Sparse Linear Algebra: Direct Methods

P. Amestoy and A. Buttari
INPT(ENSEEIHT)

A. Guermouche (Univ. Bordeaux-LaBRI),
J.-Y. L'Excellent and Bora Uçar (INRIA-CNRS/LIP-ENS Lyon) and
F.-H. Rouet (LSTC, Livermore, California)

2018-2019

Outline

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, book to appear
- 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.

Motivations

- solution of linear systems of equations $\rightarrow$ key algorithmic kernel

Continuous problem
$\downarrow$
Discretization
$\downarrow$
Solution of a linear system $Ax = b$

- Main parameters:
  - Numerical properties of the linear system (symmetry, pos. definite, conditioning, ...)
  - Size and structure:
    - Large (order $> 10^7$ to 10^8), square/rectangular
    - Dense or sparse (structured / unstructured)
  - Target computer (sequential/parallel/multicore)

$\rightarrow$ *Algorithmic choices are critical*

Motivations for designing efficient algorithms

- Time-critical applications
- Solve larger problems
- Decrease elapsed time (code optimization, parallelism)
- Minimize cost of computations (time, memory)

Difficulties

- Access to data:
  - Computer: complex memory hierarchy (registers, multilevel cache, main memory (shared or distributed), disk)
  - Sparse matrix: large irregular dynamic data structures.

  $\rightarrow$ *Exploit the locality of references to data on the computer (design algorithms providing such locality)*
- Efficiency (time and memory)
  - Number of operations and memory depend very much on the algorithm used and on the numerical and structural properties of the problem.
  - The algorithm depends on the target computer (vector, scalar, shared, distributed, clusters of Symmetric Multi-Processors (SMP), multicore).

  $\rightarrow$ *Algorithmic choices are critical*

Sparse matrices

Example:

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

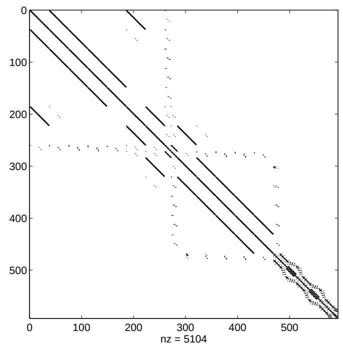
can be represented as

$$\mathbf{Ax} = \mathbf{b},$$

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.

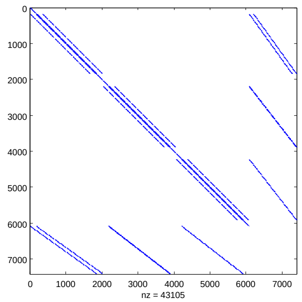
Sparse matrix ?



Matrix dwt_592.rua (N=592, NZ=5104);
Structural analysis of a submarine

Sparse matrix ?

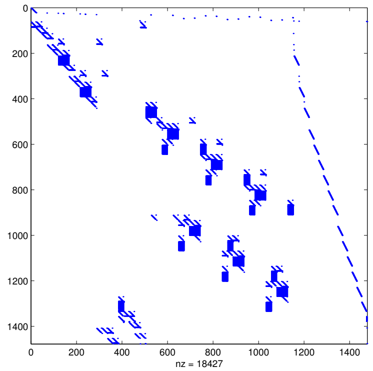
Matrix from Computational Fluid Dynamics;
(collaboration Univ. Tel Aviv)



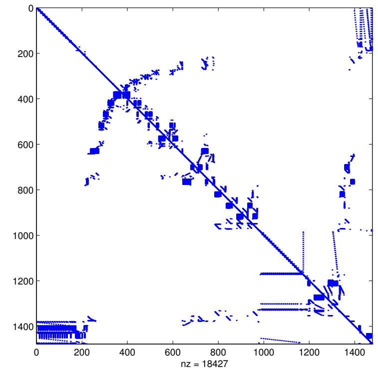
"Saddle-point" problem

Preprocessing sparse matrices

Original ($A = \text{LHR01}$)



Preprocessed matrix ($A'(\text{LHR01})$)



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:
  - positive definite $\mathbf{A} = \mathbf{LL}^{\mathsf{T}}$
  - general $\mathbf{A} = \mathbf{LDL}^{\mathsf{T}}$
- $\mathbf{A}$ is rectangular $m \times n$ with $m \geq n$ and $\min_x \|\mathbf{Ax} - \mathbf{b}\|_2$:
  - $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ is orthogonal ($\mathbf{Q}^{-1} = \mathbf{Q}^{\mathsf{T}}$) and $\mathbf{R}$ is triangular.
  - Solve: $\mathbf{y} = \mathbf{Q}^{\mathsf{T}}\mathbf{b}$ then $\mathbf{Rx} = \mathbf{y}$

Difficulties

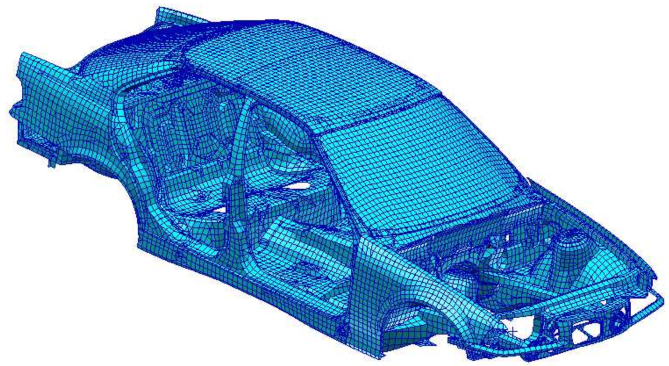
- Only non-zero values are stored
- Factors $\mathbf{L}$ and $\mathbf{U}$ have far more nonzeros than $\mathbf{A}$
- Data structures are complex
- Computations are only a small portion of the code (the rest is data manipulation)
- Memory size is a limiting factor ($\rightarrow$ *out-of-core solvers*)

Key numbers:

1- **Small sizes** : 500 MB matrix;
Factors = 5 GB; Flops = 100 Gflops ;
2- **Example of 2D problem**: Lab. Géosiences Azur, Valbonne
- Complex 2D finite difference matrix n=16×10^6 , 150×10^6 nonzeros
- Storage (single prec): 2 GB (12 GB with the factors)
- Flops: 10 TeraFlops

3- **Example of 3D problem**: EDF (Code_Aster, structural engineering)
- Real matrix finite elements $n = 10^6$, $nz = 71 \times 10^6$ nonzeros
- Storage: 3.5×10^9 entries (28 GB) for factors, 35 GB total
- Flops: 2.1×10^{13}

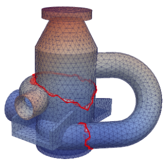
Example in structural mechanics:



Car body,
227,362 unknowns,
5,757,996 nonzeros,
MSC.Software

Size of factors:
51.1 million entries
Number of operations:
44.9×10^9

Sparse linear solvers: typical numbers

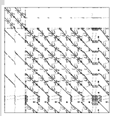


pump (credits: Code_Aster)

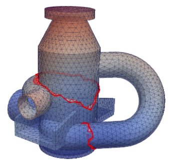
Discretisation of physical problem
→ Solve **AX**=**B** or **Ax**= **b**, **A** large and sparse
- Direct solution: factor **A**=**LU** (or $\mathbf{LDL}^T$ if symmetric) then **X**= $\mathbf{U}^{-1}(\mathbf{L}^{-1}\ \mathbf{B})$
- Iterative solvers: **x**= $\lim_k \mathbf{x}_k$

Typical numbers (illustration on the pump problem)

- **A**: $n = 5.4 \times 10^6$ with $nnz = 2 \times 10^8$ nonzeros
- Target computer:
  - Peak performance: 518 Gflops/s (14 cores × 37 Gflops/s/core)
  - Max memory bandwidth: $bw = 28$ GB/s



Sparse linear solvers: time analysis

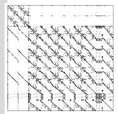


pump (credits: Code_Aster)

Discretisation of physical problem
→ Solve **AX**=**B** or **Ax**= **b**, **A** large and sparse
- **A**: $n = 5.4 \times 10^6$ with $nnz = 2 \times 10^8$ nonzeros
Peak performance of computer: 518 Gflops/s
(14 cores × 37 Gflops/s/core)
Memory bandwidth: $bw = 28$ GB/s

Typical numbers (illustration on the pump problem)

- Factor **A**= $\mathbf{LDL}^T$ → **L**: 5×10^9 nonzeros (40 GB)! (fill -in)
- Direct solution: factor **A** (**A**= $\mathbf{LDL}^T$) → 25 Tflops
Time(direct) ∼ 175 s (Tera = 10^{12}); → (143 Gflops/s)
- Iterative solution: key kernel is often **A***y*: 5×10^8 flops
Min time **A***y* ∼ 0.1 s (7 Gflops/s)
$\left(\frac{(2 \text{ flops per access to } \mathbf{A}) \times (\text{bw in GB/s})}{(8 \text{ Bytes per element of } \mathbf{A})}\right)$
Time(iterative) $\sim 0.1 \text{ s} \times NbIter \times NbCol(\mathbf{B}) + t_{precond}$



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: MxN matrix **A** with NNZ nonzeros.
- Elemental matrices (unassembled): MxN matrix **A** with NELT elements.
- Arithmetic: Real (4 or 8 bytes) or complex (8 or 16 bytes)
- Symmetric (or Hermitian)
  $\rightarrow$ 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

$$\begin{array}{llcccccc} \text{IRN} & [1:NNZ] & = & 1 & 3 & 2 & 2 & 3 \\ \text{JCN} & [1:NNZ] & = & 1 & 1 & 2 & 3 & 3 \\ \text{VAL} & [1:NNZ] & = & a_{11} & a_{31} & a_{22} & a_{23} & a_{33} \end{array}$$

- Compressed Sparse Column (CSC) format

$$\begin{array}{llcccccc} \text{IRN} & [1:NNZ] & = & 1 & 3 & 2 & 2 & 3 \\ \text{VAL} & [1:NNZ] & = & a_{11} & a_{31} & a_{22} & a_{23} & a_{33} \\ \text{COLPTR} & [1:N+1] & = & 1 & 3 & 4 & 6 & \end{array}$$

  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

	1	2	3
1	a11		
2		a22	a23
3	a31		a33

- Diagonal format (M=N):
  NDIAG = 3
  IDIAG = $-2\ 0\ 1$

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

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

Sparse Matrix-vector products $Y \leftarrow AX$

Algorithm depends on sparse matrix format:

- Coordinate format:
- CSC format:
- CSR format

Example of elemental matrix format

$$\mathbf{A}_1 = \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{pmatrix} -1 & 2 & 3 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{array}{c} 3 \\ 4 \\ 5 \end{array} \begin{pmatrix} 2 & -1 & 3 \\ 1 & 2 & -1 \\ 3 & 2 & 1 \end{pmatrix}$$

$$\mathbf{A} = \begin{pmatrix} -1 & 2 & 3 & 0 & 0 \\ 2 & 1 & 1 & 0 & 0 \\ 1 & 1 & 3 & -1 & 3 \\ 0 & 0 & 1 & 2 & -1 \\ 0 & 0 & 3 & 2 & 1 \end{pmatrix} = \mathbf{A}_1 + \mathbf{A}_2$$

Example of elemental matrix format

$$\mathbf{A}_1 = \begin{array}{c} 1 \\ 2 \\ 3 \end{array} \begin{pmatrix} -1 & 2 & 3 \\ 2 & 1 & 1 \\ 1 & 1 & 1 \end{pmatrix}, \quad \mathbf{A}_2 = \begin{array}{c} 3 \\ 4 \\ 5 \end{array} \begin{pmatrix} 2 & -1 & 3 \\ 1 & 2 & -1 \\ 3 & 2 & 1 \end{pmatrix}$$

- N=5 NELT=2 NVAR=6 $\mathbf{A} = \sum_{i=1}^{NELT} \mathbf{A}_i$
ELTPTR	[1:NELT+1]	=	1 4 7
ELTVAR	[1:NVAR]	=	1 2 3 3 4 5
ELTVAL	[1:NVAL]	=	-1 2 1 2 1 1 3 1 1 2 1 3 -1 2 2 3 -1 1
- Remarks:
  - NVAR = ELTPTR(NELT+1)-1
  - Order of element i: $S_i = ELTPTR(i+1) - ELTPTR(i)$
  - NVAL = $\sum S_i^2$ (unsym) ou $\sum S_i(S_i+1)/2$ (sym),
  - storage of elements in ELTVAL: by columns

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

- The University of Florida Sparse Matrix Collection `http://www.cise.ufl.edu/research/sparse/matrices/`
- Matrix market `http://math.nist.gov/MatrixMarket/`
- Rutherford-Boeing `http://www.cerfacs.fr/algor/Softs/RB/index.html`
- TLSE `http://gridtlse.org/`

Gaussian elimination

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

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

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

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

<u>Finally</u> $\mathbf{A}^{(3)}\mathbf{x} = \mathbf{b}^{(3)}$

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

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

Relation with $\mathbf{A} = \mathbf{LU}$ factorization

- One step of Gaussian elimination can be written: $\mathbf{A}^{(k+1)} = \mathbf{L}^{(k)}\mathbf{A}^{(k)}$, with

$$\mathbf{L}^{(k)} = \begin{pmatrix} 1 & & & & & \\ & \cdot & & & & \\ & & \cdot & & & \\ & & & 1 & & \\ & & & -\mathbf{l_{k+1,k}} & \cdot & \\ & & & \cdot & & \cdot \\ & & & -\mathbf{l_{n,k}} & & & 1 \end{pmatrix} \text{ and } l_{ik} = \frac{a_{ik}^{(k)}}{a_{kk}^{(k)}}.$$

