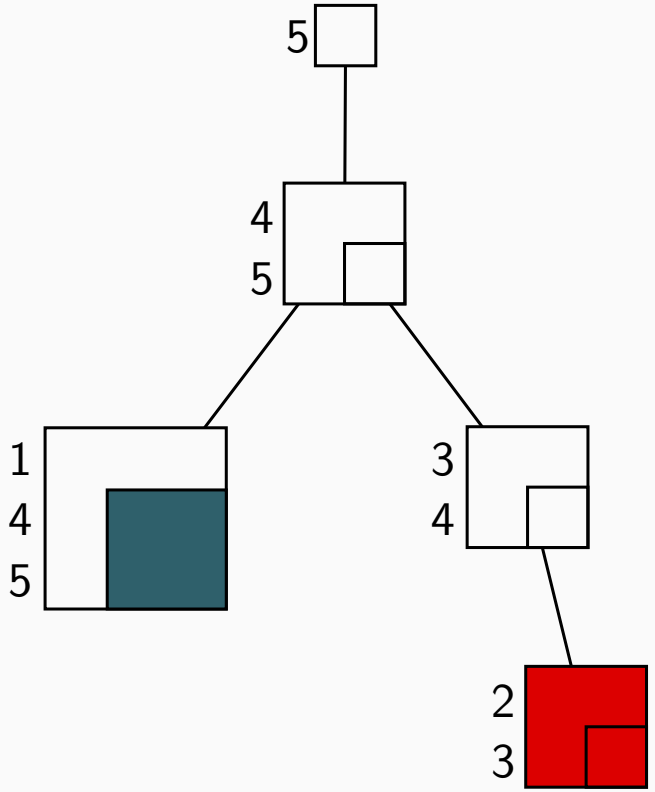


The multifrontal method [Duff & Reid '83]

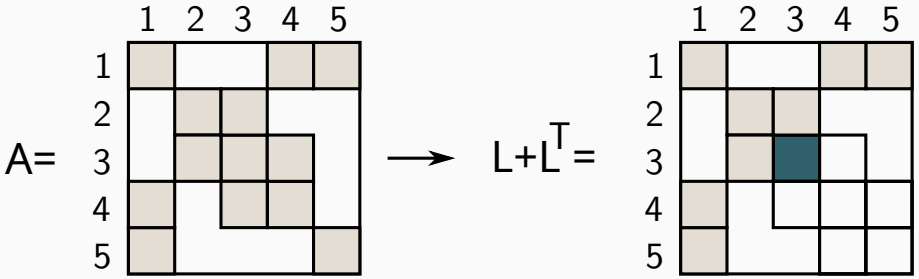


Storage is divided into two parts:

- Factors
- Active memory

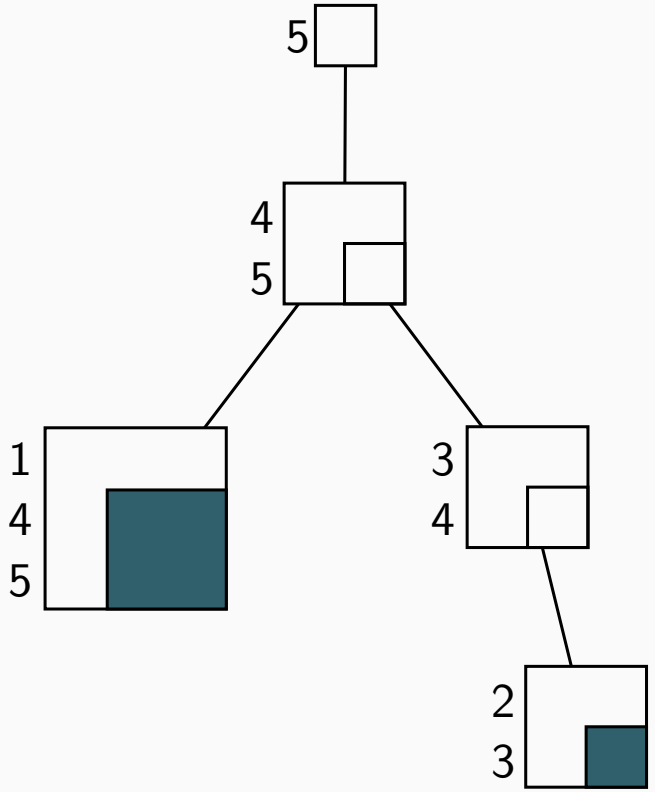


The multifrontal method [Duff & Reid '83]

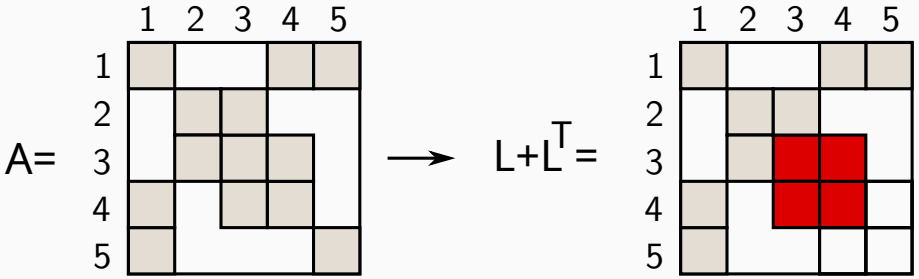


Storage is divided into two parts:

