The importance of structure in algebraic preconditioners

Jennifer Scott

Rutherford Appleton Laboratory

Miroslav Tůma

Institute of Computer Science

Academy of Sciences of the Czech Republic

Introduction

Consider the large sparse symmetric positive definite linear system

$$Ax = b$$

Long history of preconditioners based on incomplete factorizations of A:

- Threshold-based $IC(\tau)$: entries greater than τ dropped.
- Memory-based $IC(m)$: dropping of entries based on memory available.
- Structure-based $IC(\ell)$: potential fill entries allowed only if their level of fill is less than τ.

Range of variants and refinements have been proposed and used for different classes of problems, with no single approach best overall.

Potential disadvantage of $IC(\tau)$ and $IC(m)$: structural information may be lost.

Objectives

Our aim:

- Consider importance of structure of A in construction of incomplete factorization preconditioners.
- Develop a general approach $IC(\ell, \tau, m)$ that combines level-based approach with memory prediction and dropping small entries.

Objectives

Our aim:

- Consider importance of structure of A in construction of incomplete factorization preconditioners.
- Develop a general approach $IC(\ell, \tau, m)$ that combines level-based approach with memory prediction and dropping small entries.

Today: concentrate on a simple modification to level-based approach.

Objectives

Our aim:

- Consider importance of structure of A in construction of incomplete factorization preconditioners.
- Develop a general approach $IC(\ell, \tau, m)$ that combines level-based approach with memory prediction and dropping small entries.

Today: concentrate on a simple modification to level-based approach.

Integrate this with a prescribed memory and dropping strategy that aims to retain structure of A.

Objectives

Our aim:

- Consider importance of structure of A in construction of incomplete factorization preconditioners.
- Develop a general approach $IC(\ell, \tau, m)$ that combines level-based approach with memory prediction and dropping small entries.

Today: concentrate on a simple modification to level-based approach.

Integrate this with a prescribed memory and dropping strategy that aims to retain structure of A.

Notation: $L = \{l_{ij}\}$ denotes complete factor of A ($A = LL^T$);
$\hat{L} = \{\hat{l}_{ij}\}$ denotes incomplete factor.

Level-based approach

Fill path theorem (Rose, Tarjan and Leuker 1978): l_{ij} is non zero if and only if there is a fill path connecting i and j in $\mathcal{G}$ (the adjacency graph of A).

Level-based approach

Fill path theorem (Rose, Tarjan and Leuker 1978): l_{ij} is non zero if and only if there is a fill path connecting i and j in $\mathcal{G}$ (the adjacency graph of A).

Sum rule : entries of $\hat{L}$ corresponding to nonzero entries of A are assigned the level 0 while each potential fill entry is assigned a level

$$level(i,j) = \min_{1 \le l \le \min\{i,j\}} \{level(i,l) + level(l,j) + 1\}.$$

In $IC(\ell)$ a fill entry is permitted if $level(i,j) \le \ell$.

Problems as ℓ increases:

- structure may fill in quickly making $\hat{L}$ expensive to compute and apply
- efficient computation of structure of $\hat{L}$ ... addressed by Hysom and Pothen

Incomplete fill path theorem

Incomplete fill path theorem (Hysom and Pothen 2002): $level(i,j) = \ell$ if and only if there exists a shortest fill path of length $\ell + 1$ joining i and j in $\mathcal{G}$.

- Hysom and Pothen use this to develop algorithm for computing sparsity pattern of single column of $\hat{L}$.
- Procedure uses breadth first search that finds shortest path between vertex k and vertices reachable from k via a traversal of at most $\ell + 1$ edges.
- Key feature: the structure of each column of $\hat{L}$ can be computed independently (and hence in parallel).
- Once sparsity pattern known, separate factorization phase needed to compute entries of $\hat{L}$.

Preassigning levels

Our aim: try and ensure that small entries of A contribute to fewer levels of fill than larger entries.

Proposal: given $l > 0$, preassign $level(i, j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

- Compute absolute values of smallest and largest entries of A ($msmall$ and $mbig$).

