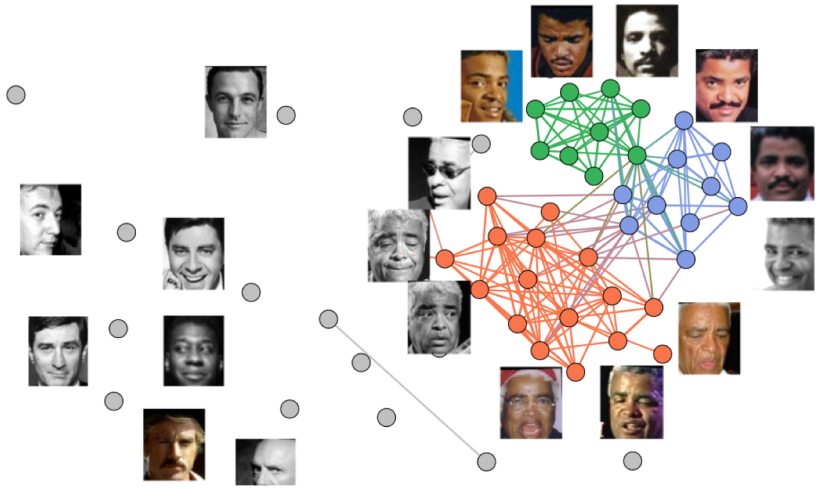
Scalable Community Detection via Parallel Correlation Clustering

Jessica Shi (Google / MIT)
Laxman Dhulipala (Google)
David Eisenstat (Google)
Jakub Łącki (Google)
Vahab Mirrokni (Google)

Graph Clustering

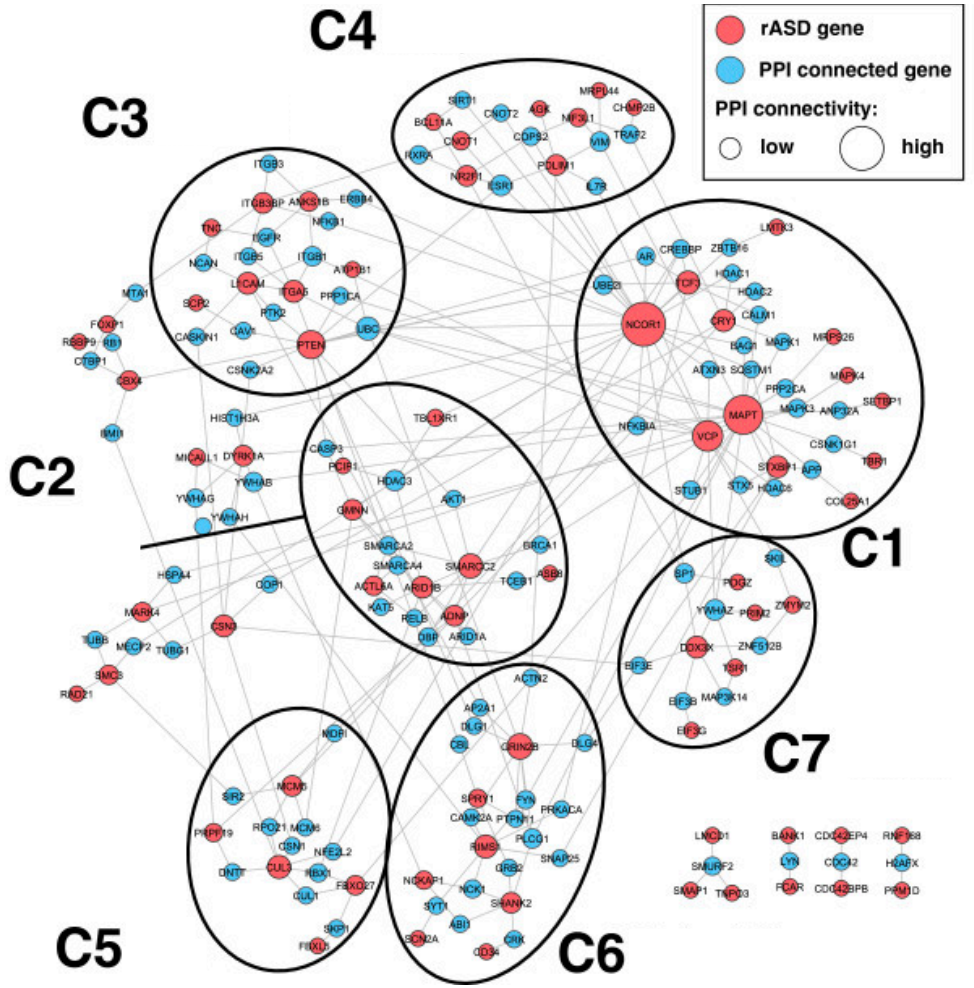
Social Networks

https://github.com/XinyueTan/Social-Network-Analysis-



Facial Recognition

A community detection approach to cleaning extremely large face database (Jin et al., 18)



Bioinformatics

DAWN: A framework to identify autism genes and subnetworks using gene expression and genetics (Liu et al., 14)

Parallelism

- Parallelism enables us to efficiently process large graphs

Apple, Microsoft, Intel, https://www.flickr.com/photos/66016217@N00/2556707493/, HP

Correlation Clustering

- Main goal: Scalable graph clustering framework with high-quality on ground truth data