- Factors
- Active memory

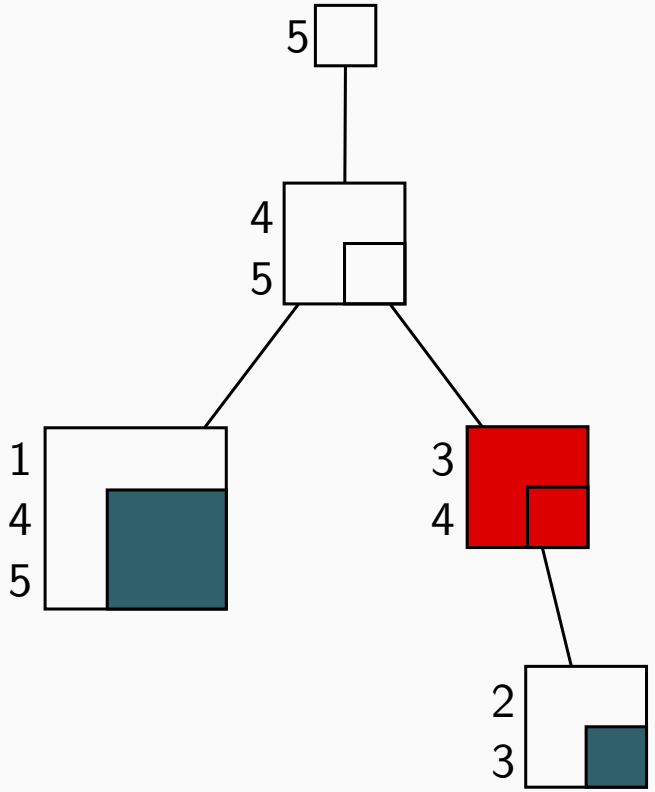


The multifrontal method [Duff & Reid '83]

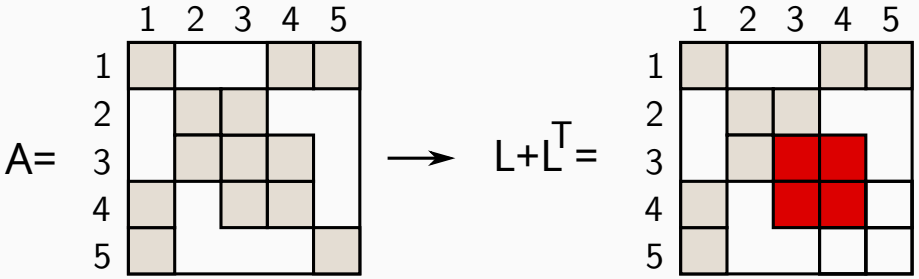


Storage is divided into two parts:

- Factors
- Active memory

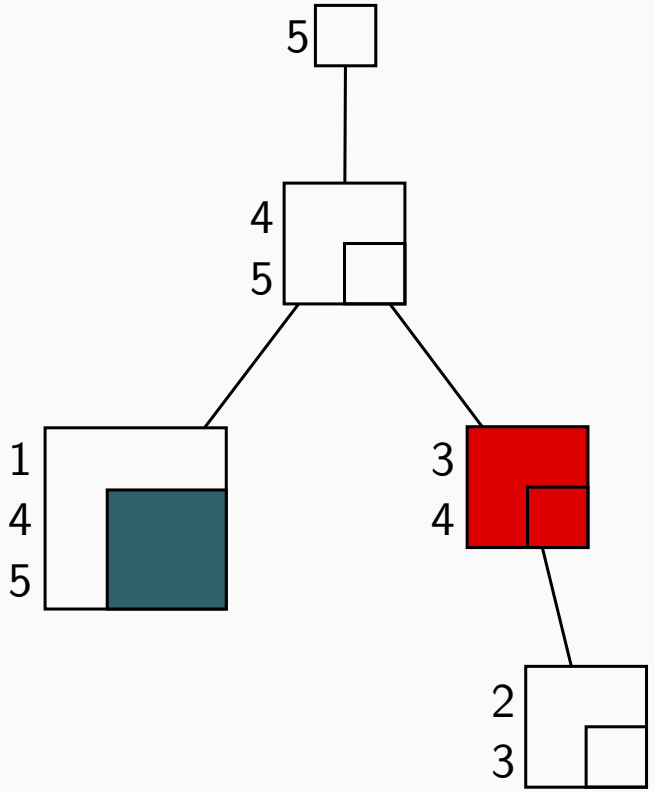


The multifrontal method [Duff & Reid '83]

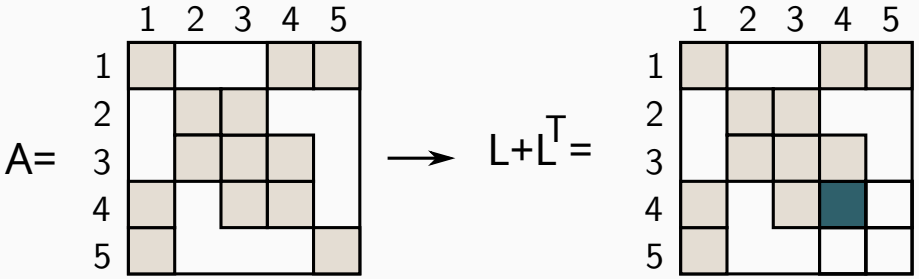


Storage is divided into two parts:

- Factors
- Active memory

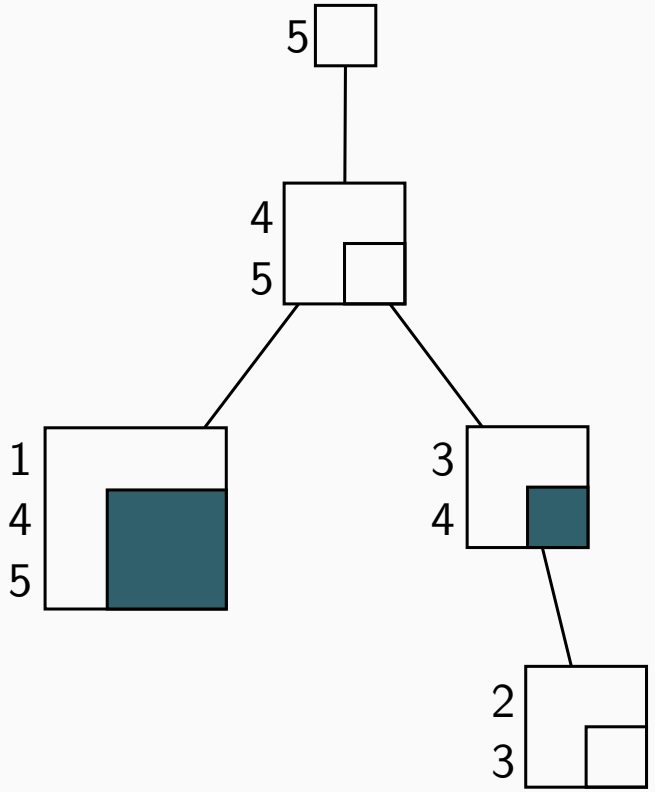


The multifrontal method [Duff & Reid '83]

