DSM Partitioning

A Pipeline for Reordering Dependency Structure Matrices

Simon Kraft Joshua Payne El Sall

CPSC 482/682 — Data Structures II
University of Northern British Columbia

April 7, 2026

Outline

1. Introduction
2. Methodology
3. Results
4. Conclusions

Motivation

The challenge:

- Modern software & engineering projects involve **hundreds to thousands of interdependent tasks** across diverse teams
- Traditional tools (Gantt, CPM) handle sequential & parallel flows well...
- ...but **fail for interdependent tasks** where outputs feed back as inputs
- These feedback dependencies are common in complex product development

Goal

Find a **task ordering** that minimises conflicts in information flow to enable faster, more predictable project execution.

Design Structure Matrix (DSM)

Definition: A binary matrix $A \in \{0,1\}^{n \times n}$ where

$$a_{ij} = 1 \iff \text{task } i \text{ depends on task } j$$

- Rows and columns labeled by the same task list
- Diagonal entries carry no dependency information.
- Equivalent to directed graph $G(A) = (V, E)$

Three dependency types:

- **Feed-Forward** ($j < i$): below diagonal — easy
- **Feed-Back** ($j > i$): above diagonal — problematic
- **Coupled**: mutual dependency — hardest

The Partitioning Problem

Objective: Find $P \in \{0,1\}^{n \times n}$ such that $A' = P^\top AP$ minimises:

Objective 1 — Feed-Back Marks

$$|\text{FBM}| = \left|\{a'_{ij} \neq 0 \mid j > i\}\right|$$

Minimise the **count** of above-diagonal nonzeros.

Objective 2 — Feedback Distance

$$\text{TFBD} = \sum_{a'_{ij} \in \text{FBM}} (j - i)$$

Minimise the **distance** of feed-back marks from the diagonal.

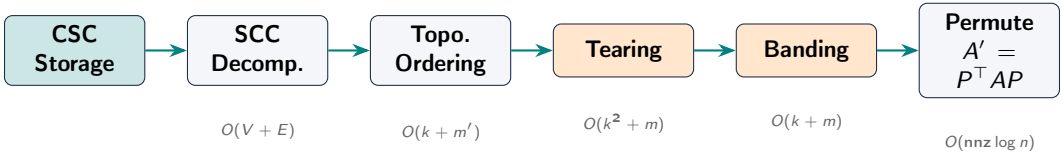
Best case

$G(A)$ has no directed cycles $\Rightarrow$ fully lower triangular,
$|\text{FBM}| = 0$, $\text{TFBD} = 0$

General case

Achieve **Block Lower Triangular Form (BLTF)**: group mutually dependent tasks into diagonal blocks and push remaining feed-back marks close to the diagonal.

Our 6-Stage Pipeline



Data Structure: CSC

- Column pointer array $\texttt{col_ptr}[0 \ldots n]$
- Row index array $\texttt{row_ind}[0 \ldots \text{nnz} - 1]$
- Storage: $\Theta(n + \text{nnz})$
- No value array needed (binary DSM)

Why CSC?

Column j gives the neighbours of vertex j in $O(\deg(j))$ — ideal for DFS traversal.

SCC Decomposition

Both algorithms built on a shared iterative DFS with callback hooks, using an explicit stack (no recursion).

Tarjan's Algorithm

- **Single DFS pass**
- Each vertex v maintains $low(v)$: smallest reachable pre-order time
- $low(v) = v.pre$ at finish $\Rightarrow$ v is SCC root; pop stack
- $O(V + E)$ time, $O(V)$ space

Kosaraju-Sharir Algorithm

- **Two DFS passes**
- Pass 1: DFS on G, record finish order
- Pass 2: DFS on $rev(G)$ in reverse finish order; each DFS tree = one SCC
- $O(V + E)$ time, $O(V + E)$ space

After SCC decomposition, the **condensation DAG** is topologically sorted with **Kahn's algorithm** to determine the block ordering.

