Sparse LU Factorization Modulo p: an Update

Claire Delaplace[1]

Joint work with Charles Bouillaguet [2]

[12]CRIStAL, Université de Lille

[1]Université de Rennes 1, IRISA

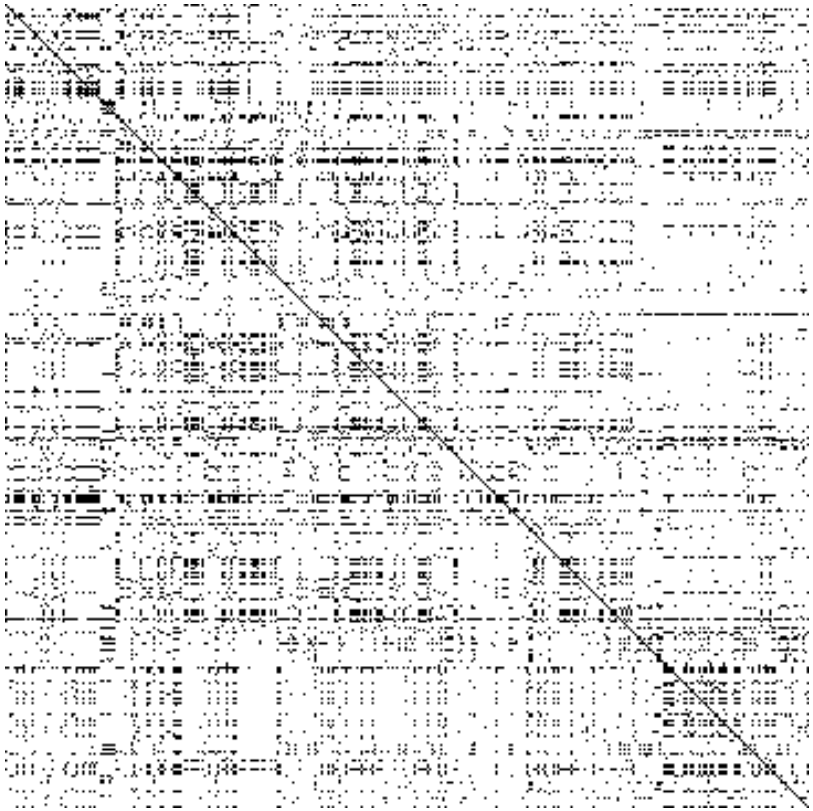
Université de Grenoble, 20 octobre 2016

Background

Sparse Linear Algebra
Modulo p (coefficients : int)

Operations

- Rank
- Linear systems
- Kernel
- etc...

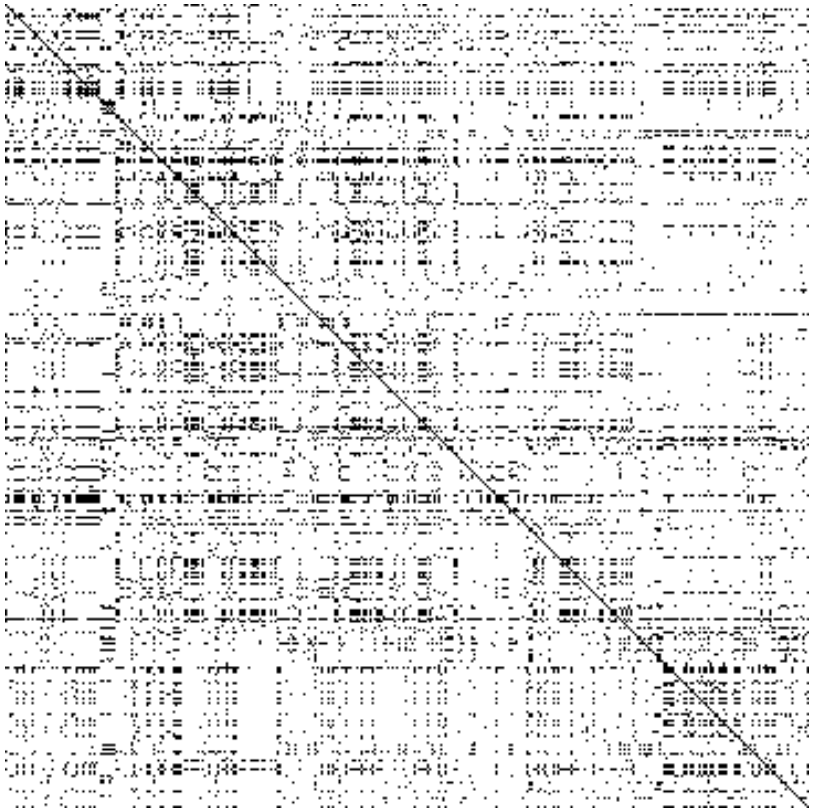


Background

Sparse Linear Algebra
Modulo p (coefficients : int)

Operations

- Rank
- Linear systems
- Kernel
- etc...



Two families of Algorithms

- **Direct methods** (Gaussian Elimination, LU, ...)
- **Iterative methods** (Wiedemann, Lanczos...)

Related Work

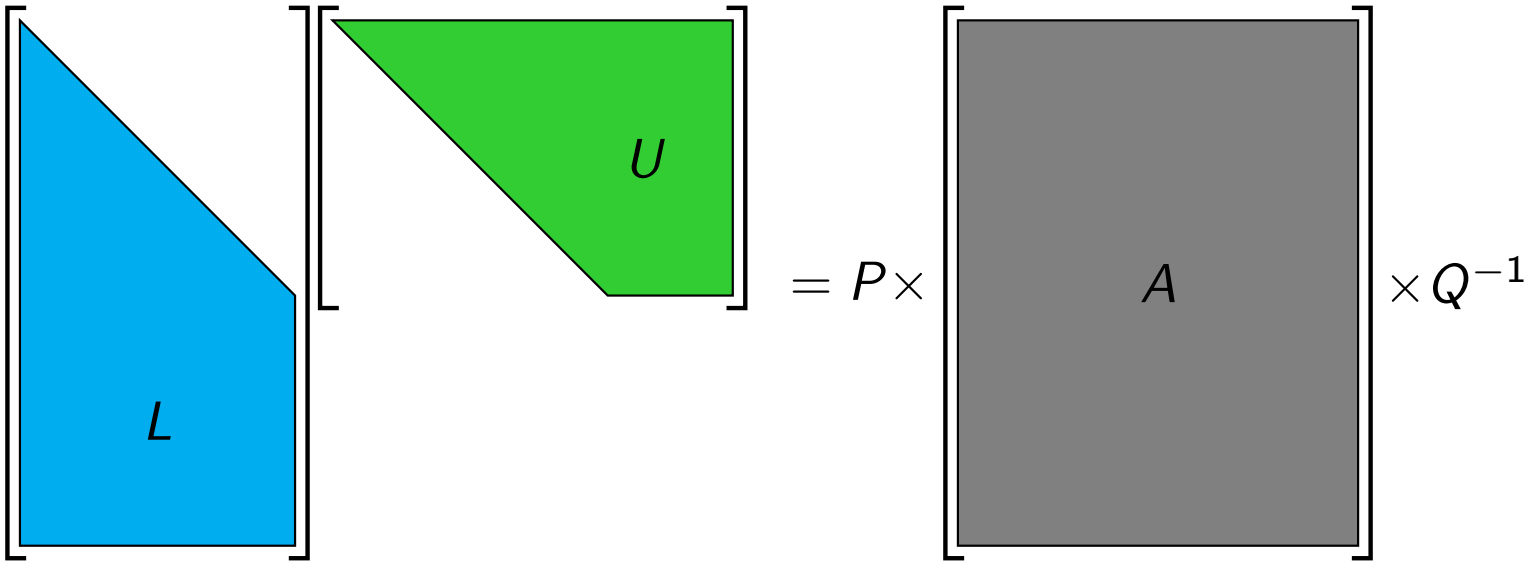
Algorithms

- Comparison between a sparse gaussian elimination and the Wiedmann algorithm: [Dumas & Villard 02]
- Direct methods in the numerical world (e.g. [Davis 06])
- Pivots selection heuristic for Gröbner Basis Matrices [Faugère & Lacharte 10]

Software

- Exact: *not much* (LinBox, GBLA (*Gröbner basis*), MAGMA)
- Numeric: *many* (SuperLU, UMFPACK, ...)

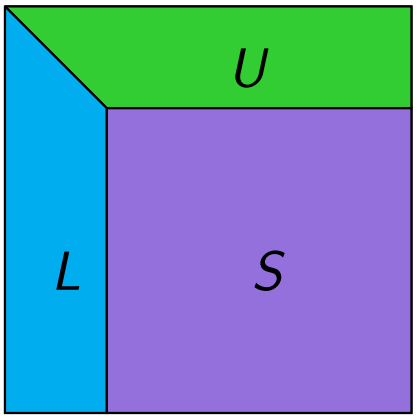
PLUQ Factorization



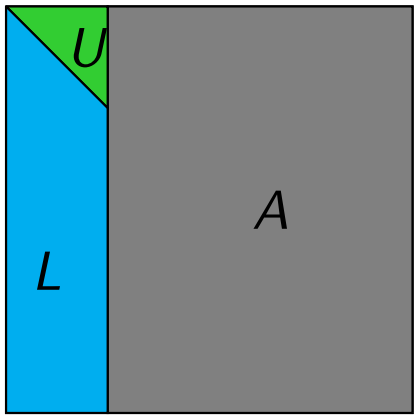
- L has non zero diagonal
- U has unit diagonal
- A can be rectangular
- A can be rank deficient

Right-looking and Left-looking LU

Usual right-looking Algorithm:

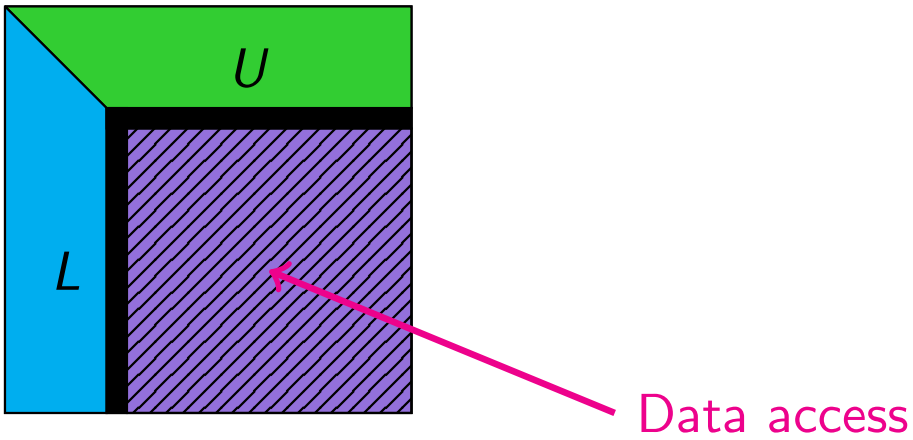


Left-looking GPLU Algorithm
[Gilbert & Peierls 88]

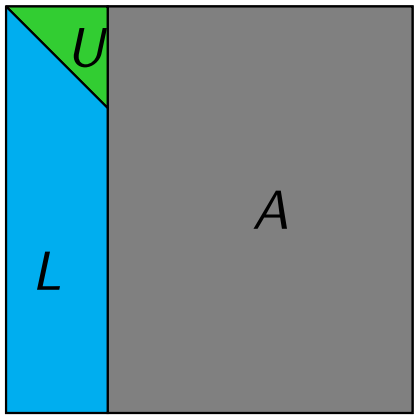


Right-looking and Left-looking LU

Usual right-looking Algorithm:

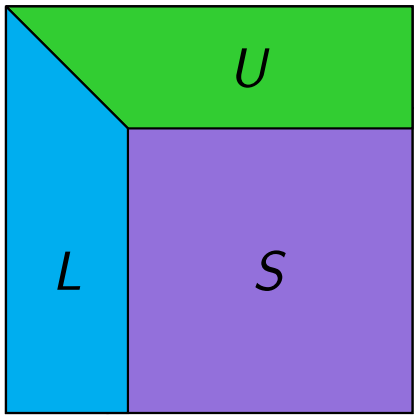


Left-looking GPLU Algorithm
[Gilbert & Peierls 88]

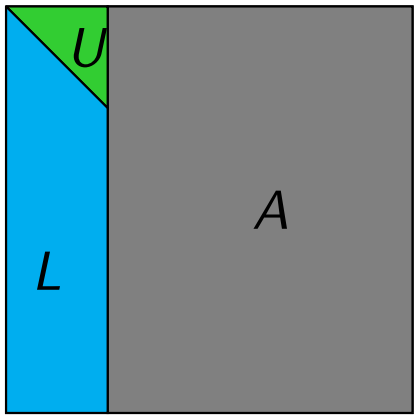


Right-looking and Left-looking LU

Usual right-looking Algorithm:

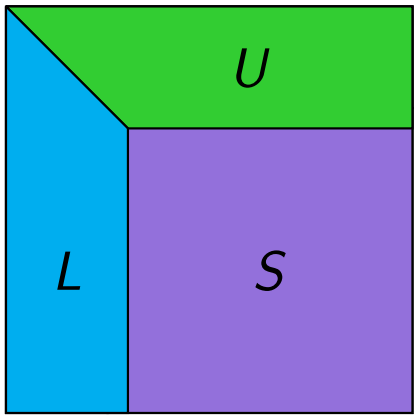


Left-looking GPLU Algorithm
[Gilbert & Peierls 88]

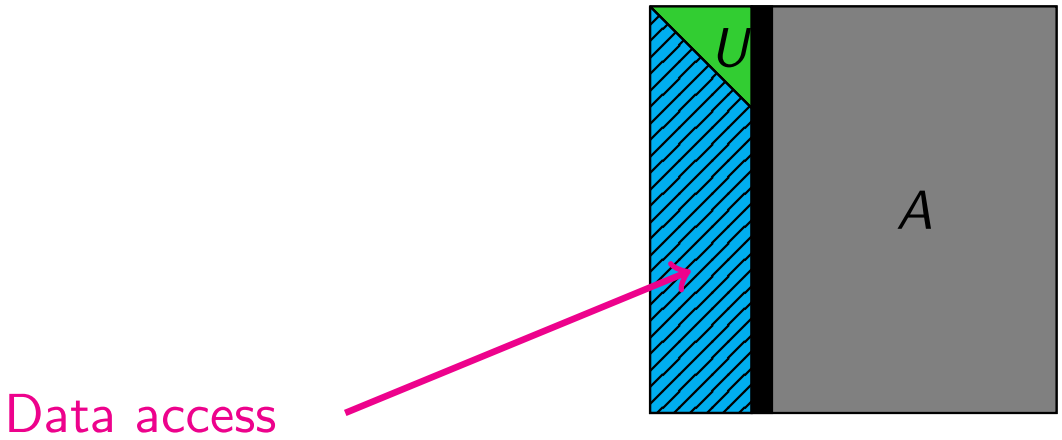


Right-looking and Left-looking LU

Usual right-looking Algorithm:

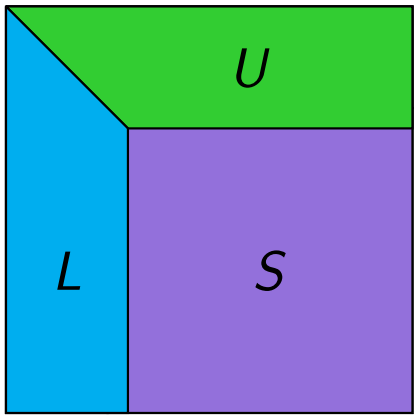


Left-looking GPLU Algorithm
[Gilbert & Peierls 88]

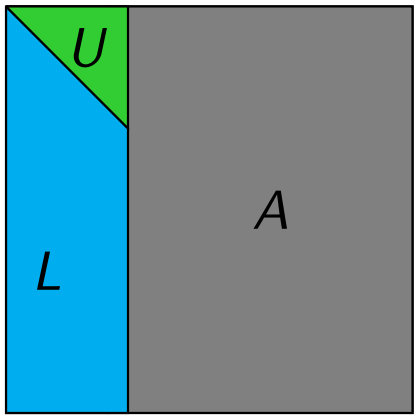


Right-looking and Left-looking LU

Usual right-looking Algorithm:

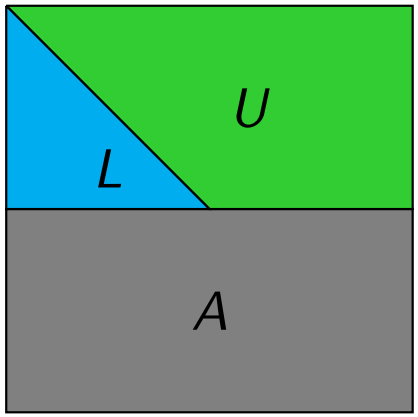


Left-looking GPLU Algorithm
[Gilbert & Peierls 88]



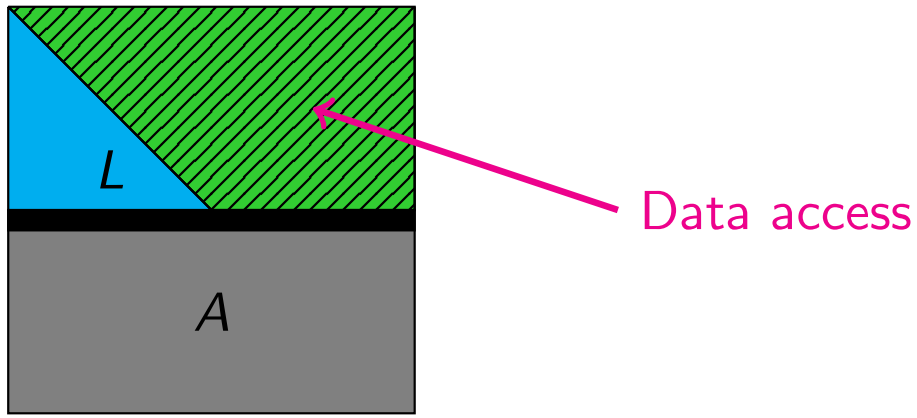
An up-looking variant of the GPLU

- Row-by-row version
- Adapted to Computer Algebra



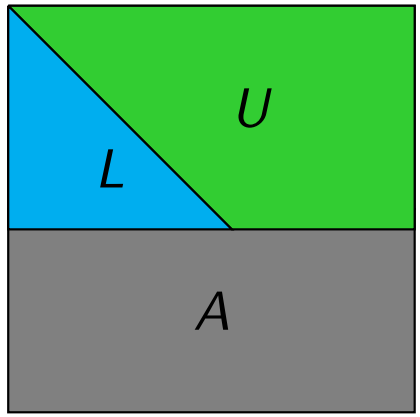
An up-looking variant of the GPLU

- Row-by-row version
- Adapted to Computer Algebra



An up-looking variant of the GPLU

- Row-by-row version
- Adapted to Computer Algebra



GPLU Algorithm: Elimination Step

Ignoring the permutations:

$$A = \begin{pmatrix} A_{00} & \mathbf{a}_{01} & A_{02} \\ \mathbf{a}_{10} & a_{11} & \mathbf{a}_{12} \\ A_{20} & \mathbf{a}_{21} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & & \\ \mathbf{l}_{10} & l_{11} & \\ L_{20} & \mathbf{l}_{21} & L_{22} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & \mathbf{u}_{01} & U_{02} \\ & 1 & \mathbf{u}_{12} \\ & & U_{22} \end{pmatrix}$$

where $(\mathbf{a}_{10}\ a_{11}\ \mathbf{a}_{12})$ is the k-th of A.

GPLU Algorithm: Elimination Step

Ignoring the permutations:

$$A = \begin{pmatrix} A_{00} & \mathbf{a}_{01} & A_{02} \\ \mathbf{a}_{10} & a_{11} & \mathbf{a}_{12} \\ A_{20} & \mathbf{a}_{21} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & & \\ \mathbf{l}_{10} & l_{11} & \\ L_{20} & \mathbf{l}_{21} & L_{22} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & \mathbf{u}_{01} & U_{02} \\ & 1 & \mathbf{u}_{12} \\ & & U_{22} \end{pmatrix}$$

where $(\mathbf{a}_{10}\ a_{11}\ \mathbf{a}_{12})$ is the k-th of A.

In particular:

$$(\mathbf{a}_{10}\ a_{11}\ \mathbf{a}_{12}) = (\mathbf{x}_0\ x_1\ \mathbf{x}_2) \cdot \begin{pmatrix} U_{00} & \mathbf{u}_{01} & U_{02} \\ & 1 & \\ & & Id \end{pmatrix}$$

with $\mathbf{x}_0 = \mathbf{l}_{10}$, $x_1 = l_{11}$ and $\mathbf{x}_2 = l_{11} \cdot \mathbf{u}_{12}$

Sparse Triangular Solving

Goal

Solve $\mathbf{x} \cdot U = \mathbf{b}$ where U and $\mathbf{b}$ are sparse.

$$x_j = b_j - \sum_i x_i \cdot u_{ij}$$

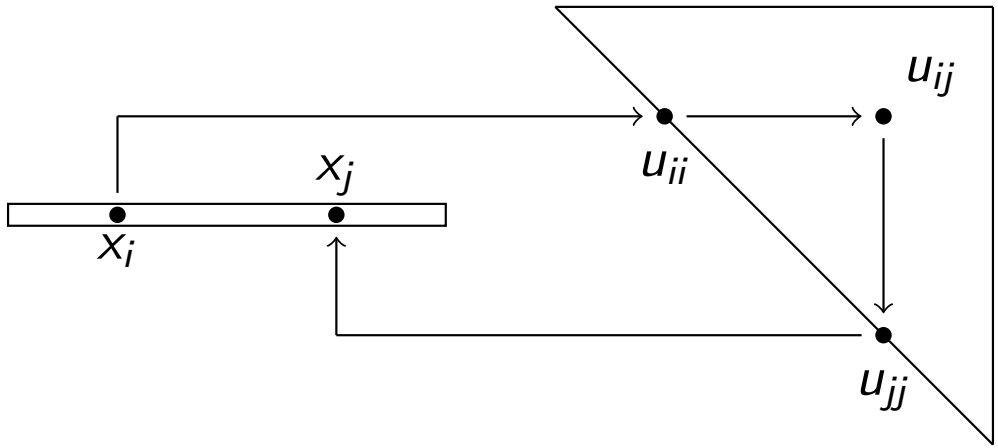
x sparse $\Rightarrow$ Compute only non-zero coefficients

Main Idea

1. Find the non-zero pattern $\mathcal{X} = \{j | x_j \neq 0\}$ of x
2. For all $j \in \mathcal{X}$, compute x_j.

Find the pattern of **x**

$$x_j = b_j - \sum_i x_i \cdot u_{ij}$$

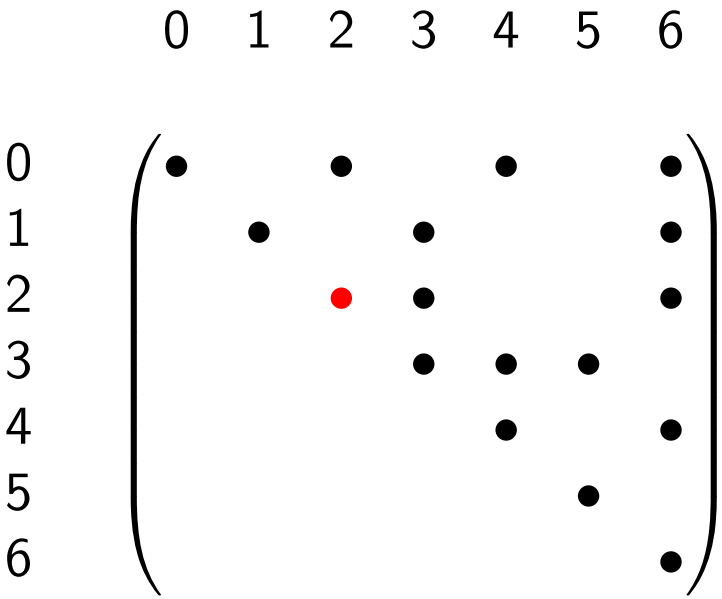


We note that:

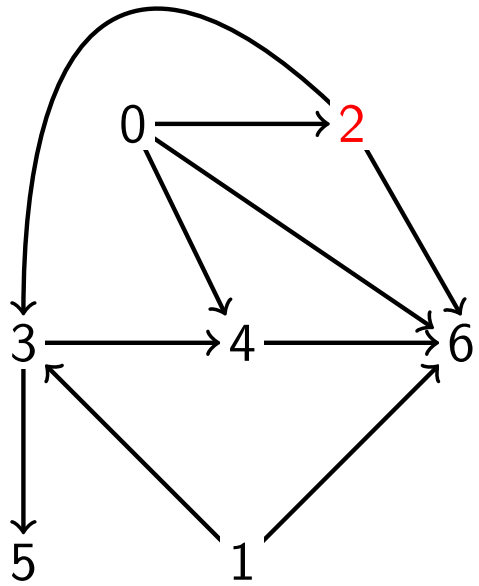
Ignoring numerical cancelations, $x_j \neq 0$ if either:

1. $b_j \neq 0$
2. $\exists i$ s.t. $x_i \neq 0$ and $u_{ij} \neq 0$

G_U : Graph of U

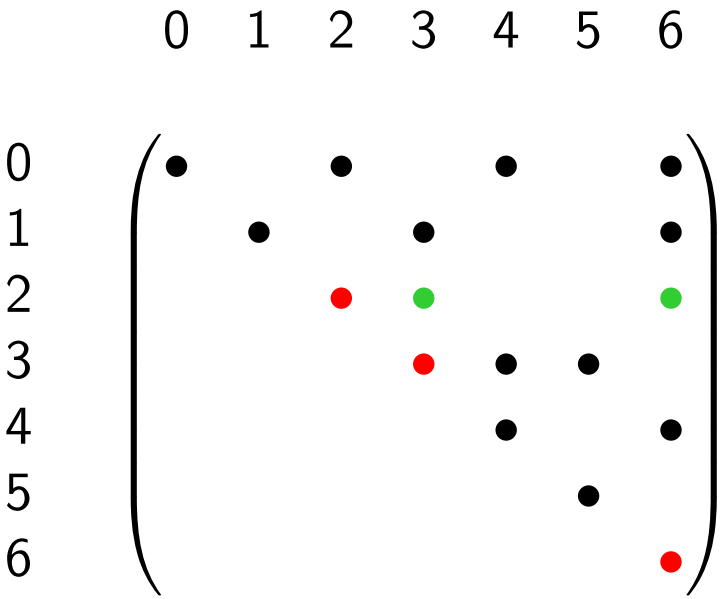


- Columns (Rows) of U : Nodes of G_U

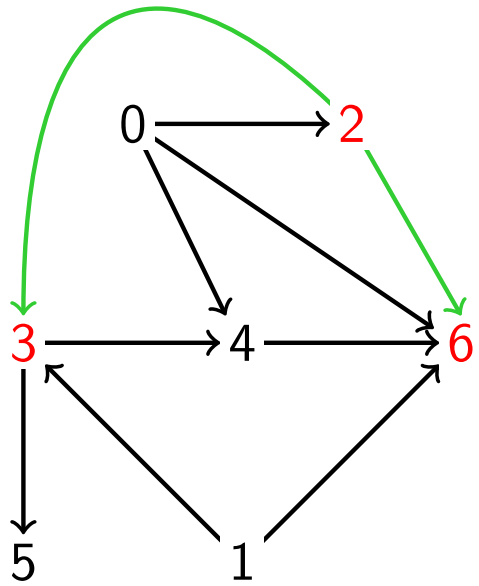


- (i, j) is an edge $\iff$ $u_{ij} \neq 0$

G_U : Graph of U

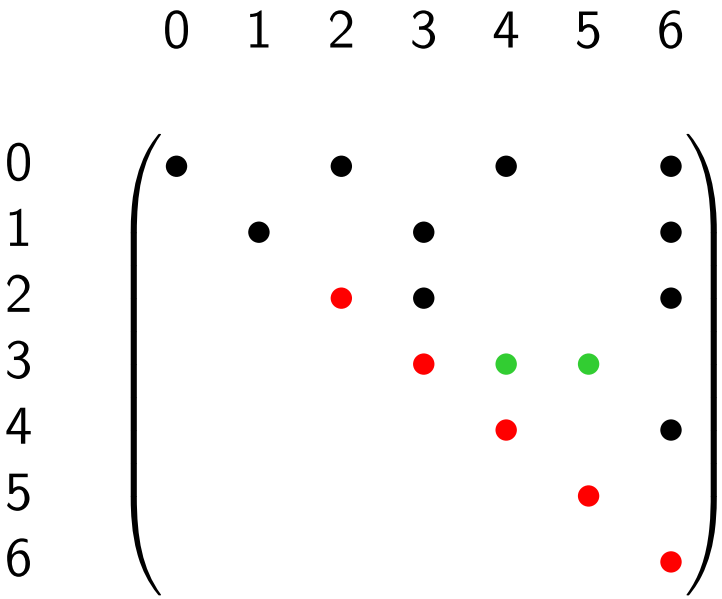


- Columns (Rows) of U : Nodes of G_U

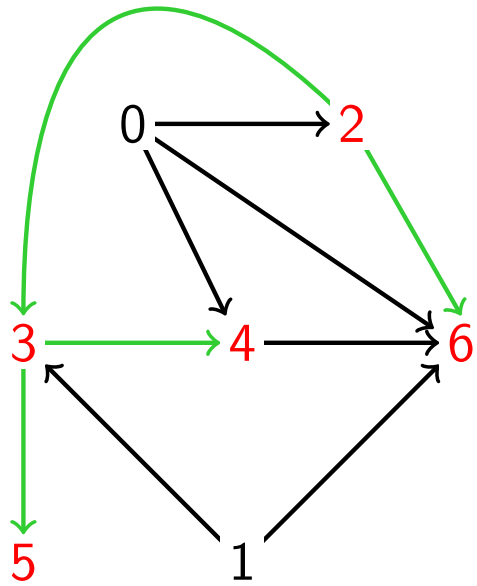


- (i, j) is an edge $\iff$ $u_{ij} \neq 0$

G_U : Graph of U

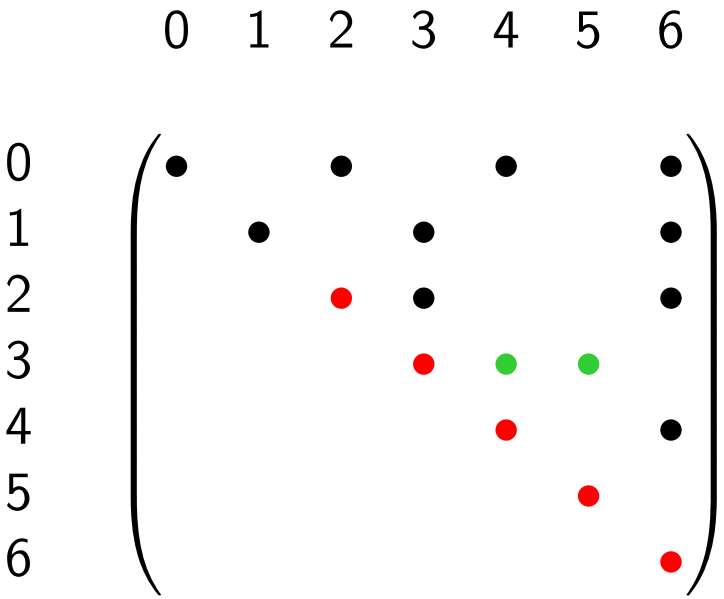


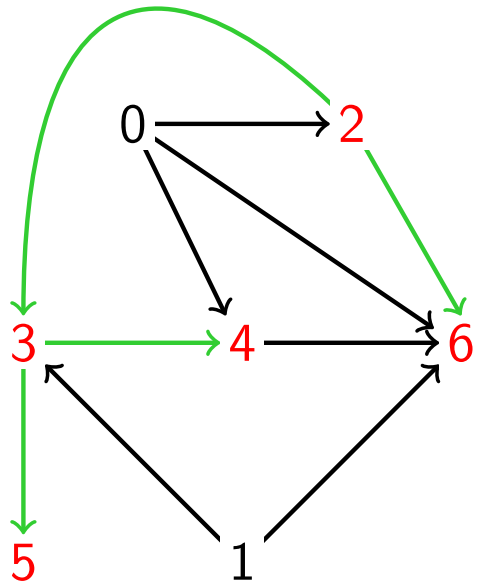
- Columns (Rows) of U : Nodes of G_U



- (i, j) is an edge $\iff$ $u_{ij} \neq 0$

G_U : Graph of U





- Columns (Rows) of U : Nodes of G_U
- (i, j) is an edge $\iff$ $u_{ij} \neq 0$

$(x_j \neq 0) \iff (\exists i \text{ s.t } b_i \neq 0 \text{ and } j \text{ can be reached from } i)$

Gilbert and Peierls Theorem

Notation

$\mathrm{Reach}_U(\mathcal{B})$: set of nodes reachable from any node in $\mathcal{B}$

Theorem ([Gilbert & Peierls 88])

Assuming there is no numerical cancelation:
$\mathcal{X} = \{j | x_j \neq 0\}$ *is given by* $\mathcal{X} = \mathrm{Reach}_U(\mathcal{B})$, *where* $\mathcal{B} = \{i | b_i \neq 0\}$.