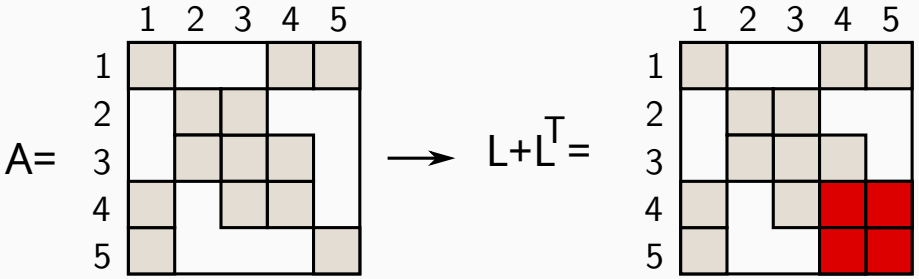


Storage is divided into two parts:

- Factors
- Active memory

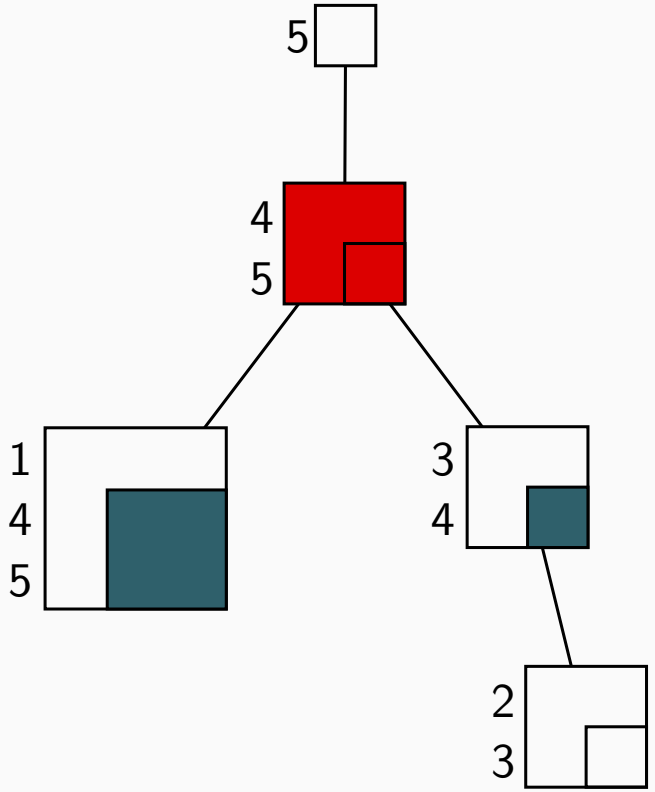


The multifrontal method [Duff & Reid '83]

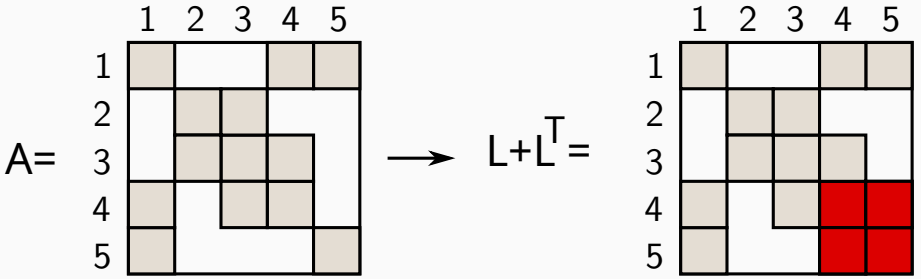


Storage is divided into two parts:

- Factors
- Active memory

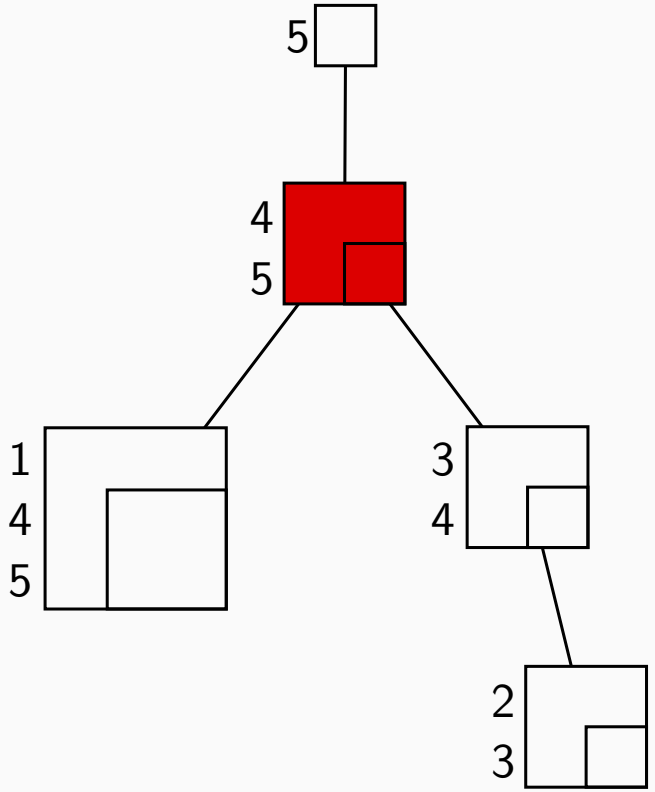


The multifrontal method [Duff & Reid '83]