- Then, $\mathbf{A}^{(n)} = \mathbf{U} = \mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)}\mathbf{A}$, which gives $\boxed{\mathbf{A} = \mathbf{LU}}$,

$$\text{with } \mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1} = \begin{pmatrix} 1 & & 0 & & \\ & \cdot & & & \\ & & \cdot & & \\ & & & \cdot & \\ & & \mathbf{l_{i,j}} & & 1 \end{pmatrix},$$

- In dense codes, entries of $\mathbf{L}$ and $\mathbf{U}$ overwrite entries of $\mathbf{A}$.
- Furthermore, if $\mathbf{A}$ is symmetric, $\boxed{\mathbf{A} = \mathbf{LDL}^T}$ with $d_{kk} = a_{kk}^{(k)}$:
  $A = LU = A^t = U^t L^t$ implies $(U)(L^t)^{-1} = L^{-1}U^t = D$ diagonal and $U = DL^t$, thus $A = L(DL^t) = LDL^t$

Dense $\mathbf{LU}$ factorization

- Step by step columns of $\mathbf{A}$ are set to zero and $\mathbf{A}$ is updated $\mathbf{L}^{(n-1)} \dots \mathbf{L}^{(1)}\mathbf{A} = \mathbf{U}$ leading to
  $\mathbf{A} = \mathbf{LU}$ where $\mathbf{L} = [\mathbf{L}^{(1)}]^{-1} \dots [\mathbf{L}^{(n-1)}]^{-1}$
- - zero entries in column of $\mathbf{A}$ can be replaced by entries in $\mathbf{L}$
  - row entries of $\mathbf{U}$ can be stored in corresponding locations of $\mathbf{A}$

Algorithm 1 Dense **LU** factorization

1: **for** $k = 1$ **to** n **do**
2: $\quad L(k:k) = 1 \;\; ; L(k+1:n,k) = \frac{A(k+1:n,k)}{A(k,k)}$
3: $\quad U(k, k:n) = A(k, k:n)$
4: $\quad$ **for** $j = k+1$ **to** n **do**
5: $\quad\quad$ **for** $i = k+1$ **to** n **do**
6: $\quad\quad\quad A(i,j) = A(i,j) - L(i,k) \times U(k,j)$
7: $\quad\quad$ **end for**
8: $\quad$ **end for**
9: **end for**

When $|\mathbf{A}(k,k)|$ is relatively too small, numerical pivoting required

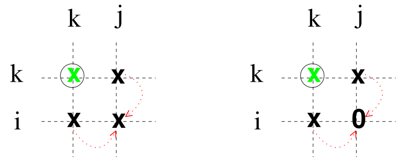
Gaussian elimination and sparsity

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

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

- 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

Gaussian elimination and sparsity

- Idem for Cholesky :
- For $i > k$ compute $l_{ik} = a_{ik}/\sqrt{a_{kk}}$ $(= a'_{ik})$,
- For $i > k, j > k, j \leq i$ (lower triang.)

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

Gaussian elimination and sparsity

- Interest of permuting a matrix:

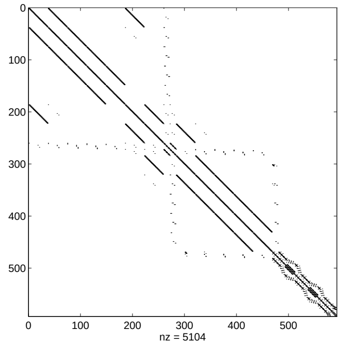
$$\begin{pmatrix} X & X & X & X & X \\ X & X & 0 & 0 & 0 \\ X & 0 & X & 0 & 0 \\ X & 0 & 0 & X & 0 \\ X & 0 & 0 & 0 & X \end{pmatrix} \quad 1 \leftrightarrow 5 \quad \begin{pmatrix} X & 0 & 0 & 0 & X \\ 0 & X & 0 & 0 & X \\ 0 & 0 & X & 0 & X \\ 0 & 0 & 0 & X & X \\ X & X & X & X & X \end{pmatrix}$$

- 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. NZ(LU factors)=58202

Original matrix



Factorized matrix

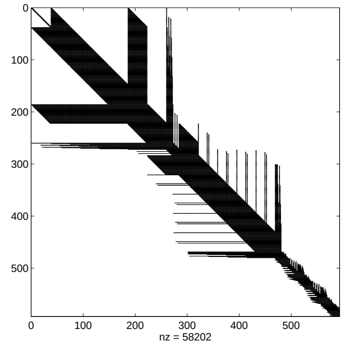
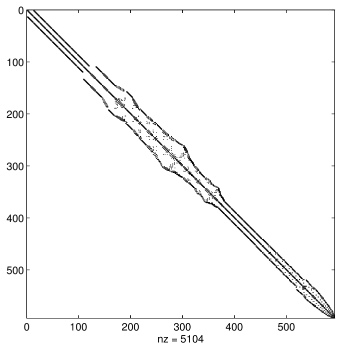


Illustration: Reverse Cuthill-McKee on matrix dwt_592.rua

NZ(LU factors)=16924

Permuted matrix (RCM)



Factorized permuted matrix

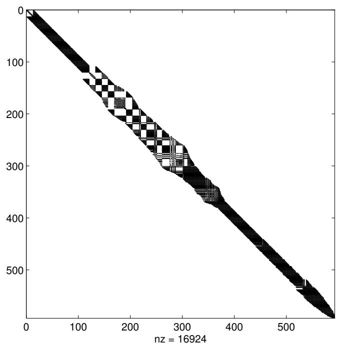


Table: Benefits of Sparsity on a matrix of order 2021 $\times$ 2021 with 7353 nonzeros. (Dongarra etal 91) .

Procedure	Total storage	Flops	Time (sec.) on CRAY J90
Full Syst.	4084 Kwords	5503 $\times 10^6$	34.5
Sparse Syst.	71 Kwords	1073$\times 10^6$	3.4
Sparse Syst. and reordering	14 Kwords	42$\times 10^3$	0.9

Sparse **LU** factorization : a (too) simple algorithm

Only non-zeros are stored and operated on

Algorithm 2 Simple sparse **LU** factorization

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

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 !!!

Control of numerical stability: numerical pivoting

- In dense linear algebra partial pivoting commonly used (at each step the largest entry in the column is selected).
- In sparse linear algebra, flexibility to preserve sparsity is offered :
  - Partial threshold pivoting : Eligible pivots are not too small with respect to the maximum in the column.
    Set of eligible pivots $= \{r \mid |a_{rk}^{(k)}| \geq u \times \max_i |a_{ik}^{(k)}|\}$, where $0 < u \leq 1$.
  - Then among eligible pivots select one preserving better sparsity.
  - u is called the threshold parameter ($u = 1 \rightarrow$ partial pivoting).
  - It restricts the maximum possible growth of: $a_{ij} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}}$ to $1 + \frac{1}{u}$ which is sufficient to the preserve numerical stability.
  - $u \approx 0.1$ is often chosen in practice.
- For symmetric indefinite problems 2×2 pivots (with threshold) is also used to preserve symmetry and sparsity.

Threshold pivoting and numerical accuracy

Table: Effect of variation in threshold parameter u on matrix 541×541 with 4285 nonzeros (Dongarra etal 91) .

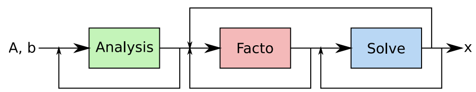
u	Nonzeros in **LU** factors	Error
1.0	16767	3×10^{-9}
0.25	14249	6×10^{-10}
0.1	13660	4×10^{-9}
0.01	15045	1×10^{-5}
10^{-4}	16198	1×10^{2}
10^{-10}	16553	3×10^{23}

Difficulty: numerical pivoting implies dynamic datastructures that can not be forecasted symbolically

Three-phase scheme to solve $Ax = b$

1. Analysis step
   - Preprocessing of **A** (symmetric/unsymmetric orderings, scalings)
   - Build the dependency graph (elimination tree, eDAG ...)
   - $\mathbf{A_{pre}} = \mathbf{P\, D_r\, A\, Q_c\, D_c\, P^T}$,
2. Factorization ($A_{pre} = \mathbf{LU}, \mathbf{LDL^T}, \mathbf{LL^T}, \mathbf{QR}$)

   Numerical pivoting
3. Solution based on factored matrices
   - triangular solves: $Ly = b_{pre}$, then $Ux_{pre} = y$
   - improvement of solution (iterative refinement), error analysis



Efficient implementation of sparse algorithms

- Indirect addressing is often used in sparse calculations: e.g. sparse SAXPY

```
do i = 1, m
   A( ind(i) ) = A( ind(i) ) + alpha * w( i )
enddo
```

- Even if manufacturers provide hardware for improving indirect addressing
  - It penalizes the performance
- Identify dense blocks or switch to dense calculations as soon as the matrix is not sparse enough

Effect of switch to dense calculations

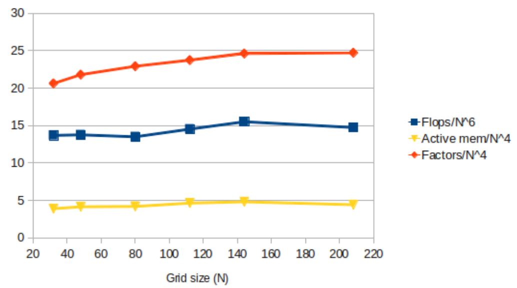
Matrix from 5-point discretization of the Laplacian on a 50×50 grid (Dongarra etal 91)

Density for switch to full code	Order of full submatrix	Millions of flops	Time (seconds)
No switch	0	7	21.8
1.00	74	7	21.4
0.80	190	8	15.0
0.60	235	11	12.5
0.40	305	21	9.0
0.20	422	50	5.5
0.10	531	100	3.7
0.005	1420	1908	6.1

Sparse structure should be exploited if density $< 10\%$.

Complexity of sparse direct methods

Regular problems (nested dissections)	2D $N \times N$ grid	3D $N \times N \times N$ grid
Nonzeros in original matrix	$\Theta(N^2)$	$\Theta(N^3)$
Nonzeros in factors	$\Theta(N^2 log\ n)$	$\Theta(N^4)$
Floating-point ops	$\Theta(N^3)$	$\Theta(N^6)$



3D example in earth science: acoustic wave propagation, 27-point finite difference grid

Current goal [Seiscope project: http://seiscope.oca.eu/]: LU on complete earth

Extrapolation on a $n = N^3 = 1000^3$ grid:
55 exaflops, 200 TBytes for factors, 40 TBytes of working memory!

Summary – sparse matrices main issues

- Widely used in engineering and industry (critical for performance of simulations)
- Irregular dynamic data structures of very large size (Tera/PetaBytes)
- Strong relations between sparse matrices and graphs
- Which graph for which matrix (symmetric, unsymmetric, triangular)?
- Efficient algorithms needed for:
  - Which graph for which algorithm ?
  - Reordering sparse matrices
  - Predict data structures of factor matrices
  - Model factorization and accomodate numerical issues

Outline

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 $1, 2, \ldots, n$.

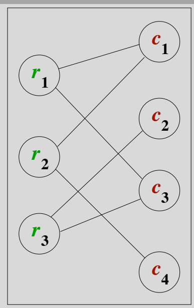
Matrices and graphs: Rectangular matrices

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

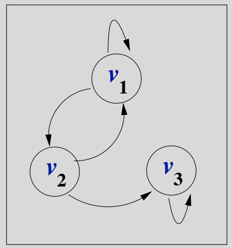
The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

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

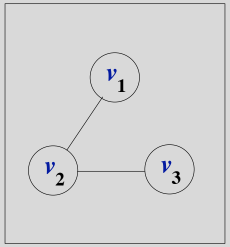
The rows/columns and nonzeros of a given sparse matrix correspond (with natural labelling) to the vertices and edges, respectively, of a graph.

Square symmetric pattern matrices

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

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.

Definitions: Trees and spanning trees

A tree is a connected, acyclic, undirected graph. If an undirected graph is acyclic but disconnected, then it is a forest.

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.

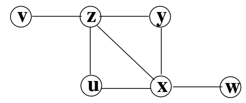
There is a unique path from the root r to every other vertex v. Any vertex y in that path is called an ancestor of v. If y is an ancestor of v, then v is a descendant of y.

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

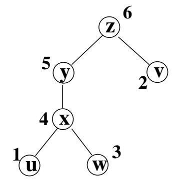
A spanning tree of a connected graph $G = (V, E)$ is a tree $T = (V, F)$, such that $F \subseteq E$.

Ordering of the vertices of a rooted tree

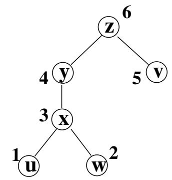
- A topological ordering of a rooted tree is an ordering that numbers children vertices before their parent.
- A postorder is a topological ordering which numbers the vertices in any subtree consecutively.



