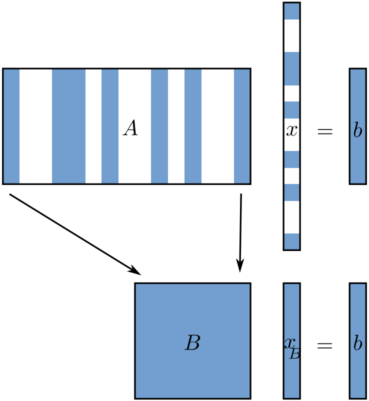
Permutations in the factorization of LP bases

Laurent Poirrier*

Joint work with Ricardo Fukasawa*

* Combinatorics & Optimization, UWaterloo

$$
\begin{aligned}
\min \quad & c^T x \\
\text{s.t.} \quad & A x = b \\
& x \geq 0
\end{aligned}
\tag{LP}
$$



A
x
$=$
b
B
x_B
$=$
b

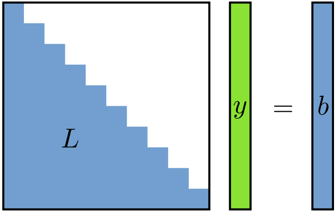
We want $\boldsymbol{x}$ in

$$B\boldsymbol{x} = b.$$

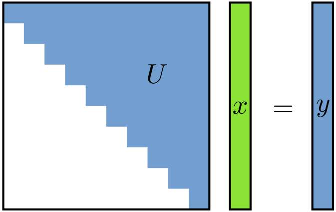
We find L, U triangular s.t. $LU = B$

$$\begin{aligned} LU\boldsymbol{x} &= b \\ L\underbrace{(U\boldsymbol{x})}_{\boldsymbol{y}} &= b \end{aligned}$$

$$L\boldsymbol{y} = b$$



$$U\boldsymbol{x} = y$$



Why a custom LU code for LP?

B is **not**:

- symmetric and positive definite
- block structured
- band structured

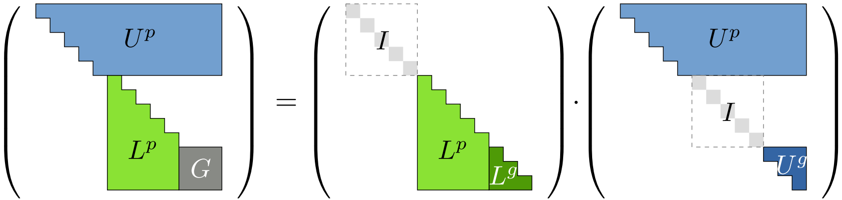
B is a submatrix of A:

Property P1: "partially" triangular

Property P2: one column changes after each iteration

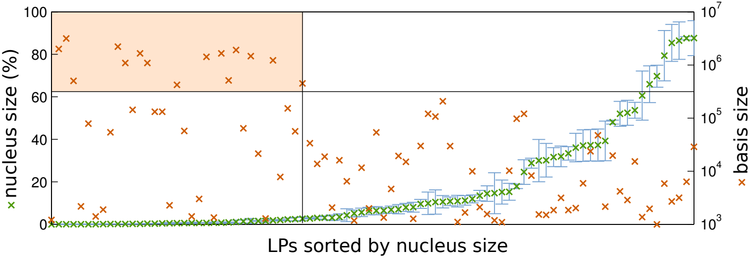
P1. Partially triangular permutation

$\exists$ permutations P, Q s.t. $PBQ =$



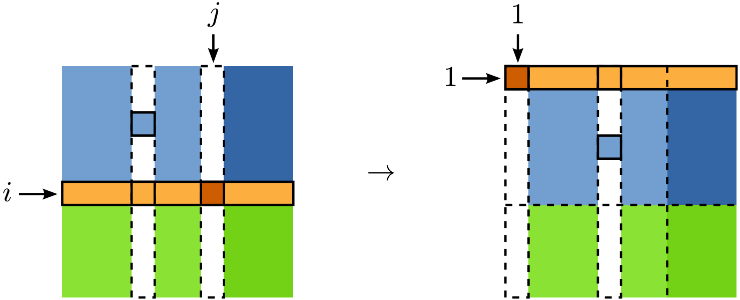
[See e.g. "Computing Sparse LU Factorizations for Large-Scale Linear Programming Bases", Suhl, Suhl, 1990]

P1. Size of the nucleus G



["On the factorization of simplex basis matrices", Luce, Duintjer Tebbens, Liesen, Nabben, Grötschel, Koch, Schenk, 2009]

