

Dynamic Supernodes in Sparse Cholesky Update/Downdate for Active Set Methods

Tim Davis, William Hager
with Y. Morris Chen, Siva Rajamanickam
with support from NSF

July 9, 2008

1 Problem statement

Cholesky update/downdate

Applicable to active set algorithms

2 Mathematical kernels

Rank-1 update

Sparse lower triangular solve

Elimination tree

3 Multiple-rank update/downdate

Multiple paths in elimination tree

Dynamic supernodes in update/downdate

Dynamic supernodes in forward solve

4 Results

5 Summary

Problem statement

- Given a Cholesky factorization $A = LL^T$
- And a low rank modification $\bar{A} = A \pm WW^T$
- Find $\bar{A} = \bar{L}\bar{L}^T$
- Applications:
 - dual active set methods for LP problems (LPDASA)
 - computation of projections
 - change in PDE solution after modifying a coefficient
 - ...

LPDASA: LP Dual Active Set Algorithm

- $\min c^T x$ subject to $Ax = b, x \geq 0$
- Need to maintain Cholesky factorization of $A_F A_F^T + \sigma I$ (update/downdate), and use it for forward/back solves.
- multiple-rank update of $A_F A_F^T + \sigma I$ during line search, as columns are added to the free set F , as the dual is maximized.
- multiple-rank downdate of $A_F A_F^T + \sigma I$ at end of each LPDASA subiteration, as columns are deleted from the free set F .

Rank-1 update, $LL^T + ww^T$ (Carlson)

$$\bar{\beta} = 1$$

for $j = 1$ to n

alpha/beta/gamma/delta computation:

$$\alpha = w_j / l_{jj}$$

$$\beta = \bar{\beta}, \quad \bar{\beta} = \sqrt{\beta^2 + \alpha^2}, \quad \gamma = \alpha / (\bar{\beta}\beta), \quad \delta = \beta / \bar{\beta}$$

update diagonal:

$$l_{jj} = \delta l_{jj} + \gamma w_j$$

$$w_j = \alpha$$

update below diagonal:

$$\mathbf{t} = \mathbf{w}_{j+1:n}$$

$$\mathbf{w}_{j+1:n} = \mathbf{w}_{j+1:n} - \alpha \mathbf{L}_{j+1:n,j}$$

$$\bar{\mathbf{L}}_{j+1:n,j} = \delta \mathbf{L}_{j+1:n,j} + \gamma \mathbf{t}$$

end

Overwrites w with solution to $Lx = w$

$$\bar{\beta} = 1$$

for $j = 1$ to n

alpha/beta/gamma/delta computation:

$$\alpha = w_j / l_{jj}$$

$$\beta = \bar{\beta}, \quad \bar{\beta} = \sqrt{\beta^2 + \alpha^2}, \quad \gamma = \alpha / (\bar{\beta}\beta), \quad \delta = \beta / \bar{\beta}$$

update diagonal:

$$l_{jj} = \delta l_{jj} + \gamma w_j$$

$$w_j = \alpha$$

update below diagonal:

$$\mathbf{t} = \mathbf{w}_{j+1:n}$$

$$\mathbf{w}_{j+1:n} = \mathbf{w}_{j+1:n} - \alpha \mathbf{L}_{j+1:n,j}$$

$$\bar{\mathbf{L}}_{j+1:n,j} = \delta \mathbf{L}_{j+1:n,j} + \gamma \mathbf{t}$$

end

Key observations

- Carlson's algorithm overwrites w with $L \setminus w$
- No change to L_{*j} if $(L \setminus w)_j$ is zero
- Columns of L that change correspond to the nonzero pattern of $L \setminus w$
- For efficient sparse update: need to find the pattern of $L \setminus w$

Sparse lower triangular solve ($x = L \setminus b$)

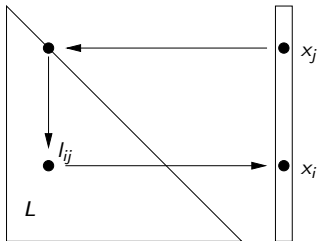
$x = b$

for $j = 1$ to n

$x_j = x_j / l_{jj}$

$x_{j+1:n} = x_{j+1:n} - x_j L_{j+1:n,j}$

end



Sparse lower triangular solve ($x = L \setminus b$)

