Initial Basis Selection for LP Crossover

Christopher Maes
Edward Rothberg, Zonghao Gu, Robert Bixby

Sparse Days
CERFACS, Toulouse, France

June 6th 2014

GUROBI
OPTIMIZATION

Gurobi Optimization

- We build high-performance Linear Programming (LP)

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

- and Mixed Integer Linear Programming (MILP) solvers

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x_i \in \mathbb{Z}, \quad i \in \mathcal{I} \\ & x \geq 0 \end{array}$$

- Also have QP, MIQP, QCP/SOCP, and MIQCP solvers.

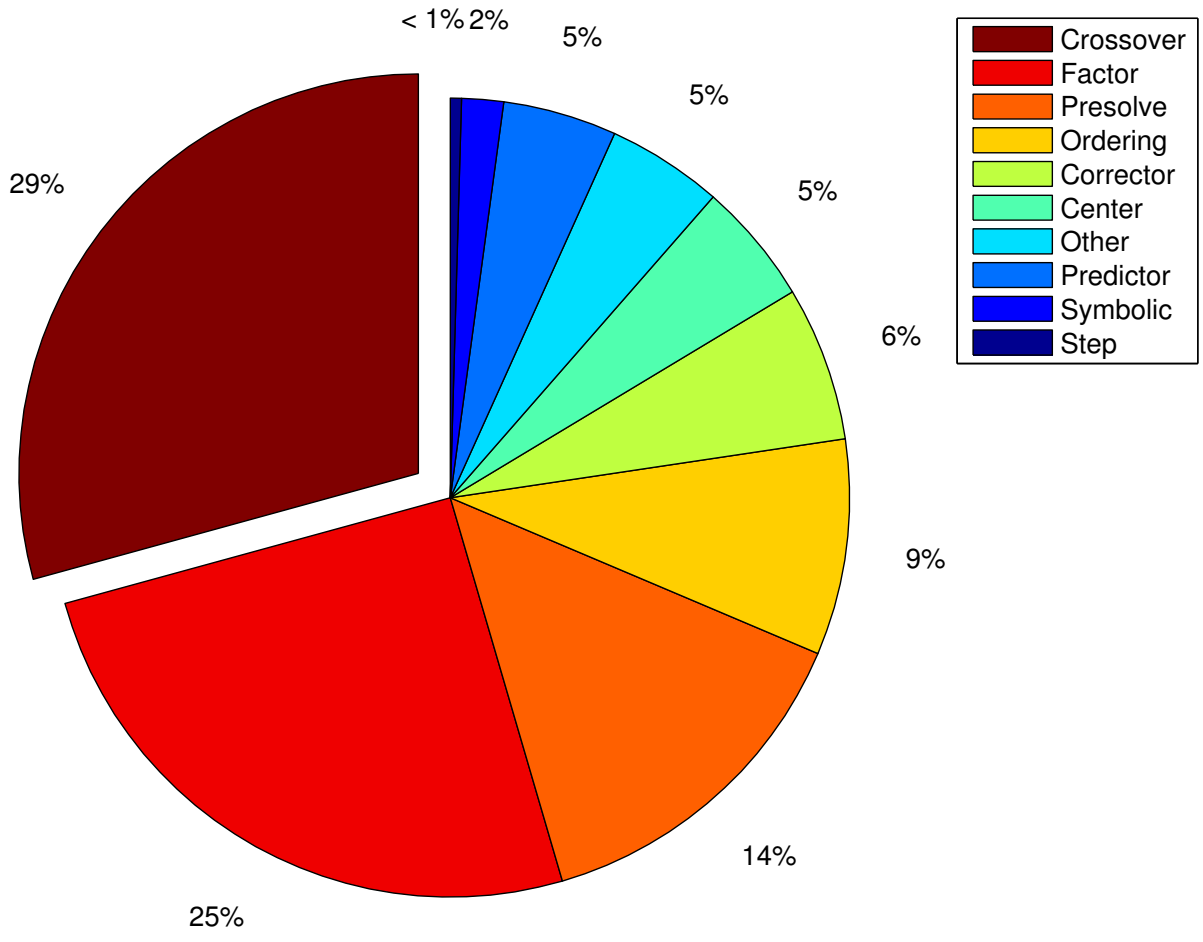
Crossover

- Post-processing algorithm used by barrier LP solver
- Computes an optimal basis and a vertex solution from an interior solution
- Enabled by default in concurrent and barrier optimizers (customer's prefer basic solutions)
- Crossover is numerically difficult

```
Warning:  2 variables dropped from basis
```

Crossover is expensive

- Breakdown of time spent in barrier
- Mean over models in internal test set with runtime $\geq 1s$



Crossover details

- Given an interior solution to

$$\begin{array}{ll} \text{minimize} & c^T x \\ \text{subject to} & Ax = b \\ & x \geq 0 \end{array}$$

  Crossover performs series of simplex-like "push" iterations to recover a vertex solution

- Primal push: (dual push is similar)
  - Begin with a square full rank basis B

$$Ax^\star = Bx_B + Sx_S = b$$

  - While there is a superbasic variable $x_j > 0$
  - Either push $x_j \to 0$ by adjusting x_B
  - Or move j into basis and push a basic variable $x_i \to 0$

- This talk is about how to compute the initial basis B

Choosing an initial basis

The ideal initial basis is

- Sparse
- Well-conditioned
- Close to the optimal basis

These properties are often in conflict:

- $B = I$ is sparse and well-conditioned — likely far from optimal
- $B^\star$ may be dense and ill-conditioned

Initial basis selection strategy:

- Begin with a set of candidate columns C

$$C = A(:, \texttt{candidates})$$

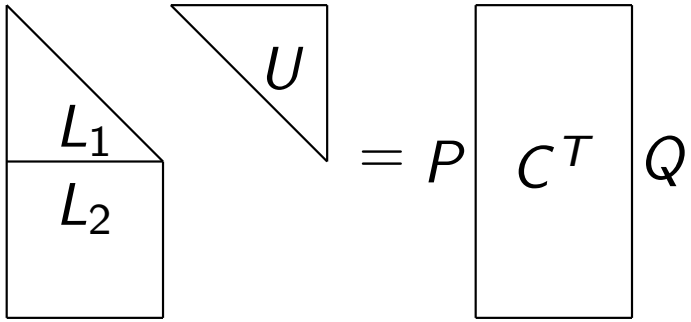
  whose corresponding variables have small reduced costs
- Compute a set of m independent columns from C
  - This is done via an LU factorization.