Connected graph G



Rooted spanning tree
with topological ordering



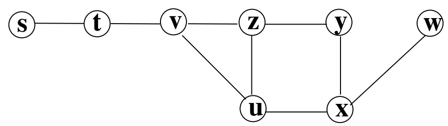
Rooted spanning tre
with postordering

How to explore a graph:

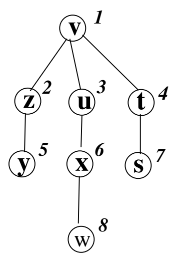
Two algorithms such that each edge is traversed exactly once in the forward and reverse direction and each vertex is visited. (Connected graphs are considered in the description of the algorithms.)

- **Depth first search** : Starting from a given vertex (*mark* it) follow a path (and mark each vertex in the path) until it 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.

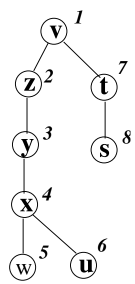
Illustration of DFS and BFS exploration



Breadth–first spanning tree (v)



Depth–first spanning tre

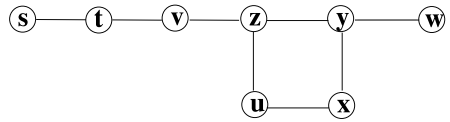


Properties of DFS and BFS exploration

- If the graph G is not connected the algorithm is restarted and in fact it detects the connected components of G.
- Each vertex is visited once.
- The visited edges of the DFS algorithm is a spanning forest of a general graph G so-called the **depth first spanning forest** of G.
- The visited edges of the BFS algorithm is a spanning forest of a general graph G so-called the **breadth first spanning forest** of G.

Graph shapes and peripheral vertices

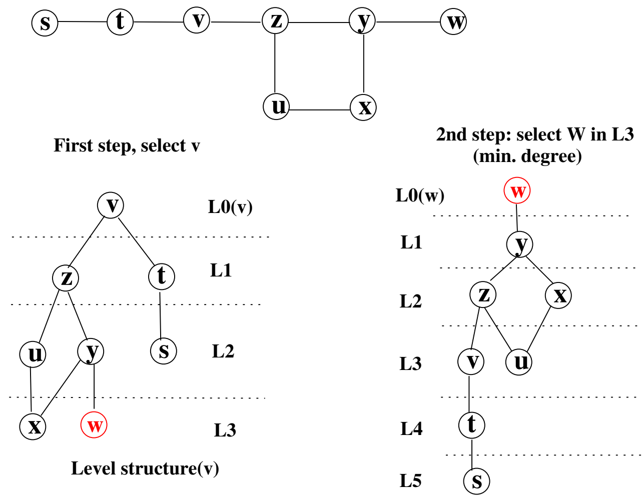
- The *distance* between two vertices of a graph is the size of the shortest path joining those vertices.
- The *eccentricity* $l(v) = max\ \{d(v, w) \mid w \in V\}$
- The *diameter* $\delta(G) = max\ \{l(v) \mid v \in V\}$
- v is a *peripheral* vertex if $l(v) = \delta(G)$
- Example of connected graph 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 $\mathcal{L}(v) = L_o(v), \dots, L_{l(v)}(v)$ such that $L_o(v) = v$ $L_i(v) = Adj(L_{i-1}(v)) \setminus L_{i-2}(v)$
- The eccentricity $l(v)$ is also-called the *length* of $\mathcal{L}(v)$. The *width* $w(v)$ of $\mathcal{L}(v)$ is defined as $w(v) = max\{|L_i(v)| \mid 0 \leq l(v)\}$
- The *aspect ratio* of $\mathcal{L}(v)$ is defined as $l(v)/w(v)$.
- 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



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. Let,

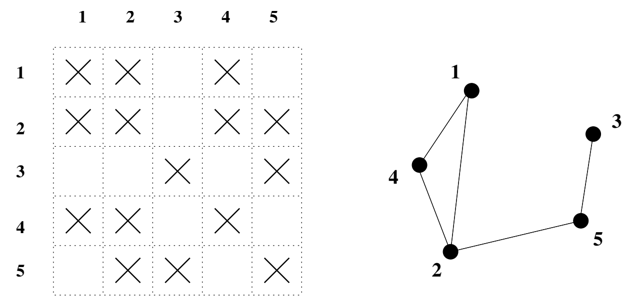
$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \end{array} \begin{array}{c} \begin{array}{ccc} 1 & 2 & 3 \end{array} \\ \begin{pmatrix} \times & \times & \\ \times & & \times \\ & & \times \end{pmatrix} \end{array}$$

and suppose we want to permute columns as $[2,1,3]$. Define $p_{2,1} = 1$, $p_{1,2} = 1$, $p_{3,3} = 1$, and $B = AP$ (if column j to be at position i, set $p_{ji} = 1$)

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

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**

Matching in bipartite graphs and permutations

A matching in a graph is a set of edges no two of which share a common vertex. We will be mostly dealing with matchings in bipartite graphs.

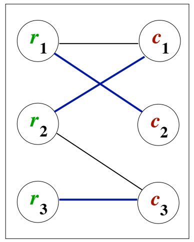
In matrix terms, a matching in the bipartite graph of a matrix corresponds to a set of nonzero entries no two of which are in the same row or column.

A vertex is said to be matched if there is an edge in the matching incident on the vertex, and to be unmatched otherwise. In a perfect matching, all vertices are matched.

The cardinality of a matching is the number of edges in it. A maximum cardinality matching or a maximum matching is a matching of maximum cardinality. Solvable in polynomial time.

Matching in bipartite graphs and permutations

Given a square matrix whose bipartite graph has a perfect matching, such a matching can be used to permute the matrix such that the matching entries are along the main diagonal.



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

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

Definitions: Reducibility

Reducible matrix: An $n \times n$ square matrix is reducible if there exists an $n \times n$ permutation matrix P such that

$$PAP^T = \begin{pmatrix} A_{11} & A_{12} \\ O & A_{22} \end{pmatrix},$$

where A_{11} is an $r \times r$ submatrix, A_{22} is an $(n-r) \times (n-r)$ submatrix, where $1 \le r < n$.

Irreducible matrix: There is no such a permutation matrix.

Theorem: An $n \times n$ square matrix is irreducible iff its directed graph is strongly connected.

Proof: Follows by definition.

Definitions: Hypergraphs

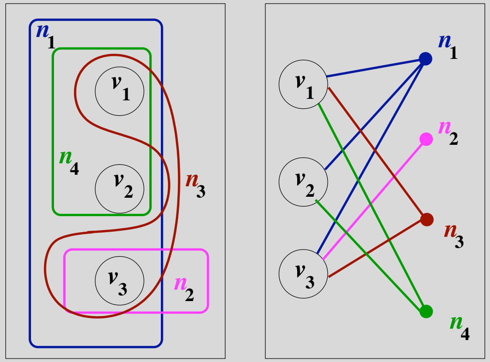
Hypergraph: A hypergraph $H = (V, N)$ consists of a finite set V called the vertex set and a set of non-empty subsets of vertices N called the hyperedge set or the net set. A generalization of graphs.

For a matrix A, define a hypergraph whose vertices correspond to the rows and whose nets correspond to the columns such that vertex v_i is in net n_j iff $a_{ij} \neq 0$ (the column-net model).

A sample matrix

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

The column-net hypergraph model



Outline

Reordering sparse matrices

- Fill-in and reordering
- The elimination graph model
- Fill-in characterization
- Fill-reducing heuristics
- Reduction to Block Triangular Form

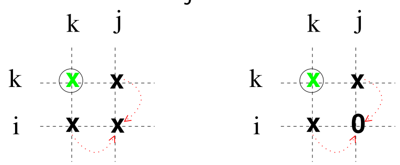
Fill-in and reordering

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

- For $i > k$ compute $l_{ik} = a_{ik}/a_{kk}$ $(= a'_{ik})$,
- For $i > k, j > k$

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

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



Interest of permuting a matrix:

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

Symmetric matrices and graphs

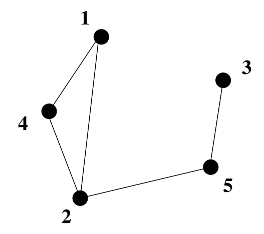
- Assumptions: **A** symmetric and pivots are chosen from the diagonal
- Structure of **A** symmetric represented by the graph $G = (V, E)$
  - Vertices are associated to columns: $V = \{1, \ldots, n\}$
  - Edges E are defined by: $(i, j) \in E \leftrightarrow a_{ij} \neq 0$
  - G undirected (symmetry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

Symmetric matrices and graphs

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

	1	2	3	4	5
1	×	×		×	
2	×	×		×	×
3			×		×
4	×	×		×	
5		×	×		×

Symmetric matrix



Corresponding graph

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

Exercice 1 (BFS traversal to order a matrix)

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

1. *Derive from BFS traversal of G an algorithm to reorder matrix **A***
2. *Explain the structural property of the permuted matrix?*
3. *Explain why such an ordering can help reducing fill-in during LU factorisation?*
4. *How can the proposed algorithm be used on an unsymmetric matrix?*

The elimination graph model for symmetric matrices

- Let **A** be a symmetric positive define matrix of order n
- The $\mathbf{LL}^T$ factorization can be described by the equation:

$$\mathbf{A} = \mathbf{A}_0 = \mathbf{H}_0 = \begin{pmatrix} d_1 & \mathbf{v}_1^T \\ \mathbf{v}_1 & \overline{\mathbf{H}_1} \end{pmatrix}$$

$$= \begin{pmatrix} \sqrt{d_1} & 0 \\ \frac{\mathbf{v}_1}{\sqrt{d_1}} & \mathbf{I}_{n-1} \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & \mathbf{H}_1 \end{pmatrix} \begin{pmatrix} \sqrt{d_1} & \frac{\mathbf{v}_1^T}{\sqrt{d_1}} \\ 0 & \mathbf{I}_{n-1} \end{pmatrix}$$

$= \mathbf{L}_1 \mathbf{A}_1 \mathbf{L}_1^T$, where

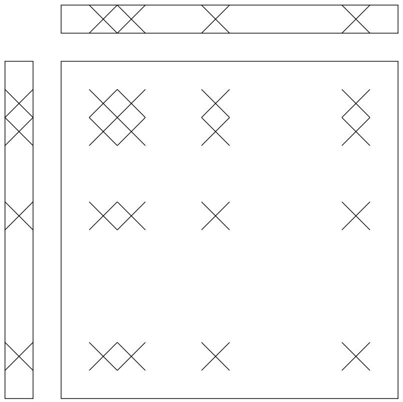
$$\mathbf{H}_1 = \overline{\mathbf{H}_1} - \frac{\mathbf{v}_1 \mathbf{v}_1^T}{d_1}$$

- The basic step is applied on $\mathbf{H}_1 \mathbf{H}_2 \cdots$ to obtain :

$$\mathbf{A} = (\mathbf{L}_1 \mathbf{L}_2 \cdots \mathbf{L}_{n-1})\, \mathbf{I}_n \left(\mathbf{L}_{n-1}^T \ldots \mathbf{L}_2^T \mathbf{L}_1^T\right) \quad = \quad \mathbf{LL}^T$$

The basic step: $\mathbf{H}_1 = \overline{\mathbf{H}_1} - \frac{\mathbf{v}_1 \mathbf{v}_1^T}{d_1}$

What is $\mathbf{v}_1 \mathbf{v}_1^T$ in terms of structure?



$\mathbf{v}_1$ is a column of $\mathbf{A}$, hence the neighbors of the corresponding vertex.

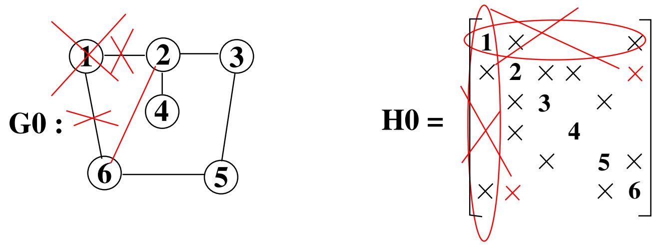
$\mathbf{v}_1 \mathbf{v}_1^T$ results in a dense sub-block in $\mathbf{H}_1$.

If any of the nonzeros in dense submatrix are not in $\mathbf{A}$, then we have fill-ins.

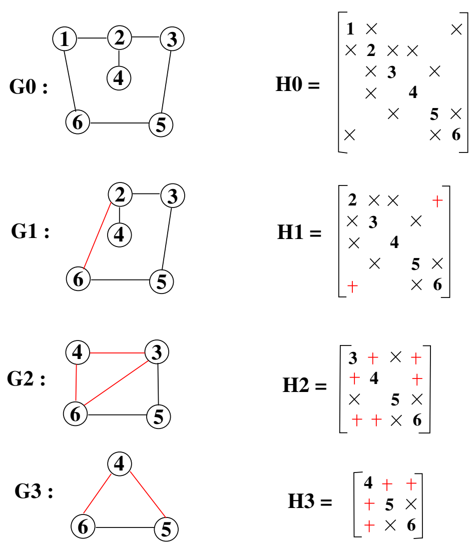
The elimination process in the graphs

$G_U(V, E) \leftarrow$ undirected graph of $\mathbf{A}$
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).