Preassigning levels

Our aim: try and ensure that small entries of A contribute to fewer levels of fill than larger entries.

Proposal: given $l > 0$, preassign $level(i, j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

- Compute absolute values of smallest and largest entries of A ($msmall$ and $mbig$).
- Distribute nonzero entries uniformly by $\log |a_{ij}|$ into

$$[log(mbig) - log(msmall)] + 1$$

groups, with smallest entries in group with index 1.

Preassigning levels

Our aim: try and ensure that small entries of A contribute to fewer levels of fill than larger entries.

Proposal: given $l > 0$, preassign $level(i, j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

- Compute absolute values of smallest and largest entries of A ($msmall$ and $mbig$).
- Distribute nonzero entries uniformly by $\log |a_{ij}|$ into

$$[log(mbig) - log(msmall)] + 1$$

  groups, with smallest entries in group with index 1.
- Sparsify A by dropping any very small entries in group 1.

Preassigning levels

Our aim: try and ensure that small entries of A contribute to fewer levels of fill than larger entries.

Proposal: given $l > 0$, preassign $level(i, j)$ for each entry of A individually to have a value ≥ 0 that depends on $|a_{ij}|$.

- Compute absolute values of smallest and largest entries of A ($msmall$ and $mbig$).
- Distribute nonzero entries uniformly by $\log |a_{ij}|$ into

$$[log(mbig) - log(msmall)] + 1$$

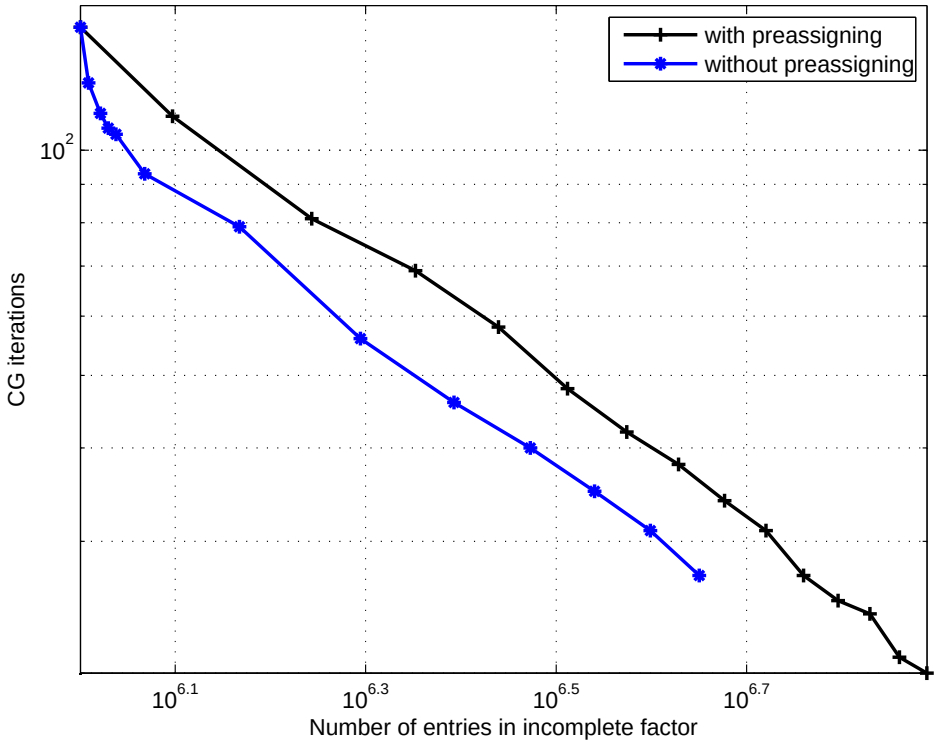
  groups, with smallest entries in group with index 1.
- Sparsify A by dropping any very small entries in group 1.
- Preassign $level(i, j)$ for individual entries according to which group they belong to. Small entries are assigned level $l - 1$ and large entries assigned level 0.