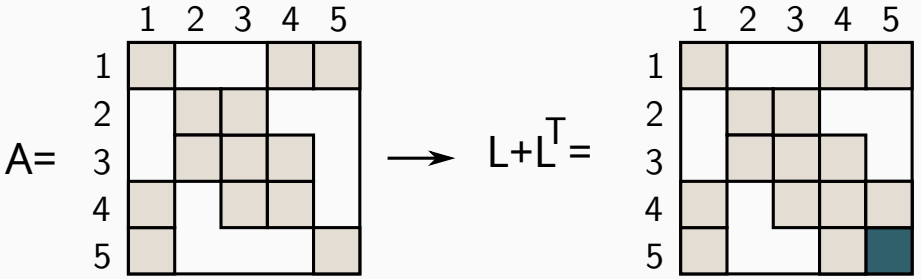


Storage is divided into two parts:

- Factors
- Active memory

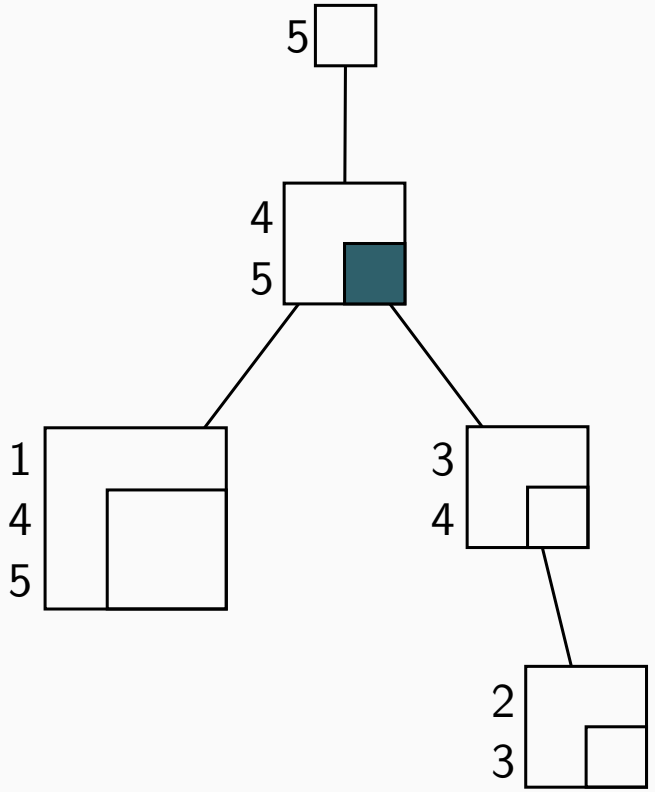


The multifrontal method [Duff & Reid '83]

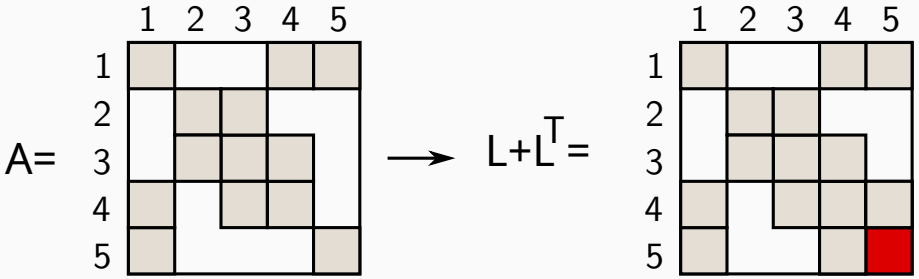


Storage is divided into two parts:

- Factors
- Active memory

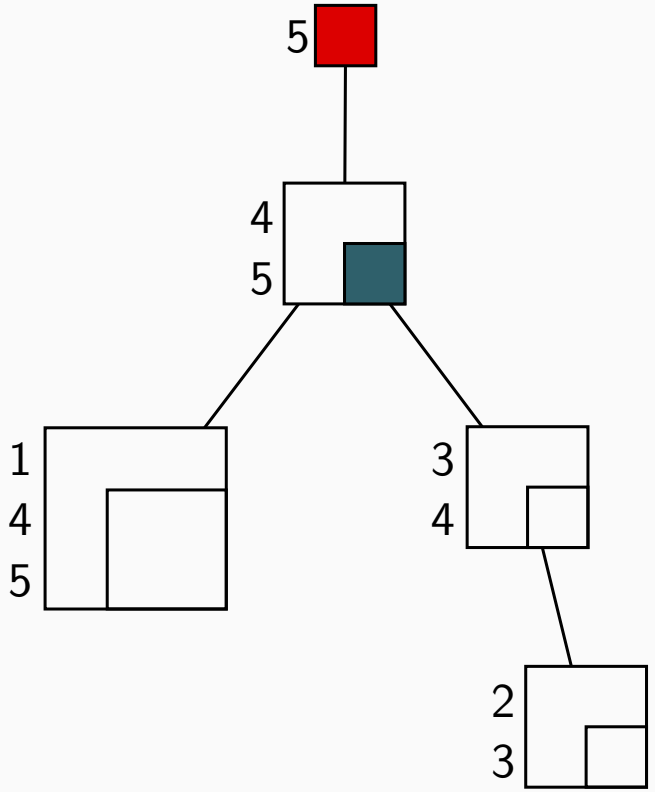


The multifrontal method [Duff & Reid '83]