$x = b$

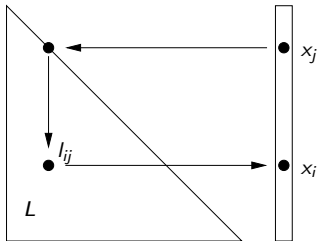
for $j = 1$ to n

$$x_j = x_j / l_{jj}$$

$$x_{j+1:n} = x_{j+1:n} - x_j L_{j+1:n,j}$$

end

- $b_i \neq 0 \Rightarrow x_i \neq 0$



Sparse lower triangular solve ($x = L \setminus b$)

$x = b$

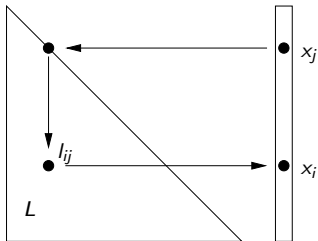
for $j = 1$ to n

$x_j = x_j / l_{jj}$

$x_{j+1:n} = x_{j+1:n} - x_j L_{j+1:n,j}$

end

- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$



Sparse lower triangular solve ($x = L \setminus b$)

$x = b$

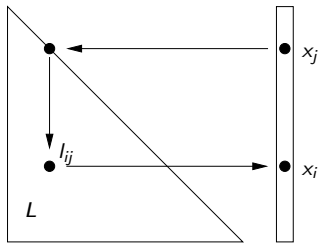
for $j = 1$ to n

$$x_j = x_j / l_{jj}$$

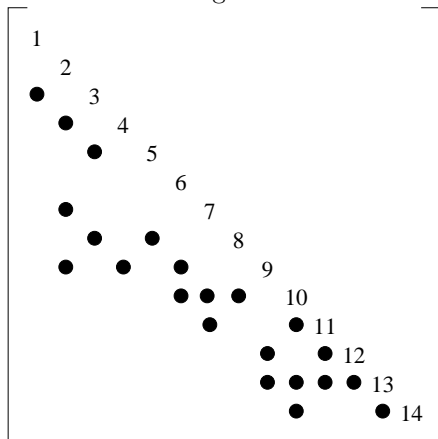
$$\mathbf{x}_{j+1:n} = \mathbf{x}_{j+1:n} - x_j \mathbf{L}_{j+1:n,j}$$

end

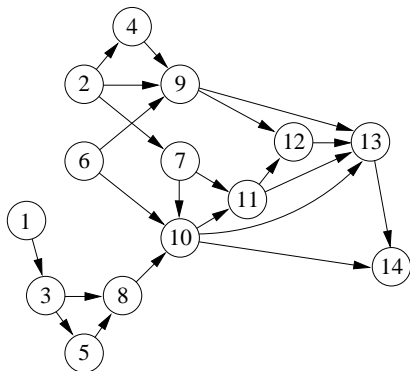
- $b_i \neq 0 \Rightarrow x_i \neq 0$
- $x_j \neq 0 \wedge l_{ij} \neq 0 \Rightarrow x_i \neq 0$
- let $G(L)$ have an edge
 $j \rightarrow i$ if $l_{ij} \neq 0$
- let $\mathcal{B} = \{i \mid b_i \neq 0\}$ and
 $\mathcal{X} = \{i \mid x_i \neq 0\}$
- then $\mathcal{X} = \text{Reach}_{G(L)}(\mathcal{B})$