- LambdaCC objective [1]: Generalized objective unifying quality measures (modularity, sparsest cut, cluster deletion)
- For edge weights w_{ij}, node weights k_i, and resolution $\lambda \in (0, 1)$, maximize:

$$\sum_{(i,j)\in V\times V \text{ where } i,j \text{ are in the same cluster}} (w_{ij} - \lambda k_i k_j)$$

[1] Veldt, Gleich, Wirth (18)

Main Results

- Highly optimized correlation clustering implementation, **Par-CC**

- Tunable optimizations with comprehensive evaluation of performance and quality improvements
- Up to 28.44x speedups over sequential baselines
- High precision and recall compared to ground truth clusters, with trade-offs depending on the resolution parameter

Main Results

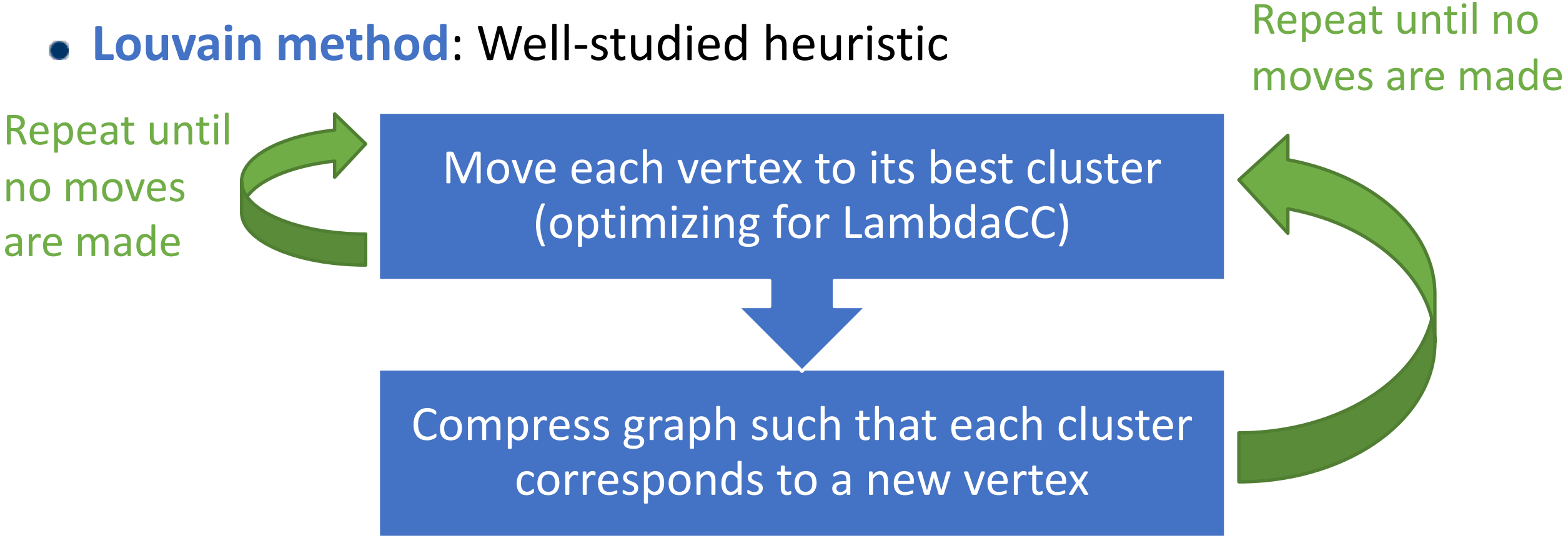
- Improved performance and quality over state-of-the-art clustering implementations
  - Significantly better objective obtained compared to pivot-based correlation clustering (C4, ClusterWild) [1]
  - Up to 3.5x speedup over parallel modularity clustering (NetworKit) [2]
  - High precision and recall compared to ground truth, outperforming triangle-based clustering (TECTONIC) [3]

[1] Pan, Papailiopoulos, Oymak, Recht, Ramchandran, Jordan (15)
[2] Staudt, Meyerhenke (16)
[3]Tsourakakis, Pachocki, Mitzenmacher (17)

Parallel Correlation Clustering Algorithm

Louvain Method

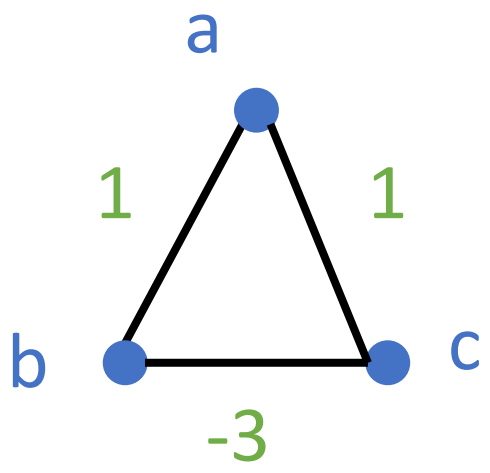
- NP-hard to optimize for the LambdaCC objective [1]
- **Louvain method**: Well-studied heuristic



[1] Dinh, Li, Thai (15)