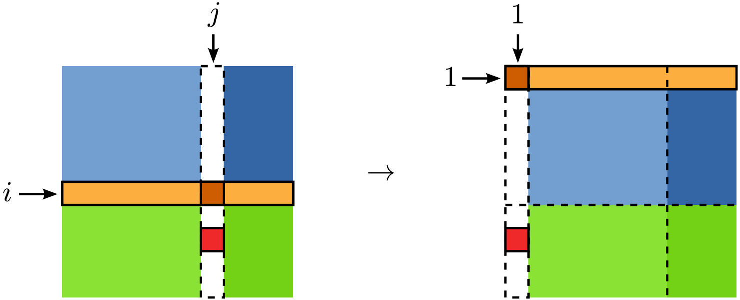
P1. Finding a partially triangular permutation



Proposition:

Permuting singletons to the front yields a minimal size nucleus.

P1. Finding a partially triangular permutation

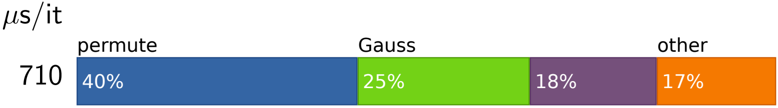


Proposition:

Permuting singletons to the front yields a minimal size nucleus.

P1. Factorization time

MIPLIB 2010 root nodes:

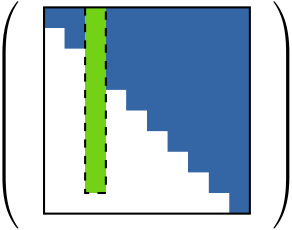


$O(m\ n_{\mathbf{nz}})$ $O(m\ n_{\mathbf{nz}}^2)$

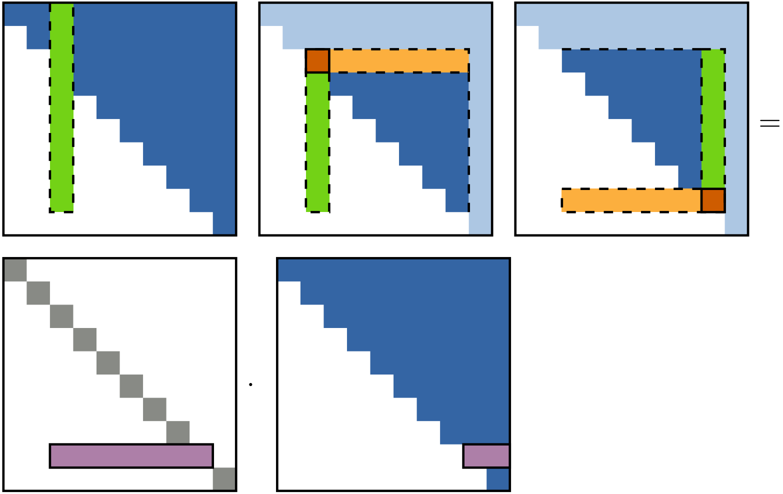
P2. Forrest-Tomlin

$$
\begin{aligned}
B' &= B - Be_ie_i^T + a_je_i^T \\
&= LU - LUe_ie_i^T + a_je_i^T \\
&= L\left(U - Ue_ie_i^T + L^{-1}a_je_i^T\right) \\
&= L\ U'
\end{aligned}
$$

$U' =$



P2. Forrest-Tomlin



P2. Forrest-Tomlin, after k iterations

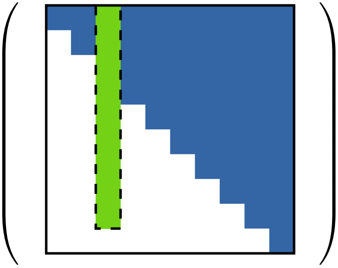
$$B = L \underbrace{\eta_1 \cdots \eta_k}_{H} U$$

$$\begin{aligned}
B' &= B &- \quad & Be_i e_i^T &+ \quad & a_j e_i^T \\
&= LHU &- \quad & LHUe_i e_i^T &+ \quad & a_j e_i^T \\
&= LH \Big(U &- \quad & Ue_i e_i^T &+ \quad & (LH)^{-1} a_j e_i^T \Big) \\
&= LH \; U'
\end{aligned}$$

P2. Refactorization time

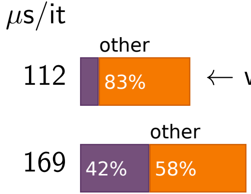
Further improvement...