Crossover's old LU factorization

- Used a right-looking Markowitz method to factor C
- Pivots selected dynamically during factorization using a Markowitz strategy to minimize fill
- Used partial pivoting for stability within a column of A (C)
- Same technique as simplex LU factorization
- Separate implementation to handle rectangular matrices. Not as highly tuned.
- Used linked list data structure for fast row/column insertions

A different approach

- Perform left-looking factorization of C^T



- Use stable partial pivoting to pick the columns of A (rows of C^T) that will form the basis
- Precompute column ordering Q for sparsity
- Advantages:
  - Better control over rank and condition of basis
  - Left-looking method uses fast sparse column data structure
  - Static column ordering takes a global view to minimize fill
- Disadvantages:
  - Unable to determine number of non-zeros in pivot rows
  - Difficult to select pivot row dynamically for sparsity

Model cont4

```
Barrier statistics:
 AA' NZ     : 6.892e+05
 Factor NZ  : 4.440e+06 (roughly 100 MBytes of memory)
 Factor Ops : 6.156e+08 (less than 1 second per iteration)
 Threads    : 1

                 Objective                Residual
Iter       Primal          Dual         Primal    Dual     Compl     Time
0    2.84863101e+06 -1.20487984e+04  9.97e+06 0.00e+00  2.85e+05  2s
1    2.85637320e+06 -3.04865388e+05  1.55e+06 8.03e-03  3.15e+04  2s
...
23   6.24603630e-03  6.24602515e-03  4.66e-10 2.44e-15  7.32e-14  9s
24   6.24603158e-03  6.24603158e-03  9.31e-10 7.03e-14  7.32e-17  9s

Barrier solved model in 24 iterations and 9.19 seconds

Crossover basis: removed 0 dense columns or rows
  100777 variables added to crossover basis                    10s
  103597 variables added to crossover basis                    15s
  104172 variables added to crossover basis                    20s
  104958 variables added to crossover basis                    25s
```

The barrier ordering

- Under certain conditions, R from Cholesky factorization

$$R^T R = Q^T A A^T Q$$

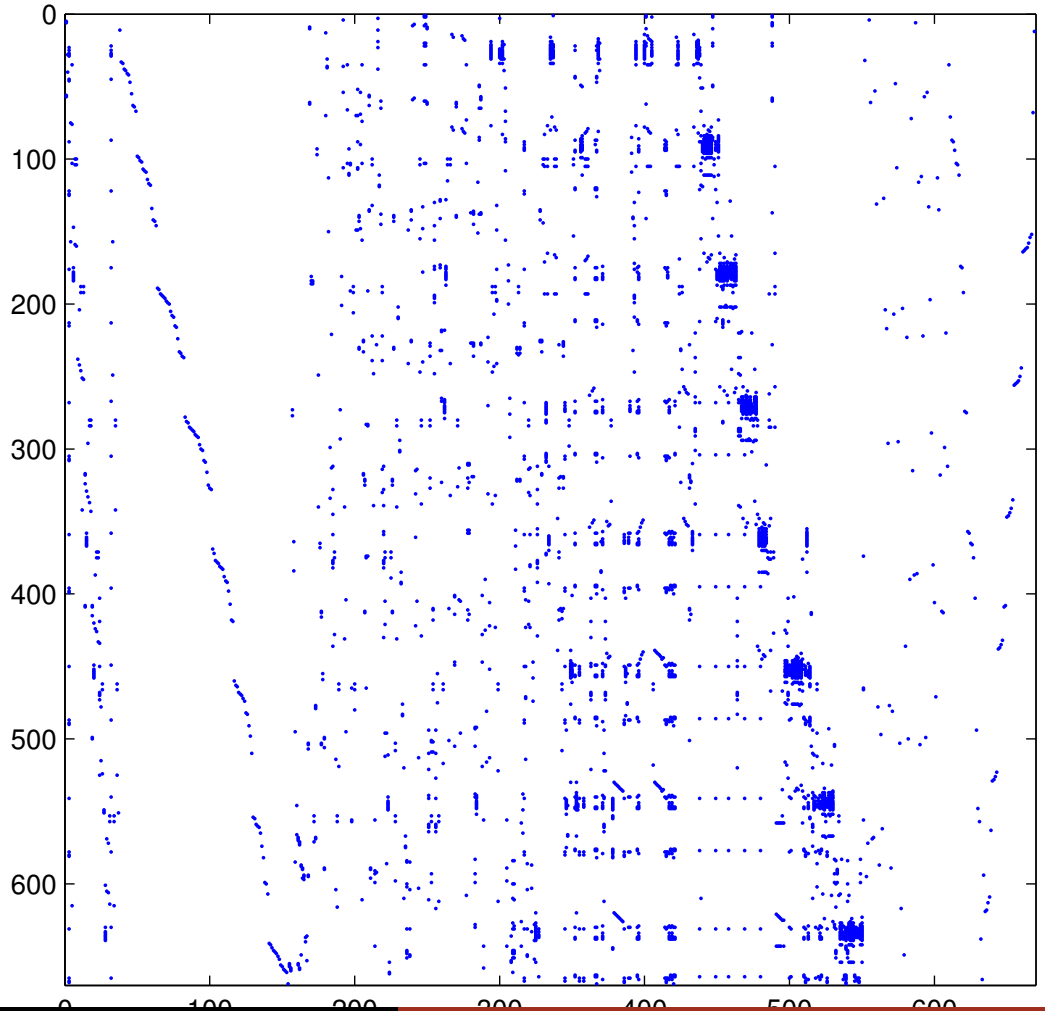
  is a loose upper bound on the non-zero pattern of U

- Already computed Q in barrier to reduce non-zeros in R

- Can we reuse this ordering for crossover LU?

