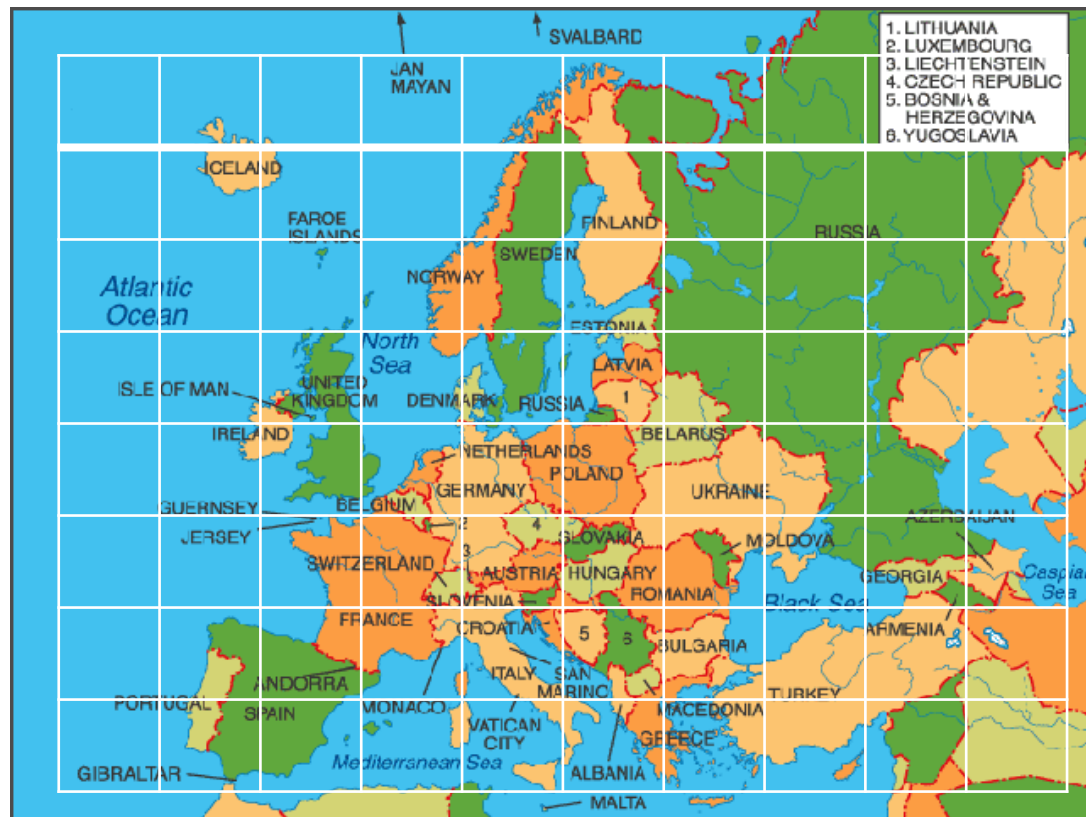


(Parallel) Sparse Matrix Computations

Sparse Matrices arise in


- Simulation of Physical/Chemical Phenomena
 - Modeled through particles/molecules/point clouds
- (Spatial) Database Applications
- Graph Computations
- Combinatorial Optimization

Example: Finite Differences




In case of a 5x5 grid this leads to 25 grid points and the following sparse matrix:

Number of grid points in the x direction

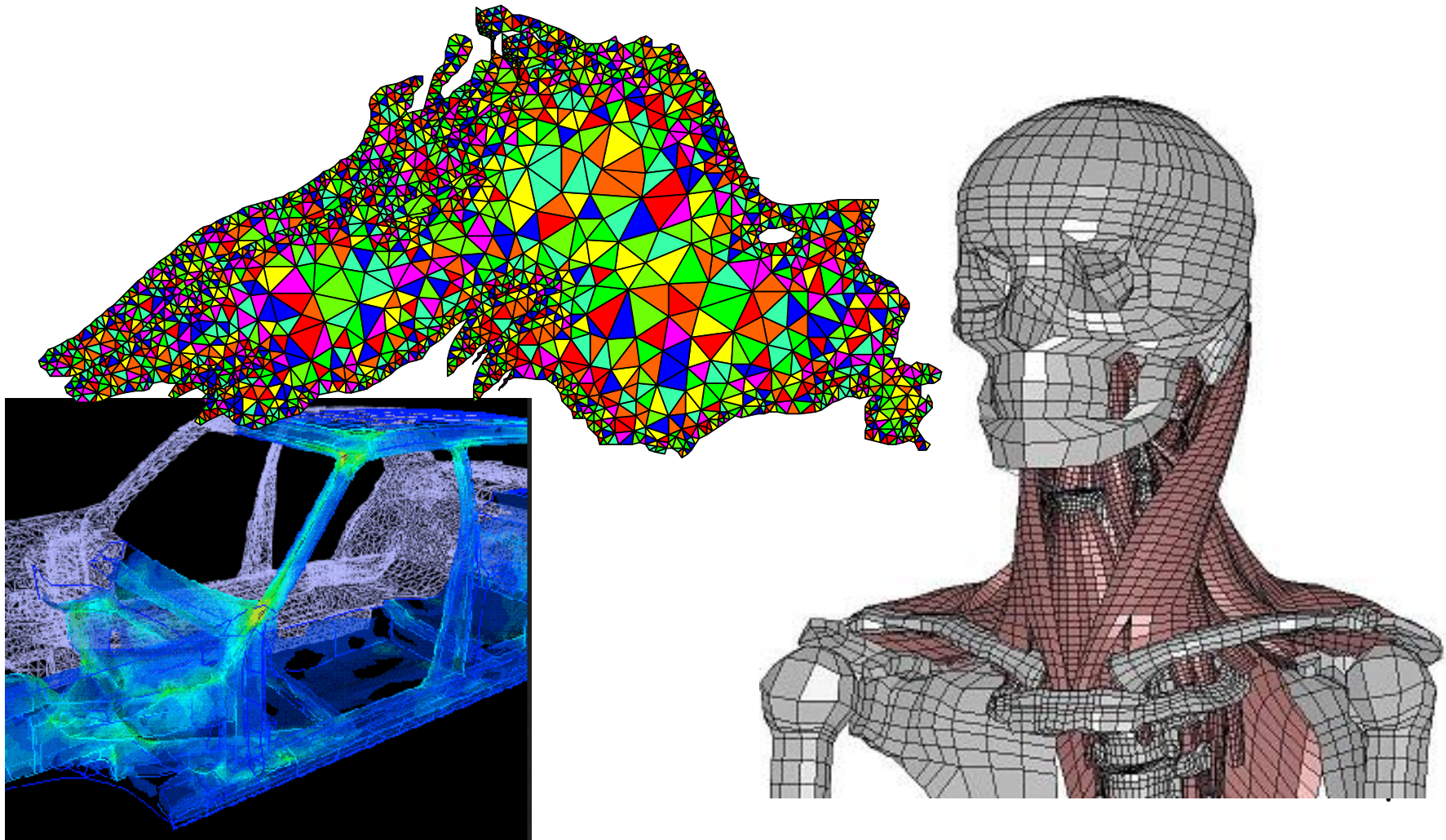


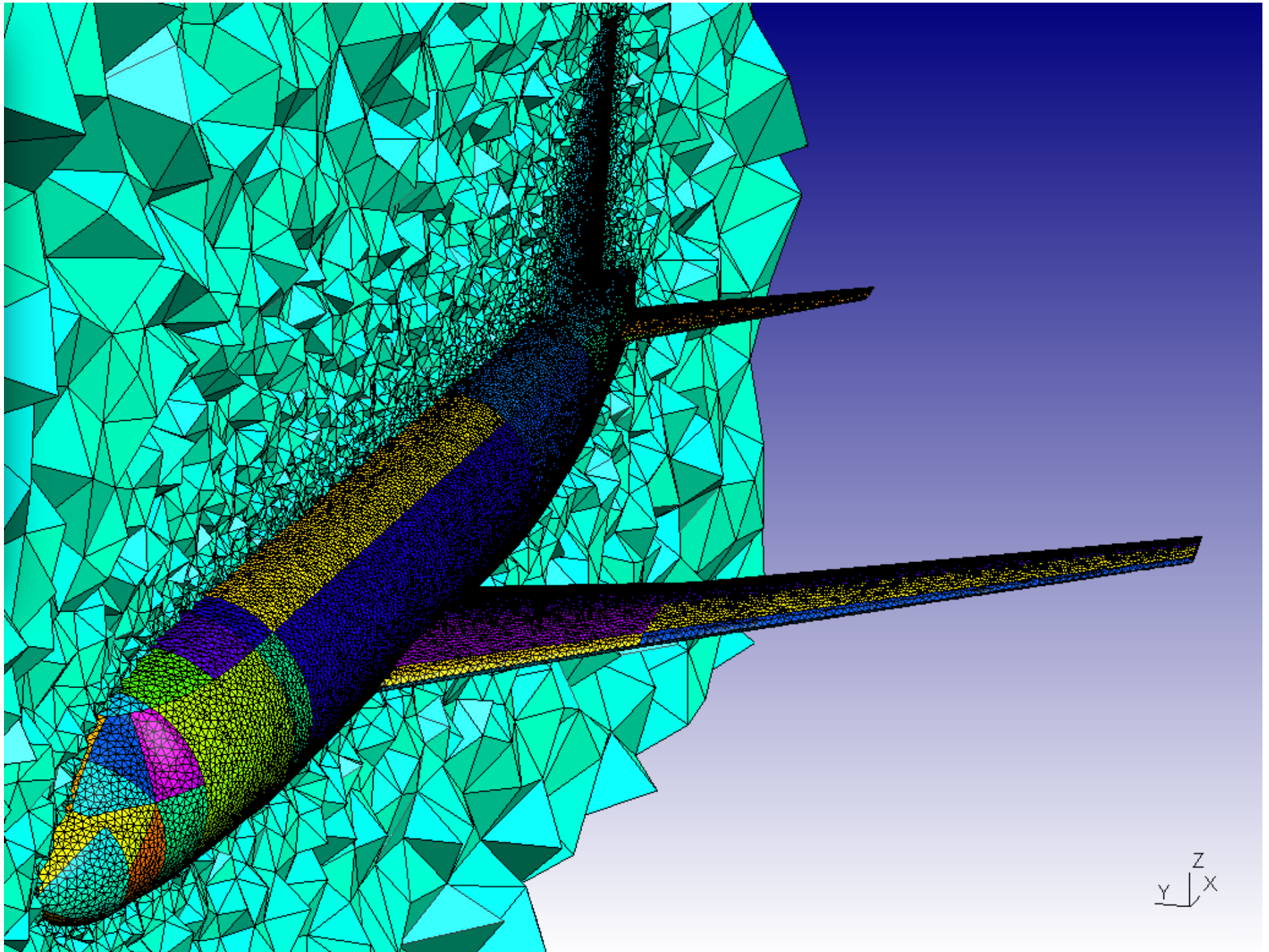
$$A = \begin{pmatrix} x & x & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x & x & x & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & x & x & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & x & x & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ x & 0 & 0 & x & x & x & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & x & 0 & 0 & x & x & x & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & x & 0 & 0 & x & x & x & 0 & 0 & x & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & x & 0 & 0 & x & x & x & 0 & 0 & x & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x & 0 & 0 & x & x & x & 0 & 0 & x & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & x & 0 & 0 & x & x & x & 0 & 0 & x & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & x & 0 & 0 & x & x & x & 0 & 0 & x & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & 0 & 0 & x & x & x & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & 0 & 0 & x & x & x & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & 0 & 0 & x & x & x \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & x & 0 & 0 & x & x \end{pmatrix}$$