Parallelizing Louvain Method

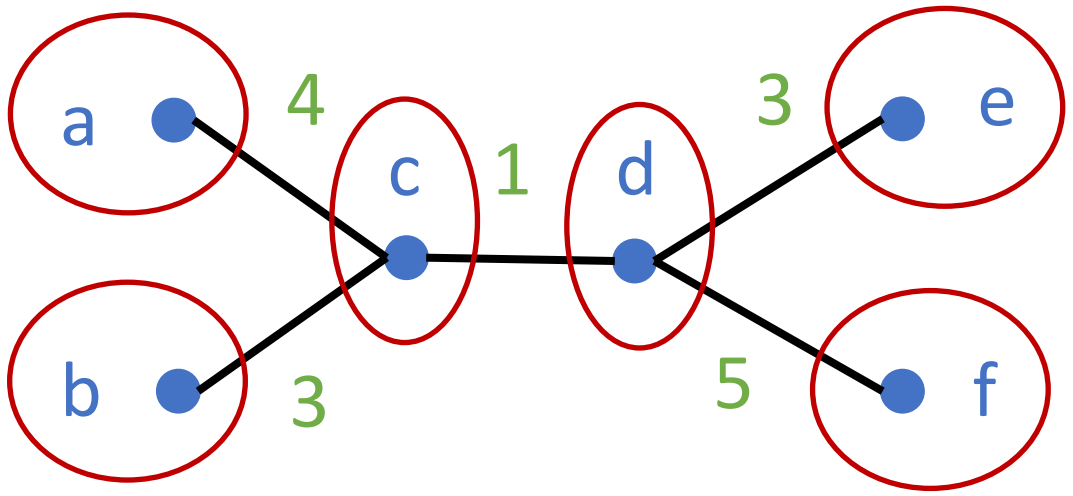
- Bottleneck: Sequential dependencies in moving vertices to best cluster



If b clusters with a, then c's best move is not to cluster with a (and vice versa)

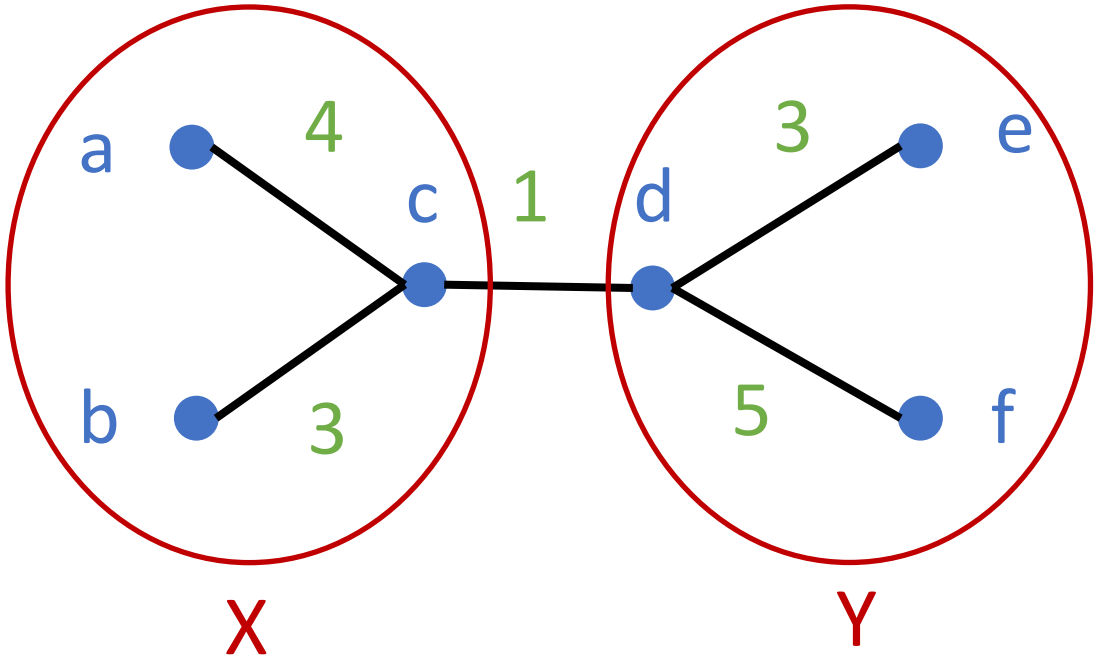
- Solution: Relax sequential dependency and allow vertices to move concurrently
  - No convergence guarantee (use a constant cutoff)