Singletons

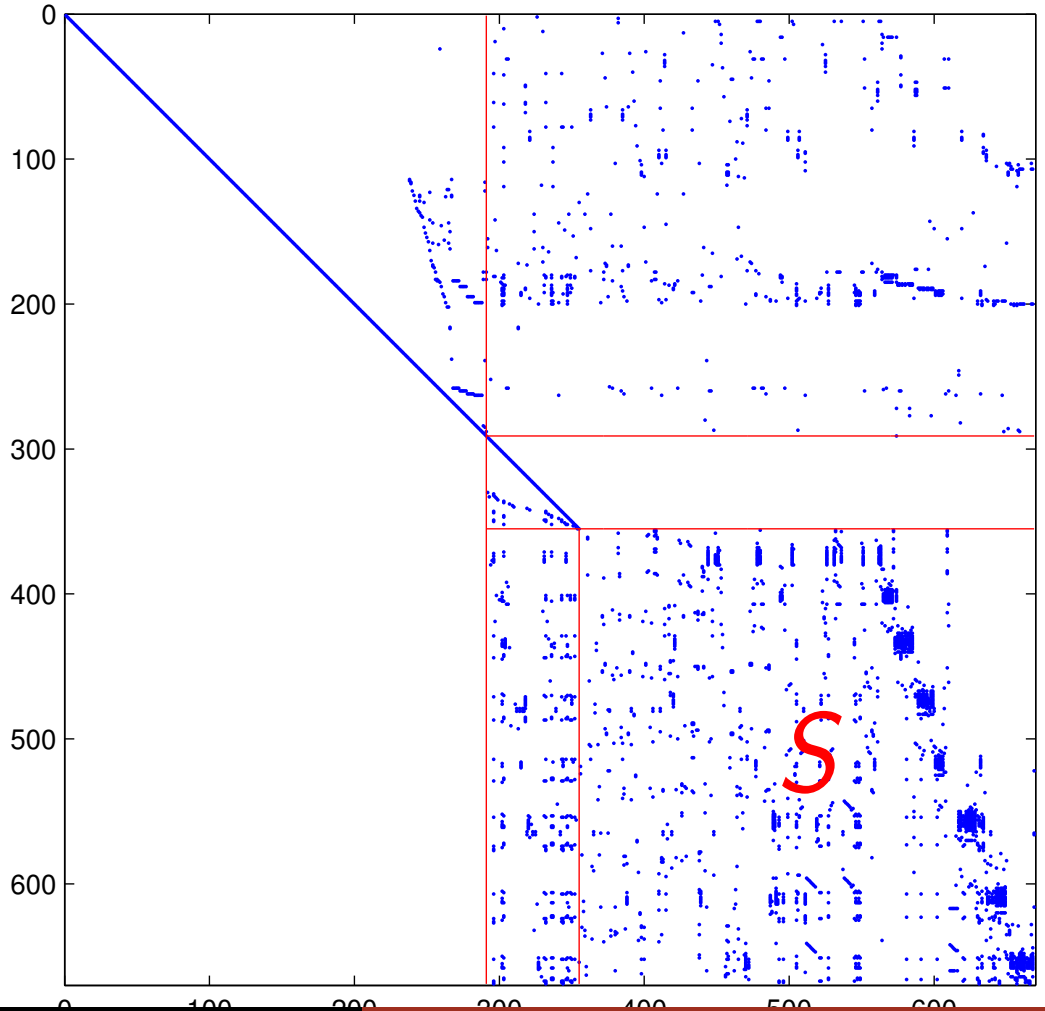
- Must first exploit special structure of LP matrices
- Once singletons are eliminated only S needs to be factored
- On average S is about 4X smaller than original matrix



Singletons

- Must first exploit special structure of LP matrices
- Once singletons are eliminated only S needs to be factored
- On average S is about 4X smaller than original matrix

P_s



Q_s

Ordering method

$$LU = PSQ$$

Column ordering Q

- Approximate minimum degree ordering of $S^T S$ with dense rows of S removed.
- Similar to Davis's COLAMD
- If S is column rank-deficient, may need to alter Q during the factorization

Row permutations P

- Threshold partial pivoting for numerical stability (and sparsity)
- Select pivot with the sparsest row estimate among those that satisfy threshold tolerance

Stopping early

- Simplex has a Phase-I method to get feasible

$$Ax + Is = b, \quad x \geq 0, s \text{ free}$$

- So we only need a basis for the matrix $\begin{pmatrix} C & I \end{pmatrix}$
- We can stop the LU factorization after r pivots

$$\begin{pmatrix} C_{11}^T & C_{21}^T \\ C_{12}^T & C_{22}^T \\ C_{13}^T & C_{23}^T \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ L_{21} & X \\ L_{31} & Y \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ 0 & Z \end{pmatrix},$$

  with L_{11}, U_{11} full rank $r \times r$ triangles
- Basis is given by

$$B = \begin{pmatrix} C_{11} & 0 \\ C_{21} & I \end{pmatrix}$$

- Basis is full-rank since

$$B^T = \begin{pmatrix} C_{11}^T & C_{12}^T \\ 0 & I \end{pmatrix} = \begin{pmatrix} L_{11} & 0 \\ 0 & I \end{pmatrix} \begin{pmatrix} U_{11} & U_{12} \\ & I \end{pmatrix}$$