Number of grid points in the y direction

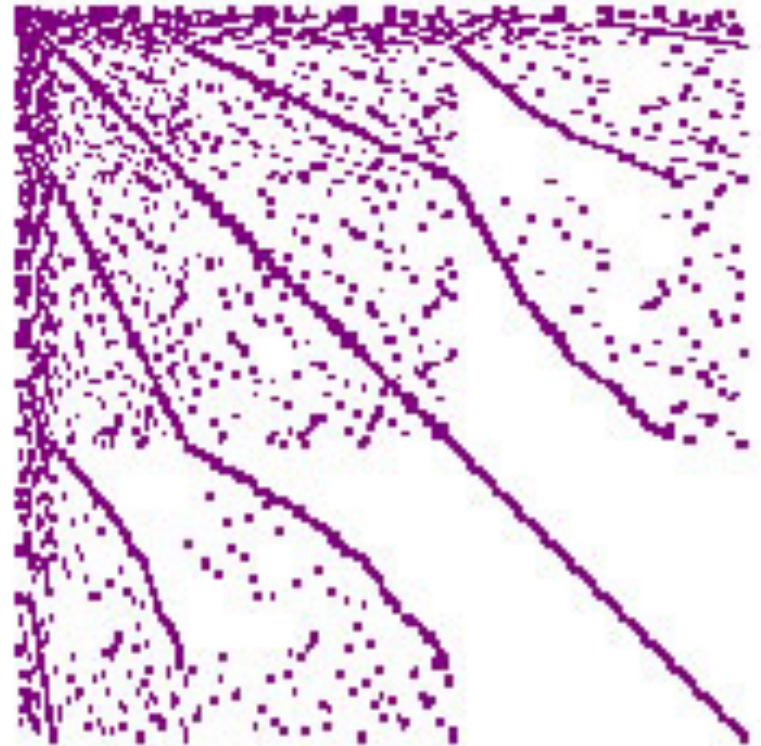
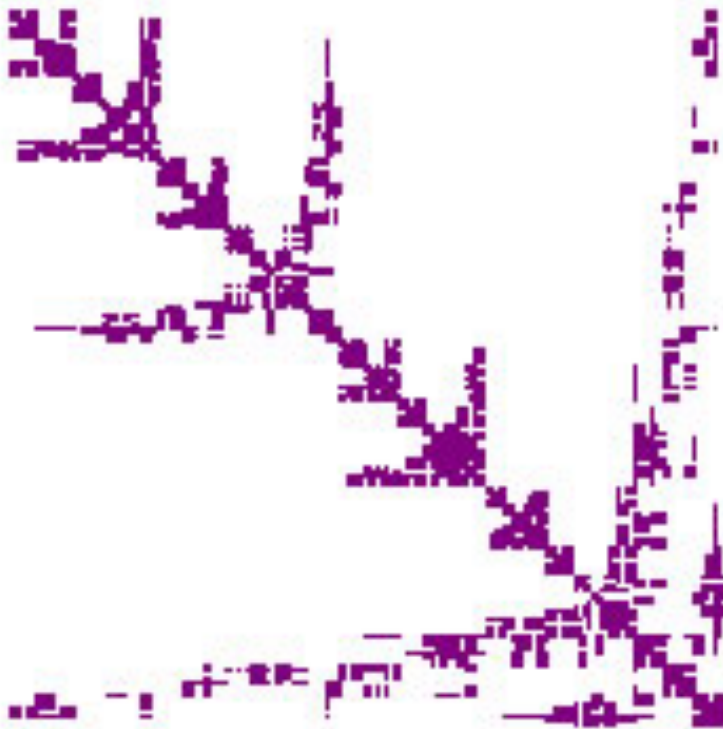