$U' =$



Reid's idea

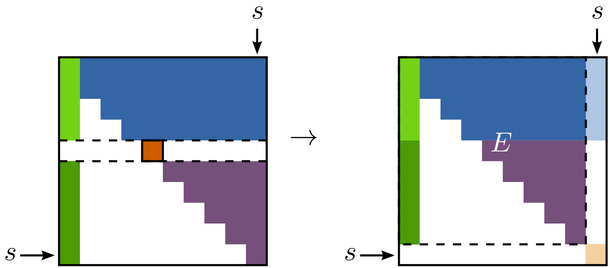
- Finds a triangular permutation: 58% of iterations
- Running time:



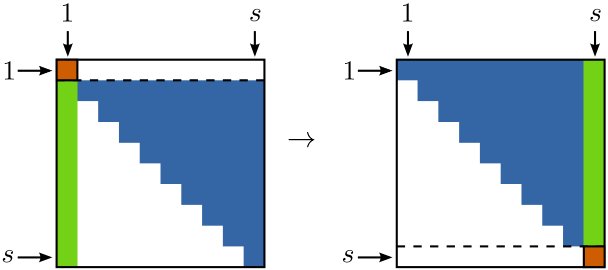
$\leftarrow$ with Forrest-Tomlin

$\leftarrow$ with Forrest-Tomlin + Reid's permute

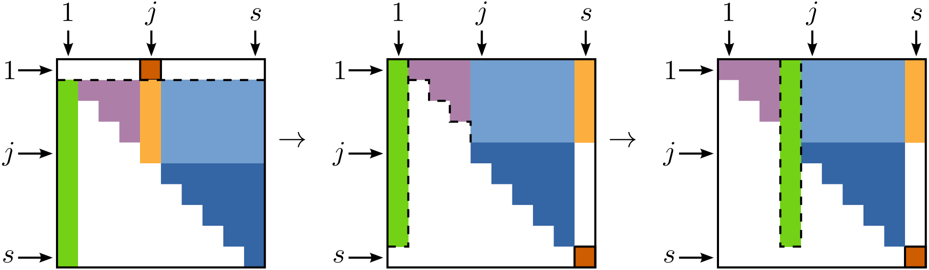
Assumption: no row-singletons in $2, \ldots, s$