Notes on preassigning levels

- Largest entries of A given initial level 0; smaller entries have positive value.
- Very easy to modify Hysom Pothen algorithm to accommodate initial levels > 0.
- Extra cost for symbolic factorization but saves on time to compute $\hat{L}$ if $nz(\hat{L})$ is sparser than for standard approach.

Effect of preassigning levels

Example: Kohn-Sham equation (`carsten3`) $n = 250500$.



Test set

- Problems taken from University of Florida Sparse Matrix Collection
- Selected all positive definite matrices of order at least 1000
- CG used with $x_0 = 0$, b computed so that $x = 1$, and stopping criteria

$$\|Ax_k - b\| \leq 10^{-6}\|b\|$$

  with limit of 800 iterations.
- Ran with $\ell = 2$ and 3 and for each run, removed problems that failed to converge with and without preassigning initial levels.

Set $\mathcal{T}$ comprises 98 examples for $\ell = 2$ and 103 for $\ell = 3$.

Test environment

For solving a given problem, preconditioner P_i is the most *efficient* of the preconditioners tested if

$$iter_i \times nz(P_i) \leq \min_{k \neq i}(iter_k \times nz(P_k)),$$

where $iter_k$ is the iteration count for P_k.

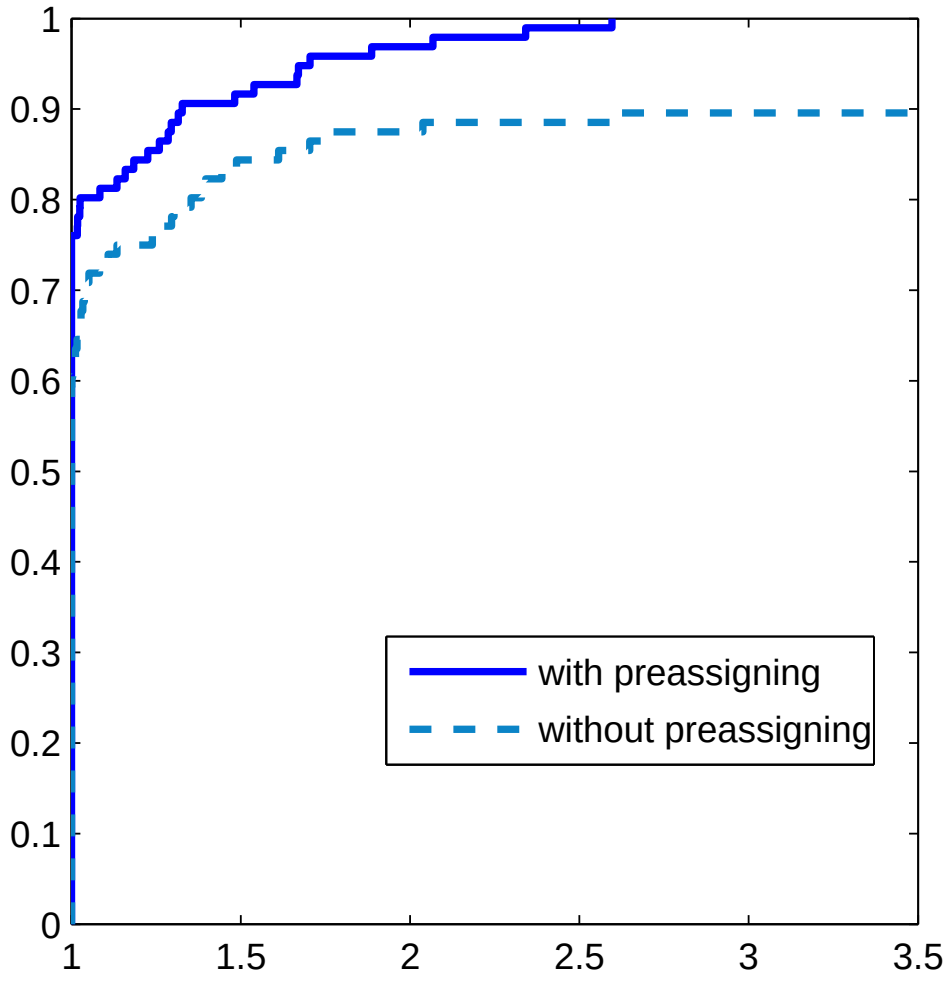
- Suppose P_i has efficiency $e_{ij} \geq 0$ when run on problem $j \in \mathcal{T}$
- Let $\hat{e}_j = \min\{e_{ij}\}$.
- For $\alpha \geq 1$ and each i define