Parallel Louvain Method: Best Move



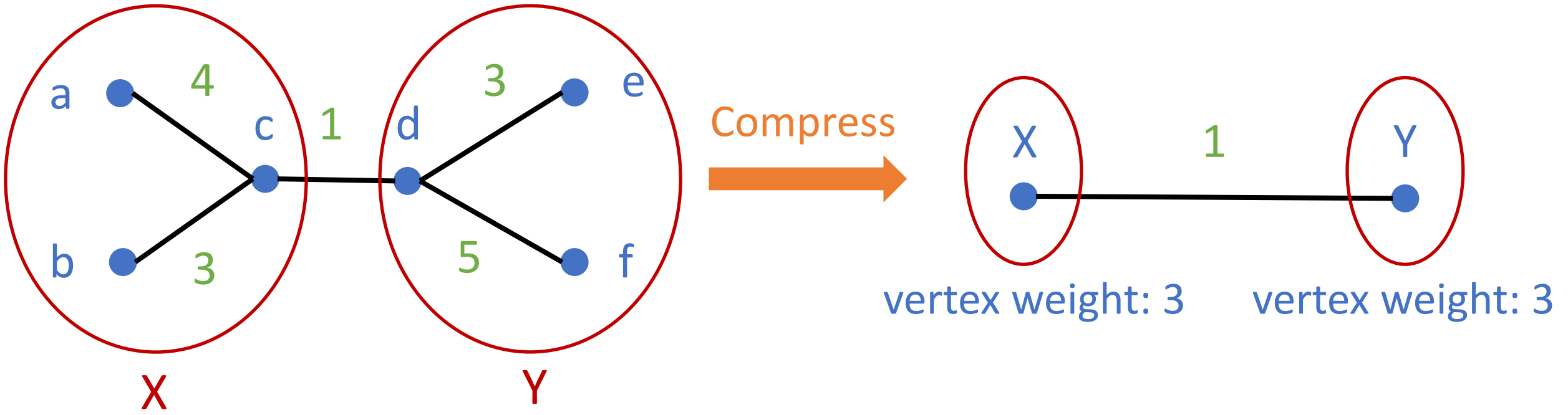
Best Move	Change in CC Objective
Vertex a → Cluster c	$4 - \lambda$
Vertex b → Cluster c	$3 - \lambda$
Vertex c → Cluster b	$4 - \lambda$
Vertex d → Cluster f	$5 - \lambda$
Vertex e → Cluster d	$3 - \lambda$
Vertex f → Cluster d	$5 - \lambda$

Parallel Louvain Method: Best Move

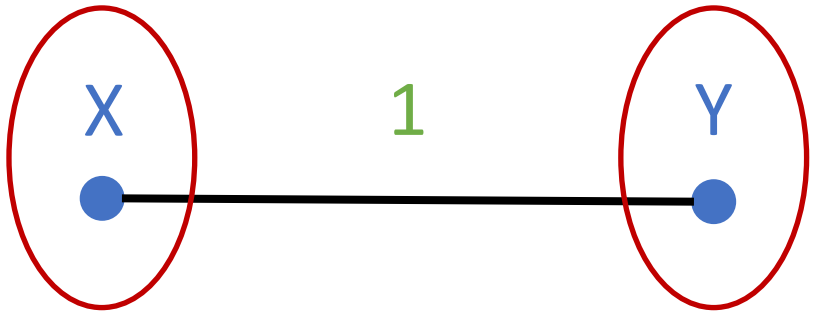


Best Move	Change in CC Objective
Vertex a → Cluster X	0
Vertex b → Cluster X	0
Vertex c → Cluster X	0
Vertex d → Cluster Y	0
Vertex e → Cluster Y	0
Vertex f → Cluster Y	0

Parallel Louvain Method: Compress



Parallel Louvain Method: Best Move



vertex weight: 3 vertex weight: 3

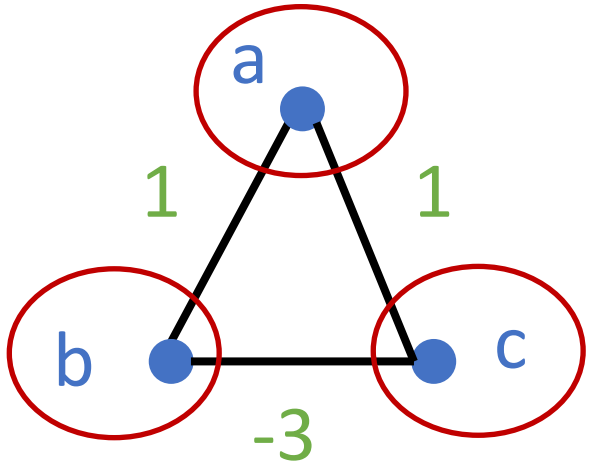
Best Move	Change in CC Objective
Vertex X → Cluster X	0
Vertex Y → Cluster Y	0

No more best moves

Practical Optimizations

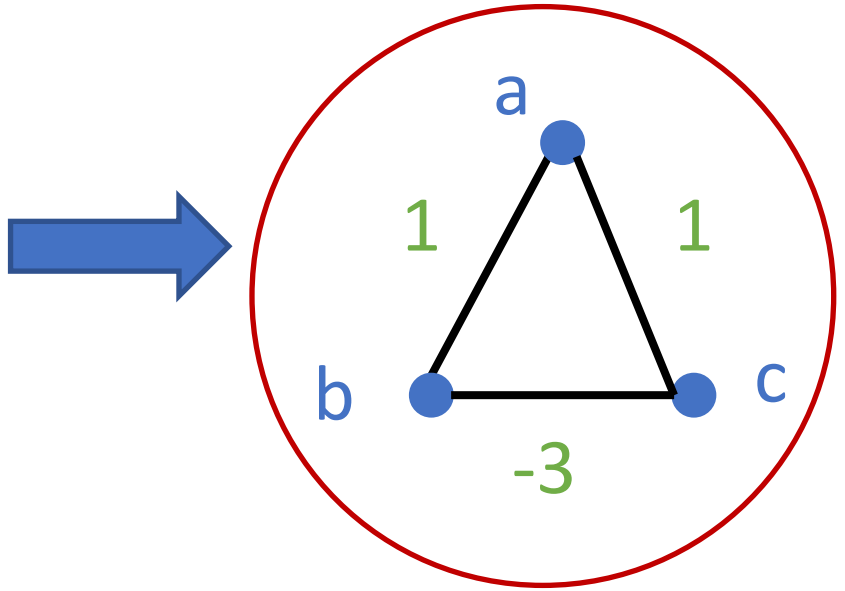
Optimization: Synchronous vs Asynchronous