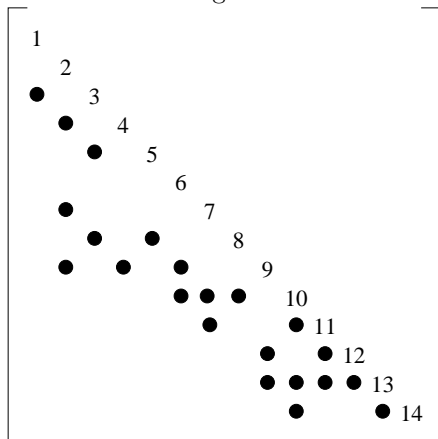
Lower triangular matrix L



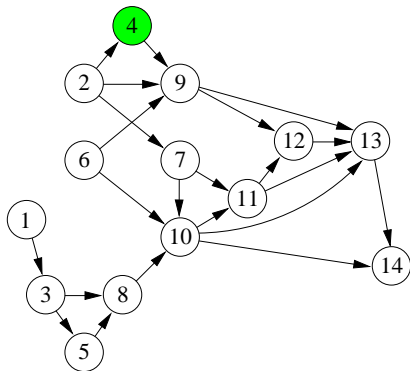
Graph G_L



Lower triangular matrix L

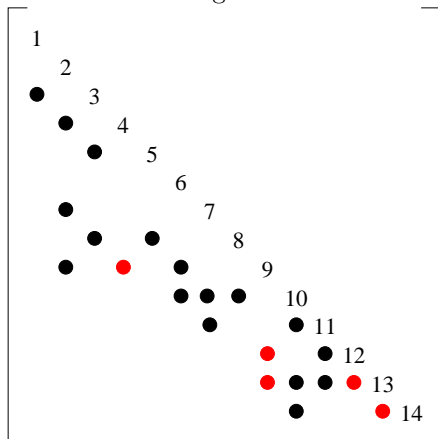


Graph G_L

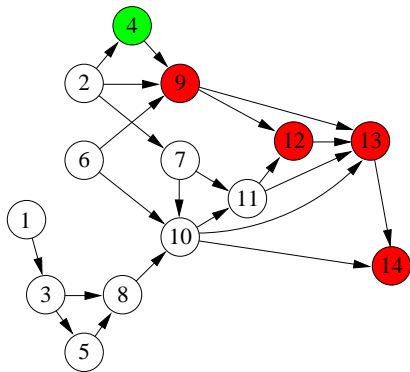


If $B = \{4\}$

Lower triangular matrix L

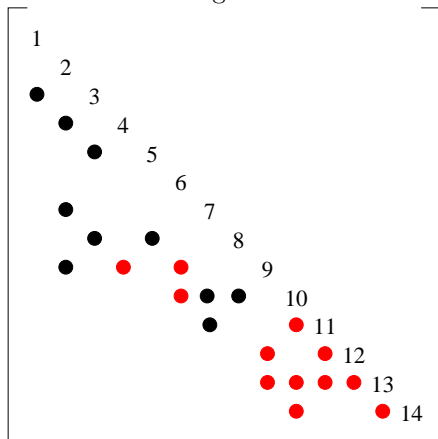


Graph G_L

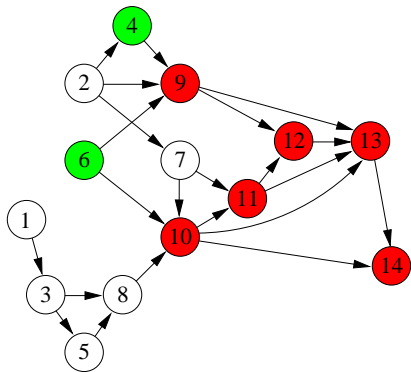


If $\mathcal{B} = \{4\}$ then $\mathcal{X} = \{4, 9, 12, 13, 14\}$

Lower triangular matrix L



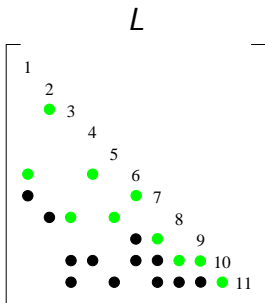
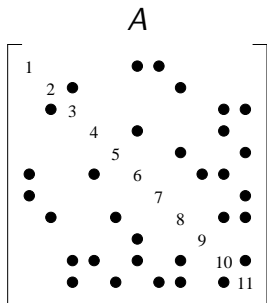
Graph G_L



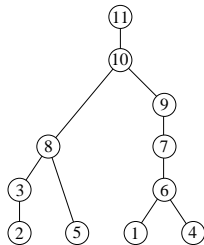
If $B = \{4, 6\}$ then $\mathcal{X} = \{6, 10, 11, 4, 9, 12, 13, 14\}$