Gilbert and Peierls Theorem

Notation

$\mathrm{Reach}_U(\mathcal{B})$: set of nodes reachable from any node in $\mathcal{B}$

Theorem ([Gilbert & Peierls 88])

Assuming there is no numerical cancelation:
$\mathcal{X} = \{j | x_j \neq 0\}$ *is given by* $\mathcal{X} = \mathrm{Reach}_U(\mathcal{B})$, *where* $\mathcal{B} = \{i | b_i \neq 0\}$.

In summary

We find $\mathcal{X}$ by performing a graph search.

Sort $\mathcal{X}$

Remark:

For each x_j in $\mathcal{X}$:

$$x_j = b_j - \sum_i x_i \cdot u_{ij}$$

$\Rightarrow$ We need to know x_i to compute x_j.
$\Rightarrow$ $\mathcal{X}$ must be sorted in topological order: if $u_{ij} \neq 0$, i comes up before j in $\mathcal{X}$.

Perform a DFS from each node from $\mathcal{B} \Rightarrow$ get $\mathcal{X}$ sorted in topological order.

GPLU Algorithm: Application to Exact Linear Algebra

- Never been used before for exact computations
- We implemented it
- We benchmarked it against LinBox (sparse right-looking)

Our Benchmarks show:

- GPLU work best when U is very sparse
- Sometimes GPLU outperform the right-looking algorithm (often)
- Sometimes the right-looking algorithm outperform GPLU (less often)

Illustrative results

Experiment carried on an Intel Core i7-3770 with 8 GB of RAM.

Right-looking (LinBox) vs up-looking GPLU, time in s:

Matrix	Right-looking	GPLU
GL7d/GL7d24	34	276
Margulies/cat_ears_4_4	3	184
Homology/ch7-9.b5	3084	8.2
Homology/ch8-8.b4	1022	0.4
Homology/ch8-8.b5	5160	6
Margulies/wheel_601	7040	4
Mgn/M0,6.data/M0,6-D11	722	0.4

$\Longrightarrow$ Can we take advantage of both methods?

Our Work: A New Hybrid Algorithm (CASC 2016)

Description

- Find many pivots without performing any arithmetical operations.
- Compute the Schur complement S, using a up-looking algorithm.
- Compute the rank of S.

Our Work: A New Hybrid Algorithm (CASC 2016)

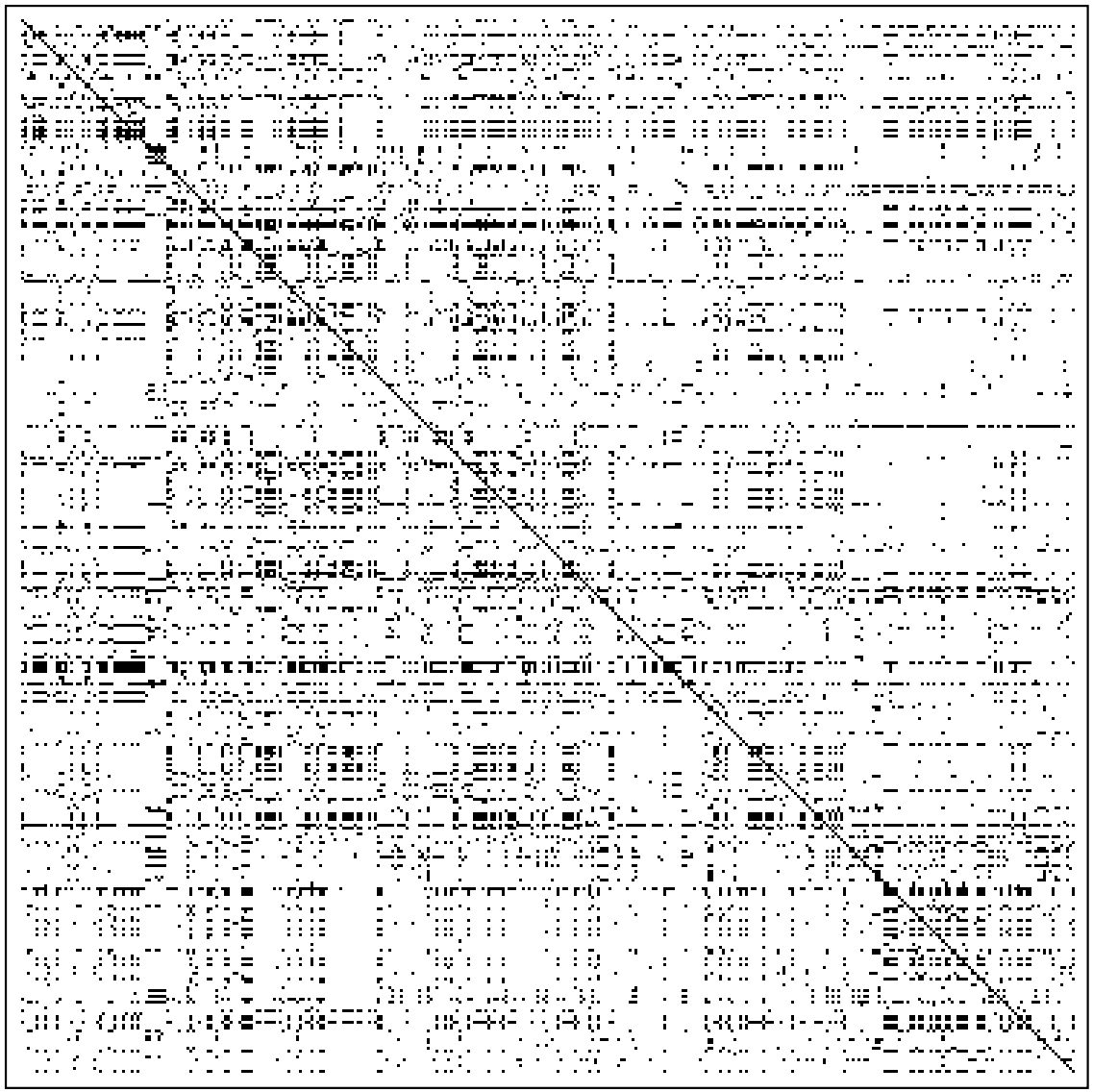
Description

- Find many pivots without performing any arithmetical operations.
- Compute the Schur complement S, using a up-looking algorithm.
- Compute the rank of S.

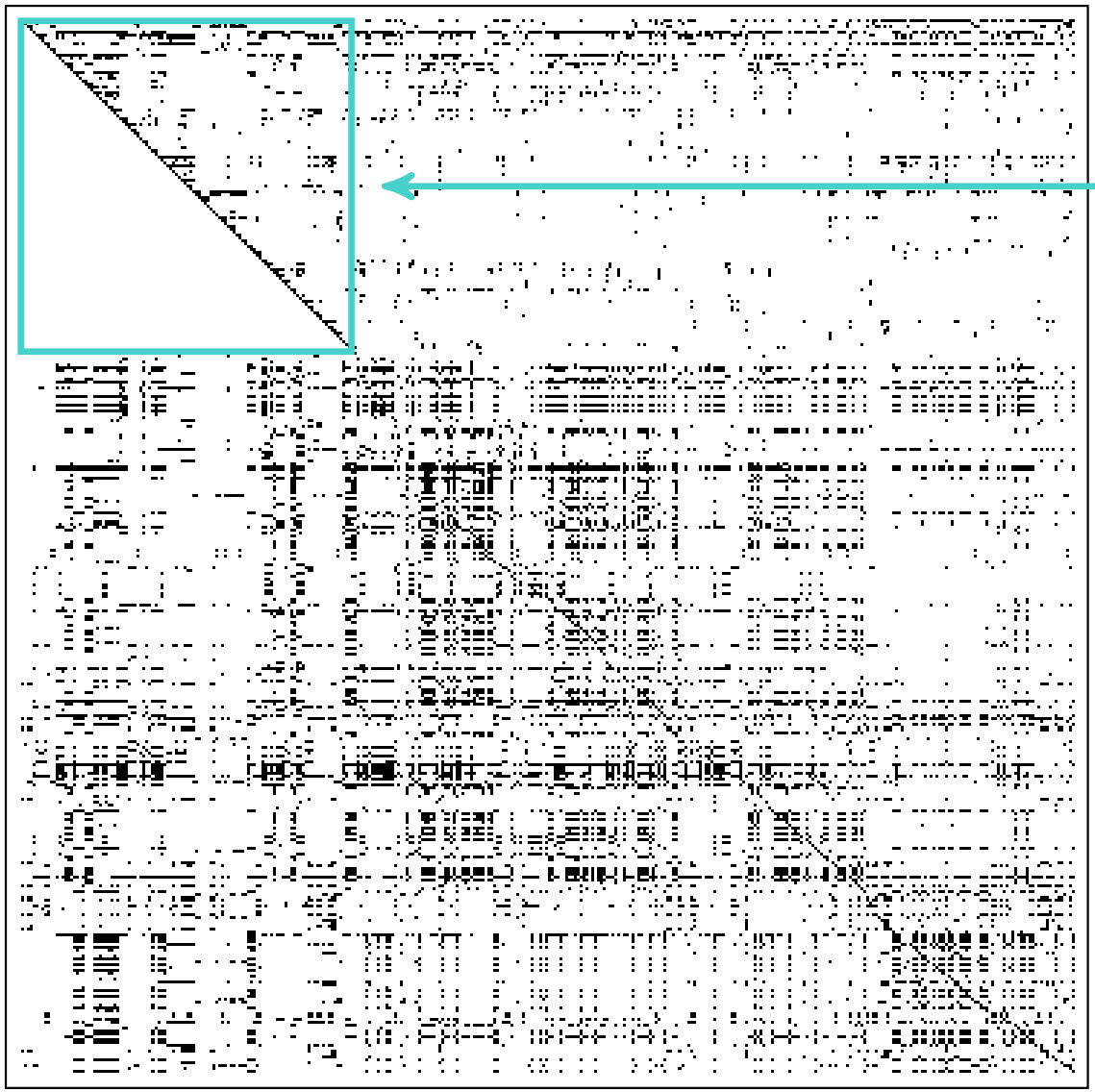
Rank of S

- Recurse
- Dense rank computation
- Wiedemann Algorithm
- ...

Example: GAG/mat364

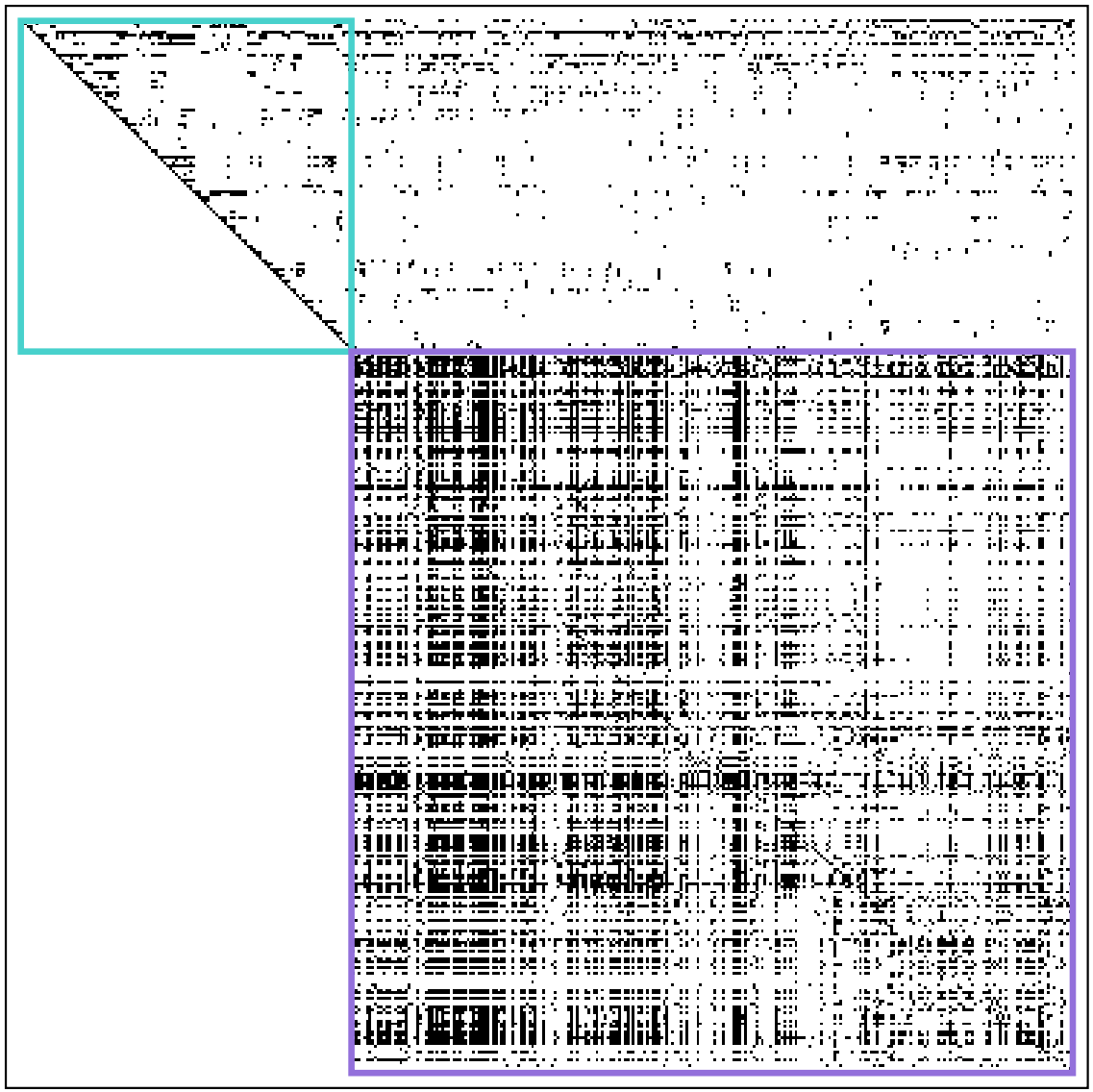


Example: GAG/mat364



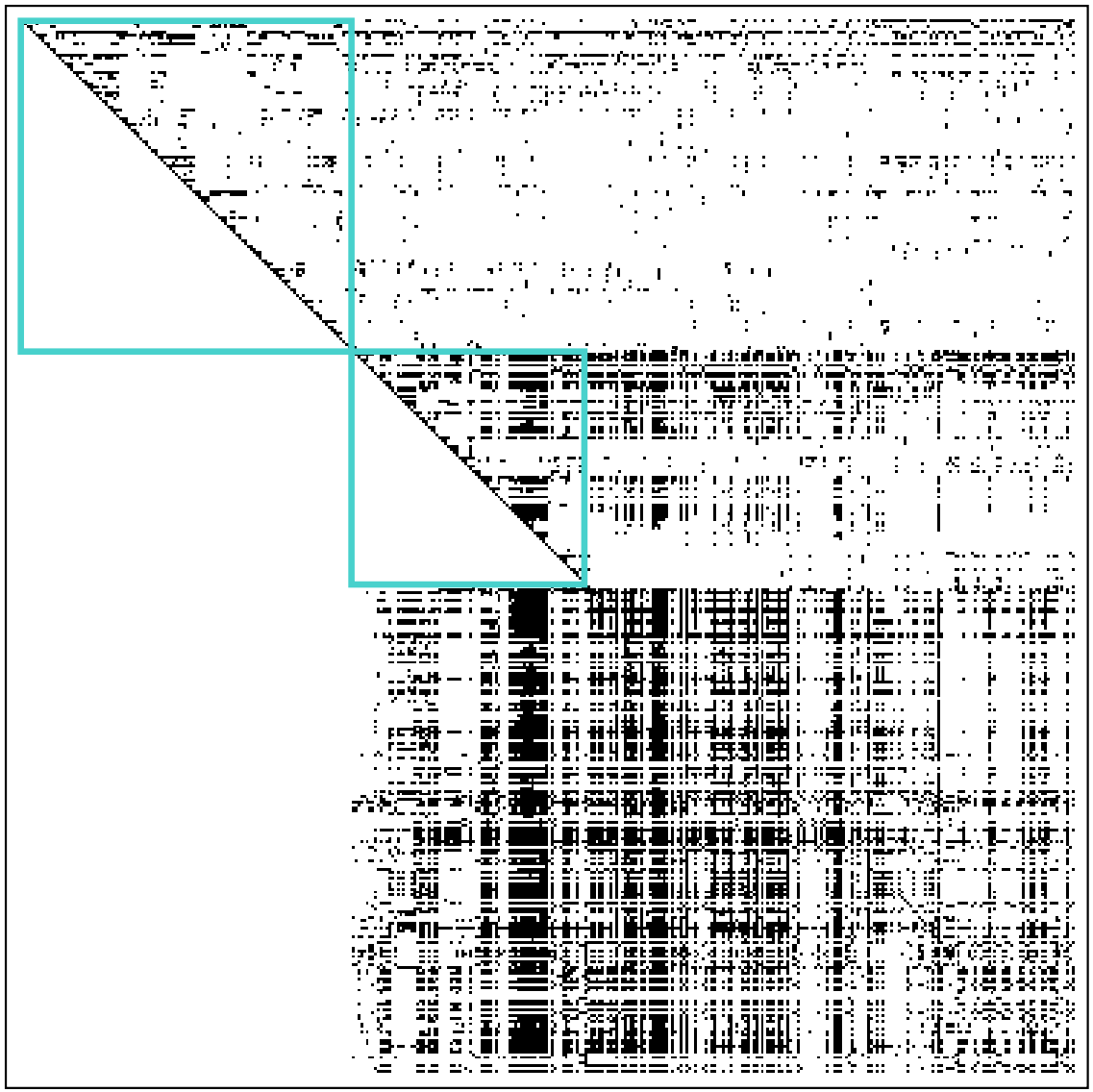
Set of pivots found without any arithmetical operations