- In performing best vertex moves,
- Synchronous: Compute the desired cluster of each vertex in parallel, and then move all vertices to their chosen clusters in parallel



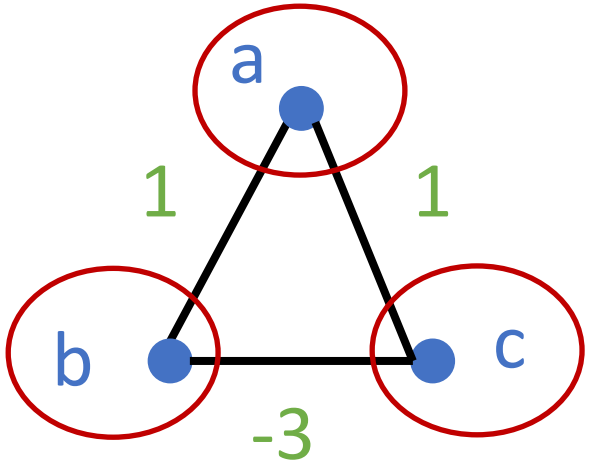
Best Move
Vertex a → Cluster b
Vertex b → Cluster a
Vertex c → Cluster a

LambdaCC Objective = -1

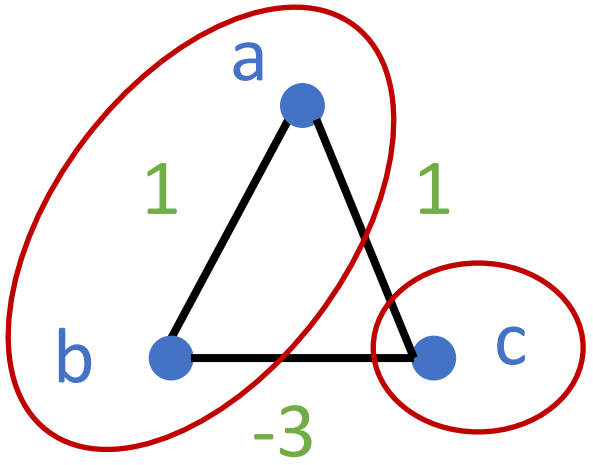


Optimization: Synchronous vs Asynchronous

- In performing best vertex moves,
- Asynchronous: Compute the desired cluster of each vertex and immediately move vertex to chosen cluster
  - Relaxes consistency guarantees



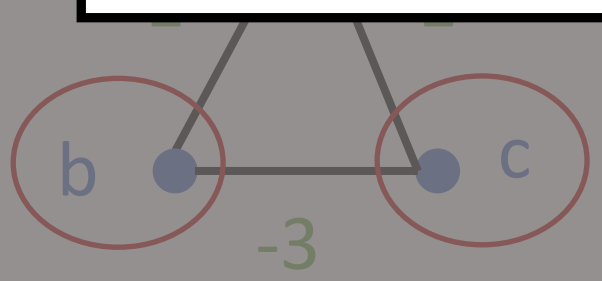
Best Move
Vertex a → Cluster b
Vertex b → Cluster a



Optimization: Synchronous vs Asynchronous

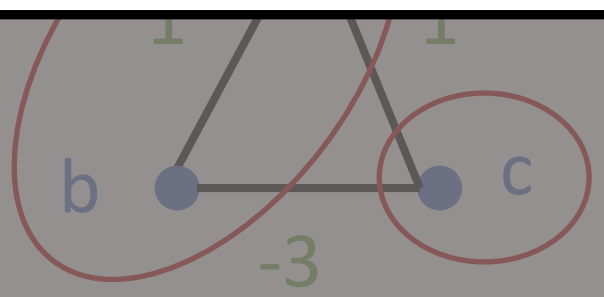
- In performing best vertex moves,
- Asynchronous: Compute the desired cluster of each vertex and

- Up to 2.5x speedups using asynchronous over synchronous (1.21x median)
- 1.29 – 156.01% increase in objective using asynchronous over synchronous