Left-looking LU (Gilbert & Peierls)

- Computes L and U one column at a time

$$\begin{pmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & l_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & u_{12} & U_{13} \\ & u_{22} & u_{23} \\ & & U_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & a_{12} & A_{13} \\ a_{21} & a_{22} & a_{23} \\ A_{31} & a_{32} & A_{33} \end{pmatrix}$$

- Key computational kernel is the sparse triangular solve with sparse rhs

$$\begin{pmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & 0 & I \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \end{pmatrix}$$

Left-looking LU (Gilbert & Peierls)

- Computes L and U one column at a time

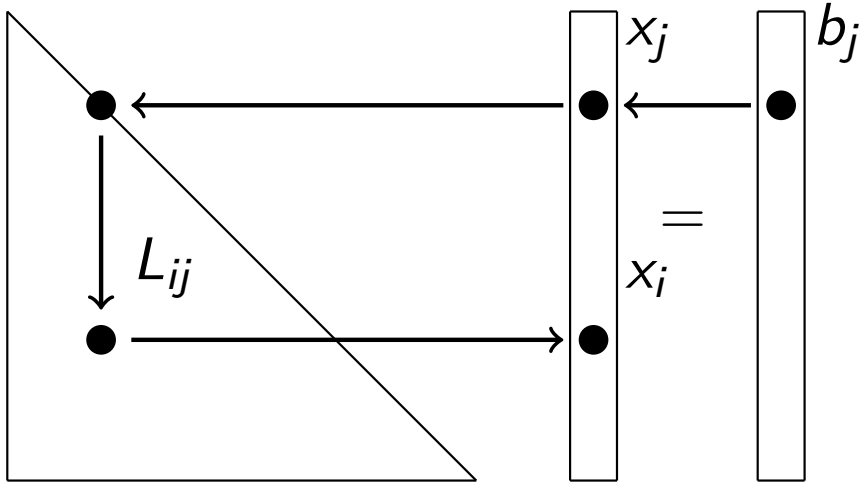
$$\begin{pmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & l_{32} & L_{33} \end{pmatrix} \begin{pmatrix} U_{11} & u_{12} & U_{13} \\ & u_{22} & u_{23} \\ & & U_{33} \end{pmatrix} = \begin{pmatrix} A_{11} & a_{12} & A_{13} \\ a_{21} & a_{22} & a_{23} \\ A_{31} & a_{32} & A_{33} \end{pmatrix}$$

- Key computational kernel is the sparse triangular solve with sparse rhs

$$\begin{pmatrix} L_{11} & & \\ l_{21} & 1 & \\ L_{31} & 0 & I \end{pmatrix} \begin{pmatrix} u_{12} \\ u_{22} \\ l_{32}\, u_{22} \end{pmatrix} = \begin{pmatrix} a_{12} \\ a_{22} \\ a_{32} \end{pmatrix}$$

Sparse triangular solve with a sparse rhs (G&P)

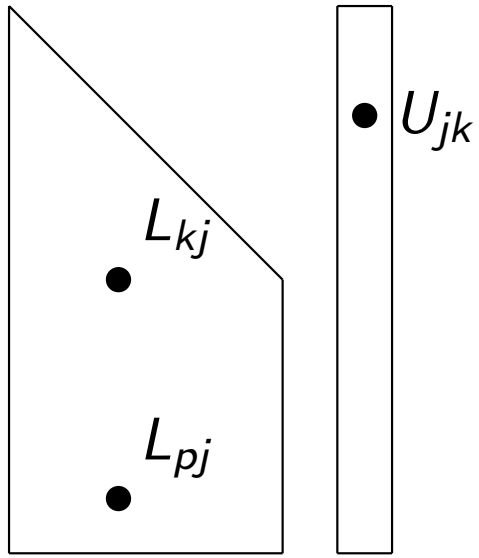
- Solve $Lx = b$ when L and b are sparse
- $O(\texttt{flops})$ if non-zero pattern $\mathcal{X} = \{j \mid x_j \neq 0\}$ known
- If $L_{ij} \neq 0$, there is an edge (j, i) in graph G_L
- $\mathcal{X}$ is set of nodes reachable from $\mathcal{B} = \{i \mid b_i \neq 0\}$



- Find $\mathcal{X}$ via a depth-first search

Symmetric pruning (Eisenstat & Liu)

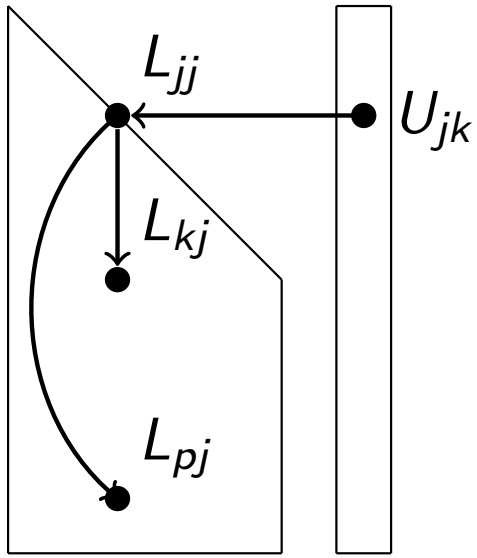
- Prune edges in G_L to reduce depth-first search time
- Suppose we have just computed the kth column of L and U
- If $U_{jk} \neq 0$ and $L_{kj} \neq 0$, we can prune edge (j, p) corresponding to $L_{pj} \neq 0$ for $j < k < p$