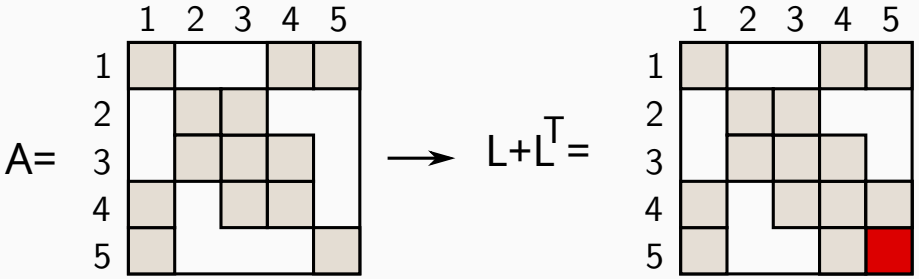


Storage is divided into two parts:

- Factors
- Active memory

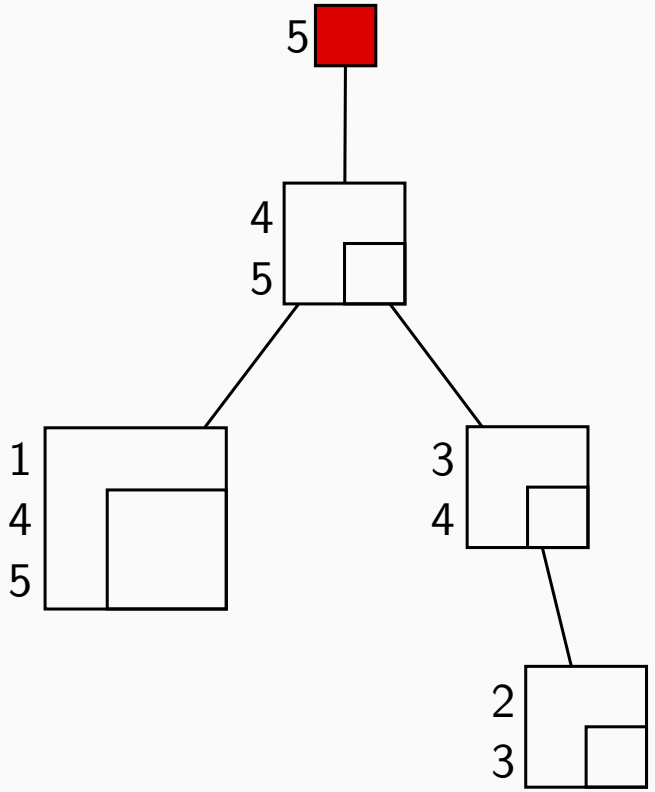


The multifrontal method [Duff & Reid '83]



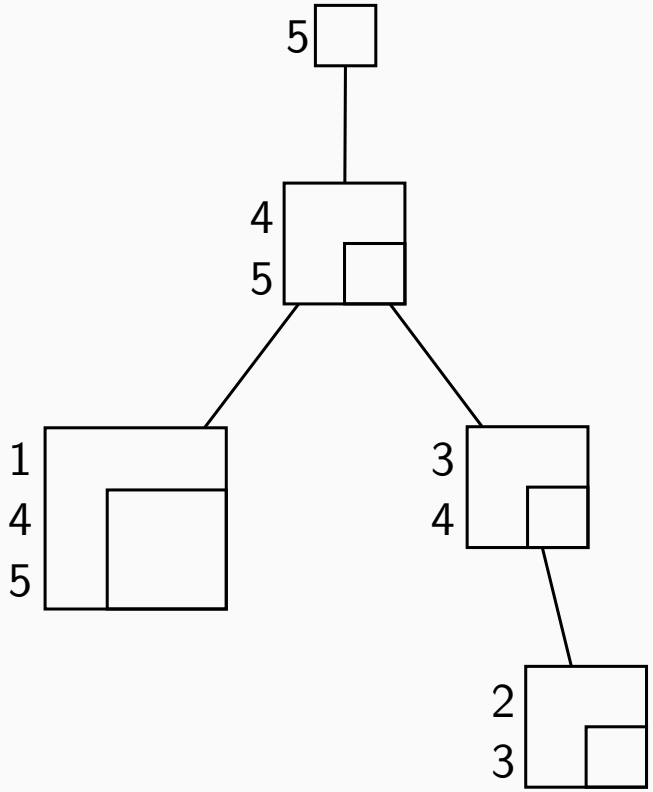
Storage is divided into two parts:

- Factors
- Active memory



The multifrontal method [Duff & Reid '83]

- Factors are incompressible and usually scale fairly; they can optionally be written on disk.
- In sequential, the traversal that minimizes active memory is known [Liu'86].
- In parallel, active memory becomes dominant.
  Example: share of active storage on the AUDI matrix using MUMPS 4.10.0
  1 processor: 11%
  256 processors: 59%