Tearing & Banding

Tearing (reduces FBM within SCCs)

- **Algorithm GR** (Eades, Lin, Smyth):
- Repeatedly peel sinks $\rightarrow$ append to back, sources $\rightarrow$ prepend to front
- When neither exists, pick vertex with max $d^+ - d^-$ as a source
- The resulting ordering defines which edges are feedback arcs (backward edges)

Banding (reduces TFBD within SCCs)

- **Reverse Cuthill-McKee (RCM)**: concentrates nonzeros near diagonal
- **Adjacent-swap refinement**: accepts swaps that lower TFBD without increasing FBM
- Best of torn + RCM orderings kept

Experimental Setup

Benchmark suite

- 10 sparse matrices from the project repository
- Dimensions: $n = 11$ to 27,770, nnz $= 4$ to 352,807

Hardware & methodology

- Apple M2 / 8 GB RAM (L1d 64 KB, L1i 128 KB, L2 4 MB), compiled with -O3, Apple LLVM 17
- Each algorithm: 2 warm-up + 5 timed runs
- Metrics: FBM count, TFBD, and mean runtime (μs)

Stage-wise Quality Improvement

Average reductions across 10 matrices:

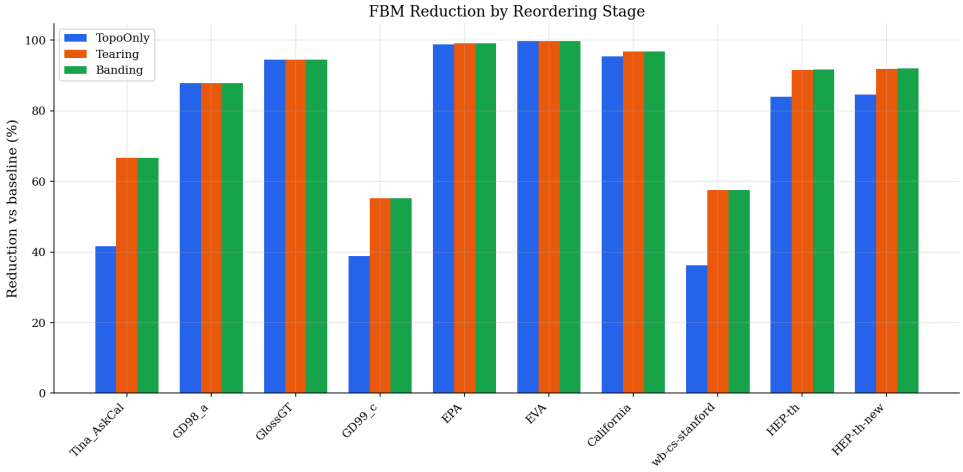
Stage	FBM ↓	TFBD ↓
Baseline	—	—
TopoOnly	76.18%	95.21%
+ Tearing	84.12%	93.95%
+ Banding	**84.15%**	**94.54%**

- Topo ordering provides the **largest single gain**
- Tearing drives further FBM reduction within SCCs
- Banding polishes TFBD without hurting FBM

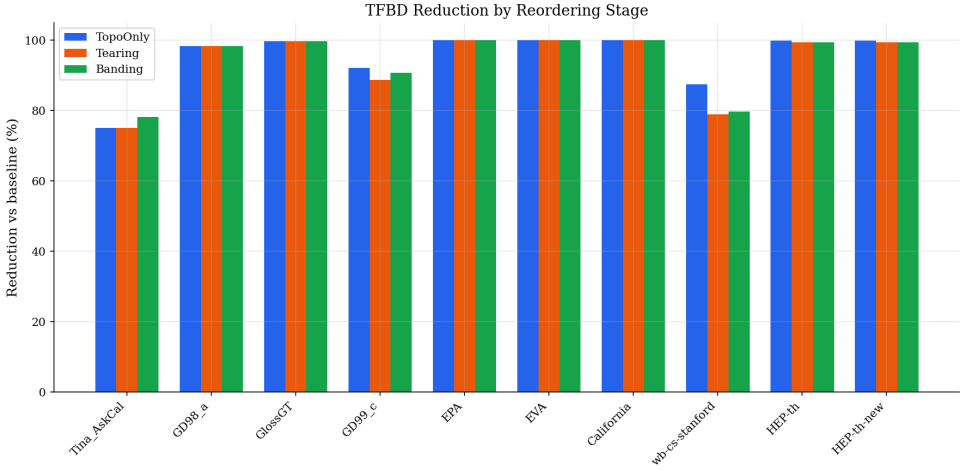
Selected matrix results

Matrix	$FBM_0 \rightarrow FBM_f$	$TFBD_0 \rightarrow TFBD_f$
Tina_AskCal	$12 \rightarrow 4$	$32 \rightarrow 7$
California	$6063 \rightarrow 192$	$18M \rightarrow 669$
wb-cs-stanford	$20131 \rightarrow 8537$	$8.7M \rightarrow 1.78M$
HEP-th-new	$104435 \rightarrow 8281$	$1.7B \rightarrow 10.1M$