Example: GAG/mat364

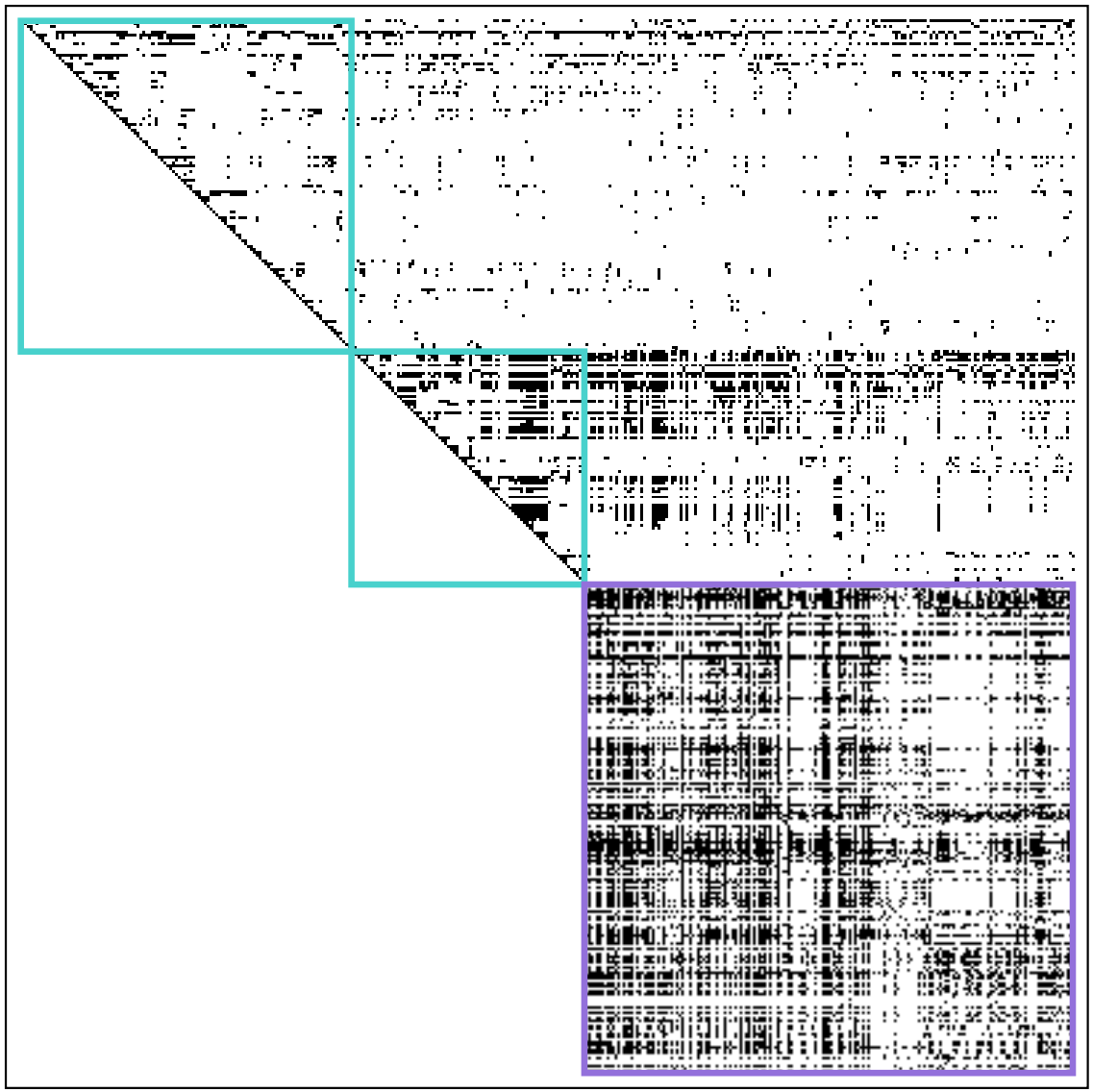


← Schur Complement

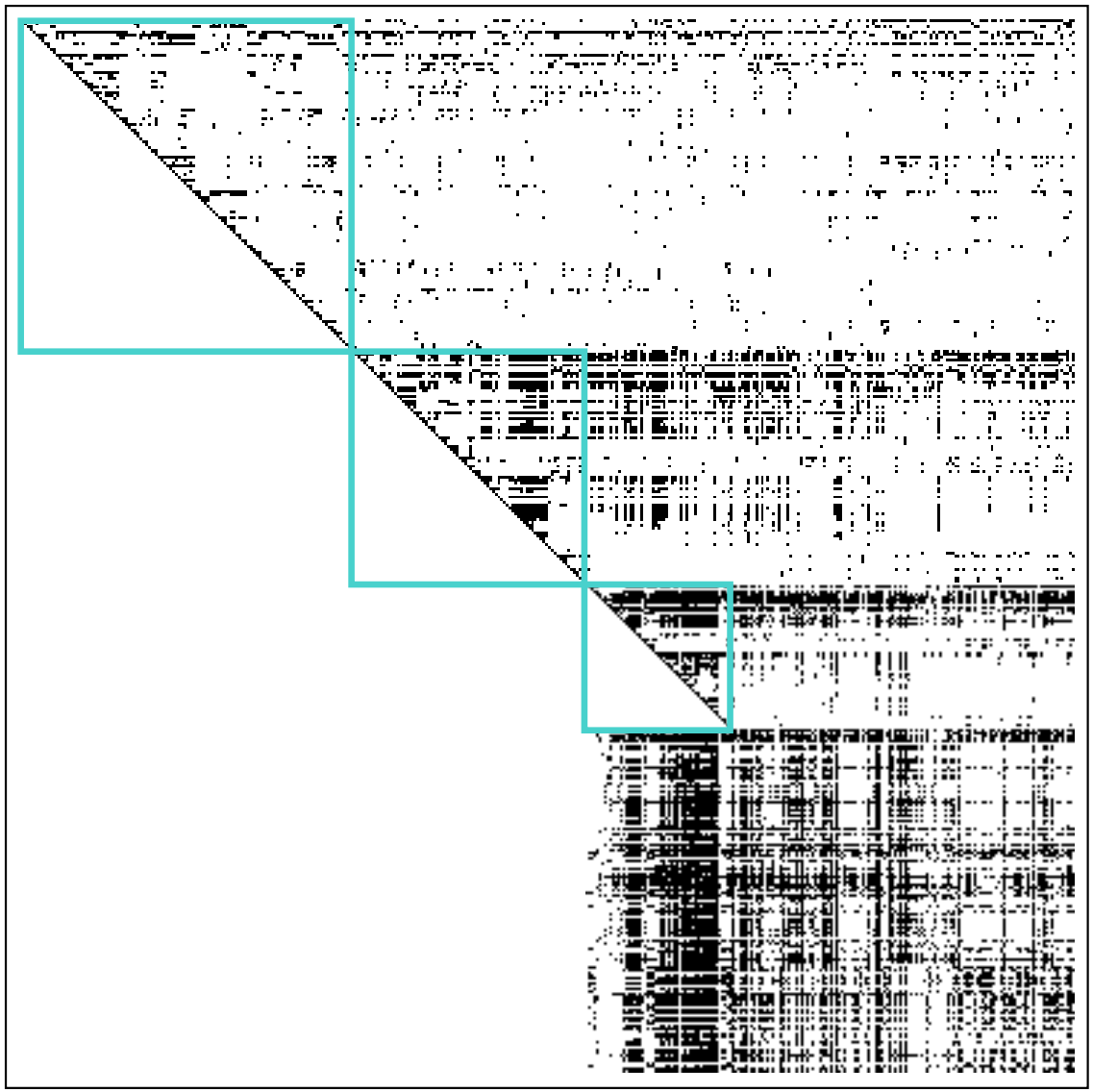
Example: GAG/mat364



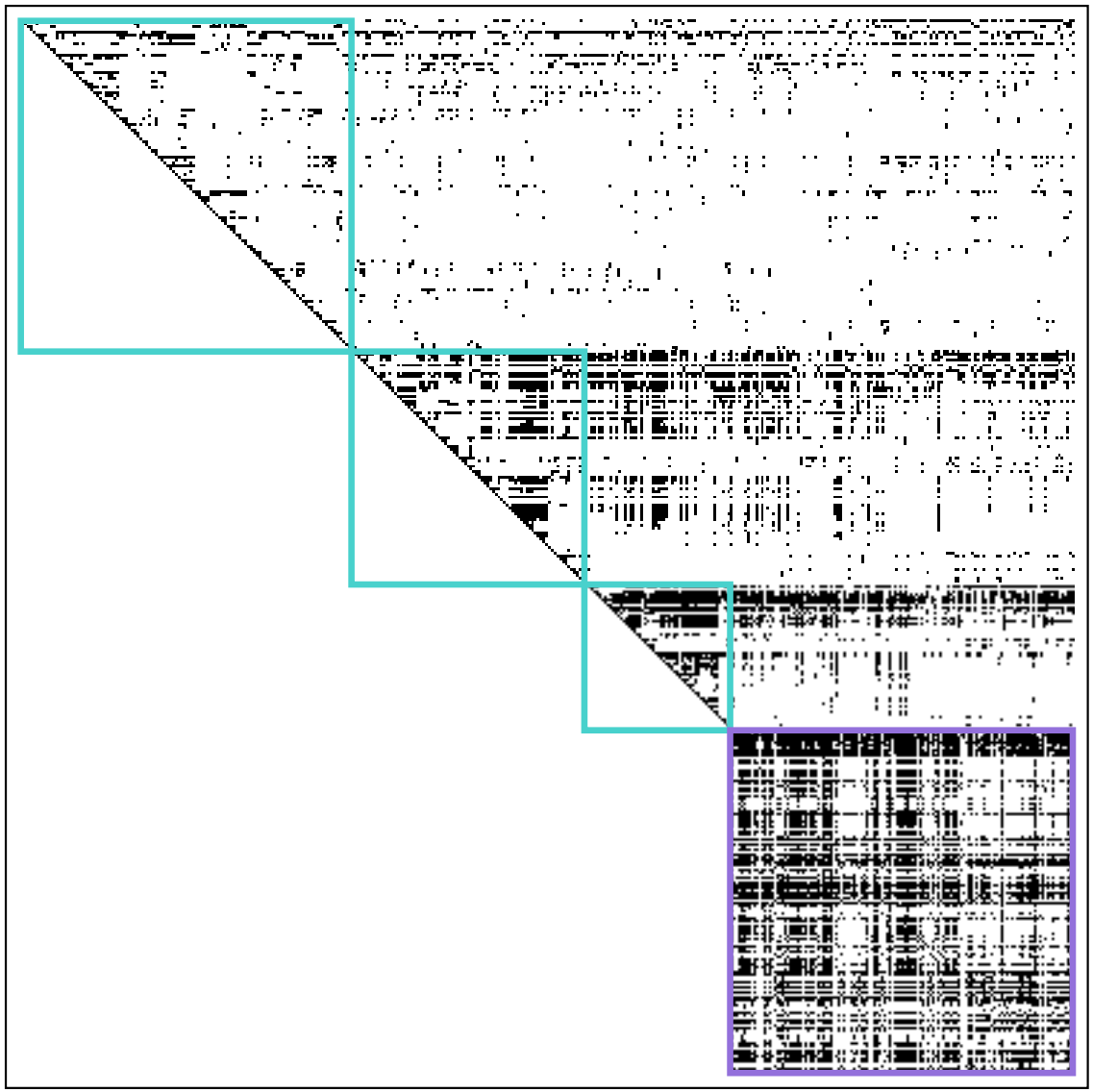
Example: GAG/mat364



Example: GAG/mat364

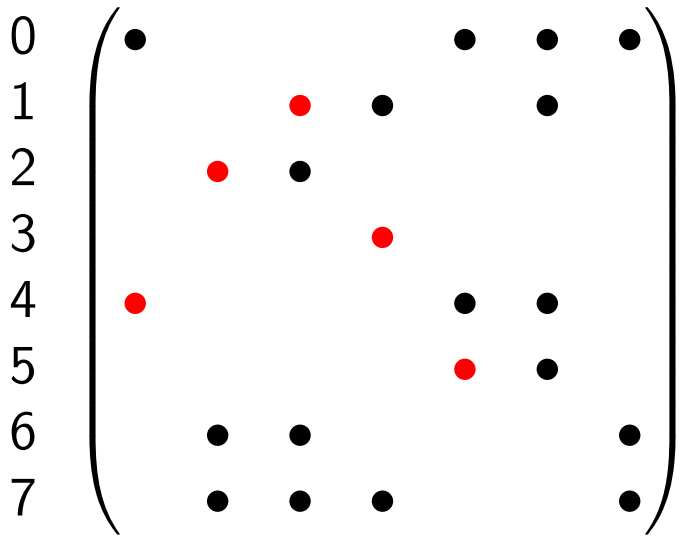


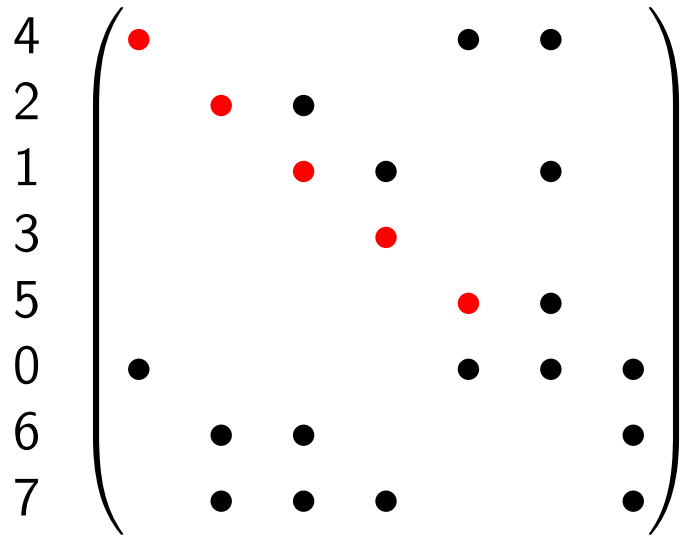
Example: GAG/mat364



← Parallelizable

Initial Pivots Selection Heuristic [Faugère & Lachartre 10]





Description

- Each row is mapped to the column of its leftmost coefficient.
- When several rows have the same leftmost coefficient, select the sparsest.
- Move the selected rows before the others and sort them by increasing position of the leftmost coefficient.