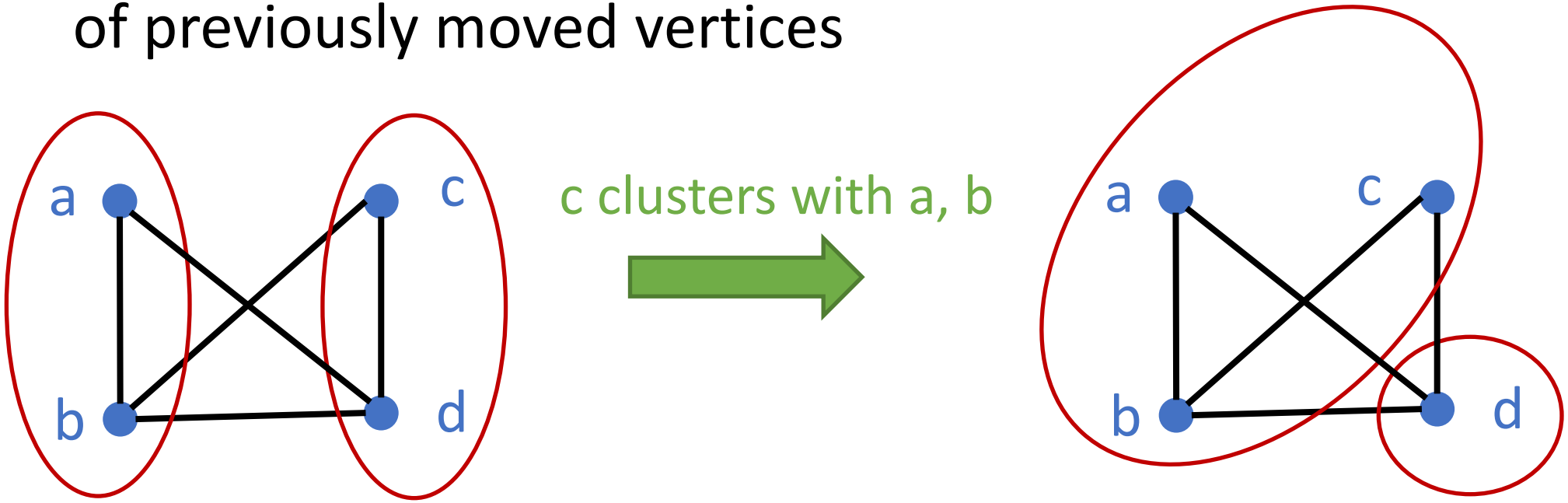
Vertex a → Cluster b

Vertex b → Cluster a



Optimization: Subset of Vertices

- Instead of considering all vertices in best moves,
- Neighbors of vertices: Consider only vertices that are neighbors of previously moved vertices



Consider only vertices b and d in the next round of best moves

Optimization: Subset of Vertices

- Instead of considering all vertices in best moves,
- Neighbors of clusters: Consider only vertices that are neighbors

- Up to 1.98x speedups using neighbors of vertices over all vertices (1.03x median)

Consider only vertices a, b, and d in the next round of best moves

Optimization: Multi-level Refinement

- Multi-level refinement: After the algorithm is finished and the last compressed graph is computed, traverse back through previous compressed graphs in order + repeat the best moves subroutine
  - Particularly helpful if best moves does not converge when graph compression occurs

Optimization: Multi-level Refinement

- Multi-level refinement: After the algorithm is finished and the last compressed graph is computed, traverse back through

- Up to 2.29x slowdowns using multi-level refinement (1.67x median)
- 1.12 – 36.92% increase in objective using multi-level refinement

Best Optimizations

- Asynchronous
- Neighbors of vertices
- Multi-level refinement

- Up to 5.85x speedups using these optimizations
- Up to a 156% increase in objective using these optimizations