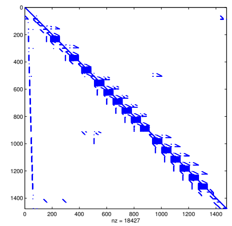


A sequence of elimination graphs

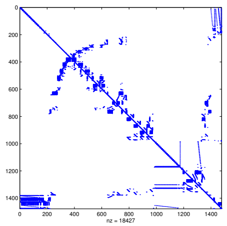


Fill-in and reordering

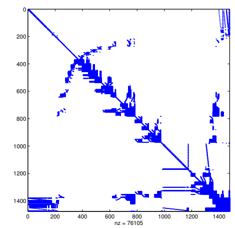
"Before permutation" (A''(LHR01))



Permuted matrix (A'(LHR01))



Factored matrix ($LU(A')$)

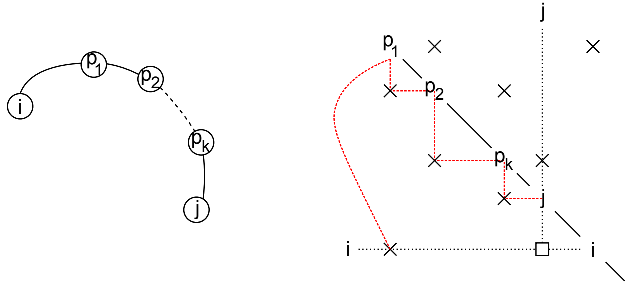


Fill-in characterization

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.

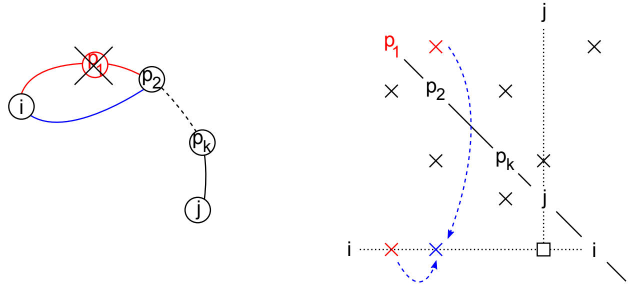


Fill-in characterization (proof intuition)

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.

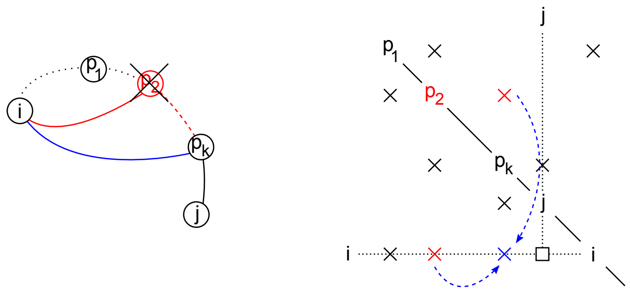


Fill-in characterization (proof intuition)

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.

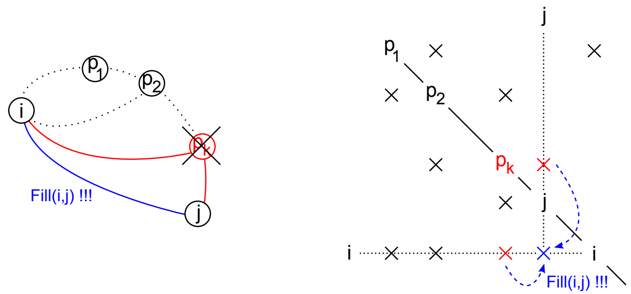


Fill-in characterization (proof intuition)

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

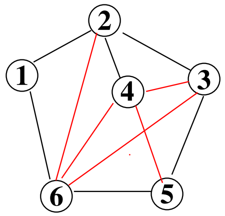
Fill path theorem, Rose, Tarjan, Leuker, 76

$l_{ij} \neq 0$ 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.

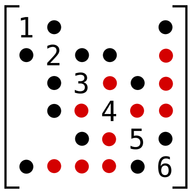


Introducing the filled graph $G^+(\mathbf{A})$

- Let $\mathbf{F} = \mathbf{L} + \mathbf{L}^T$ be the filled matrix, and $G(\mathbf{F})$ the *filled graph* of $\mathbf{A}$ denoted by $G^+(\mathbf{A})$.
- Lemma (Parter 1961) : $(v_i, v_j) \in G^+$ if and only if $(v_i, v_j) \in G$ or $\exists k < \min(i, j)$ such that $(v_i, v_k) \in G^+$ and $(v_k, v_j) \in G^+$.



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



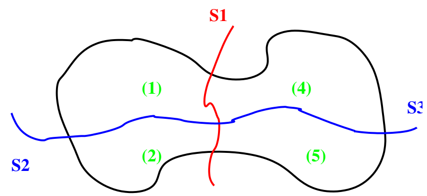
$F = L + L^T$

Fill-reducing heuristics

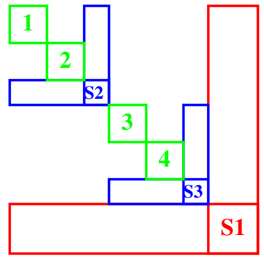
Three main classes of methods for minimizing fill-in during factorization

- 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

- 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).
- Hybrid approaches: Once the matrix is permuted to block structure, local heuristics are used within the blocks.

Local heuristics to reduce fill-in during factorization

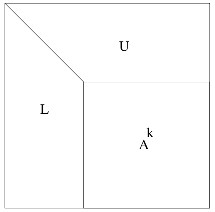
Let $G(A)$ be the graph associated to a matrix A that we want to order using local heuristics.
Let *Metric* such that $Metric(v_i) < Metric(v_j)$ implies v_i is a better than v_j

Generic algorithm
Loop until all nodes are selected
Step1: select current node p (so called pivot) with minimum metric value,
Step2: update elimination graph,
Step3: update $Metric(v_j)$ for all non-selected nodes v_j.

Step3 should only be applied to nodes for which the Metric value might have changed.

Reordering unsymmetric matrices: Markowitz criterion

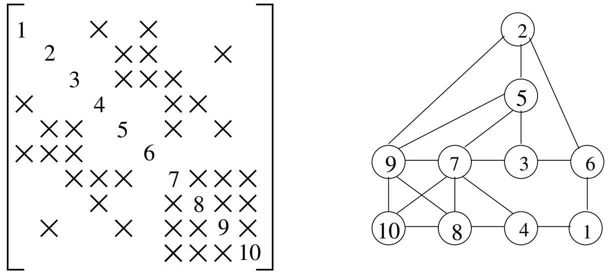
- At step k of Gaussian elimination:



  - 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 $(r_i^k - 1) \times (c_j^k - 1) \quad \forall i, j > k$
- $\boxed{\text{Minimum degree}}$: 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

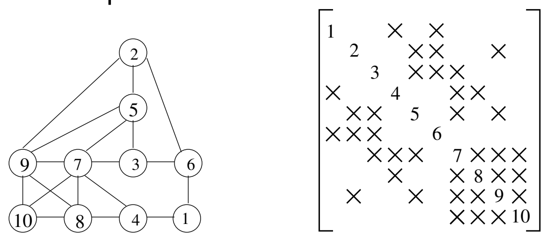
(b) Elimination graph

The node/variable selected is 1 of degree 2.

- **Notation for the elimination graph**
  - Let $G^k = (V^k, E^k)$, the graph built at step k.
  - G^k describes the structure of $\mathbf{A}_k$ after elimination of k pivots.
  - G^k is non-oriented ($\mathbf{A}_k$ is symmetric)
  - Fill-in in $\mathbf{A}_k \equiv$ adding edges in the graph.

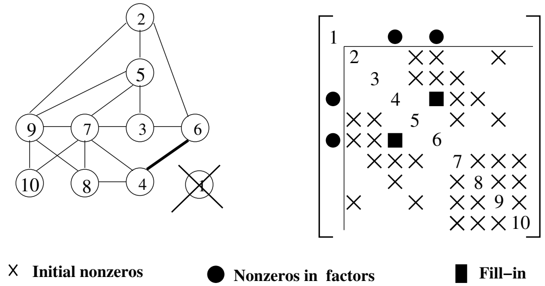
Illustration

Step 1: elimination of pivot 1



(a) Elimination graph

(b) Factors and active submatrix



Minimum degree algorithm applied to the graph:

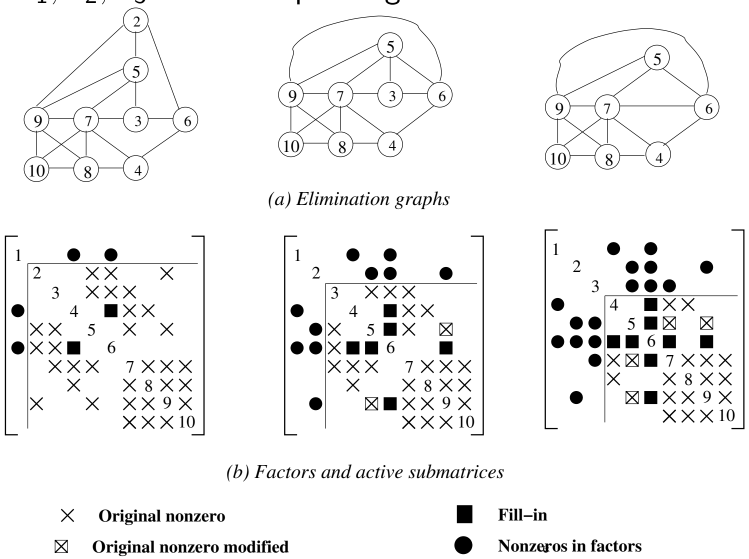
- Step k : Select the node with the smallest number of neighbors
- G^k is built from G^{k-1} by *suppressing the pivot* and *adding edges* corresponding to fill-in.

Minimum degree algorithm based on elimination graphs

$\forall i \in [1 \ldots n] \quad t_i \; = \; |\mathrm{Adj}_{G^0}(i)|$
For $k = 1$ **to** n **Do**
  $p = \min_{i \in V_{k-1}} (t_i)$
  For each $i \in \mathrm{Adj}_{G^{k-1}}(p)$ **Do**
    $\mathrm{Adj}_{G^k}(i) = (\mathrm{Adj}_{G^{k-1}}(i) \cup \mathrm{Adj}_{G^{k-1}}(p)) \setminus \{i, p\}$
    $t_i = |\mathrm{Adj}_{G^k}(i)|$
  EndFor
  $V^k = V^{k-1} \setminus p$
EndFor

Illustration (cont'd)

Graphs G_1, G_2, G_3 and corresponding reduced matrices.

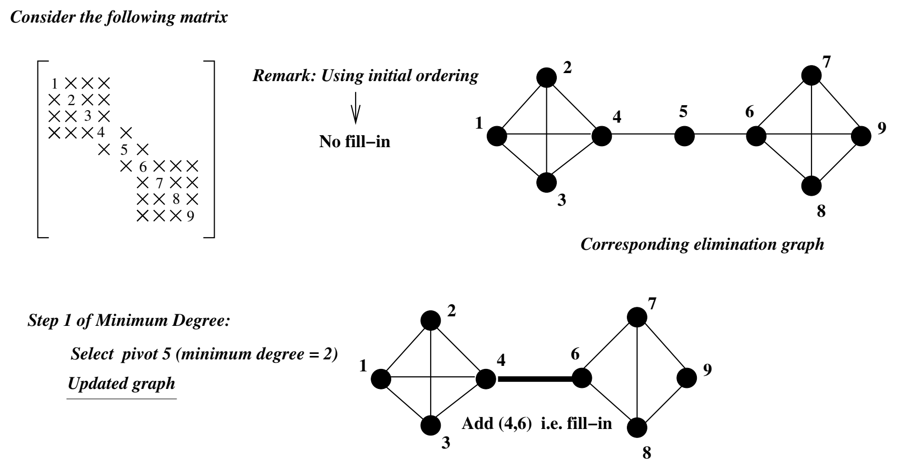


(a) Elimination graphs

(b) Factors and active submatrices

× Original nonzero ⊠ Original nonzero modified ■ Fill–in ● Nonzeros in factors

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



Reduce memory complexity

Decrease the size of working space
Using the elimination graph, working space is of order O(#nonzeros in factors).

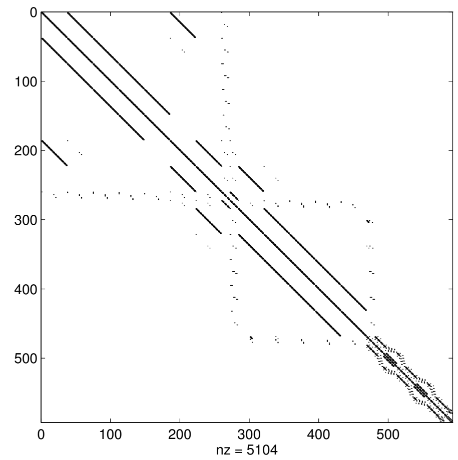
- **Property**: Let *pivot* be the pivot at step k

  If $i \in \mathrm{Adj}_{G^{k-1}}(pivot)$ Then $\mathrm{Adj}_{G^{k-1}}(pivot) \subset \mathrm{Adj}_{G^k}(i)$

- We can then use an implicit representation of fill-in by defining the notions of **element** (variable already eliminated) and **quotient graph**. A variable of the quotient graph is adjacent to variables and elements.
- We can show that $\forall k \in [1 \ldots N]$ Size of quotient graph $= O(G^0)$

Influence on the structure of factors

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



Mininimum fill-in heuristics

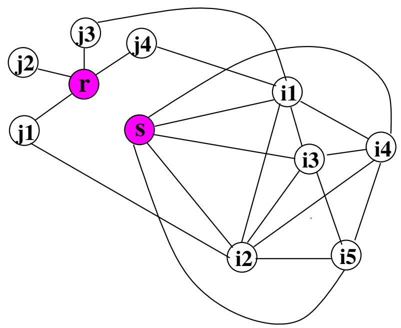
Recalling the generic algorithm
Let $G(\mathbf{A})$ be the graph associated to a matrix $\mathbf{A}$ that we want to order using local heuristics.
Let *Metric* be such that $Metric(v_i) < Metric(v_j) \equiv v_i$ is a better than v_j

Generic algorithm
Loop until all nodes are selected
Step1: Select current node p (so called pivot) with minimum metric value,
Step2: update elimination (or quotient) graph,
Step3: update $Metric(v_j)$ for all non-selected nodes v_j.