Example: Finite Elements for more complex geometries





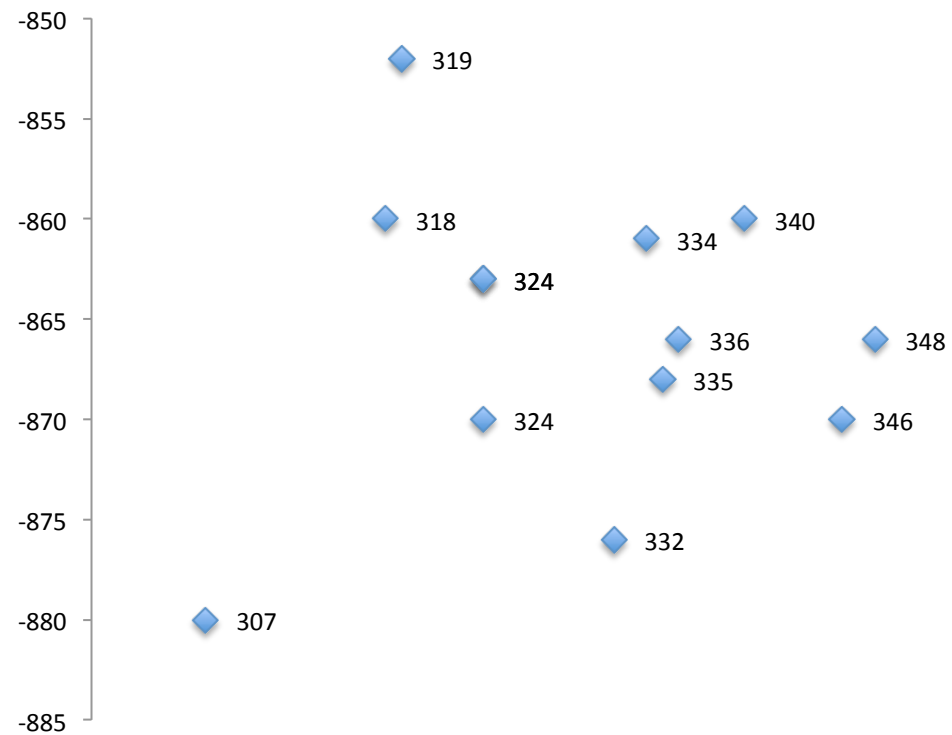
Leads to:



(Spatial) Databases Applications:

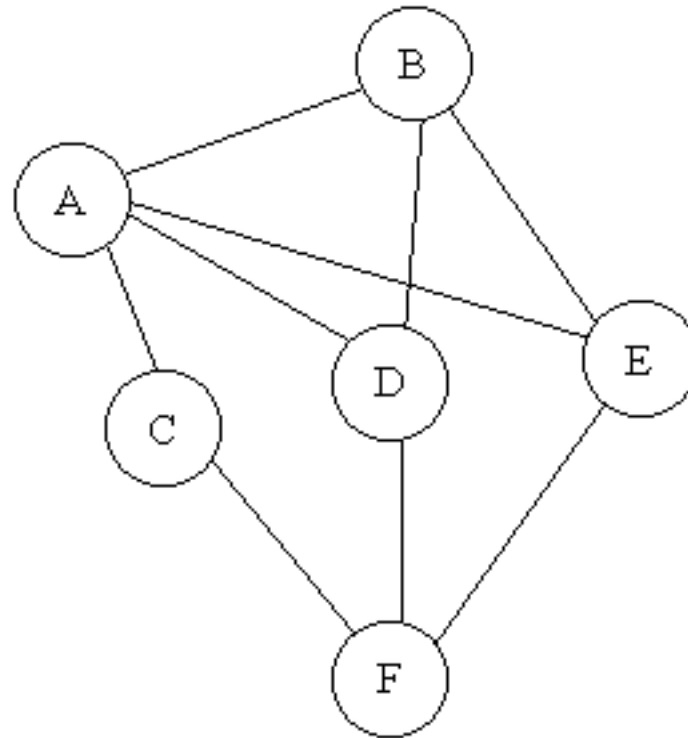
	City	State	ZipCode	Latitude	Longitude
1	Troy	AL	36081	31.809675	-85.972173
2	Mobile	AL	36685	30.686394	-88.053241
3	Trussville	AL	35173	33.621385	-86.602739
4	Montgomery	AL	36106	32.35351	-86.265837
5	Selma	AL	36701	32.41179	-87.022234
6	Talladega	AL	35161	33.43451	-86.102689
7	Tuscaloosa	AL	35402	33.209003	-87.571005
8	Huntsville	AL	35801	34.729135	-86.584979
9	Gadsden	AL	35901	34.014772	-86.007172
10	Birmingham	AL	35266	33.517467	-86.809484
11	Montgomery	AL	36124	32.38012	-86.300629
12	Decatur	AL	35602	34.60946	-86.977029
13	Eufaula	AL	36072	31.941565	-85.239689