- Don't prune (j, p) if L_{pk} dropped because $|L_{pk}| < \epsilon$

Symmetric pruning (Eisenstat & Liu)

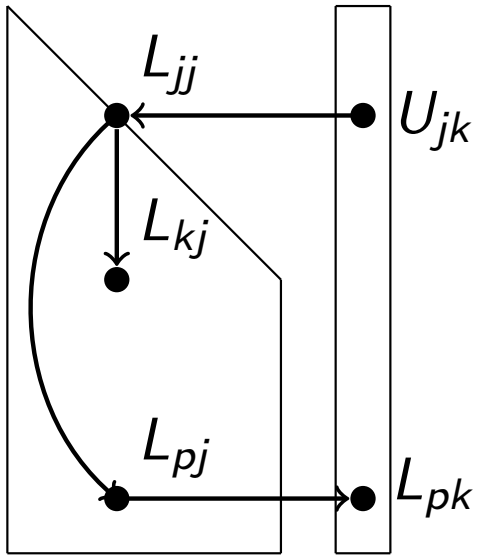
- Prune edges in G_L to reduce depth-first search time
- Suppose we have just computed the kth column of L and U
- If $U_{jk} \neq 0$ and $L_{kj} \neq 0$, we can prune edge (j, p) corresponding to $L_{pj} \neq 0$ for $j < k < p$



- Don't prune (j, p) if L_{pk} dropped because $|L_{pk}| < \epsilon$

Symmetric pruning (Eisenstat & Liu)

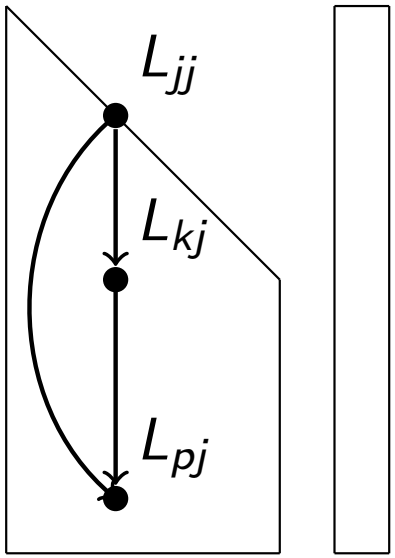
- Prune edges in G_L to reduce depth-first search time
- Suppose we have just computed the kth column of L and U
- If $U_{jk} \neq 0$ and $L_{kj} \neq 0$, we can prune edge (j, p) corresponding to $L_{pj} \neq 0$ for $j < k < p$



- Don't prune (j, p) if L_{pk} dropped because $|L_{pk}| < \epsilon$

Symmetric pruning (Eisenstat & Liu)

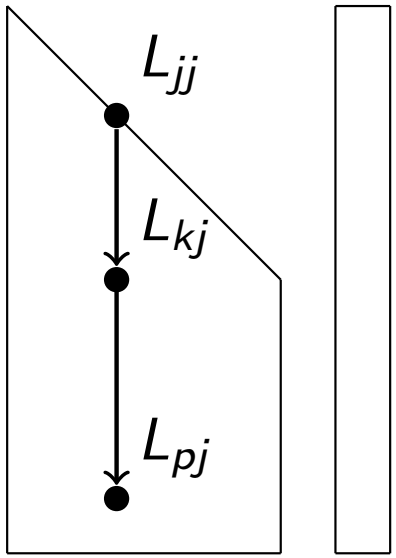
- Prune edges in G_L to reduce depth-first search time
- Suppose we have just computed the kth column of L and U
- If $U_{jk} \neq 0$ and $L_{kj} \neq 0$, we can prune edge (j, p) corresponding to $L_{pj} \neq 0$ for $j < k < p$



- Don't prune (j, p) if L_{pk} dropped because $|L_{pk}| < \epsilon$

Symmetric pruning (Eisenstat & Liu)

- Prune edges in G_L to reduce depth-first search time
- Suppose we have just computed the kth column of L and U
- If $U_{jk} \neq 0$ and $L_{kj} \neq 0$, we can prune edge (j, p) corresponding to $L_{pj} \neq 0$ for $j < k < p$