Schur Complement Computation

P denotes the permutation that pushes the pre-computed pivots in the top of A. Ignoring permutation over the columns of A:

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}$$

Schur Complement Computation

P denotes the permutation that pushes the pre-computed pivots in the top of A. Ignoring permutation over the columns of A:

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}$$

Definition

The **Schur Complement** S of PA with respect to U_{00} is given by :

$$S = A_{11} - A_{10} U_{00}^{-1} U_{01}$$

Schur Complement Computation

P denotes the permutation that pushes the pre-computed pivots in the top of A. Ignoring permutation over the columns of A:

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}$$

Definition

The **Schur Complement** S of PA with respect to U_{00} is given by :

$$S = A_{11} - A_{10} U_{00}^{-1} U_{01}$$

Denote by $(\mathbf{a}_{i0}\ \mathbf{a}_{i1})$ the i-th row of $(A_{10}\ A_{11})$, and consider the following system :

$$(\mathbf{x}_0\ \mathbf{x}_1) \cdot \begin{pmatrix} U_{00} & U_{01} \\ & Id \end{pmatrix} = (\mathbf{a}_{i0}\ \mathbf{a}_{i1})$$

Schur Complement Computation

P denotes the permutation that pushes the pre-computed pivots in the top of A. Ignoring permutation over the columns of A:

$$PA = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} = \begin{pmatrix} L_{00} & \\ L_{10} & L_{11} \end{pmatrix} \cdot \begin{pmatrix} U_{00} & U_{01} \\ & U_{11} \end{pmatrix}$$

Definition

The **Schur Complement** S of PA with respect to U_{00} is given by :

$$S = A_{11} - A_{10} U_{00}^{-1} U_{01}$$

Denote by $(\mathbf{a}_{i0}\ \mathbf{a}_{i1})$ the i-th row of $(A_{10}\ A_{11})$, and consider the following system :

$$(\mathbf{x}_0\ \mathbf{x}_1) \cdot \begin{pmatrix} U_{00} & U_{01} \\ & Id \end{pmatrix} = (\mathbf{a}_{i0}\ \mathbf{a}_{i1})$$

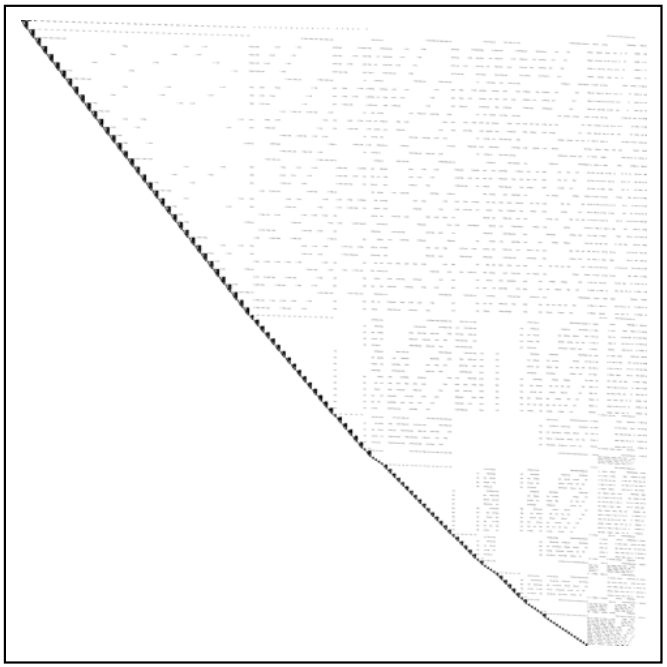
We obtain $\mathbf{x}_1 = \mathbf{a}_{i1} - \mathbf{a}_{i0} U_{00}^{-1} U_{01}$. $\mathbf{x}_1$ **is the i-th row of the** S

Schur Complement Computation

In a Nutshell:

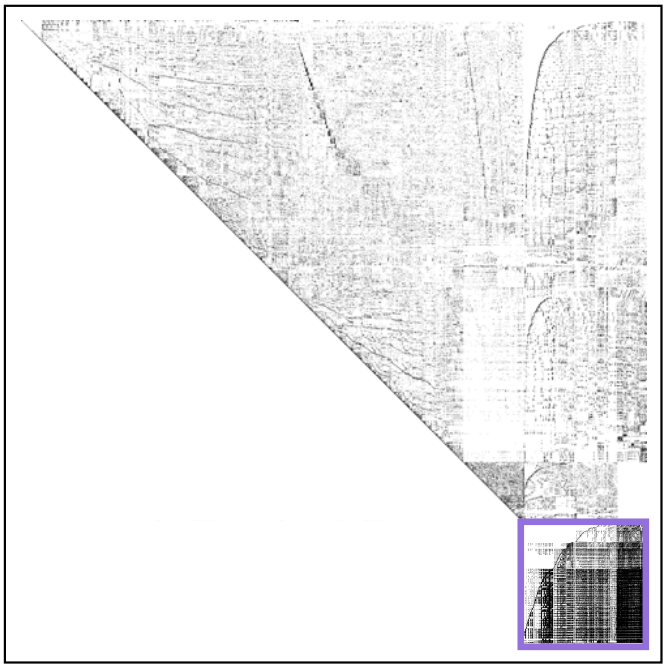
- Each row of S can be computed independently
- Guess if S is sparse or dense by computing some random rows
- If S sparse : compute S using the previous method
- The Schur complement computation is parallelizable

Example: the case of Homology/shar_te2.b3



Example: the case of Homology/shar_te2.b3

Example: the case of Homology/shar_te2.b3

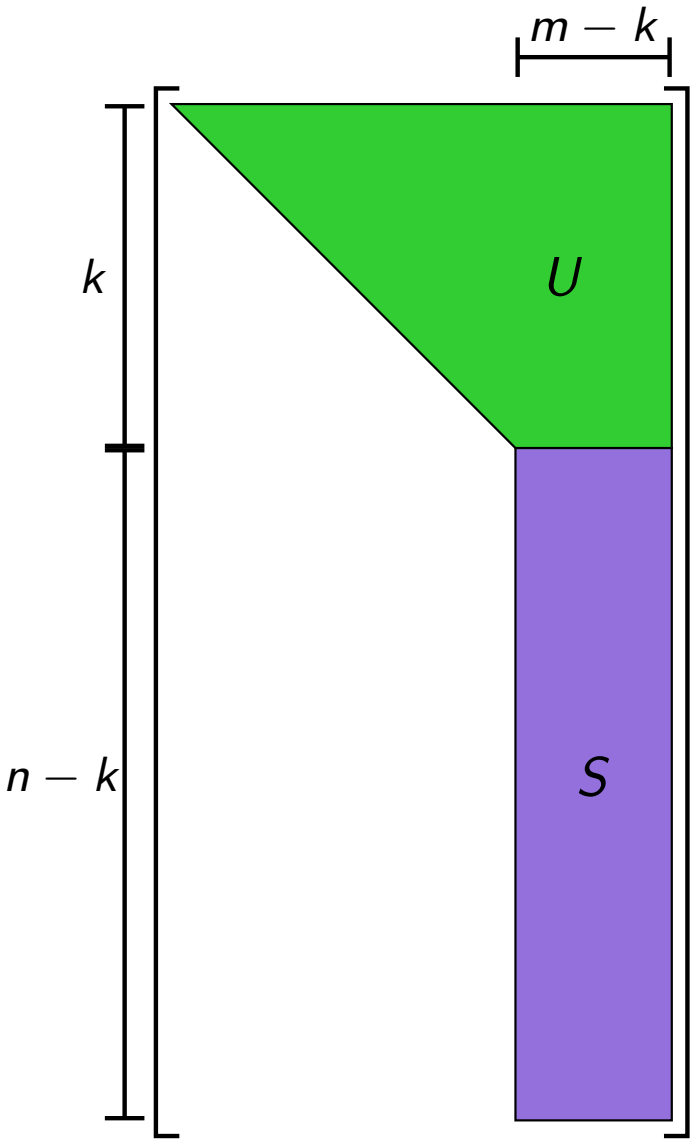


Schur Complement:

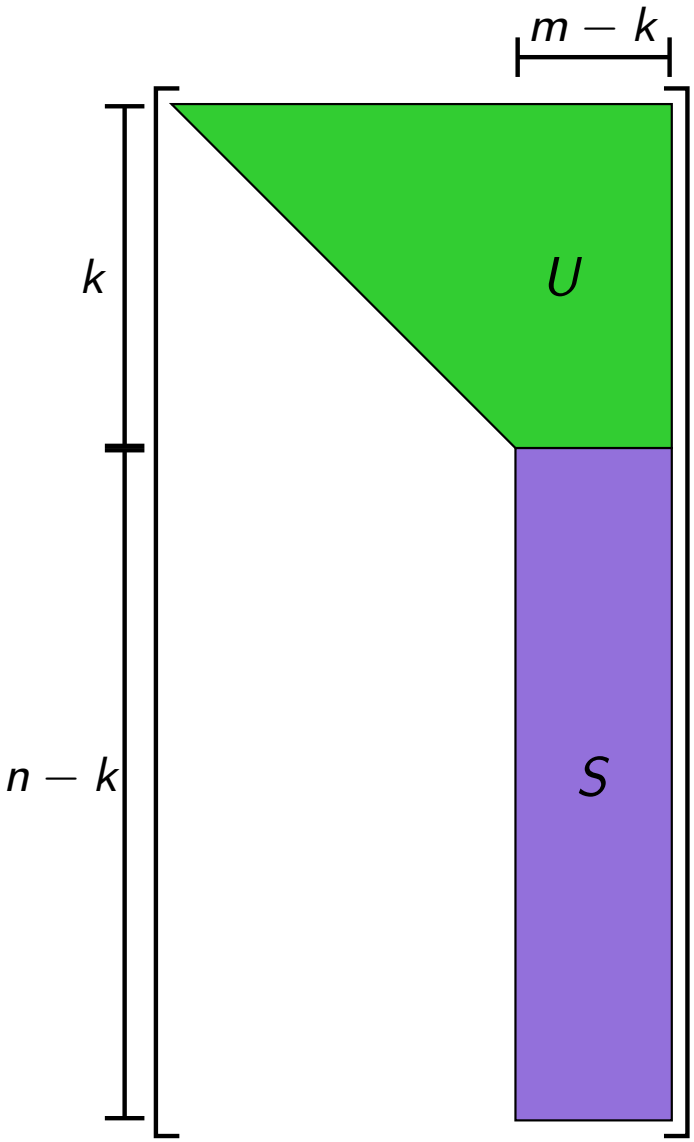
On hpac.imag.fr

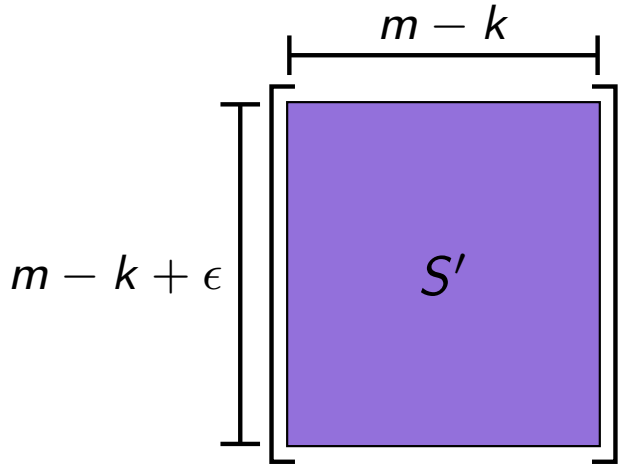
N threads	T (in s)	speedup	efficiency
1	25		
2	13.8	1.81	90.58 %
4	7.2	3.47	86.81 %
8	3.6	6.94	86.81 %
16	1.9	13.16	82.24 %
32	1.1	22.73	71.02 %

Tall and Narrow Schur Complement



Tall and Narrow Schur Complement





Idea

- Build a dense matrix S'
- The rows of S' are dense random linear combinations of the rows of S.
- Compute the rank of S'.

Example:The case of Relat/relat9

About relat9:

- Size: 9746232×274667
- Density : 0.0016
- Rank : 2745574

Results using an improved pivots selection heuristic

- Size of S: 9473991×2426
- Density of S: 0.9
- Dense rank computation on hpac.imag.fr: 22s

Performance of the Hybrid Algorithm (CASC 2016)

Experiments carried on an Intel Core i7-3770 with 8 GB of RAM
Experiments carried on all matrices from SIMC with integer coefficents
Only one core used

Our benchmarks show:

- Hybrid can compute the rank of a large set of matrices
- Hybrid is always faster than the right-looking algorithm (LinBox)
- Hybrid is almost always faster than GPLU
- Hybrid is faster than the Wiedemann algorithm on some big and very sparse matrices

New: Improvement of the Pivots Selection Heuristic

Why

- Enable us to reduce the number of the elimination steps
- Keep U sparse $\Rightarrow$ fast elimination steps

New: Improvement of the Pivots Selection Heuristic

Why

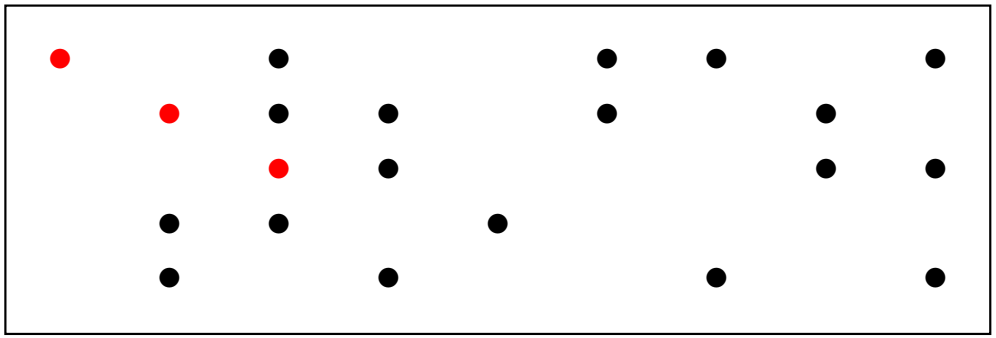
- Enable us to reduce the number of the elimination steps
- Keep U sparse $\Rightarrow$ fast elimination steps

How

- Find the largest set: NP-Complete
- Find a large set using heuristics

Our idea: First find the F.-L. pivots, then search for more.