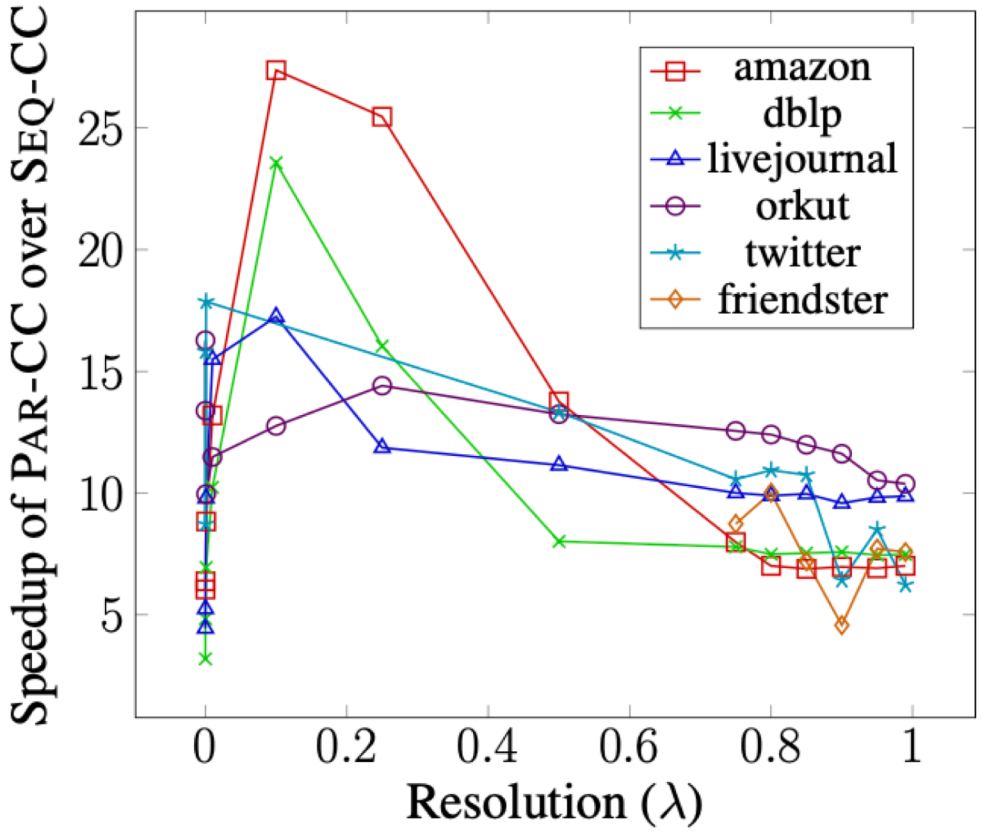
Experiments

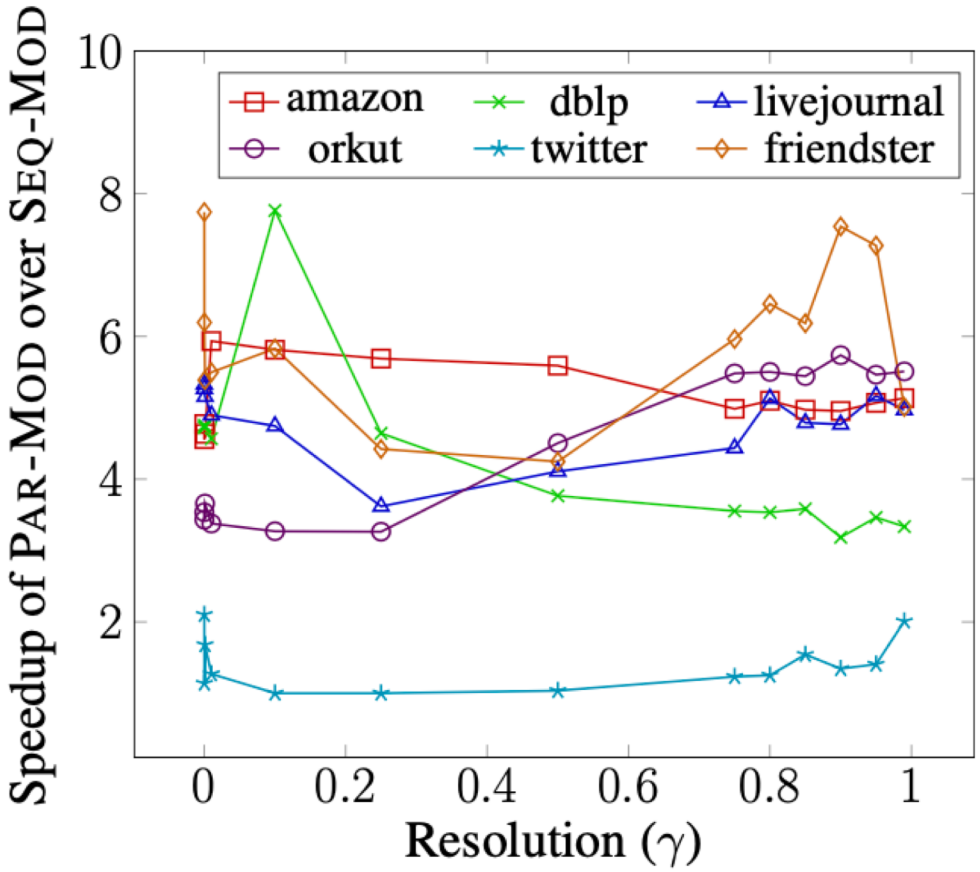
Environment

- 30-core GCP instance (2-way hyper-threading), 240 GiB main memory
- 48-core GCP instance (2-way hyper-threading), 1434 GiB main memory for large graphs

- Graphs with ground-truth communities:
- Unweighted real-world Stanford Network Analysis Platform (SNAP) graphs with up to 1.8 billion edges
- Weighted graphs from computing k-NN on real-world pointsets from the UCI Machine Learning repository