Stored using longitude and latitude values, normalized x10

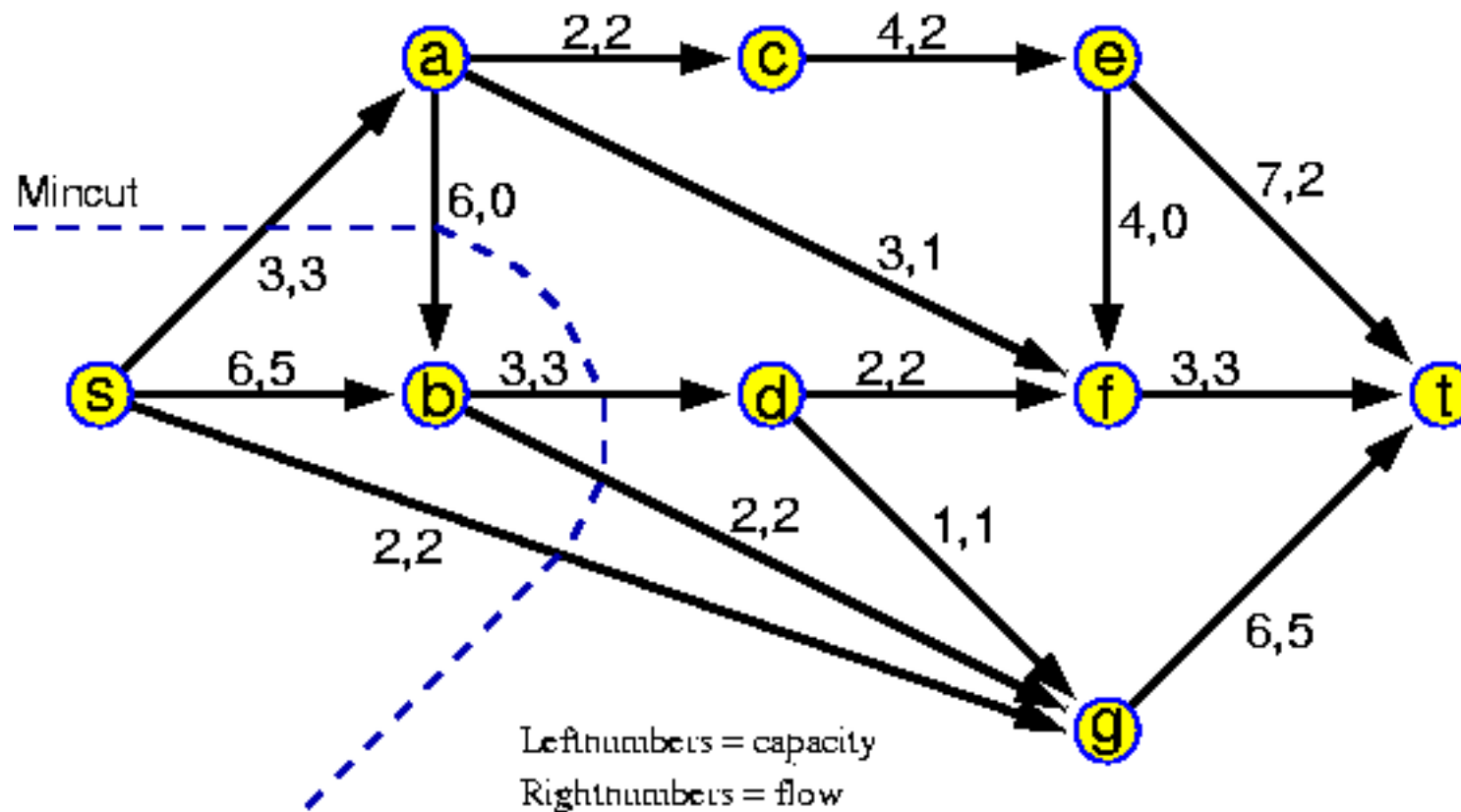


Example: Graph Algorithms

	A	B	C	D	E	F
A	-	1	1	1	1	
B	1	-		1	1	
C	1		-			1
D	1	1		-		1
E	1	1			-	1
F			1	1	1	-



Example: Combinatorial Optimization



Solving $Ax = b$, with sparse A

- Direct Methods

- $Ax = LUx = b$

- Iterative Methods

- Write $Ax = b$ as

- $$Mx = (M-A)x + b, \text{ for some matrix } M$$

- Solve each time:

- $$Mx_{k+1} = (M-A)x_k + b$$

- Until

- $$\|x_{k+1} - x_k\| < \varepsilon, \text{ for some small } \varepsilon$$

Choose **easy invertible M** :

- Diagonal part of A (Jacobi's)

- Triangular part of A (Gauss Seidel)

- Combination of the two (Successive Overrelaxation)

- If $M = A$, then we have the direct method

- Incomplete LU Factorization

Stability in direct methods

- Recapture Dense LU:

```
DO I = 1, N
  PIVOT = A(I, I)
  DO J = I+1, N
    MULT = A(J, I) / PIVOT
    A(J, I) = MULT
    DO K = I+1, N
      A(J, K) = A(J, K) - MULT * A(I, K)
    ENDDO
  ENDDO
ENDDO
```

- What if the PIVOT IS 0 (or very small) ?

Pivoting

$$\begin{pmatrix} 0 & 1 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 4 \\ 5 \end{pmatrix}$$

- Whenever $a_{kk} = 0$ (or small) for some k . Look for a_{mk} which is not zero (or large)
- Permute row m to row k (exchange row m and row k)
- a_{mk} is now on the diagonal

$$\begin{pmatrix} 2 & 3 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5 \\ 4 \end{pmatrix}$$

Numerical instability with small pivots

$$\begin{pmatrix} 0.001 & 2.42 \\ 1.00 & 1.58 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5.20 \\ 4.57 \end{pmatrix}$$

If Gaussian elimination is performed with 3 decimal floating point arithmetic (0.123 E10), then $(1.58 - 2420 = -2420$ and $4.57 - 5200 = -5200)$

$$\begin{pmatrix} 0.001 & 2.42 \\ 0 & -2420 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} 5.20 \\ -5200 \end{pmatrix}$$

Which gives as result $\tilde{x} = \begin{pmatrix} -3.00 \\ 2.15 \end{pmatrix}$ ($0.001 * x_1 = 5.20 - 2.42 * 2.15 = -0.003$)

While true solution is $x = \begin{pmatrix} 1.18 \\ 2.15 \end{pmatrix}$ ($1.18 * 1 + 2.15 * 1.58 = 4.57$
 $1.18 * 0.001 + 2.15 * 2.42 = 5.20$)

This is solved by **partial pivoting** (again).

→ Ensure that all multipliers < 1 , or
for all entries l_{ij} of L : $|l_{ij}| < 1$

This is achieved by choosing only pivots a_{kk} such that

$$|a_{kk}^{(k)}| \geq |a_{ik}^{(k)}|, i > k$$

This is again achieved by row interchanges.

Example

$$A = \begin{bmatrix} 3 & 17 & 10 \\ 2 & 4 & -2 \\ 6 & 18 & -12 \end{bmatrix}$$

At the first step 6 is chosen as pivot.

So row 1 \rightarrow row 3, row 2 \rightarrow row 2, and row 3 \rightarrow row 1

This can be represented with **permutation matrices**:

$$P_1 = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}, \text{ and } P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 2 & 4 & -2 \\ 3 & 17 & 10 \end{bmatrix}$$

The elimination step can be represented by:

$$E_1 = \begin{bmatrix} 1 & 0 & 0 \\ -1/3 & 1 & 0 \\ -1/2 & 0 & 1 \end{bmatrix}, \text{ so } E_1 P_1 A = \begin{bmatrix} 6 & 18 & -12 \\ 0 & -2 & 2 \\ 0 & 8 & 16 \end{bmatrix}$$

Solution is obtained by

$$1. \quad c = Pb$$

$$2. \quad Ly = c$$

$$3. \quad Ux = y$$

with: $P = P_{n-1}P_{n-2}\dots P_2P_1, PA = LU$

$$Ax = b \Rightarrow PAx = Pb \Rightarrow LUx = Pb \Rightarrow L(Ux) = Pb$$

Complete Pivoting

With partial pivoting the growth of the entries in the lower triangular matrix can still be as large as 2^{n-1} (if pivot ≈ 1 at each step, then entries can double at each step)