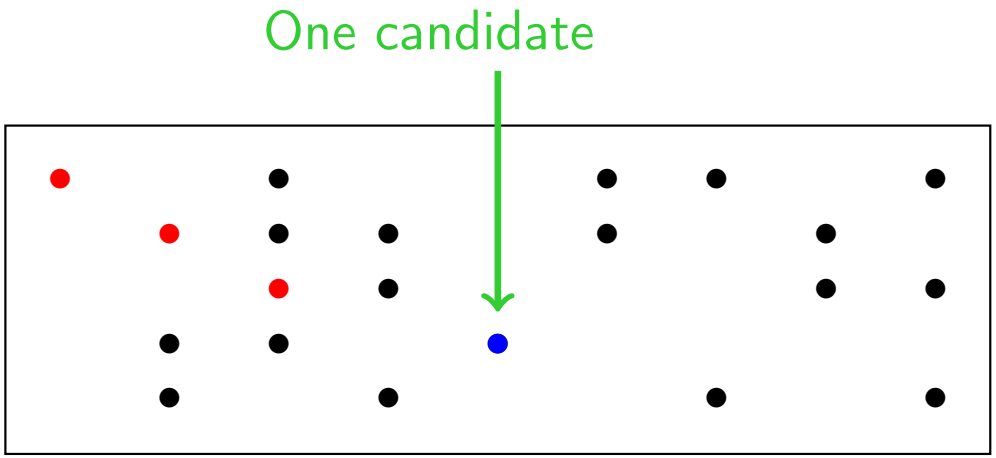
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

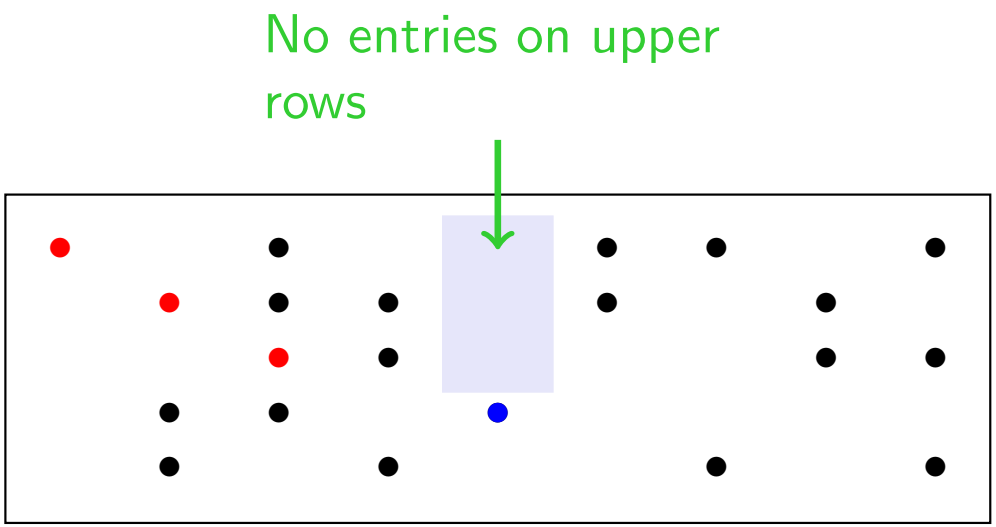
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

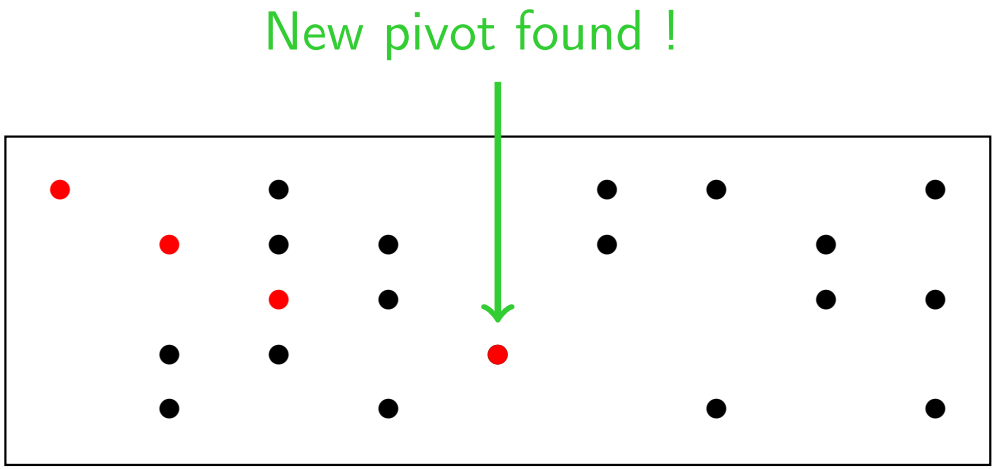
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

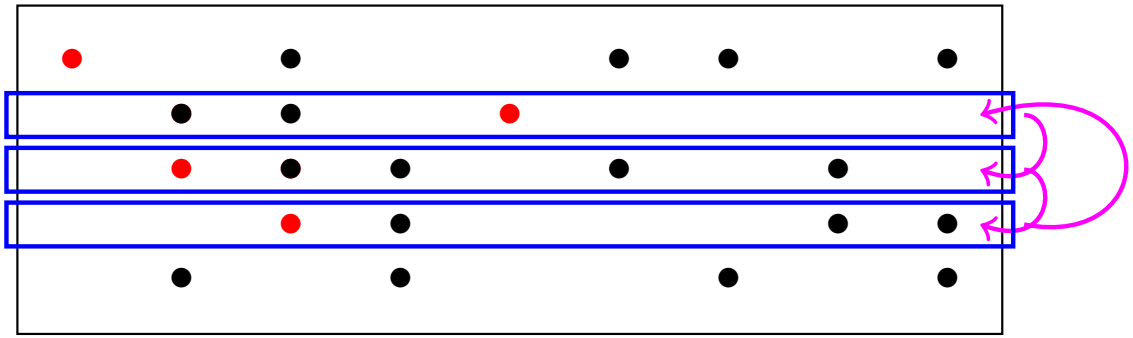
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

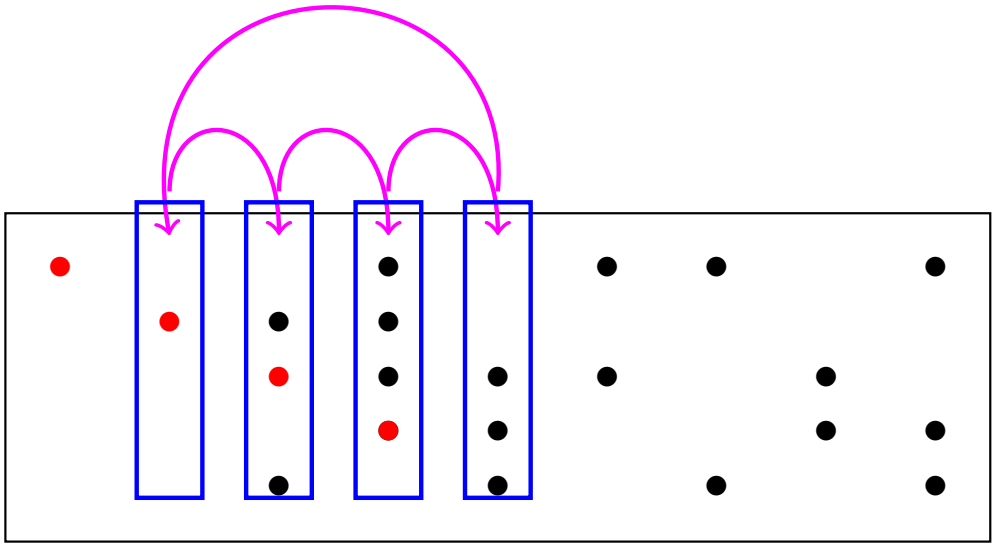
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

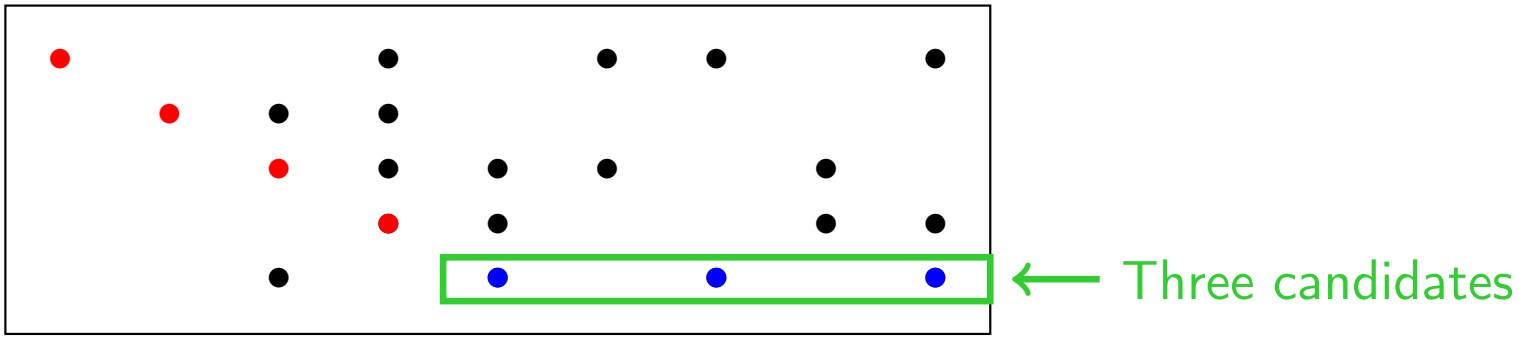
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

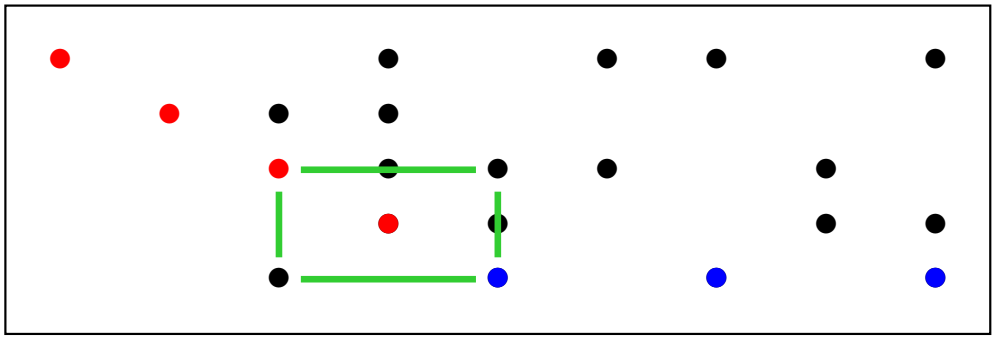
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

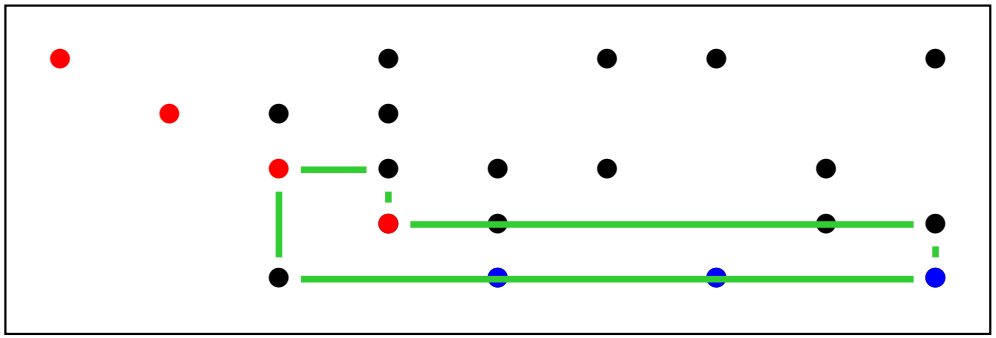
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

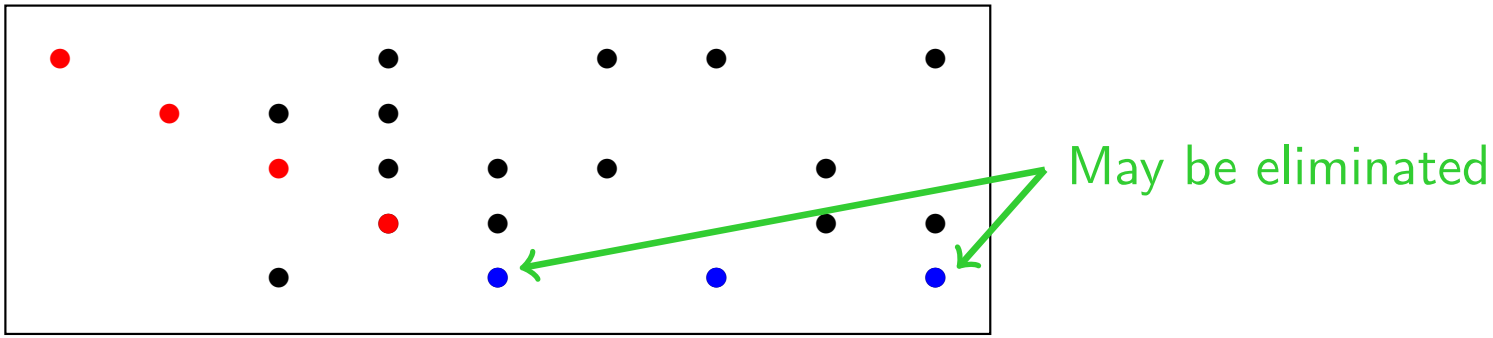
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

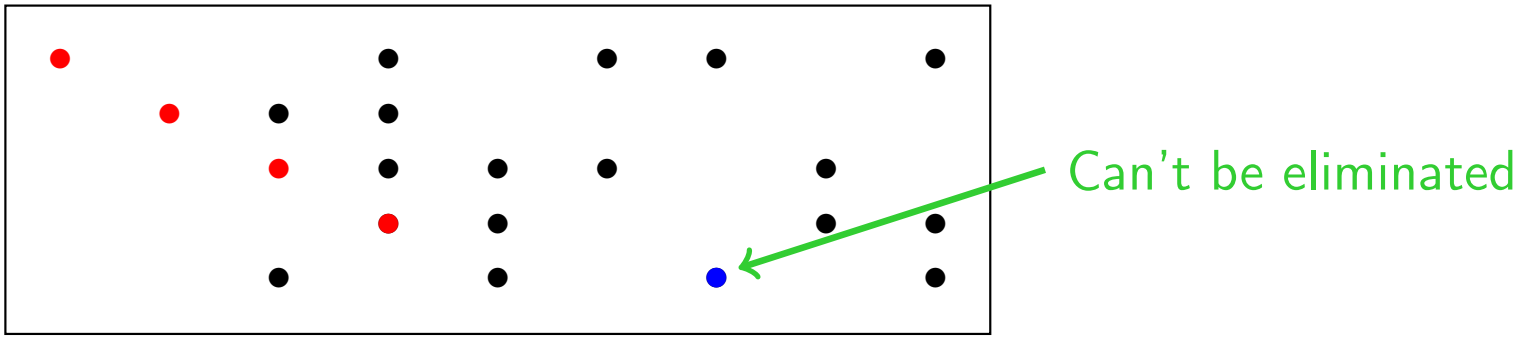
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

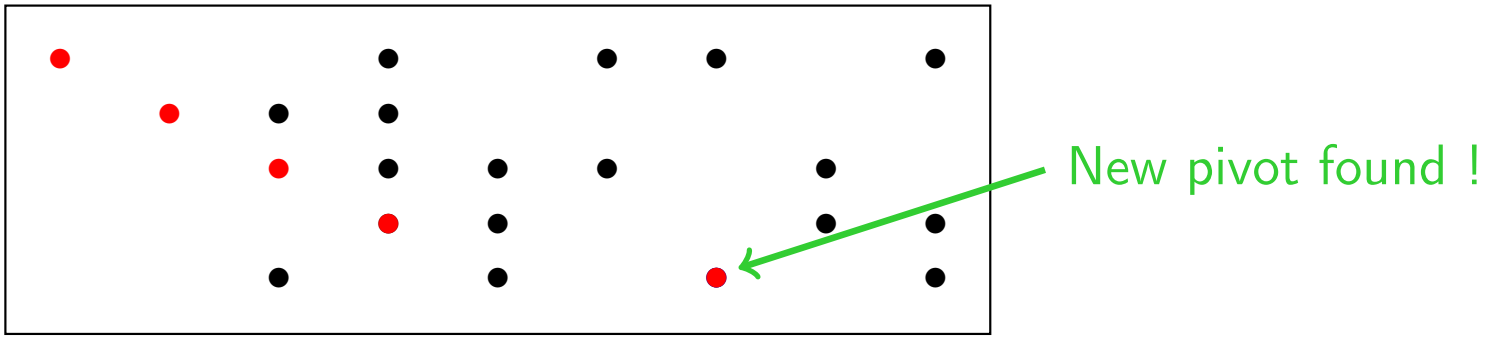
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

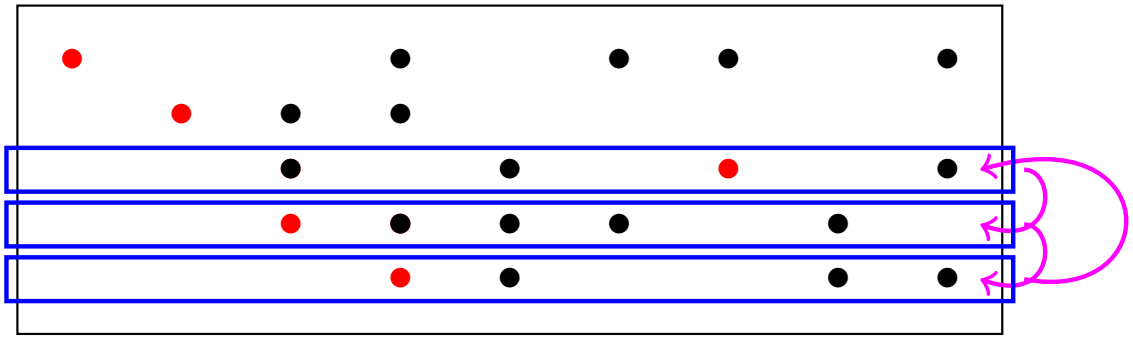
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