Cholesky update governed by the etree

parent of j is $\min\{i \mid i > j \text{ and } l_{ij} \neq 0\}$

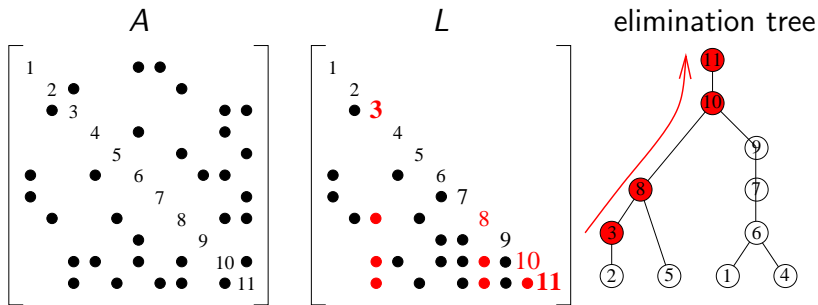


elimination tree



Reach(j) is a single path to the root

$$\text{Reach}(3) = \{3, 8, 10, 11\}$$

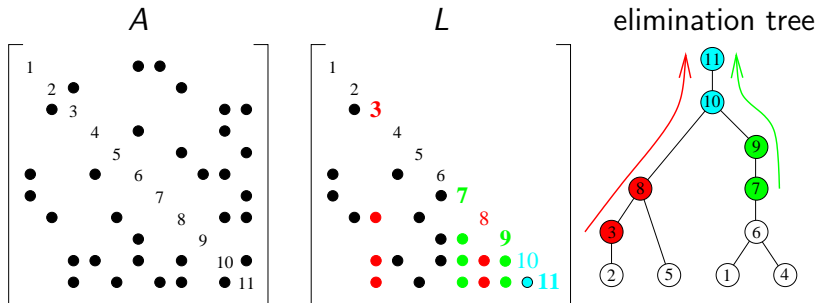


Multiple rank update/downdate

- Each rank-1 update follows a single path
- Starts at node j where j is the smallest index where $w_j \neq 0$
- Walks up to the root
- Multiple rank update when paths merge

Example rank-2 update/downdate

Starting at nodes 3 and 7.



Exploiting supernodes

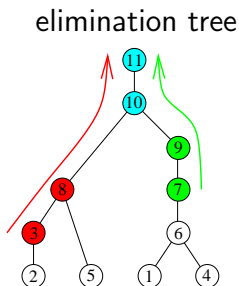
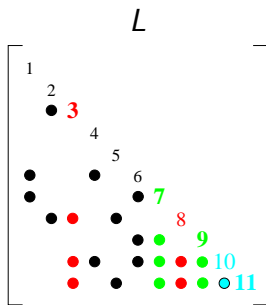
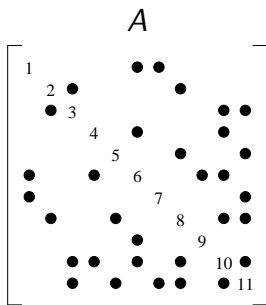
- Supernodes:
adjacent columns of L with identical nonzero pattern
- Cache reuse, dense kernels:
high performance sparse Cholesky factorization
- Change in pattern:
supernodes merge and break apart
- Need to exploit dynamic supernodes in update/downdate

Exploiting supernodes

- Let $\mathcal{L}_j = \{i \mid l_{ij} \neq 0\}$
- Let $|\mathcal{S}|$ denote the size of a set
- j and $j + 1$ are in the same **fundamental supernode** if $\text{parent}(j) = j + 1$ and $\mathcal{L}_j = \mathcal{L}_{j+1} \cup \{j\}$
- Dynamic supernodes:
 - Suppose j and $p = \text{parent}(j)$ lie on the same update/downdate path.
 - If $\mathcal{L}_j = \mathcal{L}_p \cup \{j\}$, then j and p can be part of one **dynamic supernode**
 - Quick test: $\mathcal{L}_j = \mathcal{L}_p \cup \{j\}$ if and only if $|\mathcal{L}_j| = |\mathcal{L}_p| + 1$.

Dynamic supernodes

Dynamic supernodes: (3,8), (7,9), and (10,11)



Exploiting dynamic supernodes

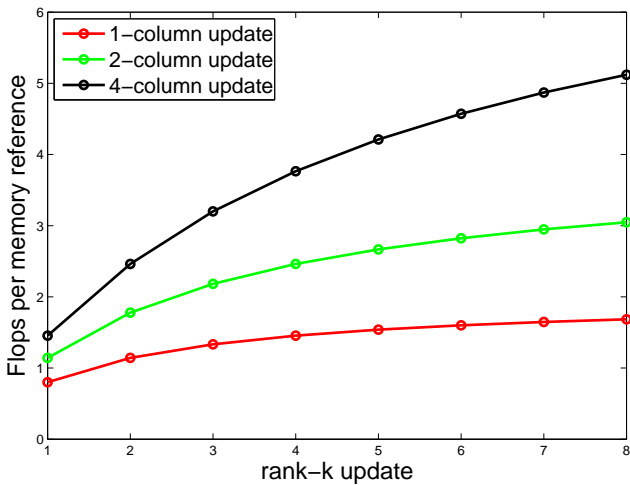
- Paths are traversed towards the root
- Constant time look-ahead:
 - look for up to 4 columns along the path with identical pattern
- If row indices of L kept sorted:
 - Just read pattern of first column in supernode
- “Dense” rank- k update of each column in the dynamic supernode