Sparse Direct Solvers - Multifrontal Method

Postorderings and memory usage

Postorderings and memory usage

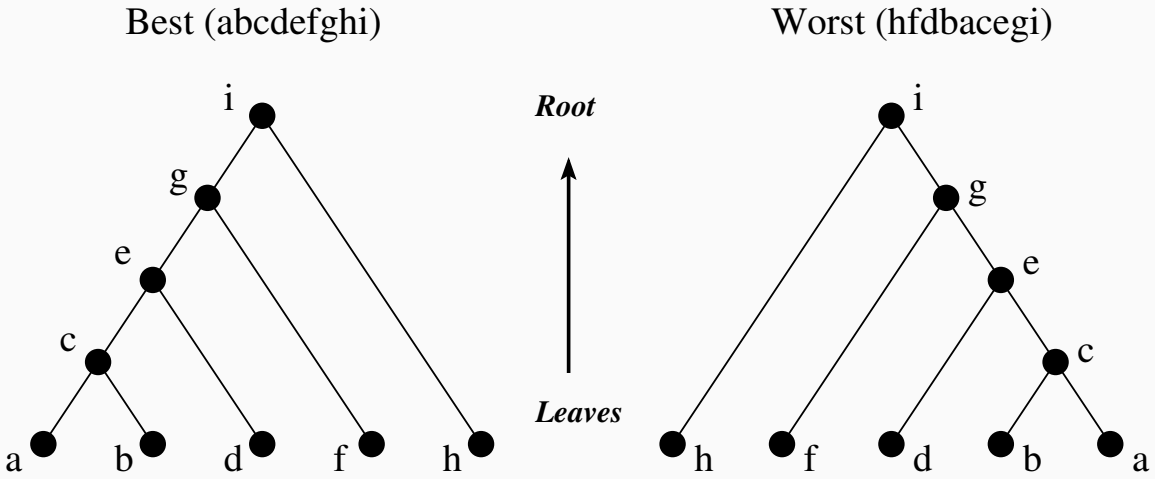
- Assumptions:
  - Tree processed from the leaves to the root
  - Parents processed as soon as all children have completed (postorder of the tree)
  - Each node produces and sends temporary data consumed by its father.
- **Exercise:** In which sense is a postordering-based tree traversal more interesting than a random topological ordering ?

Postorderings and memory usage

- Assumptions:
  - Tree processed from the leaves to the root
  - Parents processed as soon as all children have completed (postorder of the tree)
  - Each node produces and sends temporary data consumed by its father.
- **Exercise:** In which sense is a postordering-based tree traversal more interesting than a random topological ordering ?
- Answer:
  - we consume as soon possible what is produce by the child nodes,
  - we control and limit the memory usage,
  - good locality of data: we have a stack (see multifrontal method).

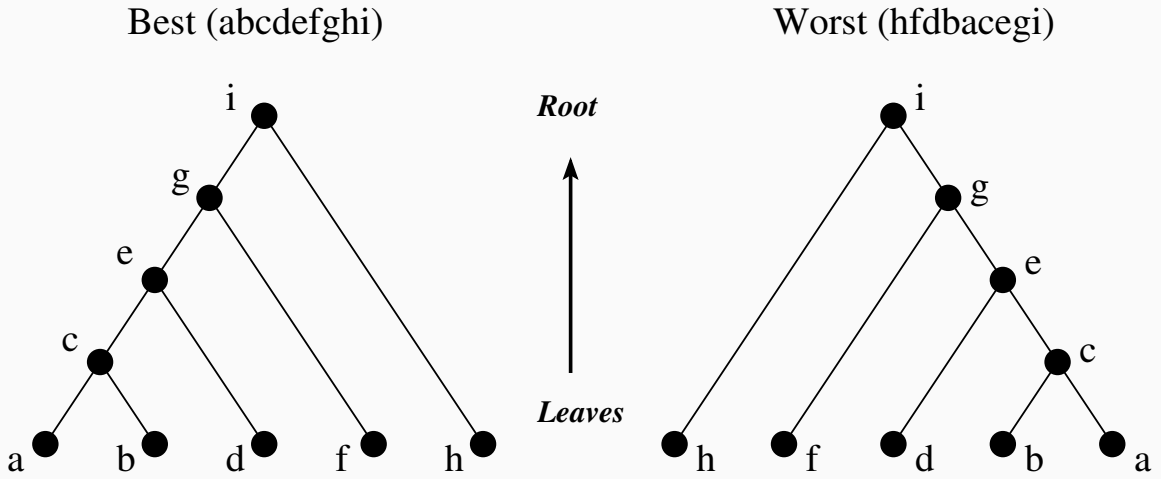
Postorderings and memory usage

- Furthermore, memory usage also depends on the postordering chosen:



Postorderings and memory usage

- Furthermore, memory usage also depends on the postordering chosen:

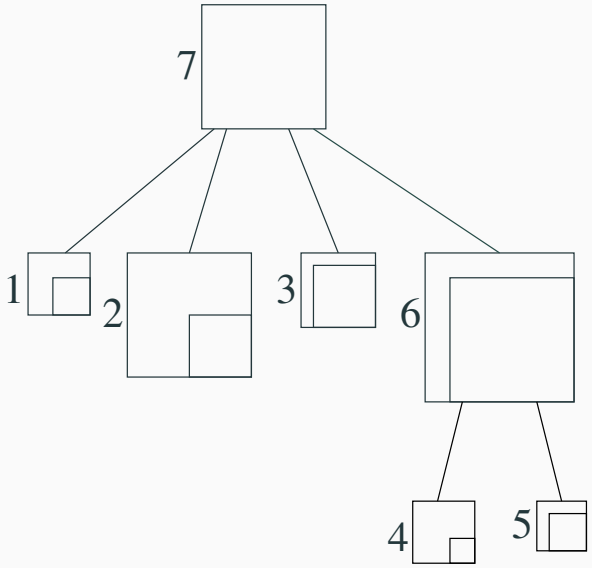


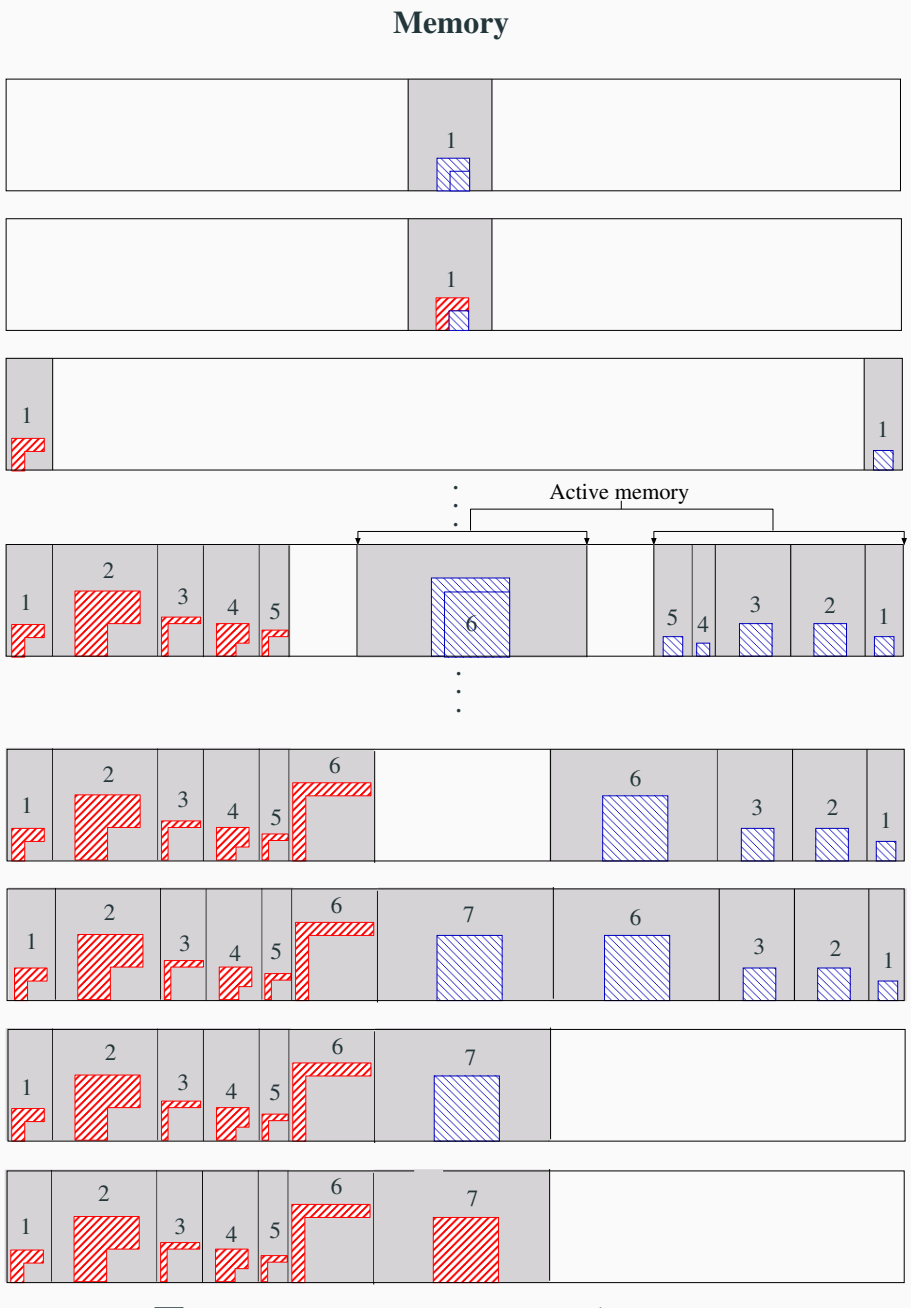
Assumptions:

- each node need 2GB for its treatment,
- each node produces a temporary data of 1GB that will be comsume by its parent.

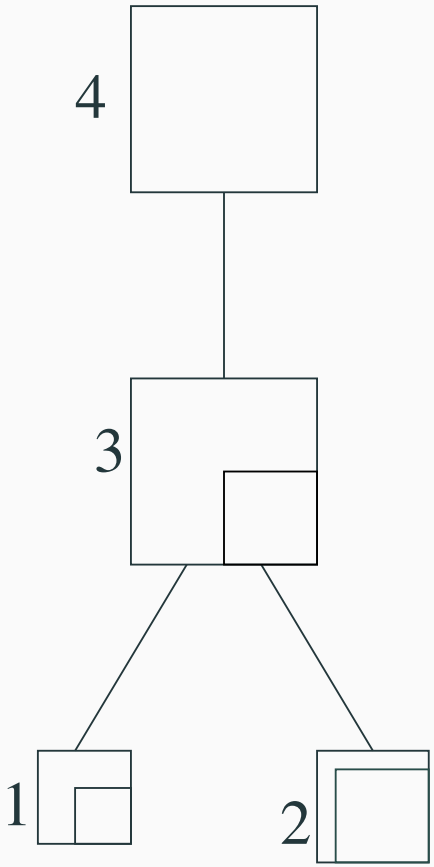
Compute the memory usage for each postordering (left and right).