FBM Reduction



TFBD Reduction



Runtime Comparison: Tarjan vs. Kosaraju-Sharir

Key findings

- Both algorithms produce **identical SCC decompositions**
- Tarjan: mean $\approx 560\,\mu$s
- Kosaraju-Sharir: mean $\approx 992\,\mu$s
- Tarjan is $\approx 1.77\times$ faster on average
- Scaling compatible with expected $O(V + E)$ near-linear behaviour

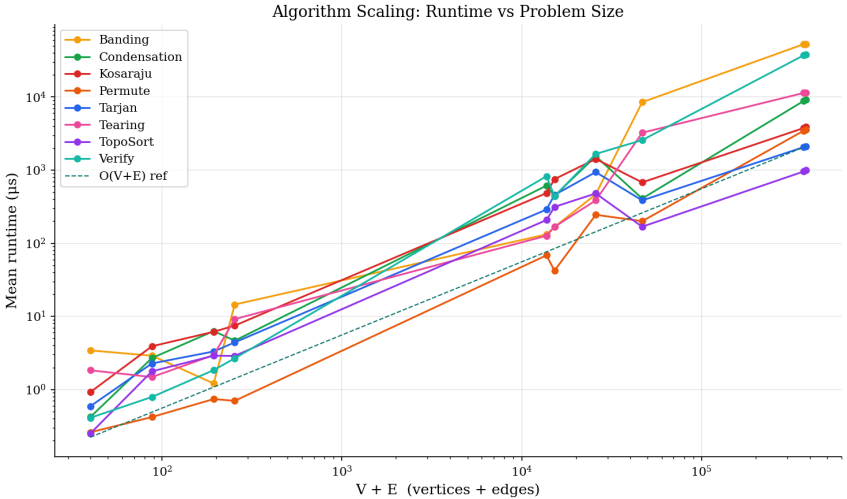
Pipeline stage runtimes

Stage	Relative cost
Tarjan SCC	low
Kosaraju SCC	$1.77\times$ Tarjan
Condensation	low
Topo sort	low
Tearing	moderate
Banding	highest
Permutation	low

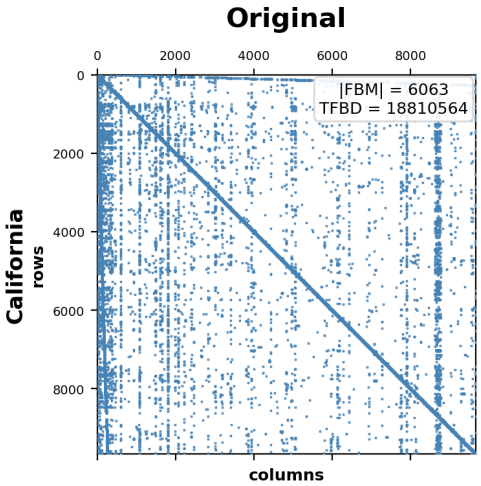
Why is Tarjan faster?

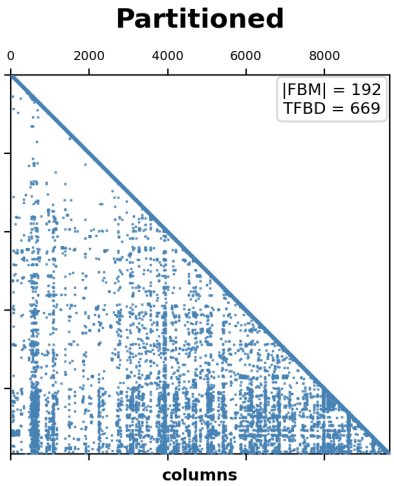
Single DFS pass vs. two passes + transpose graph construction in Kosaraju-Sharir.