Step3 should only be applied to nodes for which the Metric value might have changed.

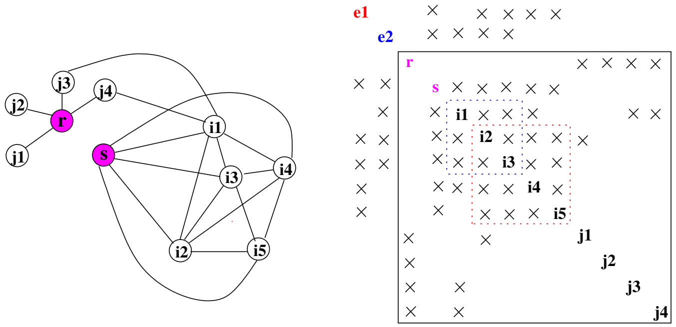
Minimum fill based algorithm

- $Metric(v_i)$ is the amount of fill-in that v_i would introduce if it were selected as a pivot.
- Illustration: r has a degree $d = 4$ and a fill-in metric of $d \times (d-1)/2 = 6$ whereas s has degree $d = 5$ but a fill-in metric of $d \times (d-1)/2 - 9 = 1$.



Minimum fill-in properties

- The situation typically occurs when $\{i_1, i_2, i_3\}$ and $\{i_2, i_3, i_4, i_5\}$ were adjacent to two already selected nodes (here e_2 and e_1)



e1 and e2 are previously selected nodes

- The elimination of a node v_k affects the degree of nodes adjacent to v_k. *The fill-in metric of $Adj(Adj(v_k))$ is also affected.*
- Illustration: selecting r affects the fill-in of i_1 (fill edge (j_3, j_4) should be deduced).

Reduction to Block Triangular Form (BTF)

- Suppose that there exists permutations matrices **P** and **Q** such that

$$\mathbf{PAQ} = \begin{pmatrix} \mathbf{B}_{11} & & & & & \\ \mathbf{B}_{21} & \mathbf{B}_{22} & & & & \\ \mathbf{B}_{31} & \mathbf{B}_{32} & \mathbf{B}_{33} & & & \\ \cdot & \cdot & \cdot & \cdot & & \\ \cdot & \cdot & \cdot & & \cdot & \\ \cdot & \cdot & \cdot & & & \cdot \\ \mathbf{B}_{N1} & \mathbf{B}_{N2} & \mathbf{B}_{N3} & \cdot & \cdot\ \cdot & \mathbf{B}_{NN} \end{pmatrix}$$

- If $N > 1$ **A** is said to be *reducible* (irreducible otherwise)
- Each $\mathbf{B}_{ii}$ is supposed to be irreducible (otherwise finer decomposition is possible)
- Advantage: to solve $\mathbf{Ax} = \mathbf{b}$ only $\mathbf{B}_{ii}$ need be factored $\mathbf{B}_{ii}x_i = (\mathbf{Pb})_i - \sum_{j=1}^{i-1} \mathbf{B}_{ij}\mathbf{y}_j,\ i = 1, \ldots, N$ with $\mathbf{y} = \mathbf{Q}^\top\mathbf{x}$

A two stage approach to compute the reduction

- Stage (1): compute a (column) permutation matrix **Q** such that $\mathbf{AQ}_1$ has non zeros on the diagonal (find a maximum transversal of **A**), then
- Stage(2): compute a (symmetric) permutation matrix **P** such that $\mathbf{PAQ}_1\mathbf{P}^t$ is BTF.

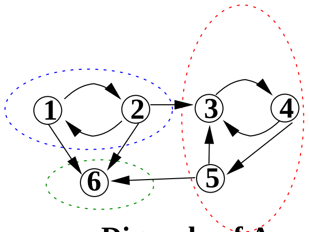
The diagonal blocks of the BTF are uniquely defined. Techniques exits to directly compute the BTF form. They do not present advantage over this two stage approach.

Main components of the algorithm

- Objective : assume $\mathbf{AQ}_1$ has non zeros on the diagonal and compute **P** such that $\mathbf{PAQ}_1\mathbf{P}^t$ is BTF.
- Use of digraph (directed graph) associated to the matrix with self loops (diagonal entries) not needed.
- Symmetric permutations on digraph $\equiv$ relabelling nodes of the graph.
- If there is no closed path through all nodes in the digraph then the digraph can be subdivided in two parts.
- *Strong components* of a graph are the set of nodes belonging to a closed path.
- The strong components of the graph are the diagonal blocks $\mathbf{B}_{ii}$ of the BTF format.

Main components of the algorithm

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



Digraph of A.

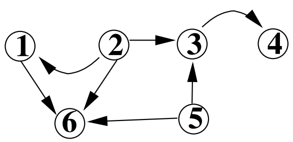
$$
\mathbf{PAPt} = \begin{bmatrix}
\boxed{\mathbf{6}} & & & & & \\
 & \mathbf{3} & \times & & & \\
 & \times & \mathbf{4} & \times & & \\
\times & \times & & \mathbf{5} & & \\
\times & & & & \mathbf{1} & \times \\
\times & \times & & & \times & \mathbf{2}
\end{bmatrix}
$$

BTF of A

Preamble : Finding the triangular form of a matrix

- <u>Observations</u>: a triangular matrix has a BTF form with blocks of size 1
  (BTF will be considered as generalization of the triangular form where each diagonal entry is now a strong component).
  Note that if the matrix has triangular form the digraph has no cycle (i.e. *acyclic*).
- Sargent and Westerberg Algorithm :
  1. Select a starting node and trace a path until finding a node with no paths leave.
  2. Number the last node first, eliminate it (and all edges pointing to it).
  3. Continue from previous node on the path (or choose a new starting node).

Preamble : Finding the triangular form of a matrix



Digraph of A.

				$\boxed{4}$					
Path		$\boxed{6}$	3	3	$\boxed{3}$			$\boxed{1}$	
	5	5	5	5	5	$\boxed{5}$	2	2	$\boxed{2}$
Steps	1	2	3	4	5	6	7	8	9

$$
\mathbf{A} = \begin{bmatrix}
\mathbf{1} & & & & & \times \\
\times & \mathbf{2} & \times & & & \times \\
 & & \mathbf{3} & \times & & \\
 & & & \mathbf{4} & & \\
 & & \times & & \mathbf{5} & \times \\
 & & & & & \mathbf{6}
\end{bmatrix}
\qquad
\mathbf{PAPt} = \begin{bmatrix}
\mathbf{6} & & & & & \\
 & \mathbf{4} & & & & \\
 & \times & \mathbf{3} & & & \\
\times & & \times & \mathbf{5} & & \\
\times & & & & \mathbf{1} & \\
\times & & \times & & \times & \mathbf{2}
\end{bmatrix}
$$

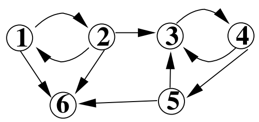
A permuted to triangular form

(not unique)

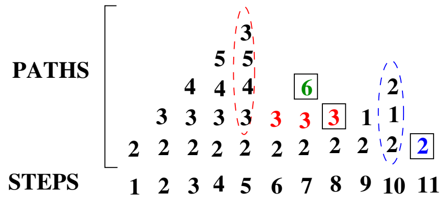
Generalizing Sargent and Westerberg Algorithm to compute BTF

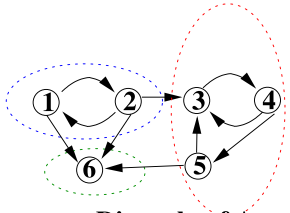
A *composite node* is a set nodes belonging to a closed path.
Strat from any node and follow a path until
(1) a closed path is found:
collapse all nodes in closed path into a composite node
the path continue from the composite node.
(2) reaching a node or composite node with no path leave:
the node or composite node is numbered next.

Generalizing Sargent and Westerberg Algorithm to compute BTF



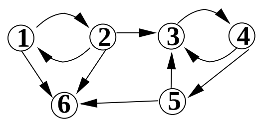
Digraph of A.



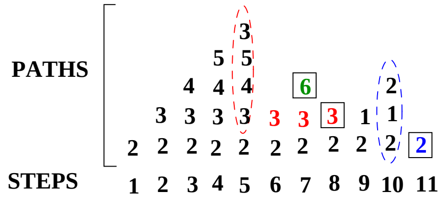


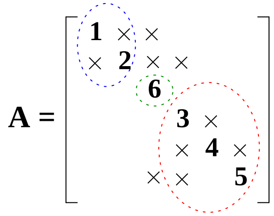
Digraph of A.

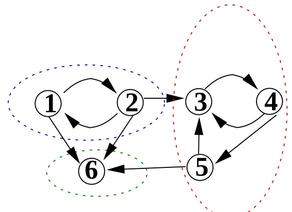
Generalizing Sargent and Westerberg Algorithm to compute BTF



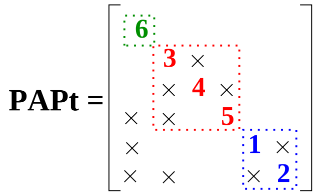
Digraph of A.







Digraph of A.



BTF of A

Concluding remarks

Summary:

Fill-in has been characterized;
Ordering strategies and associated graphs have been introduced to reduce fill-in or to permute to target structure (block tridiagonal, recursive block-bordered, block triangular form) Influence on performance (sequential and parallel)

Next step:

How to model sparse factorization, predict data structures to run the numerical factorization ?

Outline

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.

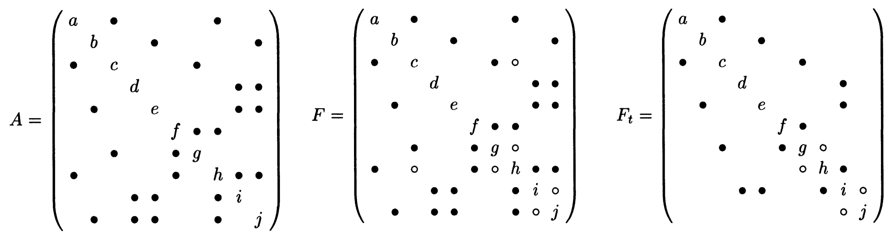
Let A be an $n \times n$ symmetric positive-definite and irreducible matrix, $A = LL^T$ its Cholesky factorization, and $G^+(A)$ its filled graph (graph of $F = L + L^T$).

A first definition

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

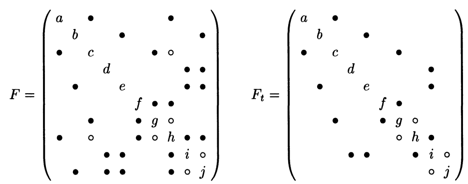
For each column $j < n$ of L, remove all the nonzeros in the column j except the first one below the diagonal.

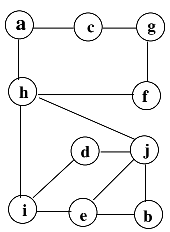
Let L_t denote the remaining structure and consider the matrix $F_t = L_t + L_t^T$. The graph $G(F_t)$ is a tree called the elimination tree.



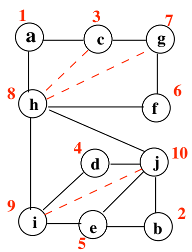
A first definition

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

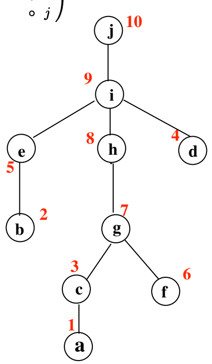




G(A)



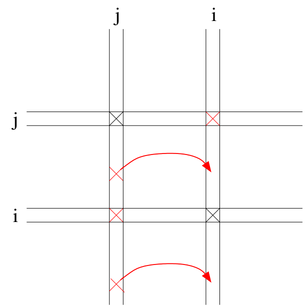
$G^+(A) = G(F)$



T(A)

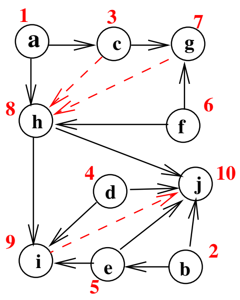
A second definition: Represents column dependencies

- 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)

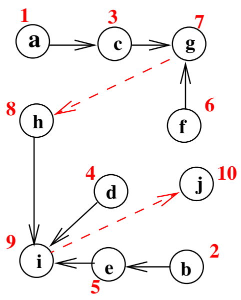


- The transitive reduction of the directed filled graph gives the elimination tree structure. Remove a directed edge (j, i) if there is a path of length greater than one from j to i.