$$k(e_{ij}, \hat{e}_j, \alpha) = \begin{cases} 1 & \text{if } e_{ij} \leq \alpha \hat{e}_j \\ 0 & \text{otherwise.} \end{cases}$$

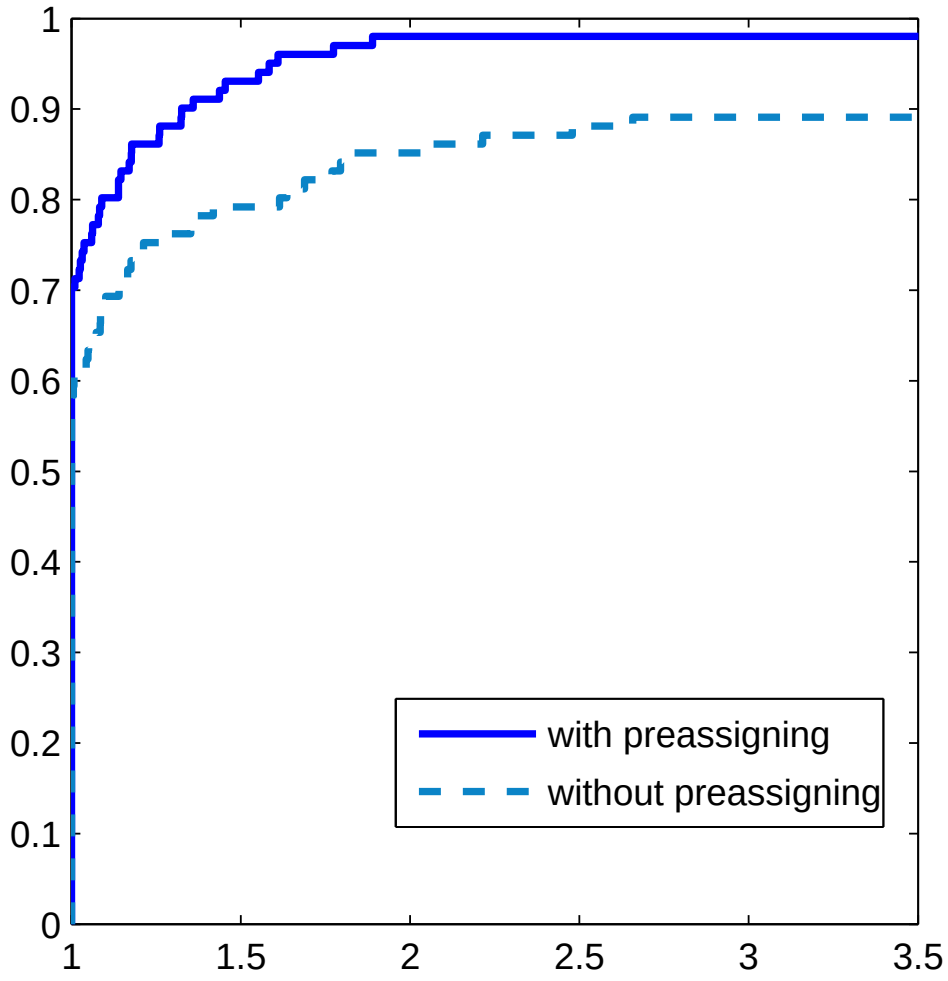
- The performance profile for P_i is then given by

$$prof_i(\alpha) = \frac{1}{|\mathcal{T}|} * \sum_{j \in \mathcal{T}} k(e_{ij}, \hat{e}_j, \alpha), \quad \alpha \geq 1.$$