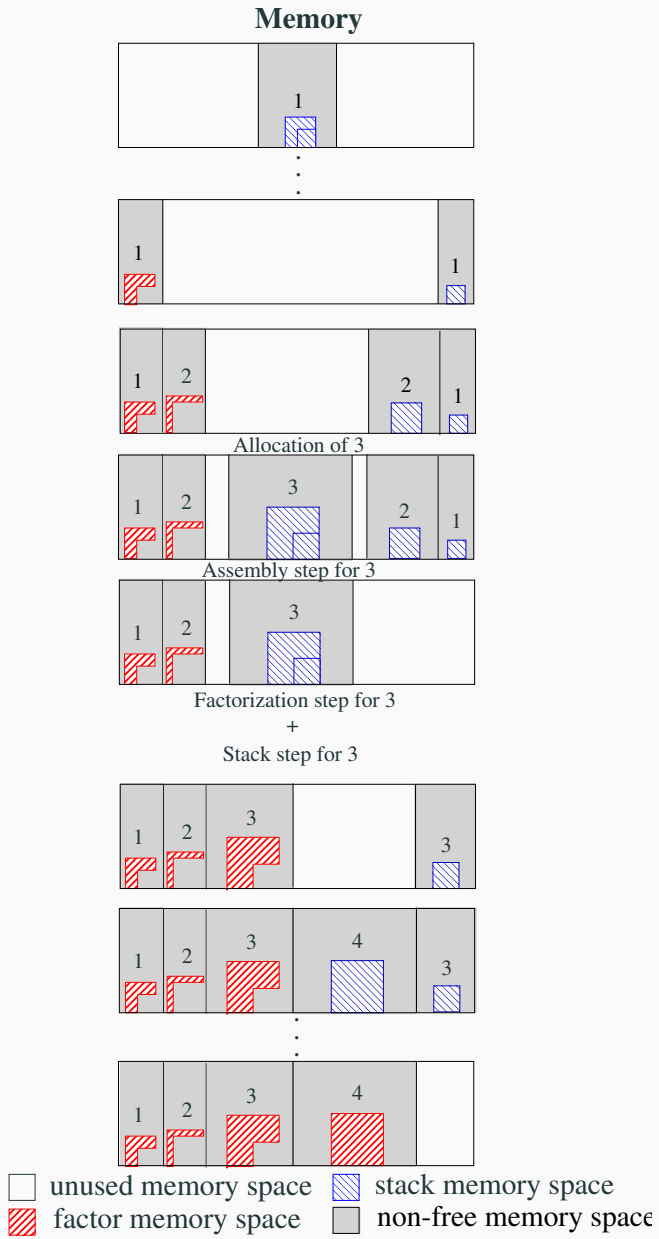
Example 1: Processing a wide tree





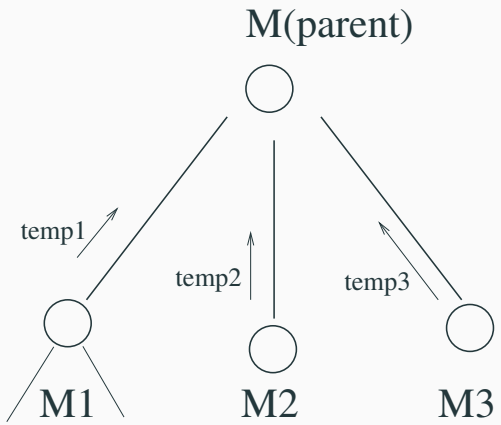
Example 2: Processing a deep tree





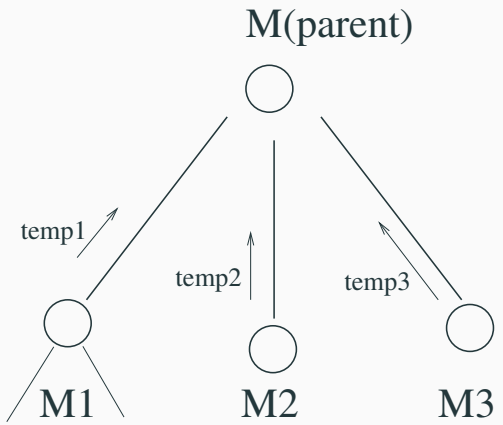
Minimizing the active memory: modelization of the problem

- M_i: memory peak for complete subtree rooted at i,
- $temp_i$: temporary memory produced by node i,
- m_{parent}: memory for storing the parent.



Minimizing the active memory: modelization of the problem

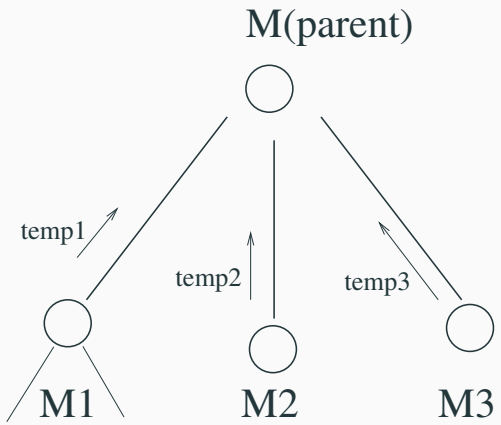
- M_i: memory peak for complete subtree rooted at i,
- $temp_i$: temporary memory produced by node i,
- m_{parent}: memory for storing the parent.



$$M_{parent} = \max\Big(\begin{array}{l} \max_{j=1}^{nbchildren}(M_j + \sum_{k=1}^{j-1} temp_k), \\ m_{parent} + \sum_{j=1}^{nbchildren} temp_j \end{array} \Big) \quad (1)$$

Minimizing the active memory: modelization of the problem

- M_i: memory peak for complete subtree rooted at i,
- $temp_i$: temporary memory produced by node i,
- m_{parent}: memory for storing the parent.



$$M_{parent} = \max\Big(\begin{array}{l} \max_{j=1}^{nbchildren}(M_j + \sum_{k=1}^{j-1} temp_k), \\ m_{parent} + \sum_{j=1}^{nbchildren} temp_j \end{array} \Big) \quad (1)$$

Objective: order the children to minimize M_{parent}

Memory-minimizing schedules

Theorem

[Liu,86] *The minimum of* $\max_j(x_j + \sum_{i=1}^{j-1} y_i)$ *is obtained when the sequence* (x_i, y_i) *is sorted in decreasing order of* $x_i - y_i$.

Corollary

An optimal child sequence is obtained by rearranging the children nodes in decreasing order of $M_i - temp_i$.

Interpretation: At each level of the tree, child with relatively large peak of memory in its subtree (M_i large with respect to $temp_i$) should be processed first.

$\Rightarrow$ Apply on complete tree starting from the leaves
(or from the root with a recursive approach)

Sparse Direct Solvers - Multifrontal Method

Concluding remarks

Concluding remarks

- **Key parameters in selecting a method**
  1. Functionalities of the solver
  2. Characteristics of the matrix
     - Numerical properties and pivoting.
     - Symmetric or general
     - Pattern and density
  3. Preprocessing of the matrix
     - Scaling
     - Reordering for minimizing fill-in
  4. Target computer (architecture)
- Substantial gains can be achieved with an adequate solver: in terms of numerical precision, computing and storage
- Good knowledge of matrix and solvers
- **Many challenging problems**
  - Active research area