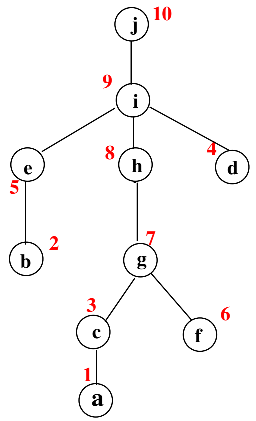
Directed filled graph and its transitive reduction



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

Outline

Efficiency of the solution phase

Sparsity in the right hand side and/or solution

Exploit sparsity of the right-hand-side/solution

Applications

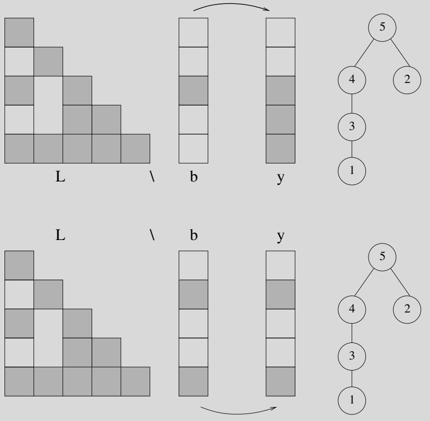
- Highly reducible matrices and/or sparse right-hand-sides (linear programming, seismic processing)
- Null-space basis computation
- Partial computation of A^{-1}
  - Computing variances of the unknowns of a data fitting problem = computing the diagonal of a so-called variance-covariance matrix.
  - Computing short-circuit currents = computing blocks of a so-called impedance matrix.
  - Approximation of the condition number of a SPD matrix.

Core idea

An efficient algorithm has to take advantage of the sparsity of A and of both the right-hand sides and the solution.

Exploit sparsity in RHS : an quick insight of main properties

solve $y \leftarrow L \setminus b$

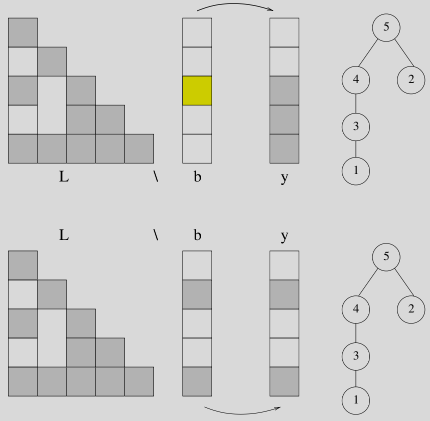


- In all application cases, only part of factors/operations needs to be loaded/performed
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded
  - Characterize operations to be performed

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

Exploit sparsity in RHS : an quick insight of main properties

solve $y \leftarrow L \setminus b$

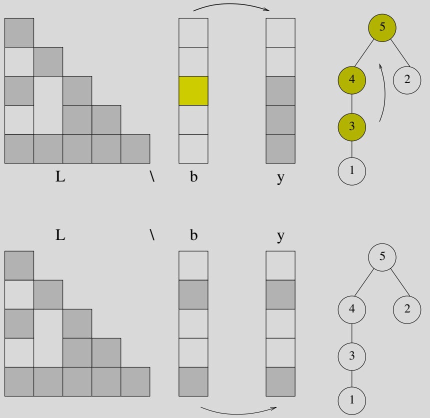


- In all application cases, only part of factors/operations needs to be loaded/performed
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded
  - Characterize operations to be performed

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

Exploit sparsity in RHS : an quick insight of main properties

solve $y \leftarrow L \setminus b$

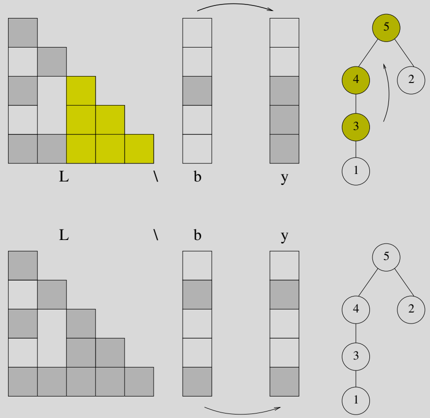


- In all application cases, only part of factors/operations needs to be loaded/performed
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded
  - Characterize operations to be performed

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

Exploit sparsity in RHS : an quick insight of main properties

solve $y \leftarrow L \setminus b$

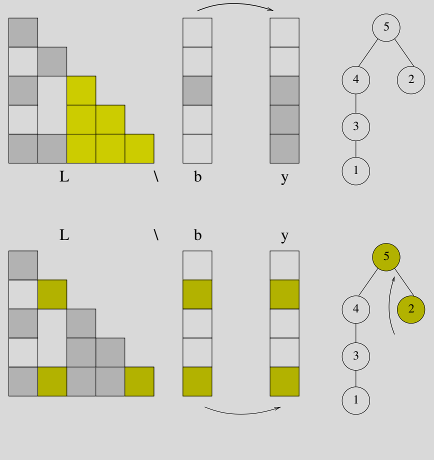


- In all application cases, only part of factors/operations needs to be loaded/performed
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded
  - Characterize operations to be performed

(1) Predicting structure of the solution vector, *Gilbert-Liu*, '93

Exploit sparsity in RHS : an quick insight of main properties

solve $y \leftarrow L \setminus b$



- In all application cases, only part of factors/operations needs to be loaded/performed
- Objectives with sparse RHS
  - Efficient use of the RHS sparsity
  - Characterize LU factors to be loaded
  - Characterize operations to be performed

(1) Predicting structure of the solution vector, *Gilbert-Liu, '93*

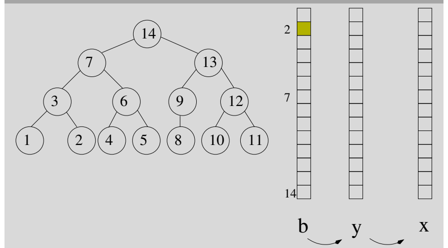
Application : elements in A^{-1}

$AA^{-1} = I$, specific entry: $a_{ij}^{-1} = (A^{-1}e_j)_i$,
$A^{-1}e_j$ – column j of A^{-1}

Theorem: structure of x (based on Gilbert and Liu '93)

For any matrix A such that $A = LU$, the structure of the solution (x) is given by the set of nodes reachable from nodes associated with right-hand side entries by paths in the e-tree.

compute some elements in A^{-1}



Which factors needed to compute a_{82}^{-1}?
$a_{82}^{-1} = (U^{-1}(L^{-1}e_2))_8$

We have to load :
L factors associated with nodes $2, 3, 7, 14$
and U factors associated with nodes $14, 13, 9, 8$

Note:
A part of the tree is concerned

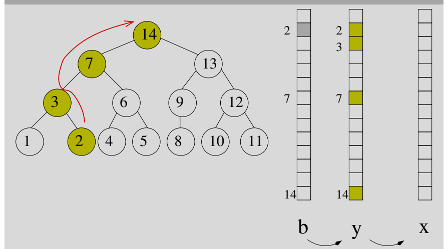
Application : elements in A^{-1}

$AA^{-1} = I$, specific entry: $a_{ij}^{-1} = (A^{-1}e_j)_i$,
$A^{-1}e_j$ – column j of A^{-1}

Theorem: structure of x (based on Gilbert and Liu '93)

For any matrix A such that $A = LU$, the structure of the solution (x) is given by the set of nodes reachable from nodes associated with right-hand side entries by paths in the e-tree.

compute some elements in A^{-1}



Which factors needed to compute a_{82}^{-1}?
$a_{82}^{-1} = (U^{-1}(L^{-1}e_2))_8$

We have to load :
L factors associated with nodes $2, 3, 7, 14$
and U factors associated with nodes $14, 13, 9, 8$

Note:
A part of the tree is concerned

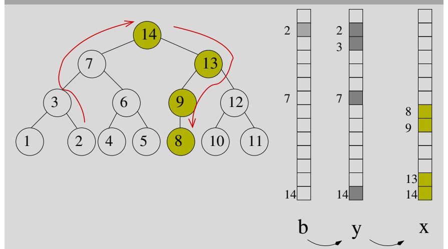
Application : elements in A^{-1}

$AA^{-1} = I$, specific entry: $a_{ij}^{-1} = (A^{-1}e_j)_i$,
$A^{-1}e_j$ – column j of A^{-1}

Theorem: structure of x (based on Gilbert and Liu '93)

For any matrix A such that $A = LU$, the structure of the solution (x) is given by the set of nodes reachable from nodes associated with right-hand side entries by paths in the e-tree.

compute some elements in A^{-1}



Which factors needed to compute a_{82}^{-1}?
$a_{82}^{-1} = (U^{-1}(L^{-1}e_2))_8$

We have to load :
L factors associated with nodes $2, 3, 7, 14$
and U factors associated with nodes $14, 13, 9, 8$

Note:
A part of the tree is concerned

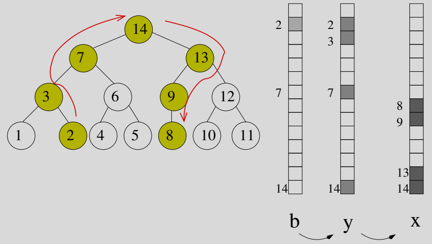
Application : elements in A^{-1}

$AA^{-1} = I$,   specific entry: $a_{ij}^{-1} = (A^{-1}e_j)_i$,
$A^{-1}e_j$ – column j of A^{-1}

Theorem: structure of x (based on Gilbert and Liu '93)

For any matrix A such that $A = LU$, the structure of the solution (x) is given by the set of nodes reachable from nodes associated with right-hand side entries by paths in the e-tree.

compute some elements in A^{-1}



Which factors needed to compute a_{82}^{-1}?
$a_{82}^{-1} = (U^{-1}(L^{-1}e_2))_8$

We have to load :
L factors associated with nodes $2, 3, 7, 14$
and U factors associated with nodes $14, 13, 9, 8$

Note:
A part of the tree is concerned

Entries of the inverse: a single one

Notation for later use

$P(i)$: denotes the nodes in the unique path from the node i to the root node r (including i and r).

$P(S)$: denotes $\bigcup_{s \in S} P(s)$ for a set of nodes S.

Use the elimination tree

For each requested (diagonal) entry a_{ii}^{-1},
(1) visit the nodes of the elimination tree from the node i to the root: at each node access necessary parts of **L**,

(2) visit the nodes from the root to the node i again; this time access necessary parts of **U**.

Experiments: interest of exploiting sparsity

Implementation

These ideas have been implemented in MUMPS solver during Tz. Slavova's PhD.

Experiments: computation of the diagonal of the inverse of matrices from data fitting in Astrophysics (CESR, Toulouse)

Matrix size	Time (s)	
	No ES	ES
46,799	6,944	472
72,358	27,728	408
148,286	>24h	1,391

Interest

Exploiting sparsity of the right-hand sides reduces the number of accesses to the factors (in-core: number of flops, out-of-core: accesses to hard disks).

Outline

Outline

1. Introduction
2. Elimination tree and multifrontal method
3. Postorderings, equivalent orderings and memory usage
4. Concluding remarks

Recalling the Gaussian elimination

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

- For $i > k$ compute $l_{ik} = a_{ik}/a_{kk}$ $(= a'_{ik})$,
- For $i > k, j > k$ such that a_{ik} and a_{kj} are nonzeros

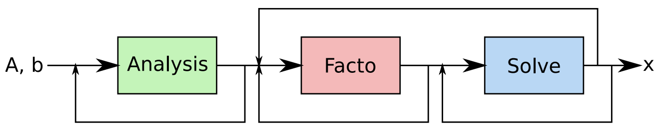
$$a'_{ij} = a_{ij} - \frac{a_{ik} \times a_{kj}}{a_{kk}}$$

- If $a_{ik} \neq 0$ et $a_{kj} \neq 0$ then $a'_{ij} \neq 0$
- If a_{ij} was zero $\rightarrow$ its non-zero value must be stored
- Orderings (minimum degree, Cuthill-McKee, ND) limit fill-in, the number of operations and modify the tasks graph

Three-phase scheme to solve $Ax = b$

1. Analysis step
   - Preprocessing of **A** (symmetric/unsymmetric orderings, scalings)
   - Build the dependency graph (elimination tree, eDAG ...)
2. Factorization ($A =$ **LU**, **LDL**^{T}, **LL**^{T}, **QR**)

   Numerical pivoting
3. Solution based on factored matrices
   - triangular solves: $Ly = b$, then $Ux = y$
   - improvement of solution (iterative refinement), error analysis



Elimination tree and Multifrontal approach

We recall that:

The elimination tree is the dependency graph of the factorization.

Building the elimination tree (small matrices)

- Permute matrix (to reduce fill-in) **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

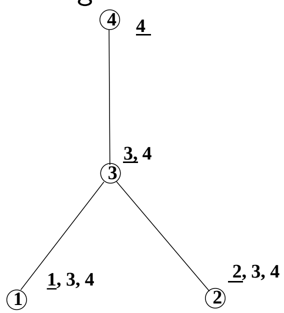
$\rightarrow$ (multifrontal) Each node k of the tree corresponds to the partial factorization of a frontal matrix whose row and column structure is that of column k of $\mathbf{A}_F$.