Performance profile $\ell = 2$



Performance profile $\ell = 3$



Findings so far

- Overall, some advantage in preassigning levels.
- With ℓ fixed, able to solve some problems if levels preassigned that were not solved otherwise.
- For other problems, efficiency improved by preassigning levels.
- **But** for some problems, better not to preassign levels. So far, not able to predict when this is the case.
- If the matrix is well-scaled, $ngroup = 1$ and so presassigning has no effect.

Second component of our approach: keeping structure during factorization phase.

Allowing additional memory

- We have used $m = 1$, that is $nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization (modified Hysom-Pothen algorithm).

Allowing additional memory

- We have used $m = 1$, that is $nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization (modified Hysom-Pothen algorithm).
- What if we allow $m > 1$?

Allowing additional memory

- We have used $m = 1$, that is $nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization (modified Hysom-Pothen algorithm).
- What if we allow $m > 1$?
- Retain the predicted structure but allow some extra entries outside this.

Allowing additional memory

- We have used $m = 1$, that is $nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization (modified Hysom-Pothen algorithm).
- What if we allow $m > 1$?
- Retain the predicted structure but allow some extra entries outside this.
- Obvious approach: keep the largest extra entries

Allowing additional memory

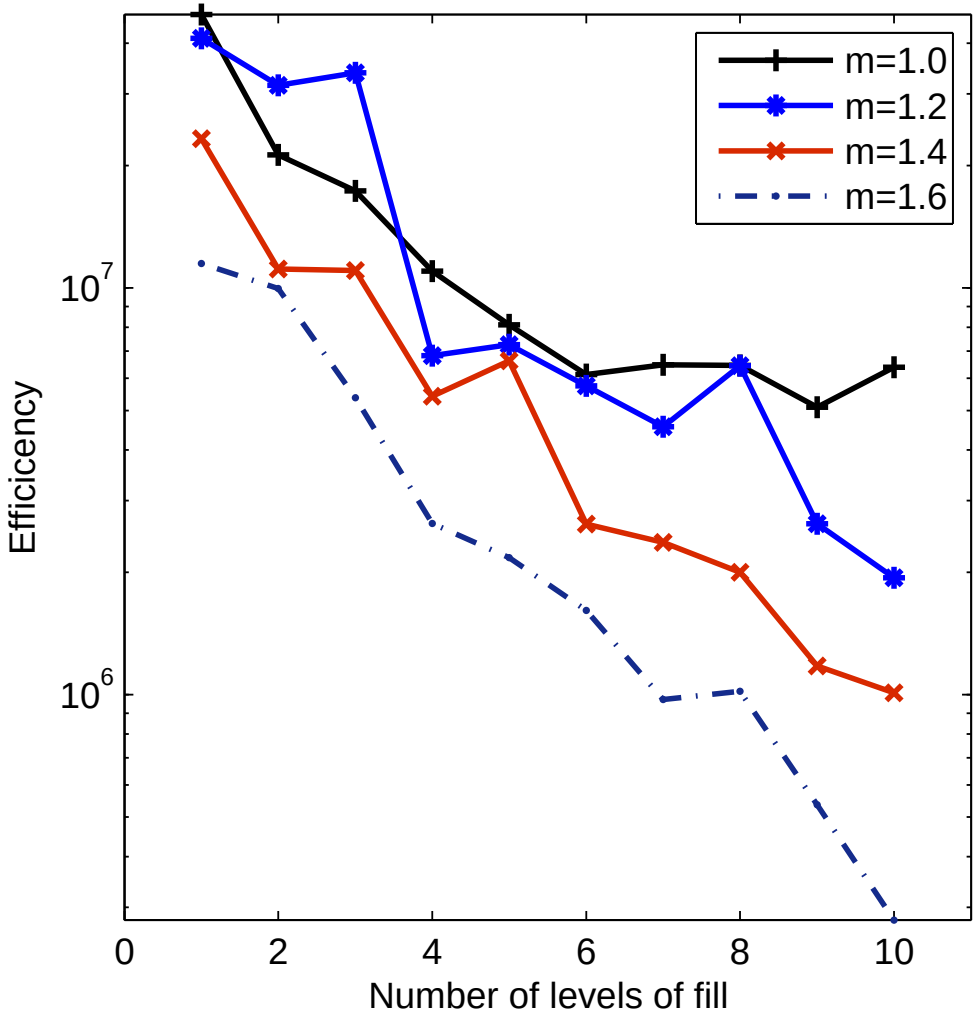
- We have used $m = 1$, that is $nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization (modified Hysom-Pothen algorithm).
- What if we allow $m > 1$?
- Retain the predicted structure but allow some extra entries outside this.
- Obvious approach: keep the largest extra entries
- How to assign extra space? Two possibilities:
  - uniformly share space between columns
  - non-uniform distribution, using column counts for exact factor L.