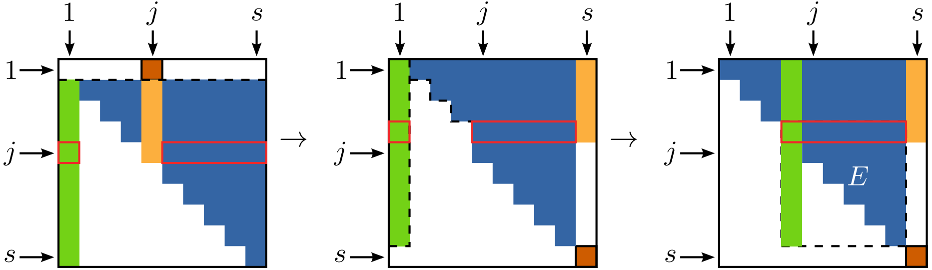
Case 1: E_{11} row-singleton



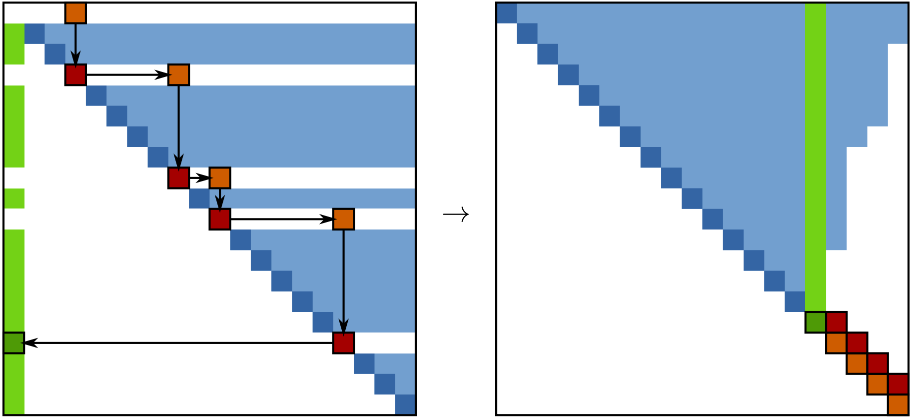
Case 2: E_{1j} row-singleton



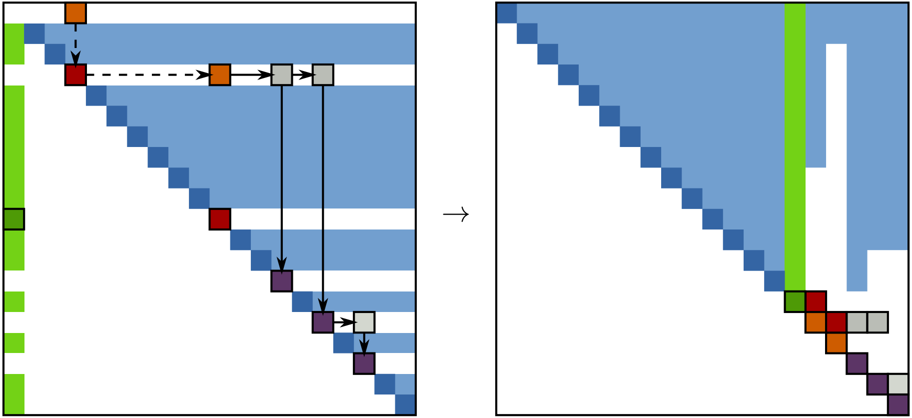
Case 2: E_{1j} row-singleton



One-path

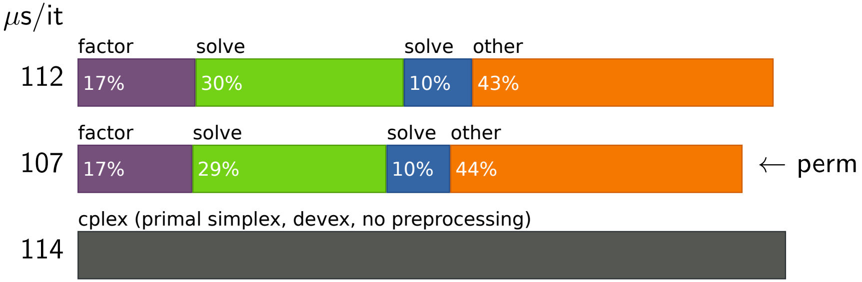


Dropping the assumption



Results

- Finds a triangular permutation: 58% of iterations
- Running time:



Why?

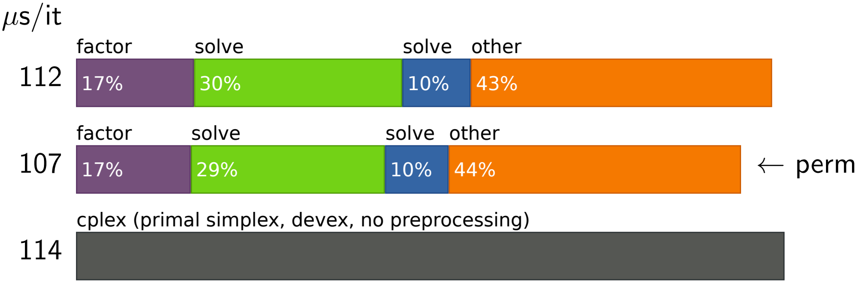
- Avoids Forrest-Tomlin
- Fewer η matrices
- Sparser rhs

Summary

- A sparse method for permuting out the spike.
- Can replace Forrest-Tomlin in 58% of iterations.
- Decreases density of triangular solve intermediate vectors.

Results

- Finds a triangular permutation: 58% of iterations
- Running time:



Benchmark framework

Instances:

MIPLIB 2010 benchmark set root nodes, after CPLEX MIP preprocessing.

In CPLEX:

- disable LP preprocessing
- force devex pricing
- force primal simplex

Comparison unfairness

Our code:

- trained and tested on same instances, same computer
- finetuned for minimal time per iteration
- does more iterations (devex? tradeoff on sparsity?)
- all-logical starting basis (sparser!)

CPLEX:

- unusual setting (no LP preprocessing, forced devex and primal)
- stricter constraints on robustness (numerical accuracy)

Important implementation caveat

Updating the permutation vectors:

- asymmetric row/column permutations $\rightarrow \sim 2\times$ more work
- sliding the mapping needs to be implemented carefully!
  the number of pivots is usually small $\rightarrow$
  - sort pivots
  - innermost loop: slide chunks of contiguous indices by a constant number

  This is like in F-T, but in F-T there is only one chunk and we slide indices by exactly 1.