→ Need for **finding better pivots**

Instead of

$$|a_{kk}^{(k)}| \geq \max (|a_{ik}^{(k)}|, i > k)$$

choose

$$|a_{kk}^{(k)}| \geq \max (|a_{ij}^{(k)}|, i, j > k)$$

So with complete pivoting each step can be expressed as:

$$E_{n-1}P_{n-1}E_{n-2}P_{n-2} \dots E_1P_1 A Q_1 Q_2 \dots Q_{n-1} = U.$$

So,

$$PAQ = LU$$

with $P = P_{n-1}P_{n-2} \dots P_2P_1$, $Q = Q_1Q_2 \dots Q_{n-2}Q_{n-1}$

So, the solution x can be obtained by

1. $c = Pb$

2. $Ly = c$

3. $Uz = y$

4. $Q^T x = z$ ($Q^T = Q^{-1}$)

For many systems pivoting is not required

1. A is strictly **diagonally dominant**, if $|A_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}|$.

Theorem 1 *If A^T is strictly diagonally dominant, then LU obtained with no pivoting has the property that $|L_{ij}| \leq 1$, for all i, j .*

2. A is symmetric, if $A_{ij} = A_{ji}$ for all i, j . A is positive definite, if for every $x \neq 0$

$$x^T A x > 0$$

($x^T A x$ often reflects the energy of the underlying physical system and is therefore often positive.)

Theorem 2 *If A is **symmetric positive definite**, then*

$$\rho = \max_{i,j,k} |a_{ij}^{(k)}| \leq \max_{i,j} |a_{ij}|.$$

In this case LU can be written as $A = L \cdot L^T$ (or LDL^T , avoiding the calculation of square roots). This is called **Choleski Factorization**.

Iterative Methods

$$Mx_{k+1} = (M-A)x_k + b$$

with M easy invertible, meaning that in most of the cases M^{-1} can be directly expressed by a single matrix \mathcal{M}

→ So, the solution can be obtained by simply performing (sparse) matrix multiplications

$$x_{k+1} = \mathcal{M} ((M-A)x_k + b)$$

Implementation Issues

- **Data Storage:** Pointer structures, Linked lists, Linear Arrays
- **Pivot Search:** Multiple storage schemes
- **Masking Operations:** Gather/Scatter Operations
- **Garbage collection:** Fill-in, Explicit garbage collection
- **Permutation Issues:** Implicit and/or explicit

$$A = (a_{ij}) = \begin{pmatrix} 1. & 0. & 0. & -1. & 0. \\ 2. & 0. & -2. & 0. & 3. \\ 0. & -3. & 0. & 0. & 0. \\ 0. & 4. & 0. & -4. & 0. \\ 5. & 0. & -5. & 0. & 6. \end{pmatrix}$$

Coordinate Scheme Storage

```
int  IRN[11], JCN[11];  
float VAL[11];
```

	1	2	3	4	5	6	7	8	9	10	11
IRN	1	2	2	1	5	3	4	5	2	4	5
JCN	4	5	1	1	5	2	4	3	3	2	1
VAL	-1.	3.	2.	1.	6.	-3.	-4.	-5.	-2.	4.	5.

- No explicit order of the nonzero entries is enforced
- Fetching row/column requires the whole data structure to be searched
- Insertion and/or deletion of nonzero entries is simple

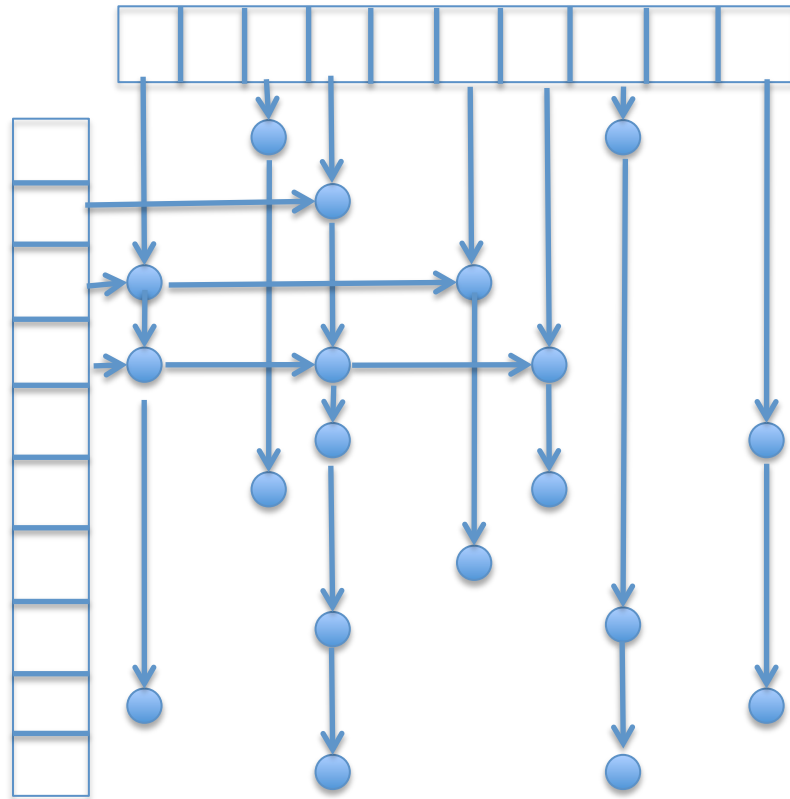
Sparse Compressed Row/Column Format

```
int    LENROW[5], POINTER[5], ICN[11]
float VAL[11]
```