Illustration of multifrontal factorization

We assume pivots are chosen down the diagonal in order.

$$\begin{bmatrix} \times & & \times & \times \\ & \times & \times & \times \\ \times & \times & \times & F \\ \times & \times & F & \times \end{bmatrix}$$

Filled matrix



Elimination graph

Treatment at each node:

- Assembly of the frontal matrix using the contributions from the sons.
- Gaussian elimination on the frontal matrix

- Elimination of variable 1 (a_{11} pivot)
  - Assembly of the frontal matrix

$$\begin{array}{c|ccc} & 1 & 3 & 4 \\ \hline 1 & x & x & x \\ 3 & x & & \\ 4 & x & & \end{array}$$

  - Contributions : $a_{ij} = \frac{-(a_{i1} \times a_{1j})}{a_{11}} \quad i > 1, j > 1$ on a_{33}, a_{44}, a_{34} and a_{43} :

$$a_{33}^{(1)} = -\frac{(a_{31} \times a_{13})}{a_{11}} \quad a_{34}^{(1)} = -\frac{(a_{31} \times a_{14})}{a_{11}}$$
$$a_{43}^{(1)} = -\frac{(a_{41} \times a_{13})}{a_{11}} \quad a_{44}^{(1)} = -\frac{(a_{41} \times a_{14})}{a_{11}}$$

Terms $-\frac{a_{i1} \times a_{1j}}{a_{11}}$ of the contribution matrix are stored for later updates.

- Elimination of variable 2 (a_{22} pivot)
  - Assembly of frontal matrix: update of elements of pivot row and column using contributions from previous updates (none here)

$$\begin{array}{c|ccc} & 2 & 3 & 4 \\ \hline 2 & x & x & x \\ 3 & x & & \\ 4 & x & & \end{array}$$

  - Contributions on a_{33}, a_{34}, a_{43}, and a_{44}.

$$\begin{aligned} a_{33}^{(2)} &= -\frac{(a_{32} \times a_{23})}{a_{22}} \\ a_{34}^{(2)} &= -\frac{(a_{32} \times a_{24})}{a_{22}} \\ a_{43}^{(2)} &= -\frac{(a_{42} \times a_{23})}{a_{22}} \\ a_{44}^{(2)} &= -\frac{(a_{42} \times a_{24})}{a_{22}} \end{aligned}$$

- Elimination of variable 3.
  - Assembly of frontal matrix using the previous contribution matrices $\begin{pmatrix} a_{33}^{(1)} & a_{34}^{(1)} \\ a_{43}^{(1)} & a_{44}^{(1)} \end{pmatrix}$ and $\begin{pmatrix} a_{33}^{(2)} & a_{34}^{(2)} \\ a_{43}^{(2)} & a_{44}^{(2)} \end{pmatrix}$:

$$\begin{aligned} a'_{33} &= a_{33} + a_{33}^{(1)} + a_{33}^{(2)} \\ a'_{34} &= a_{34} + a_{34}^{(1)} + a_{34}^{(2)}, \quad (a_{34} = 0) \\ a'_{43} &= a_{43} + a_{43}^{(1)} + a_{43}^{(2)}, \quad (a_{43} = 0) \\ a'_{44} &= a_{44}^{(1)} + a_{44}^{(2)} \end{aligned}$$

  - Contribution on variable 4: $a_{44}^{(3)} = a'_{44} - \frac{(a'_{43} \times a_{34})}{a_{33}}$

$$\begin{array}{c|cc} & 3 & 4 \\ 3 & x & x \\ 4 & x & \end{array}$$

- Contribution on a_{44} : $a_{44}^{(3)} = a'_{44} - \frac{(a'_{43} \times a'_{34})}{a'_{33}}$
  Note that a_{44} is partially summed since it still does not include the entry from the initial matrix
- Elimination of variable 4
  - Frontal involves only a_{44} : $a_{44} = a_{44} + a_{44}^{(3)}$

$$\begin{array}{c|c} & 4 \\ 4 & x \end{array}$$

Supernodes/supervariables

Definition 1

A supernode (or supervariable) is a set of contiguous columns in the factors $\mathbf{L}$ *that share essentially the same sparsity structure. Columns* $i_1, i_2, \ldots, i_p$ *in the filled graph form a supernode:* $|L_{i_{k+1}}| = |L_{i_k}| - 1$.

- All algorithms (ordering, symbolic factor., factor., solve) generalize to blocked versions.
- Use of efficient matrix-matrix kernels (improve cache usage).
- Same concept as *supervariables* for elimination tree/minimum degree ordering.
- Supernodes and pivoting: pivoting inside a supernode does not increase fill-in.

Amalgamation

- GOAL
  - Exploit a more regular structure in the original matrix
  - Decrease the amount of indirect addressing
  - Increase the size of frontal matrices
- HOW?
  - Relax the number of nonzeros of the matrix
  - Amalgamation of nodes of the elimination tree

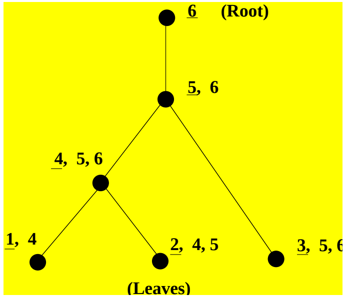
- CONSEQUENCES?
  - Increase in the total amount of flops
  - But decrease of indirect addressing
  - And increase in performance
- Amalgamation of supernodes (same lower diagonal structure) is without fill-in

Illustration of amalgamation

Original Matrix

	1	2	3	4	5	6
1	X			X		
2		X		X	X	
3			X		X	X
4	X	X		X	F3	X
5		X	X	F3	X	F4
6			X	X	F4	X

Elimination tree

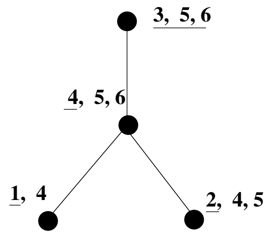


Structure of node i = frontal matrix noted $\underline{\mathbf{i}}, i_1, i_2 \ldots i_f$

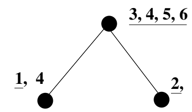
Illustration of amalgamation

Amalgamation

(WITHOUT fill-in)



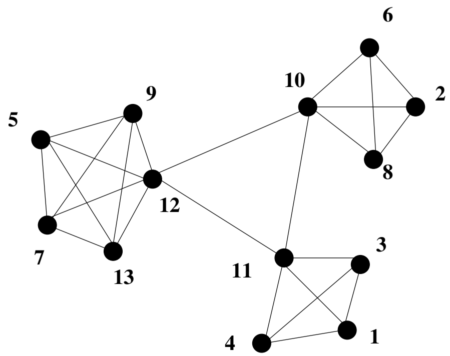
(WITH fill-in)



fill-in : (3,4) and (4,3)

Amalgamation and Supervariables

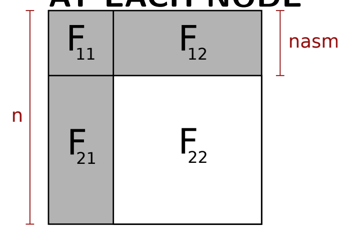
Amalgamation of supervariables does not cause fill-in
Initial Graph:



Reordering: $1, 3, 4,\ 2, 6, 8,\ 10, 11,\ 5, 7, 9, 12, 13$
Supervariables: $\{1, 3, 4\}$; $\{2, 6, 8\}$; $\{10, 11\}$; $\{5, 7, 9, 12, 13\}$

Supervariables and multifrontal method

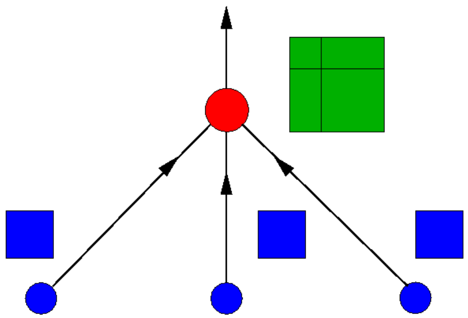
AT EACH NODE



$$F_{22} \leftarrow F_{22} \ - \ F_{12}^T F_{11}^{-1} F_{12}$$

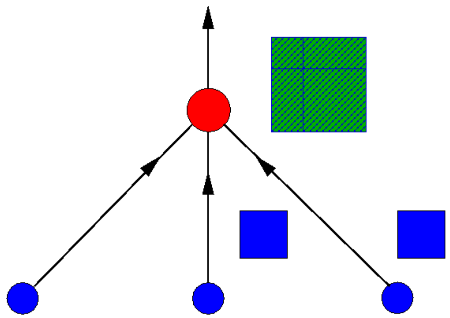
Pivot can ONLY be chosen from F_{11} block since F_{22} is **NOT** fully summed

Multifrontal method



From children to parent

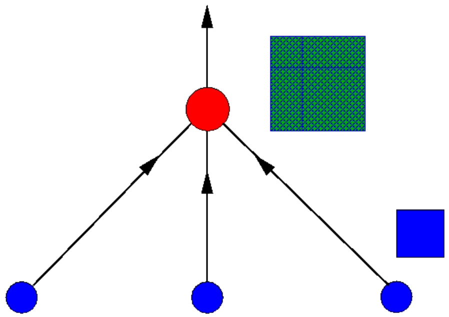
Multifrontal method



From children to parent

- ASSEMBLY: Scatter-add operations (indirect addressing)

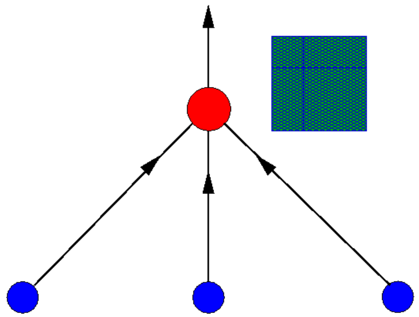
Multifrontal method



From children to parent

- ASSEMBLY: Scatter-add operations (indirect addressing)

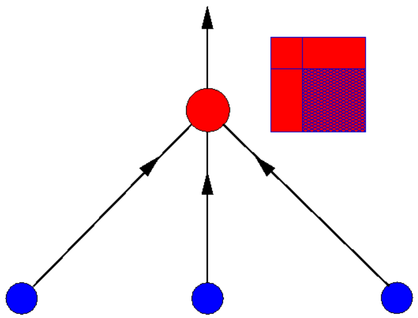
Multifrontal method



From children to parent

- ASSEMBLY: Scatter-add operations (indirect addressing)

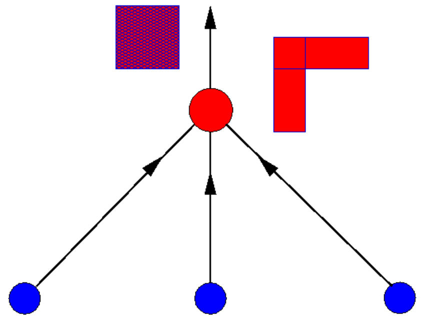
Multifrontal method



From children to parent

- ASSEMBLY: Scatter-add operations (indirect addressing)
- ELIMINATION: Dense partial Gaussian elimination, Level 3 BLAS (TRSM, GEMM)

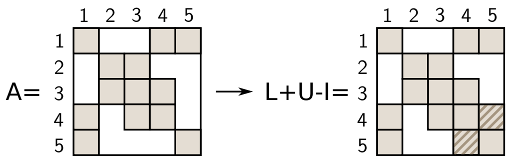
Multifrontal method



From children to parent

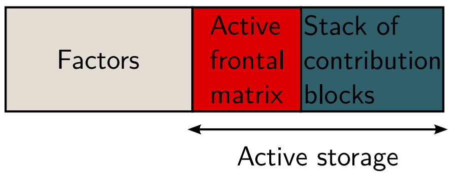
- ASSEMBLY: Scatter-add operations (indirect addressing)
- ELIMINATION: Dense partial Gaussian elimination, Level 3 BLAS (TRSM, GEMM)
- CONTRIBUTION to parent

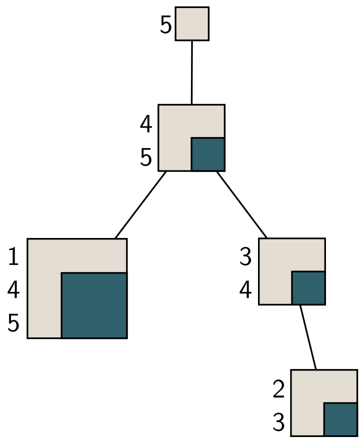
The multifrontal method [Duff & Reid '83]



Storage is divided into two parts:

- Factors
- Active memory





Elimination tree

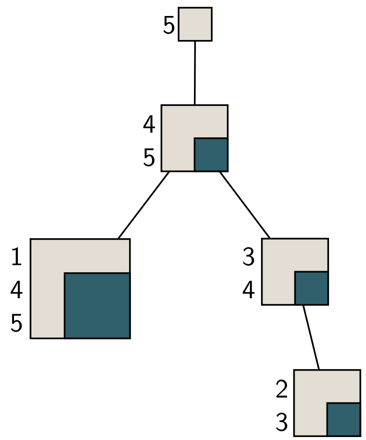
The multifrontal method [Duff & Reid '83]