Exploiting dynamic supernodes

- Performance governed by flops per memory reference
- Let $s = |\mathcal{L}_j|$, for first column j in dynamic supernode
- Rank- k update flop count and memory traffic:

Supernode	flops	reads/writes	flops/mem
1-column	$4sk$	$2sk + 3s$	$4sk / (2sk + 3s) \leq 2$
2-column	$8sk$	$2sk + 5s$	$8sk / (2sk + 3s) \leq 4$
4-column	$16sk$	$2sk + 9s$	$16sk / (2sk + 3s) \leq 8$

Exploiting dynamic supernodes



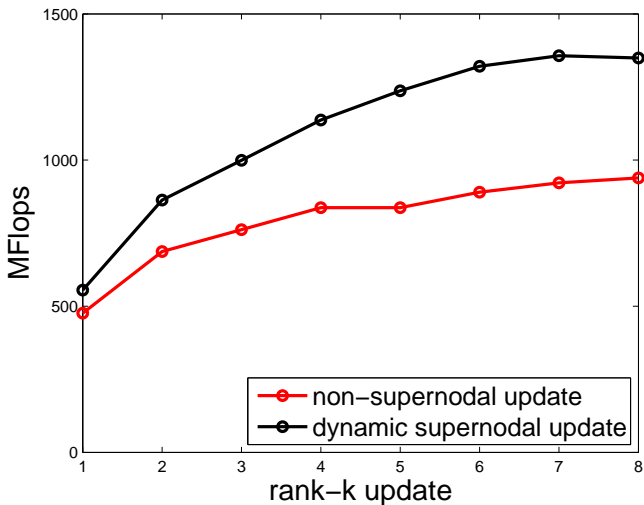
Dynamic supernodal forward solve

- Same strategy can be used for $x = L \setminus b$
- Three kinds of $x = L \setminus b$ forward solves:
 - non-supernodal (used in $x = L \setminus b$ in MATLAB)
 - dynamic supernodes
 - fixed supernodes (as output from supernodal Cholesky, used in $x = A \setminus b$ in MATLAB when A is symmetric positive definite)

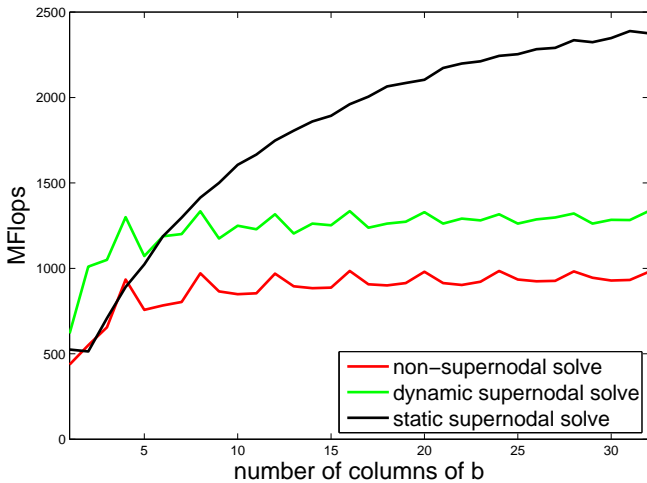
Update/downdate results: nug15 problem

Matrix name:	QAPLIB/LP_NUG15
source:	linear programming problem
n :	6330
$ A^T A $, lower triangular part:	129×10^3
ordering method:	CHOLMOD nested dissection
ordering time:	0.58 seconds
CHOLMOD symbolic Cholesky factorization:	0.02 seconds
CHOLMOD numeric Cholesky factorization:	5.89 seconds
time to convert to non-supernodal LDL^T :	0.14 seconds
initial $ L $:	7.57×10^6
Cholesky factorization flop count:	16.4×10^9
Cholesky factorization Gflops:	2.7
update/downdate rank:	128
CHOLMOD update flop count:	2.5×10^9

Dynamic supernodal update: nug15



Dynamic supernodal solve: ND3k



Summary

- Cholesky update/downdate:
key kernel in many applications
- Fixed supernodes:
great for Cholesky factorization, but hard to change
- Dynamic supernodes:
great for Cholesky update/downdate and forward solve
 - Rank-8 dynamic supernodal update:
almost 3x faster than rank-1 non-supernodal update
 - Forward solve:
dynamic supernodes faster (2x speedup over fixed supernodes and non-supernodal solve when b has 2 columns)
- Code available in CHOLMOD
- CHOLMOD in MATLAB 7.2 and later
(but not the update/downdate ... yet)