Runtime Comparison: Pipeline

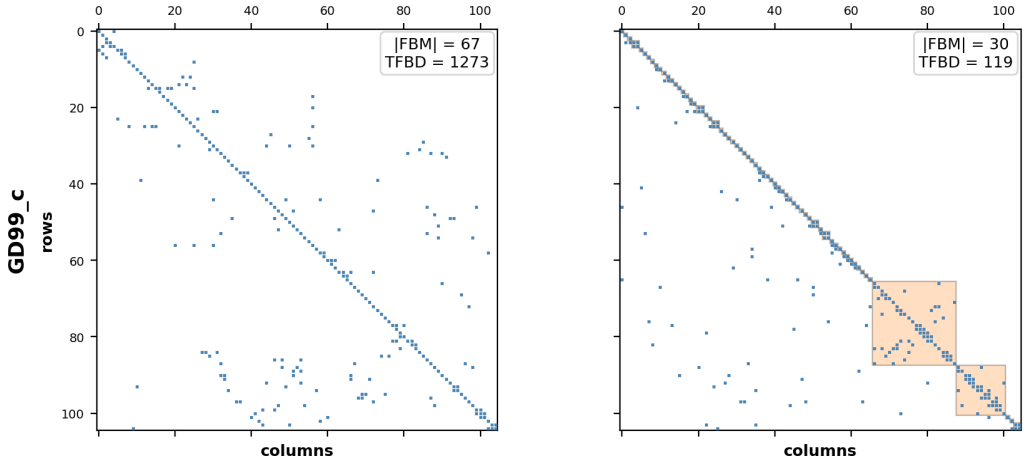


Qualitative Results: California

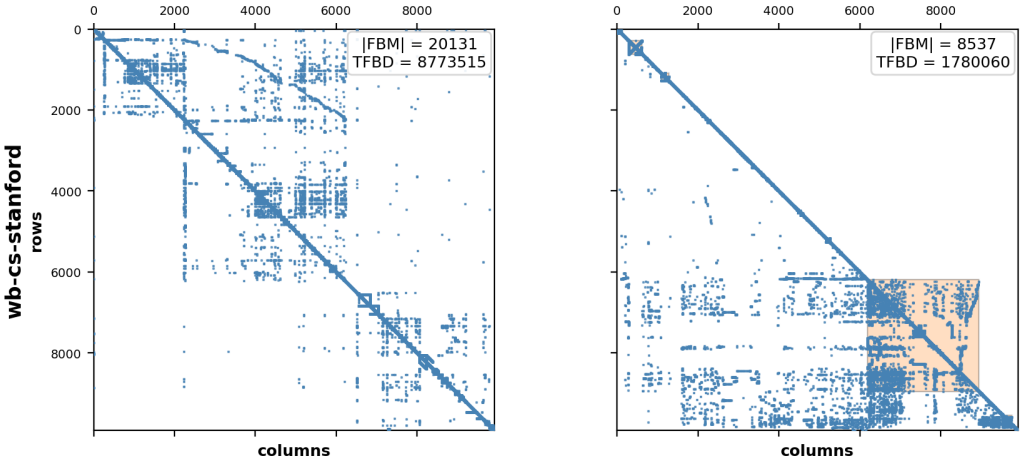




Qualitative Results: GD99_c



Qualitative Results: wb-cs-stanford



Conclusions & Future Work

What we achieved

- Implemented full 6-stage DSM reordering pipeline in C++ with CSC
- Both Tarjan and Kosaraju-Sharir SCC algorithms — identical decompositions
- Tearing via **Algorithm GR** (greedy FBM reduction), banding via **Reverse Cuthill-McKee** + adjacent-swap refinement (TFBD reduction)
- **84.15% average FBM reduction**, **94.54% average TFBD reduction** across 10 benchmark matrices

Future directions

- **Better tearing**: replace greedy Algorithm GR with an exact or improved heuristic for minimum feedback arc set
- **Parallelism**: per-SCC tearing & banding are independent — parallelise for large matrices
- **Weighted DSMs**: extend to numerical dependency strengths; optimise weighted feedback cost

Acknowledgements & Team Contributions

AI Disclosure

Claude (Opus 4.6 & Sonnet 4.6) and ChatGPT (GPT-4o) were used to understand algorithms/terminology and to write Python plotting scripts. All C++ code and report text were written by the authors without AI assistance.

Team Member Contributions

Member	Contribution
Simon Kraft	Implemented Tarjan's algorithm, Topological Sort, and python scripts for original vs permuted DSM. Wrote report introduction and abstract.
Joshua Payne	Implemented Kosaraju-Sharir algorithm, tearing & banding. Wrote report description of solution.
El Sall	Implemented permutation application, benchmark suite, python scripts for report figures. Wrote report numerical results section.

Thank you — questions welcome!