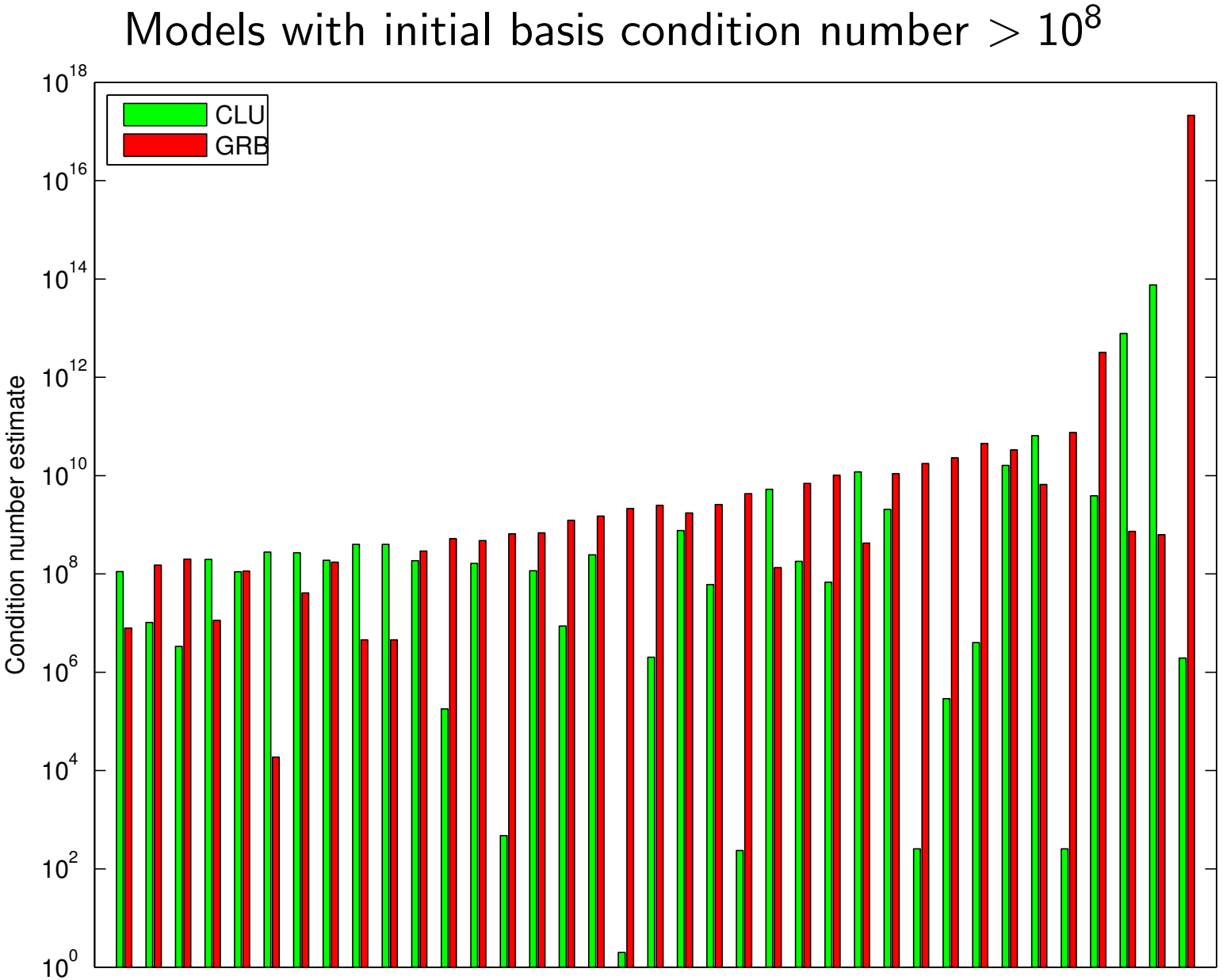
- Don't prune (j, p) if L_{pk} dropped because $|L_{pk}| < \epsilon$

Numerical Results

Summary of results

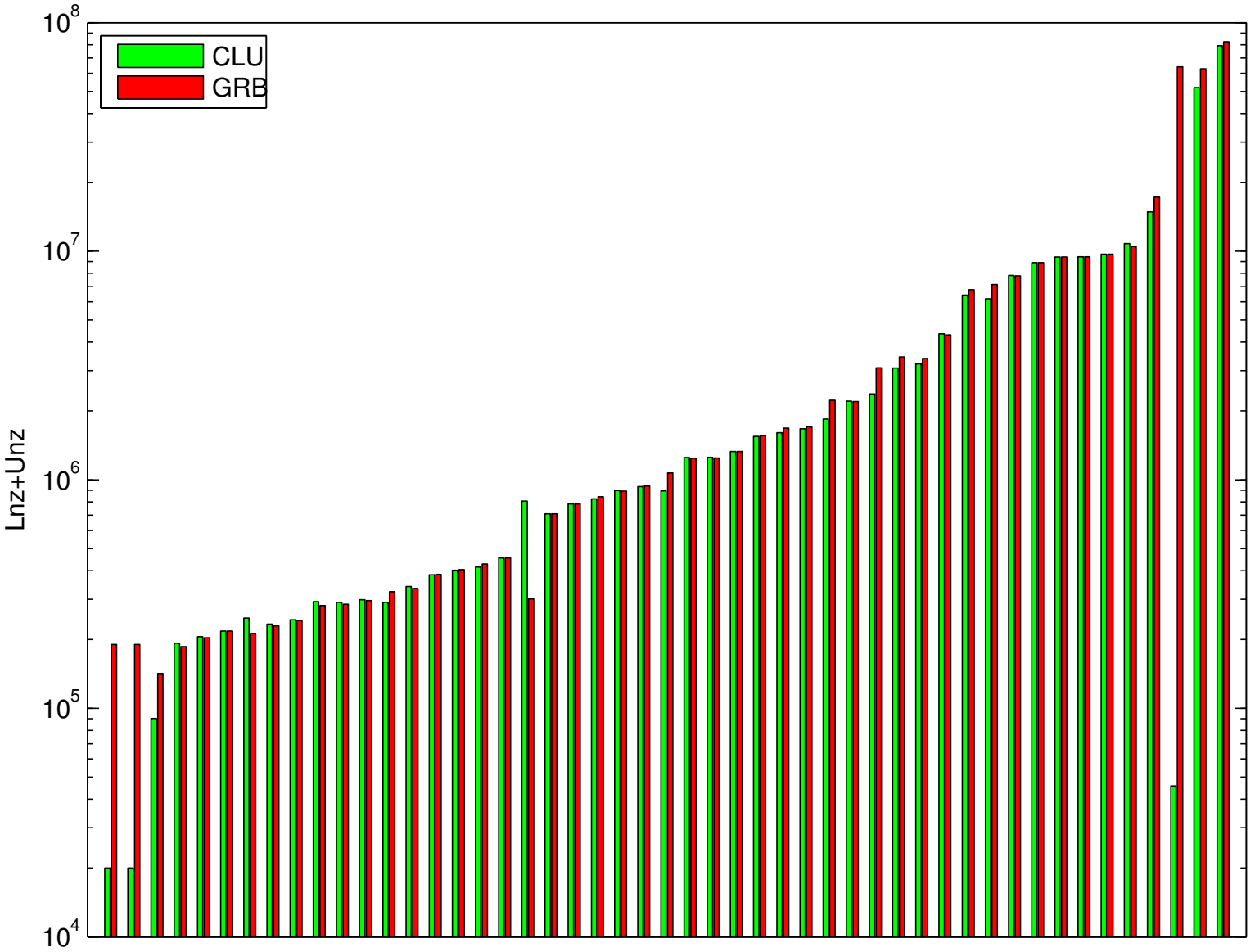
- Compared old method GRB with new approach CLU
- 10X faster when initial basis selection $\geq 1s$
- 1.5X better basis condition number overall
- 36X better basis condition number when $\kappa > 10^8$