Speedups over Sequential Baselines



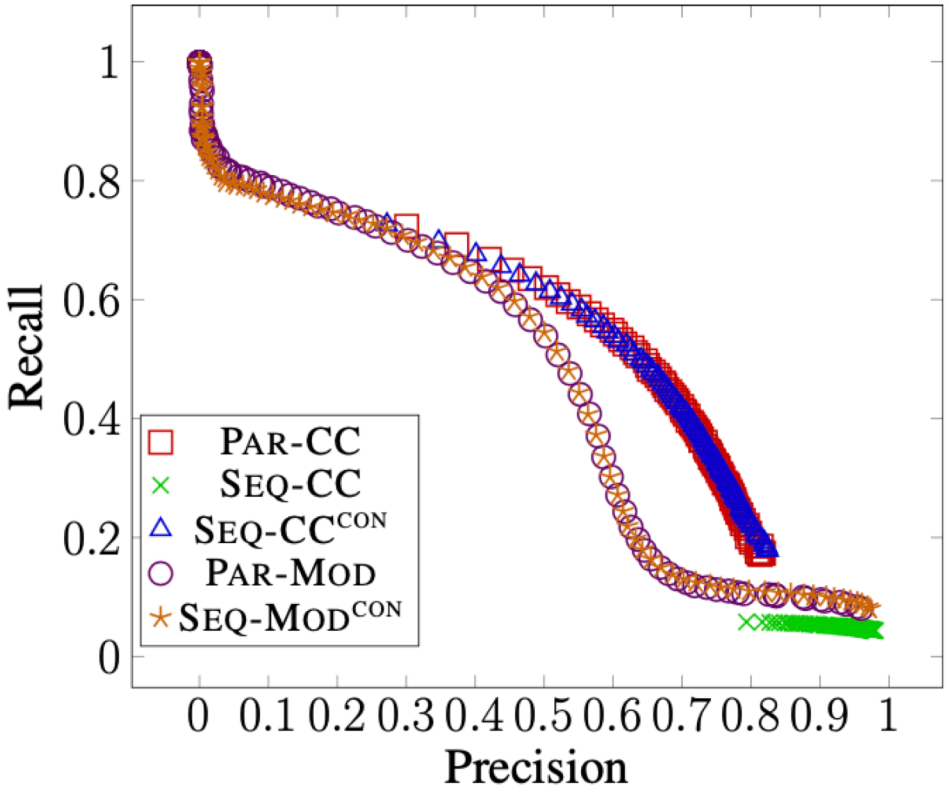


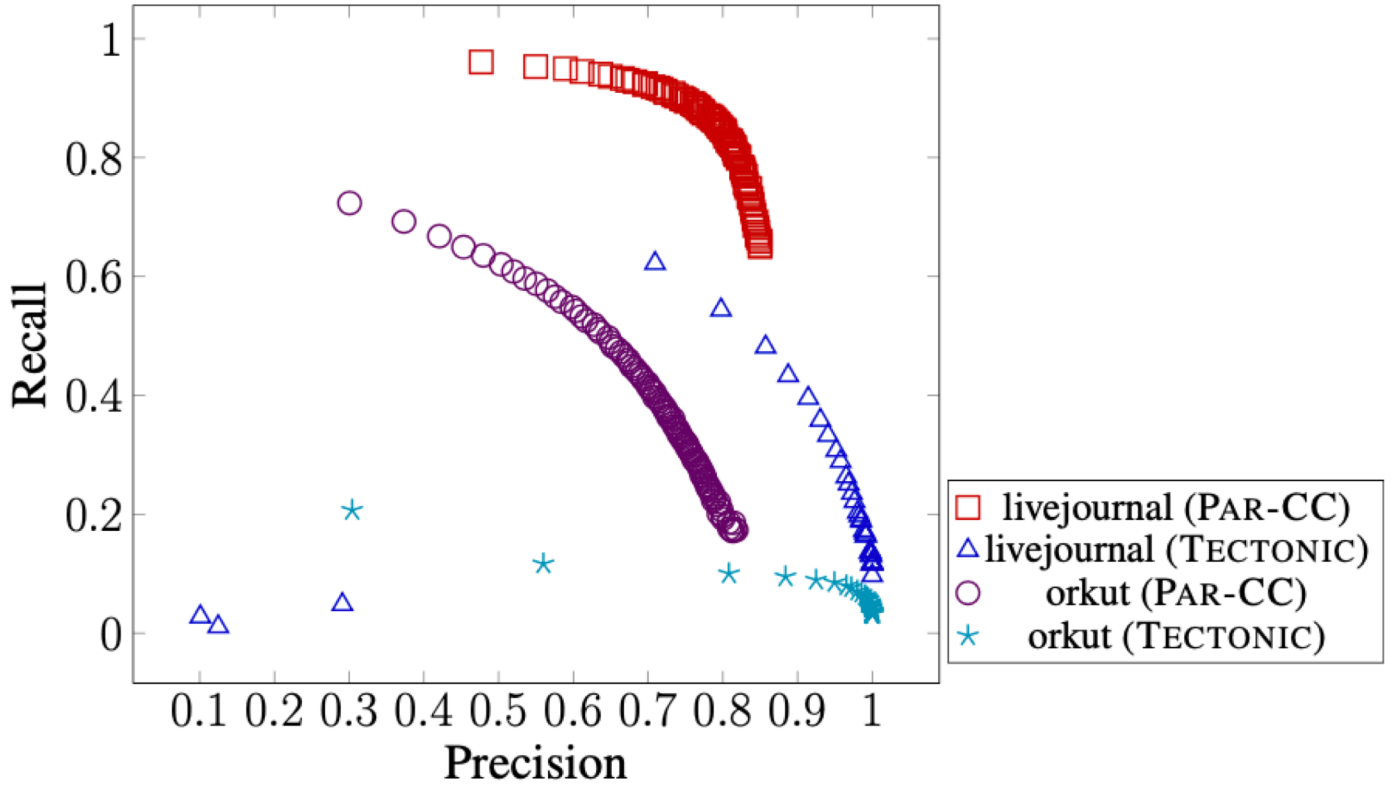
Comparison to Existing Baselines

- **Pivot-based correlation clustering**:
  - C4, ClusterWild! [1] are up to 429x faster than Par-CC
  - C4, ClusterWild! give a 273 – 433% decrease in objective compared to Par-CC
- **Parallel modularity clustering**:
  - Par-Mod is up to 3.5x faster than NetworKit [2]
- **Triangle-based clustering**:
  - Par-CC is up to 67.62x faster than TECTONIC [3]

[1] Pan, Papailiopoulos, Oymak, Recht, Ramchandran, Jordan (15)
[2] Staudt, Meyerhenke (16)
[3]Tsourakakis, Pachocki, Mitzenmacher (17)

Comparison to Ground Truth





Conclusion

Conclusion

- Scalable graph clustering framework Par-CC with high-quality on ground truth data
- Improved performance and quality over state-of-the-art clustering implementations

- Code: https://github.com/jeshi96/parallel-correlation-clustering