LENROW	2	3	1	2	3						
POINTER	1	3	6	7	9						
ICN	4	1	5	1	3	2	4	2	3	1	5
VAL	-1.	1.	3.	2.	-2.	-3.	-4.	4.	-5.	5.	6.

- LENCOL, POINTER, and IRN are used for compressed column format
- Fetching row or column is very easy in corresponding format
- Insertion of nonzero elements is a big problem – expanded row/column is put at the end, and the LENROW/LENCOL is updated correspondingly
- Instead of LENROW/LENCOL the last element in each row in ICN is negated

Linked List (Pointer) Implementations



- Very flexible
- Access to data very inefficient
 - Pointer chasing
 - Addresses not consecutive: bad spatial locality

ExtendedColumn/ITpack/JaggedDiagonal Format

Shift all nonzero entries to the beginning of each row

```
int  INDEX[5][max]
float VALUE[5][max]
```

$$\text{INDEX: } \begin{pmatrix} 1 & 4 & 0 \\ 1 & 3 & 5 \\ 2 & 0 & 0 \\ 2 & 4 & 0 \\ 1 & 3 & 5 \end{pmatrix} \text{ and VALUE: } \begin{pmatrix} 1. & -1. & 0. \\ 2. & -2. & 3. \\ -3. & 0. & 0. \\ 4. & -4. & 0. \\ 5. & -5. & 6. \end{pmatrix}$$

- Especially suited for vector processing
- Commonly used in sparse matrix multiplication
- Very good use of spatial locality

Full Dense Format

float A[i][j]

- Seems wasteful
- Mostly restricted to sub-blocks of the matrix which contain many nonzero's
- Used to locally expand rows and/or columns
- Often used in hybrid storage schemes with other formats

Pivot Search

- When doing Gaussian Elimination: rows are added to other rows
 - Compressed row storage seems to be the natural choice
 - However, for partial pivoting for instance: each time all elements in a column need to be inspected
- ➔ Both row AND column compressed storage are required

Masking Operations (GATHER/SCATTER)

Adding one sparse row to another:

- Two incrementing pointers
- Scattering target row into a dense row, with a masking array indicating which position in the row are nonzero

```
DO J = POINTER (K), POINTER (K+1) - 1      |  
    TARGET ( ICN (K) ) = VAL ( K )          | SCATTER  
    MASK ( ICN (K) ) = TRUE                 |
```

```
DO J = POINTER (I), POINTER (I+1) - 1  
    TARGET ( ICN (J) ) = TARGET ( ICN (J) ) + PIV * VAL ( J )  
    IF MASK (ICN(J)) = FALSE THEN MASK (ICN(J)) = True
```

```
DO J = 1, N  
    IF ( MASK (ICN(J)) = TRUE ) THEN write TARGET (ICN(J)) back | GATHER
```


Fill-in / Garbage Collection

- Note that the write back will cause problems in general
- Additional space is reserved to store the expanded columns or rows and the old location will have to be released at some point
- In direct solvers this is mostly explicitly controlled!!!!
- In any case: it is extremely important to minimize the amount of fill-in

Fill-in Control (Markowitch counts)

$r_i^{(k)}$ = the number of nonzero elements in row i of the active $(n-k) \times (n-k)$ sub-matrix

$c_j^{(k)}$ = the number of nonzero elements in column j of the active $(n-k) \times (n-k)$ sub-matrix

→ Instead of complete pivoting, choose pivot based on:

$|a_{ij}^{(k)}| \geq u \cdot$ values in column j of the active submatrix |
such that $(r_i^{(k)} - 1)(c_j^{(k)} - 1)$ is *minimized*.

u ($0 < u \leq 1$) is threshold parameter balancing between stability and fill-in control

Permutations

- If $Q = P^T$ then $PAQ (= PAP^T)$ is a symmetric permutation
 - Diagonal elements stay on the diagonal
 - The associated (di)graph stays the same
- Permutations can be executed explicitly (beforehand), on the fly, or implicitly by referring each time to $P(I)$ instead of I

Lab Assignment

Write a C-program which implements LU factorization with partial pivoting.

See course website for details.