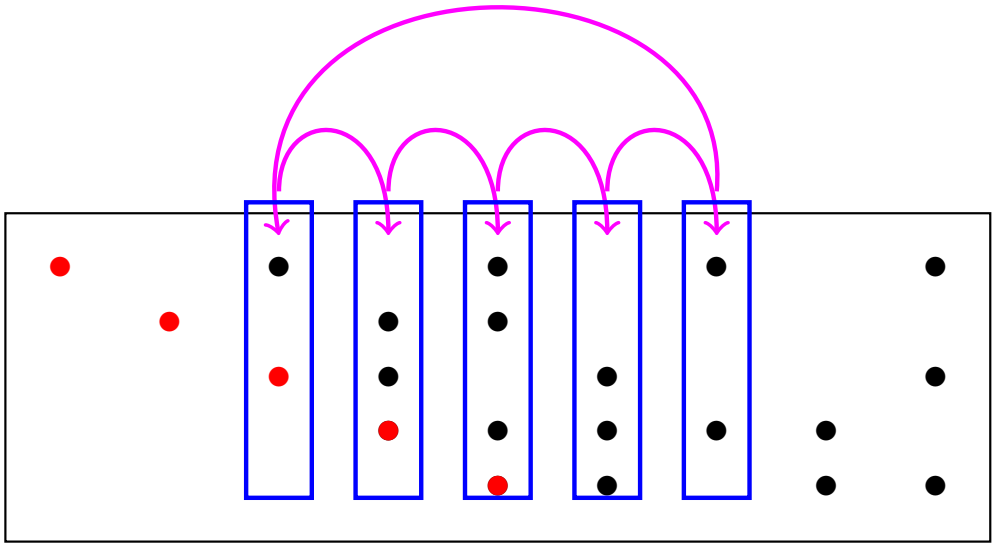
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

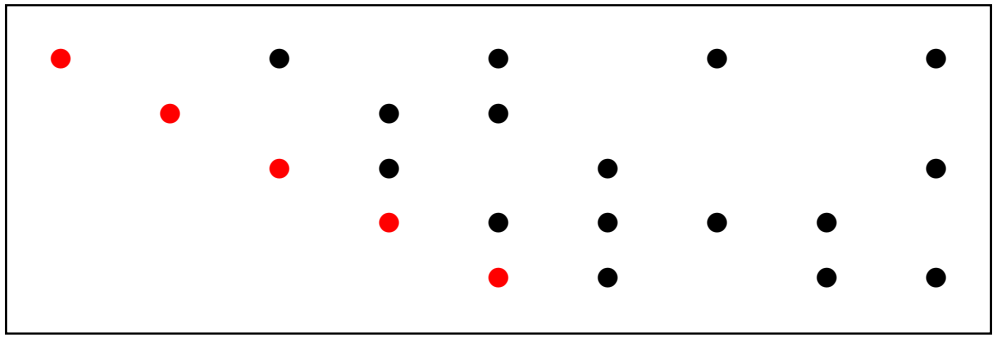
How to Find Remaining Pivots:



Choice of new pivots:

The pivotal submatrix must form a DAG $\Rightarrow$ If there is a cycle, the entry can't be selected.

Our new pivot selection algorithm



In a Nutshell:

- Find the F.-L. pivots
- Push the F.-L. pivots on the top of A.
- For all non-pivotal column j, if the upmost coefficient a_{ij} is on a non-pivotal row, select a_{ij}
- Perform a graph search to find more pivots

Graph Search

Idea

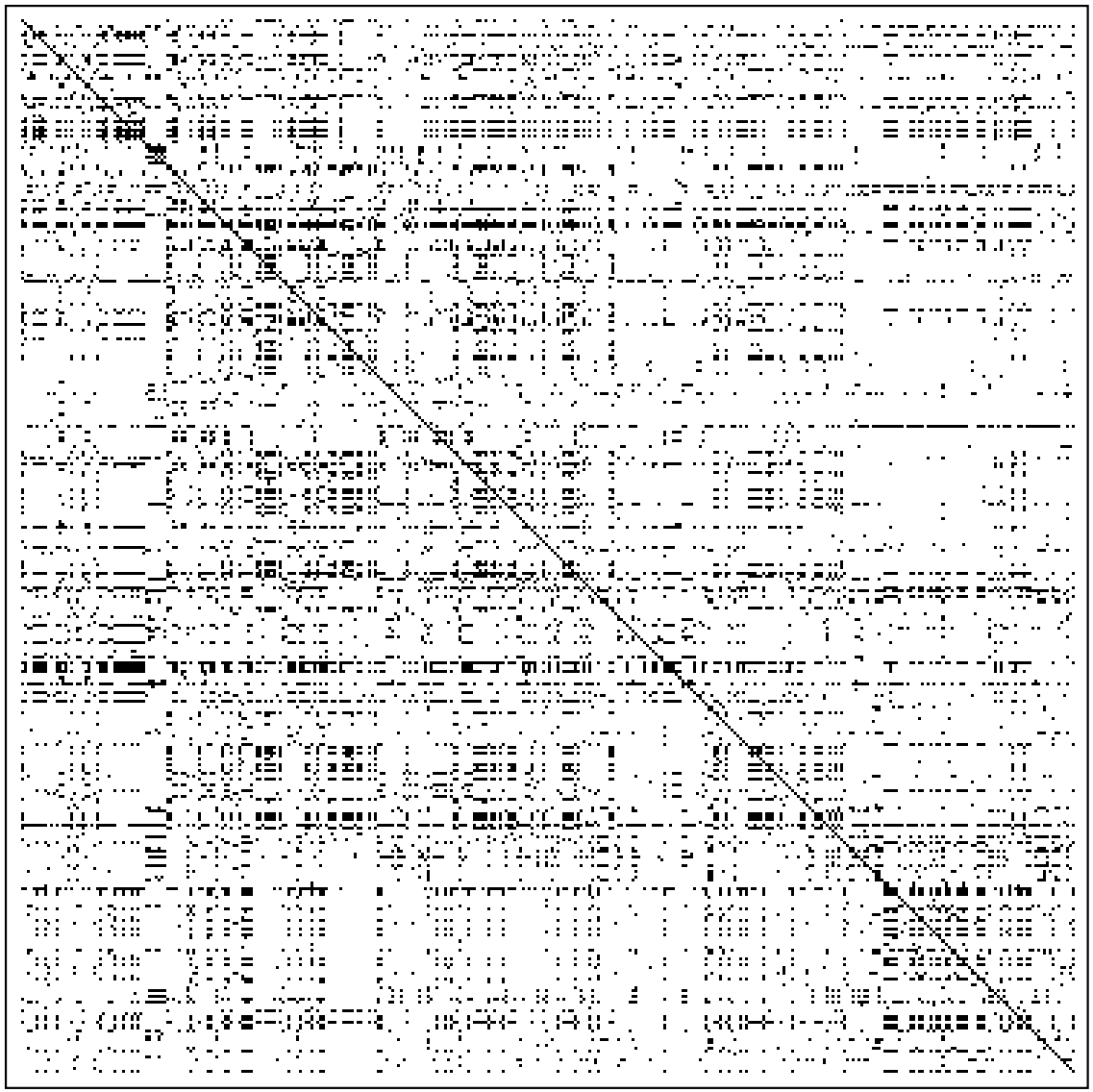
For each non pivotal row i:

- Start a BFS from each pivotal column j such that $a_{ij} \neq 0$
- When a column k such that $a_{ik} \neq 0$ is reached, remove a_{ik} from the list of potential pivots.
- If there are still candidates after the BFS end, we select the leftmost one.

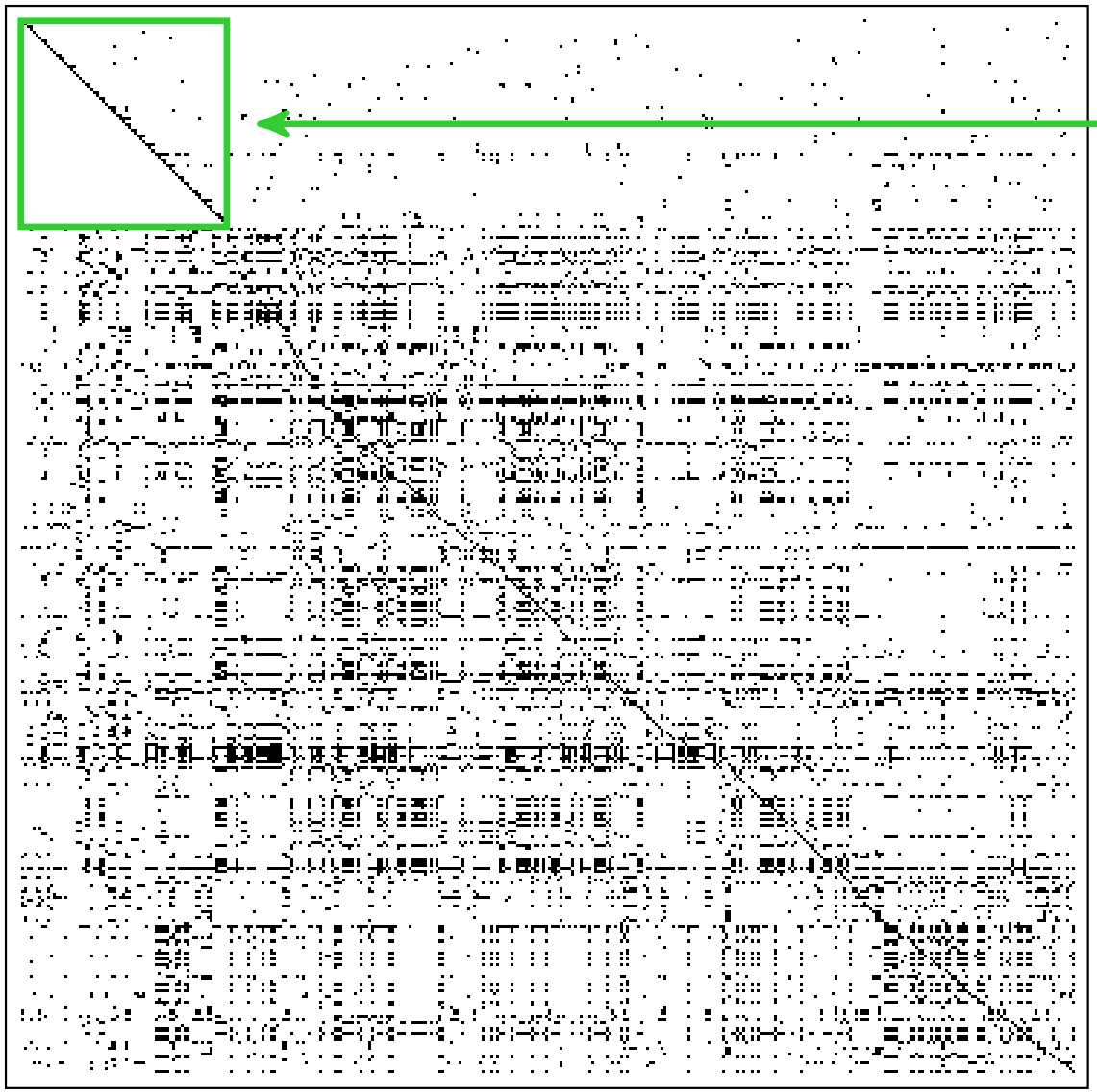
Remark

- This method is parallelizable.
- The worst case complexity of the search is $\mathcal{O}(n|A|)$ (same as Wiedemann)
- In practice : much faster than Wiedmann