Allowing additional memory

- We have used $m = 1$, that is $nz(\hat{L})$ is equal to the number of entries predicted by symbolic factorization (modified Hysom-Pothen algorithm).
- What if we allow $m > 1$?
- Retain the predicted structure but allow some extra entries outside this.
- Obvious approach: keep the largest extra entries
- How to assign extra space? Two possibilities:
  - uniformly share space between columns
  - non-uniform distribution, using column counts for exact factor L.
- Here we use **uniform** distribution.

Advantage of extra space

Example: `Nasa/nasa4704`



Integrate predefined structure with dropping

- **Aim:** To drop small entries (those smaller in absolute value than chosen tolerance τ) without effecting quality of preconditioner

Integrate predefined structure with dropping

- **Aim:** To drop small entries (those smaller in absolute value than chosen tolerance τ) without effecting quality of preconditioner
- If entries are dropped, we have spare space.

Integrate predefined structure with dropping

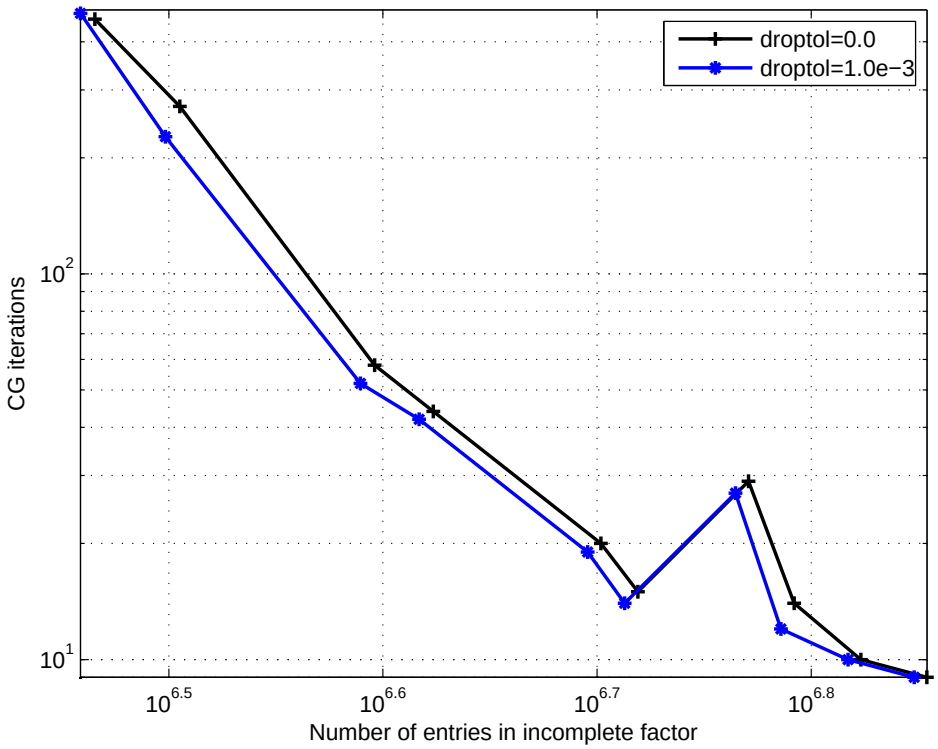
- **Aim:** To drop small entries (those smaller in absolute value than chosen tolerance τ) without effecting quality of preconditioner
- If entries are dropped, we have spare space.
- Allow entries outside the sparsity structure of $\hat{L}$ to be retained.