Results: Condition number of initial basis

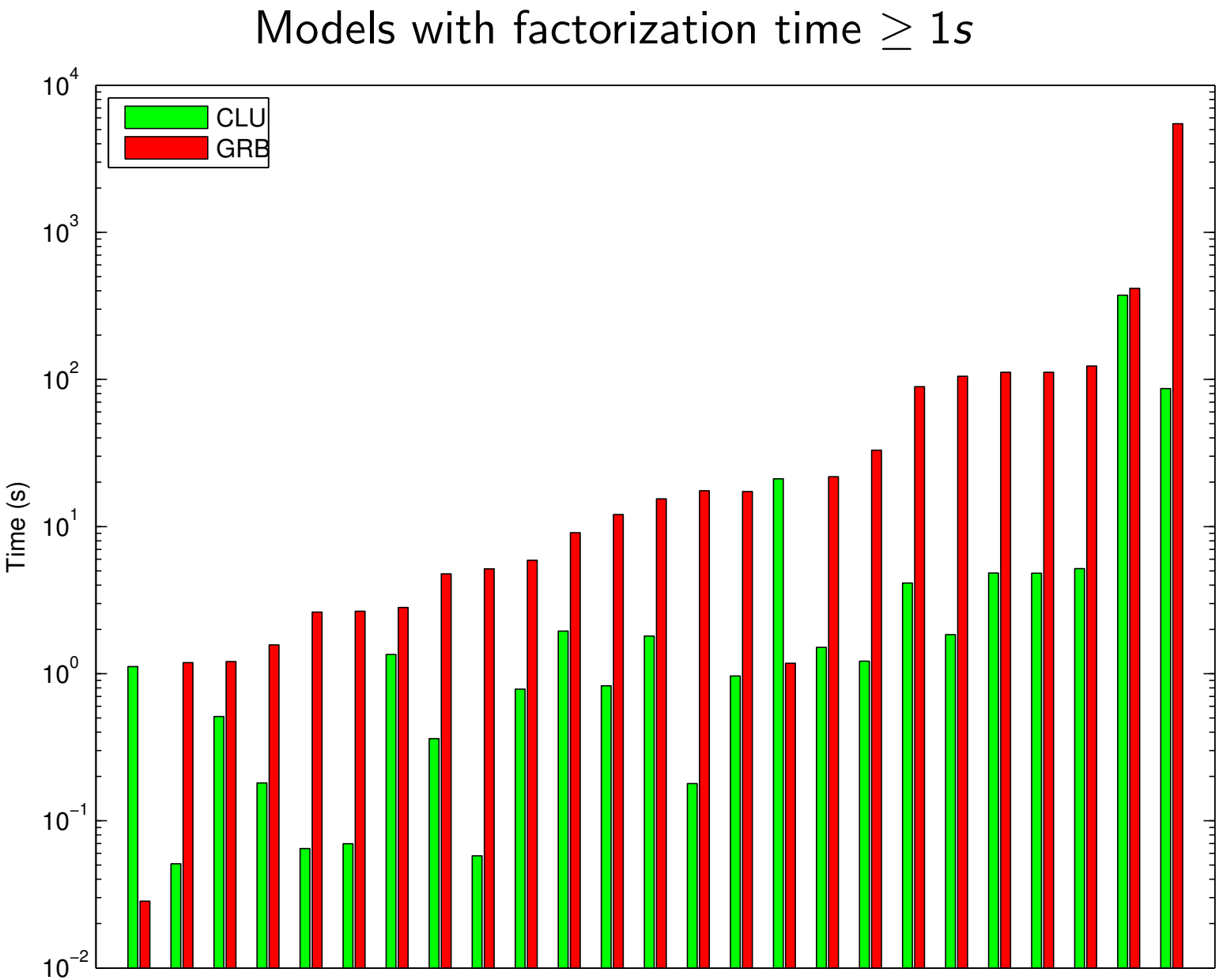


Results: Sparsity

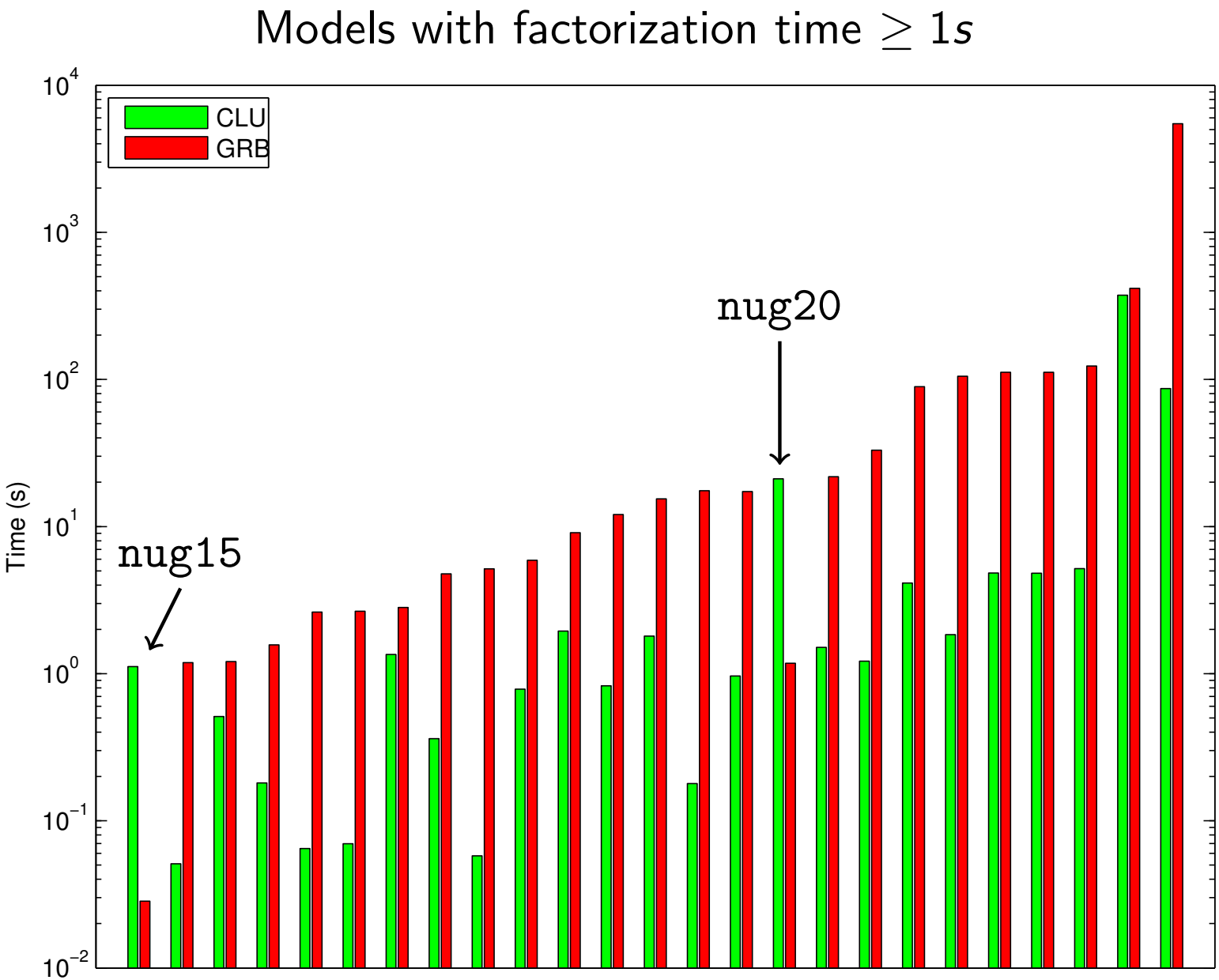
Models where $B = LU$ had $\texttt{nnz}(L) + \texttt{nnz}(U) \geq 10^5$



Results: Time to select initial basis

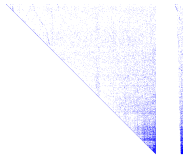
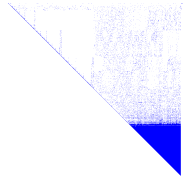
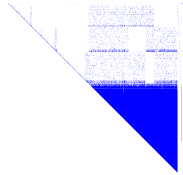


Results: Time to select initial basis



Model nug15

- *nug** models are from a test set of QAPs by Nugent *et al.*

Solver	Method	nnz(L)	nnz(U)	spy(U)
GRB	Right Markowitz	32100	53363	-
LUSOL	Right Markowitz	63301	27020	
UMFPACK	Multifrontal COLAMD	2220031	942162	
CLU (early)	Left COLAMD	5127929	2697188	
CLU	+ sparsest row & drop	1816983	511944	-

- On these models, Markowitz methods produce sparser factors than those with COLAMD preordering

Improvement possible in simplex LU?

- Left-looking LU with a COLAMD preordering is effective for crossover
- Could LU factorization for simplex also be improved?
- Compared non-zeros in L and U factors of $LU = B$ for:
  - Gurobi's right-looking Markowitz method
  - UMFPACK's multifrontal method with COLAMD preordering
- Similar to study by R. Luce *et al.* on the *Linear Algebra Kernel of Simplex-Based LP solvers*
- UMFPACK produced slightly sparser factors ($< 5\%$ overall)
- Little room for improvement
- Right-looking Markowitz methods are well-suited for LP

Conclusions

- Improves overall crossover behavior
- 10X mean reduction in initial basis selection time for models where selection takes more than a second
- Initial basis selection is the dominant cost in several models
- Produces better conditioned initial bases
- Avoids later numerical trouble on several numerically challenging models
- Progress towards more robust crossover on difficult models
- Appeared in version 5.5 of Gurobi
- Just a small step on the way to a faster crossover!

Thank you

Extra slides

Abstract

Singular and ill-conditioned bases arise frequently when performing crossover from an interior solution of a linear program. These ill-conditioned bases can slow the crossover algorithm and even cause it to fail in extreme cases. The sparsity and condition of later bases, and the total number of crossover steps, is heavily influenced by the choice of the initial basis. The ideal initial basis is sparse, well-conditioned, and contains few artificial variables. However, these properties often conflict with one another. We present a new sparse LU factorization and ordering algorithm for selecting an initial basis that seeks a balance between these different factors. We compare this new method to the approach used in version 5 of the Gurobi Optimizer on a large test set of linear programming problems.