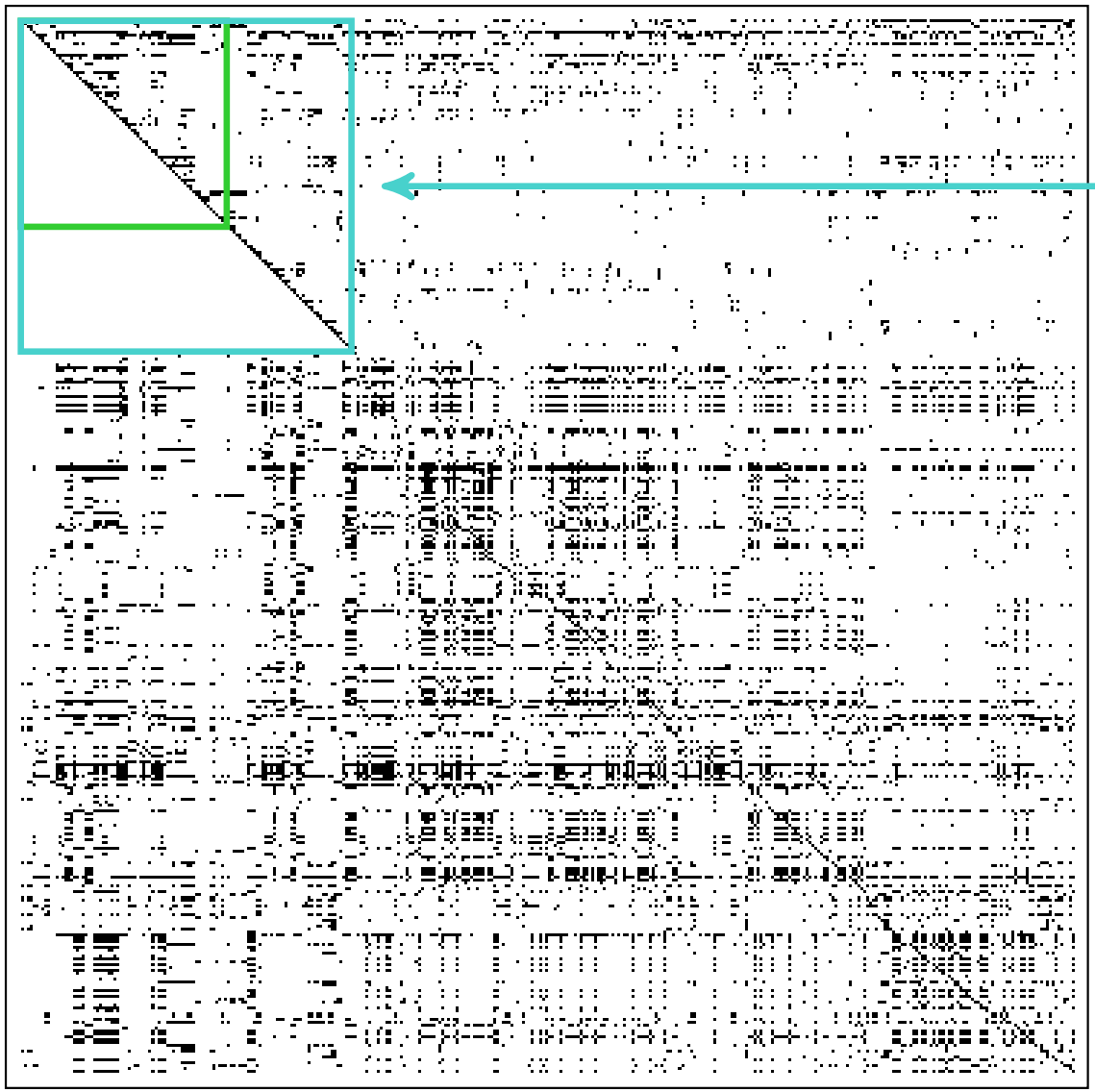
Example: GAG/mat364



Example: GAG/mat364

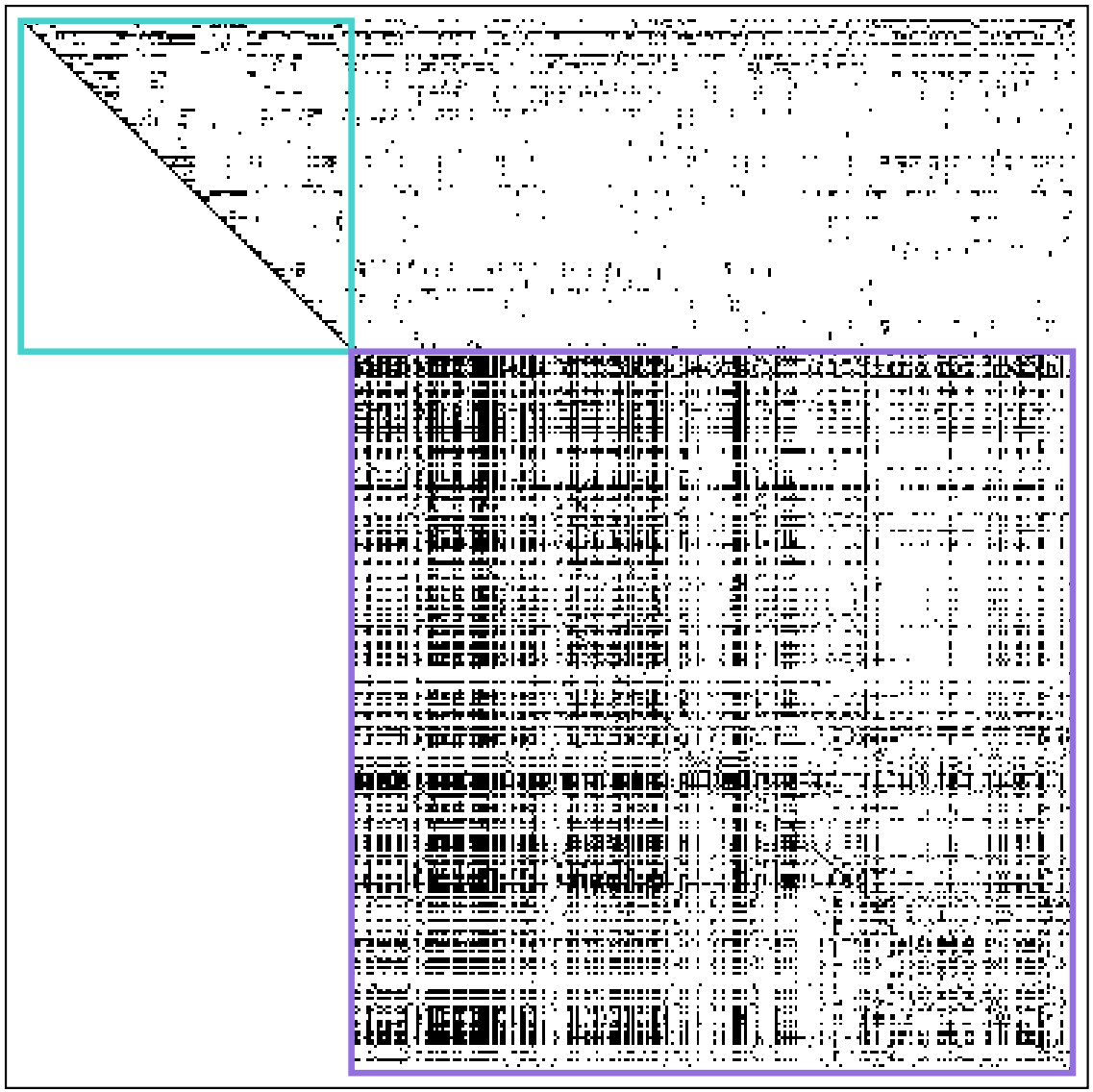


Example: GAG/mat364



More pivots after BFS

Example: GAG/mat364



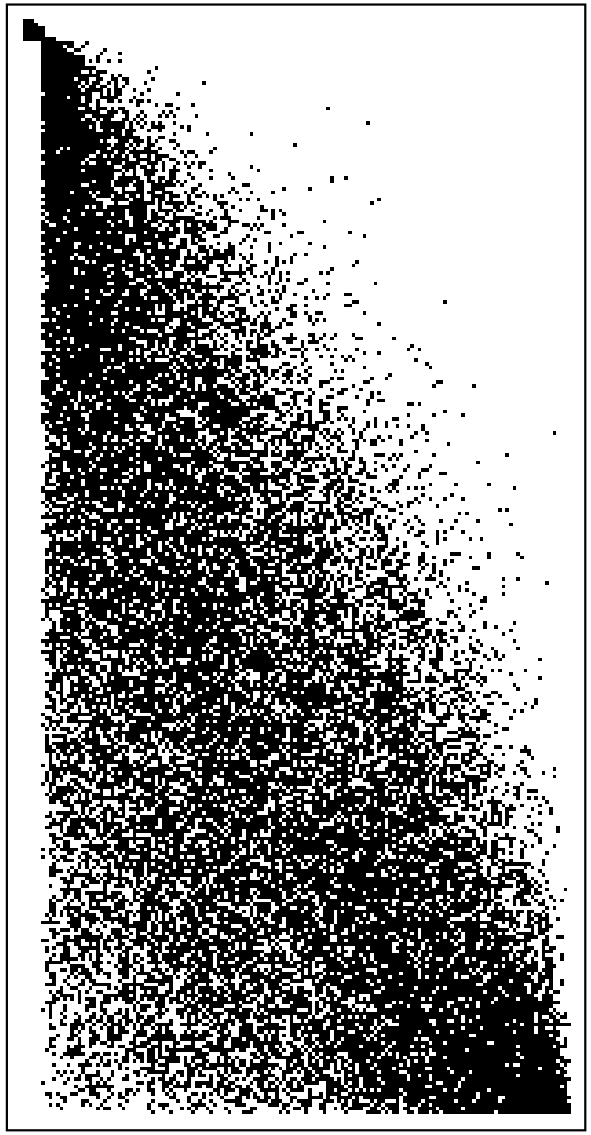
← Schur Complement

Some useful heuristics

To improve the performance, in some case we can:

- transpose the matrix
- swap the matrix

Example: GL7/GL7d12

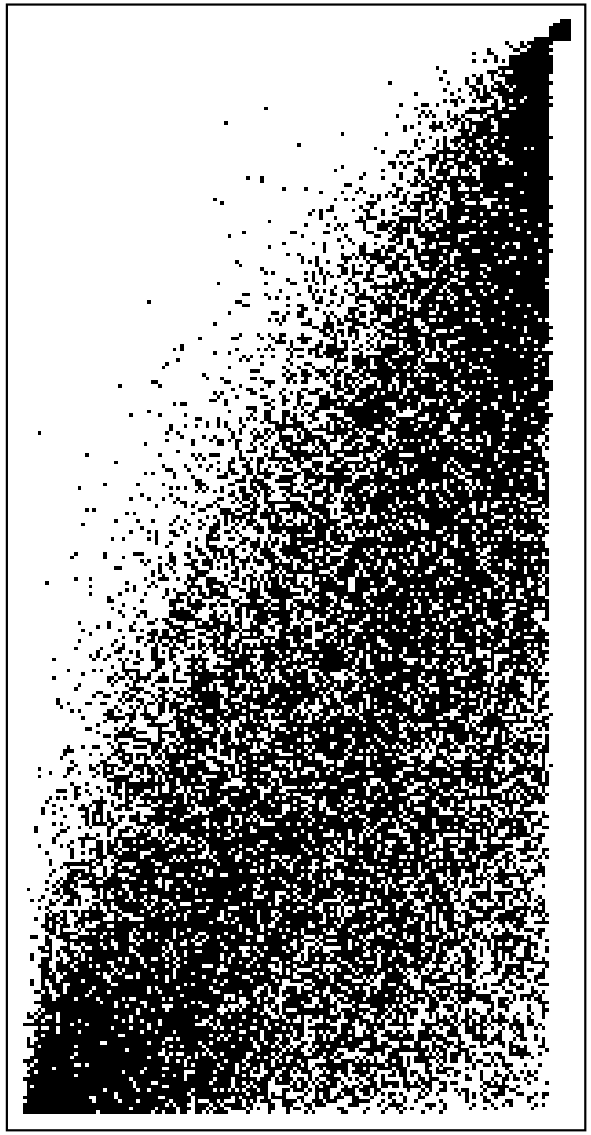


Some useful heuristics

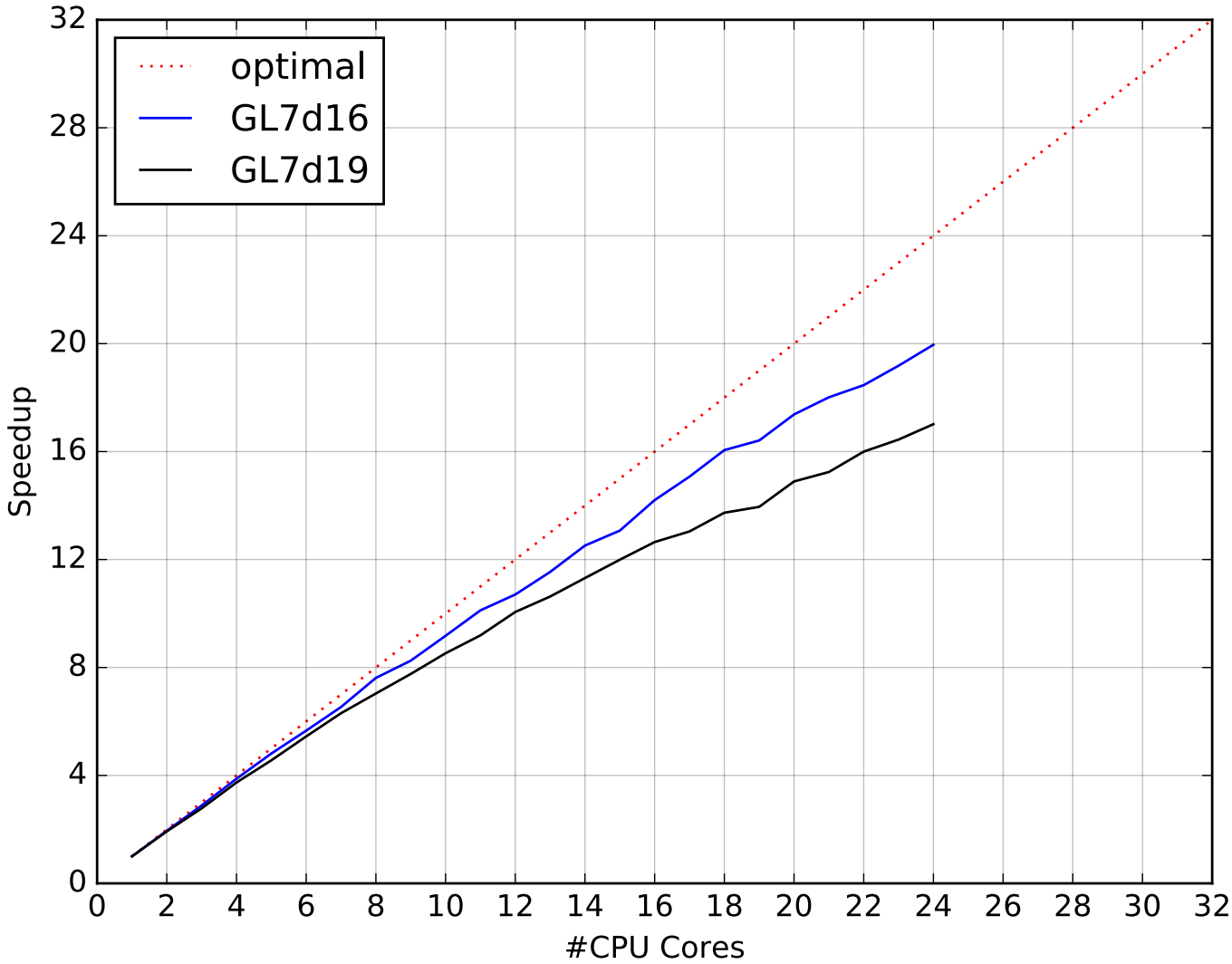
To improve the performance, in some case we can:

- transpose the matrix
- swap the matrix

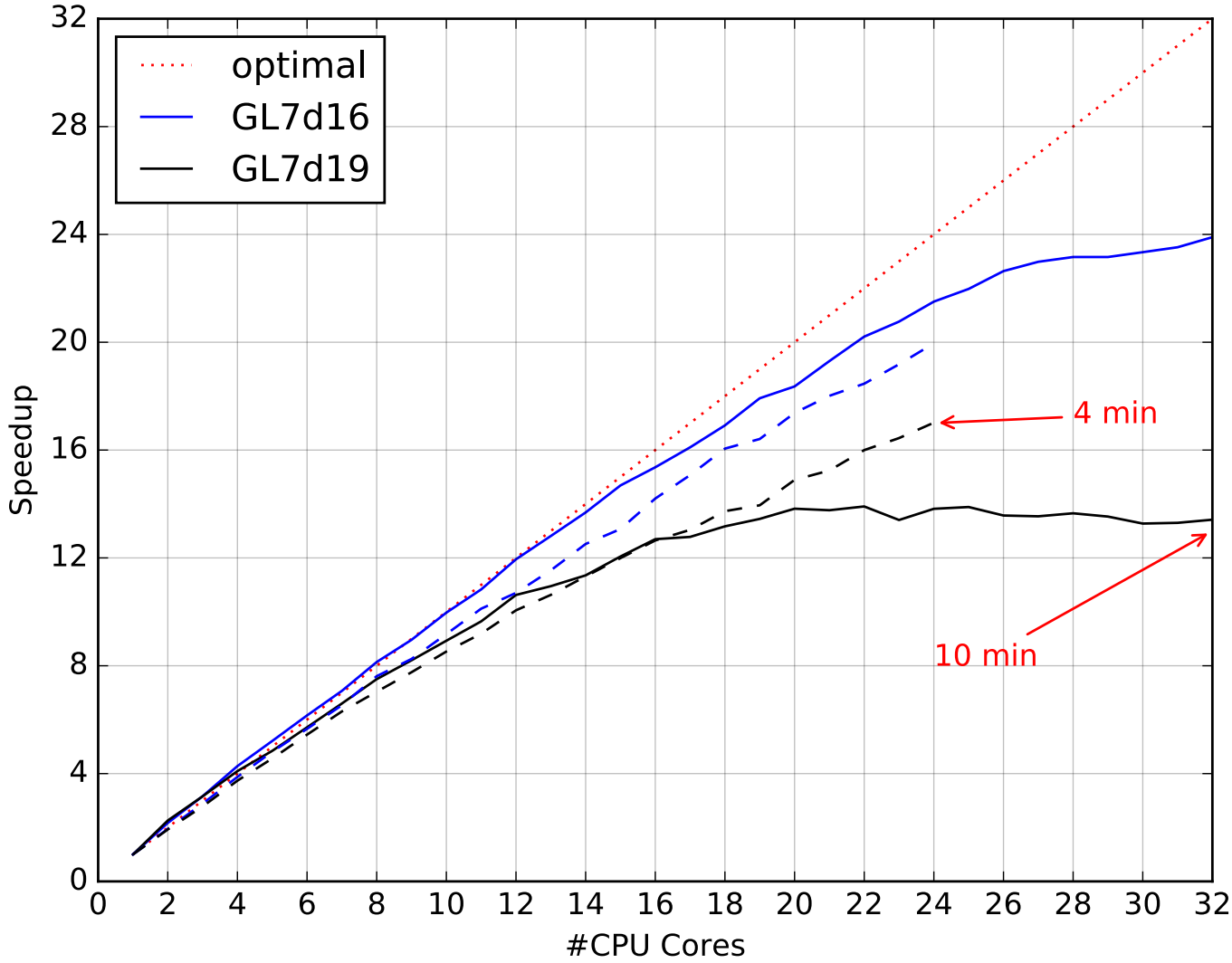
Example: GL7/GL7d12



Machine	# CPU	# cores	Cache
Lille-1	2× Xeon E5-2670 v3	12	30Mb



Machine	# CPU	# cores	Cache
Lille-1	2× Xeon E5-2670 v3	12	30Mb
hpac.imag.fr	4× Xeon E5-4620	8	16Mb



Conclusion

- Implemented in C and C++ in the SpaSM (SPArse System Modulo p) library and publicy available at:

`https://github.com/cbouilla/spasm`

Conclusion

- Implemented in C and C++ in the SpaSM (SPArse System Modulo p) library and publicy available at:

`https://github.com/cbouilla/spasm`

Open questions:

- Can we still improve the pivots selection?
- Is it possible to adapt this method for cryptanalysis purpose?

Thank you for your time !