Integrate predefined structure with dropping

- **Aim:** To drop small entries (those smaller in absolute value than chosen tolerance τ) without effecting quality of preconditioner
- If entries are dropped, we have spare space.
- Allow entries outside the sparsity structure of $\hat{L}$ to be retained.
- If there is insufficient space to accommodate all such extra entries, sort and retain only the largest.

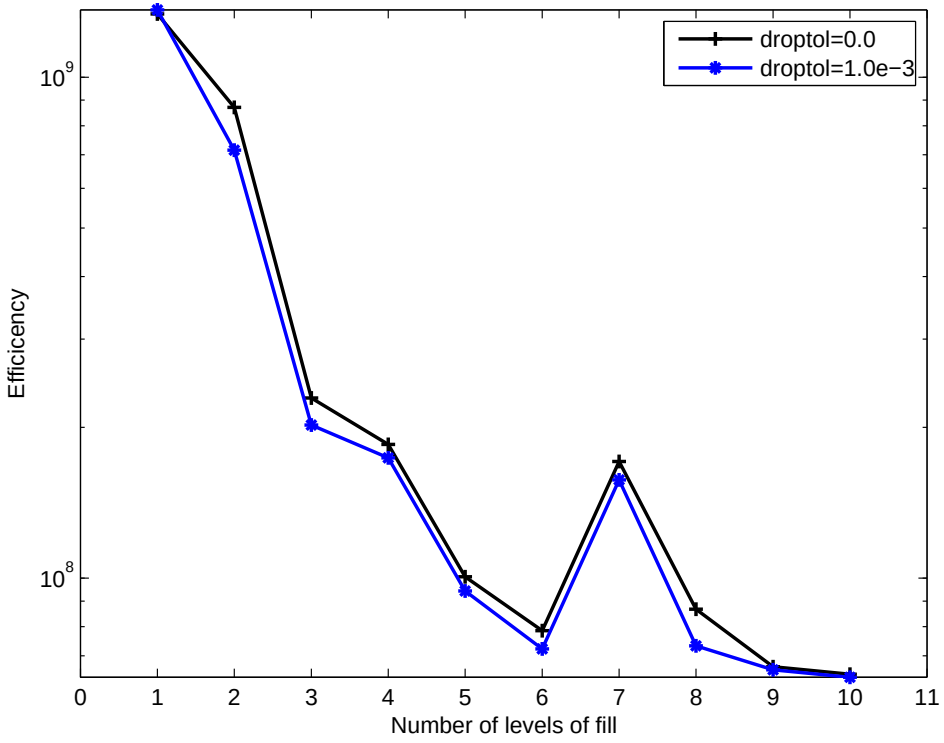
Improving efficiency from dropping

Example: `Boeing/ct20stif`. $n = 52329, m = 1.$



Improving efficiency by dropping

Example: `Boeing/ct20stif`



Some concluding remarks

- More work to be done to refine our approach but appears that retaining structure can enhance the incomplete factorization as a preconditioner.
- A new HSL code `HSL_MI22` is beginning developed that will include all the ideas mentioned here.
- Open problem: determine more sophisticated rules to preassign levels.

Some concluding remarks

- More work to be done to refine our approach but appears that retaining structure can enhance the incomplete factorization as a preconditioner.
- A new HSL code `HSL_MI22` is beginning developed that will include all the ideas mentioned here.
- Open problem: determine more sophisticated rules to preassign levels.

Thank you!