Permutations in the factorization of LP bases

Laurent Poirrier*

Joint work with Ricardo Fukasawa*

* Combinatorics & Optimization, UWaterloo
\[
\begin{align*}
\text{min} & \quad c^T x \\
\text{s.t.} & \quad A x = b \\
& \quad x \geq 0
\end{align*}
\]
We want $x$ in

$$Bx = b.$$ 

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

$$LUx = b$$

$$L(Ux) = y$$

$$Ly = b$$  \hspace{1cm}  | \hspace{1cm}  Ux = 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
There exist permutations $P, Q$ such that $PBQ = U^p$

\[
\begin{pmatrix}
U^p \\
L^p \\
G
\end{pmatrix}
= \begin{pmatrix}
I \\
L^p \\
L^g
\end{pmatrix}
\cdot \begin{pmatrix}
U^p \\
I \\
U^g
\end{pmatrix}
\]

[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:

\[ \mu s/it \]

- Permute: 40%
- Gauss: 25%
- Other: 17%

\[ O(m \ n_{nz}) \quad O(m \ n_{nz}^2) \]
P2. Forrest-Tomlin

\[
B' = B - Be_i e_i^T + a_j e_i^T
= LU - LU e_i e_i^T + a_j e_i^T
= L \left( U - U e_i e_i^T + L^{-1} a_j e_i^T \right)
= L \ U'
\]

\[
U' = \begin{pmatrix}
\end{pmatrix}
\]
P2. Forrest-Tomlin
P2. Forrest-Tomlin, after \( k \) iterations

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

\[
B' = B - Be_i e_i^T + a_j e_i^T
\]

\[
= LHU - LHU e_i e_i^T + a_j e_i^T
\]

\[
= LH \left( U - U e_i e_i^T + (LH)^{-1} a_j e_i^T \right)
\]

\[
= LH \ U'
\]
P2. Refactorization time

\[ \mu s/it \]

- Factorize: 710 μs, 83%
- Other: 112 μs, 17%

← with Forrest-Tomlin
Further improvement...

\[ U' = \begin{pmatrix} \end{pmatrix} \]
Reid’s idea

- Finds a triangular permutation: 58% of iterations

- Running time:

\[ \mu s/it \]

112

\begin{tabular}{c|c}
other & 83% \\
\end{tabular}

\[ \leftarrow \text{with Forrest-Tomlin} \]

169

\begin{tabular}{c|c}
other & 42% \\
58% & \\
\end{tabular}

\[ \leftarrow \text{with Forrest-Tomlin + Reid’s permute} \]
Assumption: no row-singletons in $2, \ldots, s$
Case 1: $E_{11}$ row-singleton
Case 2: $E_{1,j}$ row-singleton
Case 2: $E_{1j}$ row-singleton
One-path
Dropping the assumption
Results

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

<table>
<thead>
<tr>
<th>μs/it</th>
<th>factor</th>
<th>solve</th>
<th>solve</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>112</td>
<td>17%</td>
<td>30%</td>
<td>10%</td>
<td>43%</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>μs/it</th>
<th>factor</th>
<th>solve</th>
<th>solve</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>107</td>
<td>17%</td>
<td>29%</td>
<td>10%</td>
<td>44%</td>
</tr>
</tbody>
</table>

Why?

- Avoids Forrest-Tomlin
- Fewer $\eta$ 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:

\[ \mu s/it \]

<table>
<thead>
<tr>
<th></th>
<th>factor</th>
<th>solve</th>
<th>solve</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>112</strong></td>
<td>17%</td>
<td>30%</td>
<td>10%</td>
<td>43%</td>
</tr>
<tr>
<td><strong>107</strong></td>
<td>17%</td>
<td>29%</td>
<td>10%</td>
<td>44%</td>
</tr>
<tr>
<td>perm</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

cplex (primal simplex, devex, no preprocessing)

114
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.