- 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%

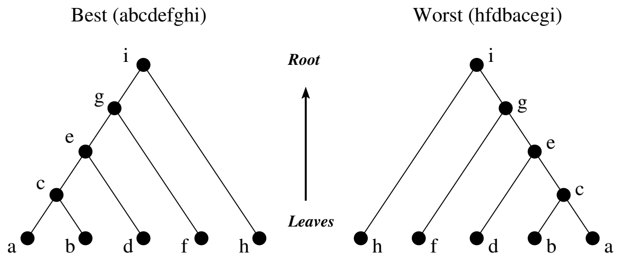
256 processors: 59%



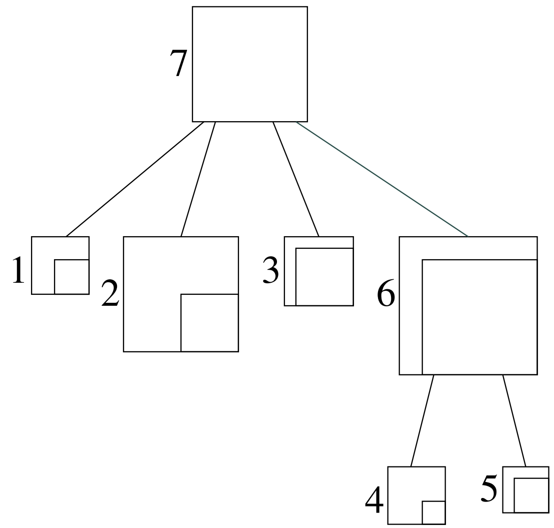
Elimination tree

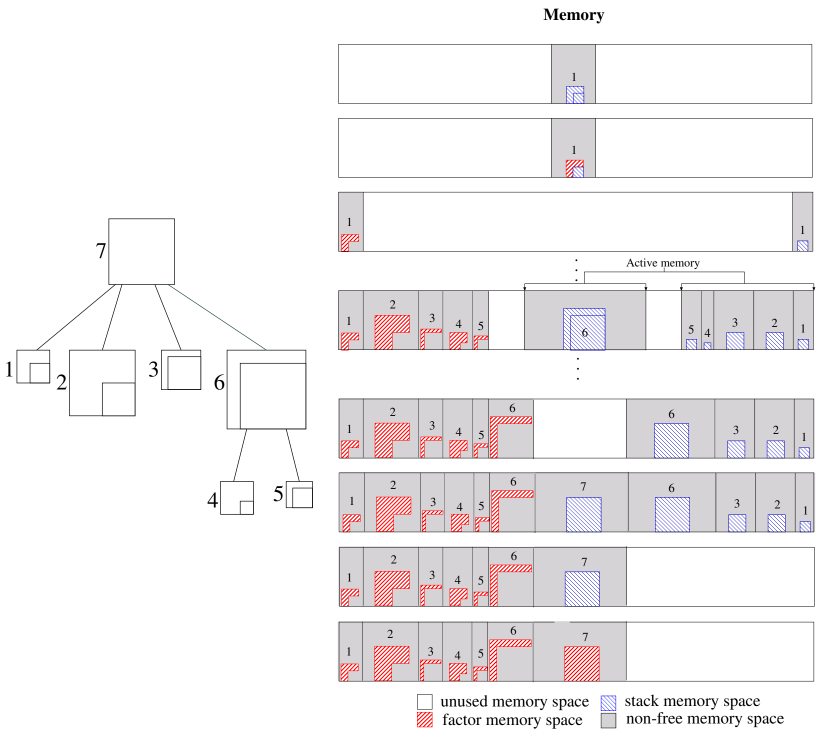
Postorderings and memory usage

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

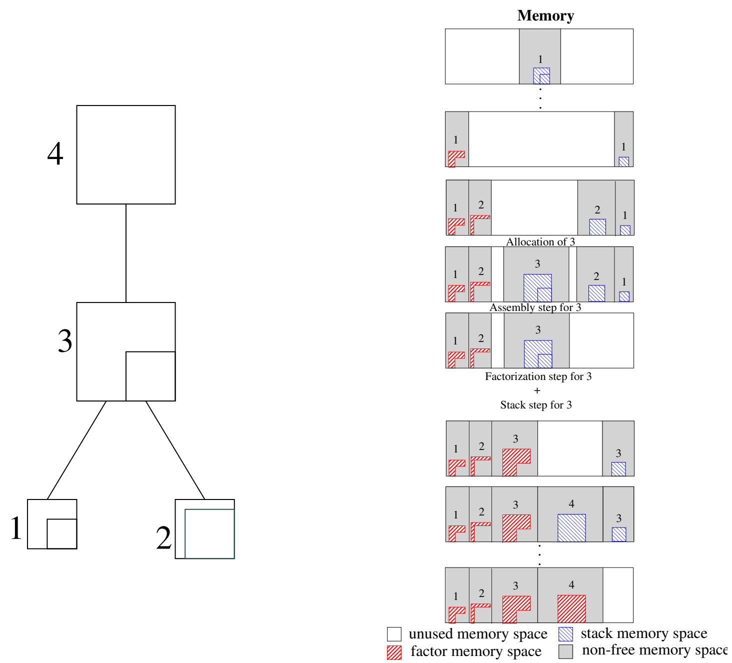


Example 1: Processing a wide tree



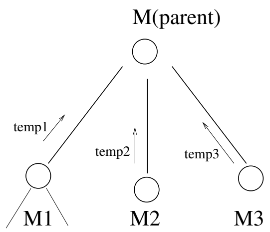


Example 2: Processing a deep tree



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.



$$M_{parent} = \max\left(\max_{j=1}^{nbchildren}\left(M_j + \sum_{k=1}^{j-1} temp_k\right),\ m_{parent} + \sum_{j=1}^{nbchildren} temp_j \right) \qquad (1)$$

Objective: order the children to minimize M_{parent}

Memory-minimizing schedules

Theorem 1

[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 1

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)

Optimal tree reordering

Objective: Minimize peak of stack memory

Tree_Reorder (T):
  Begin
    for all i in the set of root nodes **do**
      Process_Node(i);
    end for
  End
Process_Node(i):
  if i is a leaf **then**
    $M_i = m_i$
  else
    for $j = 1$ to *nbchildren* **do**
      Process_Node(j^{th} child);
    end for
    Reorder the children of i in decreasing order of $(M_j - temp_j)$;
    Compute M_{parent} at node i using Formula (1);
  end if

Equivalent orderings of symmetric matrices

Let $\mathbf{F}$ be the *filled matrix* of a symmetric matrix $\mathbf{A}$ (that is, $\mathbf{F} = \mathbf{L} + \mathbf{L}^t$, where $\mathbf{A} = \mathbf{L}\mathbf{L}^T$)
$G^+(\mathbf{A}) = G(\mathbf{F})$ is the associated *filled graph*.

Definition 2 (Equivalent orderings)

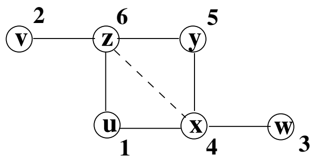
$\mathbf{P}$ *and* $\mathbf{Q}$ *are said to be equivalent orderings iff* $G^+(\mathbf{PAP}^T) = G^+(\mathbf{QAQ}^T)$
By extension, a permutation $\mathbf{P}$ *is said to be an equivalent ordering of a matrix* $\mathbf{A}$ *iff* $G^+(\mathbf{PAP}^T) = G^+(\mathbf{A})$

It can be shown that an equivalent reordering also preserves the amount of arithmetic operations for sparse Cholesky factorization.

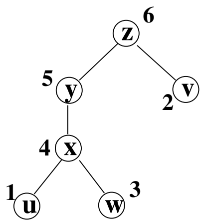
Relation with elimination trees

- Let **A** be a reordered matrix, and $G^+(\mathbf{A})$ be its filled graph
- In the elimination tree, any topological tree traversal (that processes children before parents) corresponds to an equivalent ordering **P** of **A** and the elimination tree of $\mathbf{PAP}^\top$ is identical to that of **A**.

$$\mathbf{A} = \begin{bmatrix} \mathbf{u} & & & \times & & \times \\ & \mathbf{v} & & & & \times \\ & & \mathbf{w} & \times & & \\ \times & & \times & \mathbf{x} & \times & \mathbf{F} \\ & & & \times & \mathbf{y} & \times \\ \times & \times & & \mathbf{F} & \times & \mathbf{z} \end{bmatrix}$$



Graph G^+(A)



Elimination Tree of A
with topological ordering

Tree rotations (Liu, 1988)

Definition 3

An ordering that does not introduce any fill is referred to as a perfect elimination ordering (or PEO in short)

Natural ordering is a PEO of the filled matrix **F**.

Theorem 2

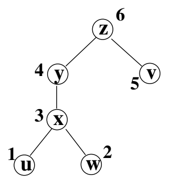
For any node x of $G^+(\mathbf{A}) = G(\mathbf{F})$, there exists a PEO on $G(\mathbf{F})$ such that x is numbered last.

- Essence of tree rotations :
  - Nodes in the clique of x in **F** are numbered last
  - Relative ordering of other nodes is preserved.

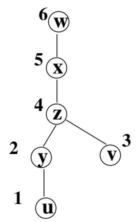
Example of equivalent orderings

On the right-hand side tree rotation applied on w:
(clique of w is $\{w, x\}$ and for other nodes relative ordering w.r.t. tree on the left is preserved).

$$\mathbf{F} = \begin{bmatrix} \mathbf{u} & & \times & & & \times \\ & \mathbf{w} & \times & & & \\ \times & \times & \mathbf{x} & \times & & \mathbf{F} \\ & & \times & \mathbf{y} & & \times \\ & & & & \mathbf{v} & \times \\ \times & & \mathbf{F} & \times & \times & \mathbf{z} \end{bmatrix} \qquad \begin{bmatrix} \mathbf{u} & & & \times & \times & \\ & \mathbf{y} & & \times & \times & \\ & & \mathbf{v} & \times & & \\ \times & \times & \times & \mathbf{z} & \mathbf{F} & \\ \times & \times & & \mathbf{F} & \mathbf{x} & \times \\ & & & & \times & \mathbf{w} \end{bmatrix}$$



with a postordering



Remark: Tree rotations can help reducing the temporary memory usage!

Some (shared memory) sparse direct codes

Code	Technique	Scope	Availability (www.)
BCSLIB	Multifrontal	SYM/UNS	Boeing → Access Analytics
HSL MA87	Supernodal	SPD	cse.clrc.ac.uk/Activity/HSL
MA41	Multifrontal	UNS	cse.clrc.ac.uk/Activity/HSL
MA49	Multifr. QR	RECT	cse.clrc.ac.uk/Activity/HSL
PanelLLT	Left-looking	SPD	Ng
PARDISO	Left-right	SYM/UNS	Schenk
PSL†	Left-looking	SPD/UNS	SGI product
qr_mumps	Multifr. QR	RECT	http://buttari.perso.enseeiht.fr/qr_mumps/
SuperLU_MT	Left-looking	UNS	nersc.gov/∼xiaoye/SuperLU
SuiteSparseQR	Multifr. QR	RECT	cise.ufl.edu/research/sparse/SPQR
TAUCS	Left/Multifr.	SYM/UNS	tau.ac.il/∼stoledo/taucs
WSMP†	Multifrontal	SYM/UNS	IBM product

† Only object code is available.

Code	Technique	Scope	Availability (www.)
DSCPACK	Multifr./Fan-in	SPD	cse.psu.edu/∼raghavan/Dscpack
MUMPS	Multifrontal	SYM/UNS	`graal.ens-lyon.fr/MUMPS`
			`mumps.enseeiht.fr`
PaStiX	Fan-in	SPD	labri.fr/perso/ramet/pastix
PSPASES	Multifrontal	SPD	cs.umn.edu/∼mjoshi/pspases
SPOOLES	Fan-in	SYM/UNS	netlib.org/linalg/spooles
SuperLU	Fan-out	UNS	nersc.gov/∼xiaoye/SuperLU
S+	Fan-out†	UNS	cs.ucsb.edu/research/S+
WSMP †	Multifrontal	SYM	IBM product

‡ Only object code is available.

- **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

Patrick R. Amestoy, Iain S. Duff, Jean-Yves L'Excellent, and Xiaoye S. Li.
Impact of the implementation of MPI point-to-point communications on the performance of two general sparse solvers.
Parallel Computing, 29(7):833–847, 2003.

Message Passing Interface Forum.
http://www-unix.mcs.anl.gov/mpi/index.html.

A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam.
PVM 3 User's Guide and Reference Manual.
Technical Report ORNL/TM-12187, Engineering Physics and Mathematics Division, Oak Ridge National Laboratory, Tennessee, 1993.

Abdou Guermouche and Jean-Yves L'Excellent.
Memory-based scheduling for a parallel multifrontal solver.

In *18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, page 71a (10 pages), 2004.

Abdou Guermouche, Jean-Yves L'Excellent, and G. Utard.
Analysis and improvements of the memory usage of a multifrontal solver.
Research report RR-4829, INRIA, 2003.
Also LIP report RR2003-08.

A. Gupta, G. Karypis, and V. Kumar.
Highly scalable parallel algorithms for sparse matrix factorization.
IEEE Trans. Parallel and Distributed Systems, 8(5):502–520, 1997.

J. H. Hennessy and D. A. Patterson.
Computer Architecture: a Quantitative Approach.
Morgan Kaufmann Publishers, Inc, San Francisco, California, 1996.
Second Edition.

G. Karypis and V. Kumar.

METIS – *Unstructured Graph Partitioning and Sparse Matrix Ordering System – Version 2.0*.
University of Minnesota, June 1995.

G. Padiou.
Systèmes opératoires, cours et notes de cours, 2ième et 3ième année informatique et mathématiques apppliquées, 2007.

M. Snir, S. W. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra.
MPI: The Complete Reference.
The MIT Press, Cambridge, Massachusetts, 1996.

R. Clint Whaley, Antoine Petitet, and Jack J. Dongarra.
Automated empirical optimization of software and the ATLAS project.
Parallel Computing, 27(1–2):3–35, 2001.
Also available as University of Tennessee LAPACK Working Note #147, UT-CS-00-448, 2000 (`www.netlib.org/lapack/lawns/lawn147.ps`).