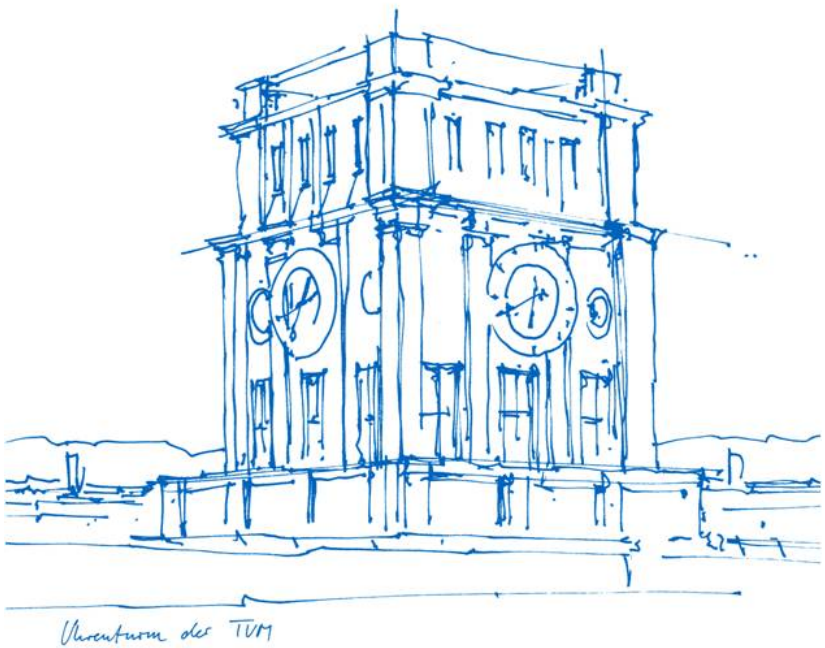
Efficient Batched Distance and Centrality Computation in Unweighted and Weighted Graphs

Manuel Then, Stephan Günnemann, Alfons Kemper, Thomas Neumann

Technische Universität München

Chair for Database Systems



Graph Centrality

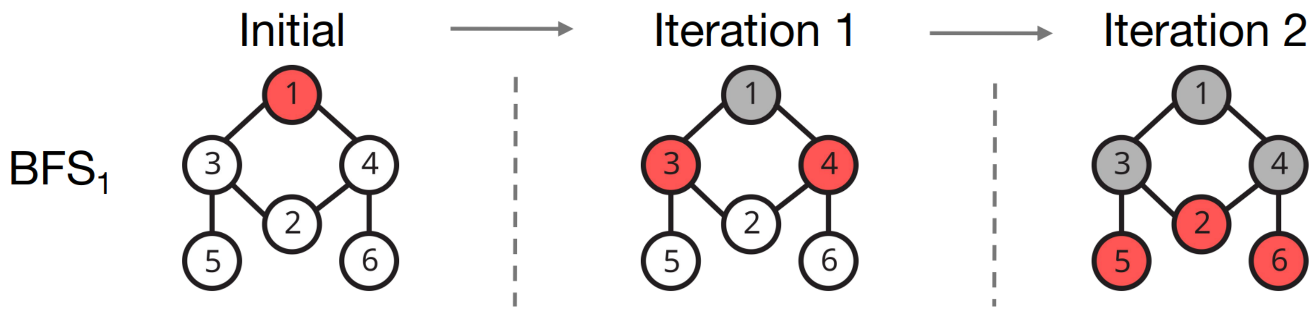
Goal: Find the most **central vertices**
- Influencers in social networks
- Critical routers in computer networks

Centrality measures
- **Degree**: degree centrality, PageRank
- **Distances**: closeness centrality
- **Paths**: betweenness centrality

Challenges
- Algorithmic complexity
- Random data access
- Redundant computation, hard to vectorize

Challenges Visualized

Unweighted closeness centrality build on BFSs

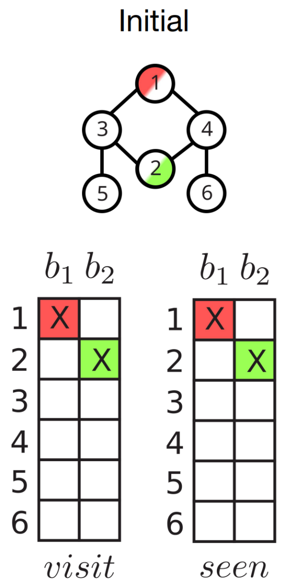


Goal: Run multiple BFSs concurrently and share common traversals

Background: Multi-Source BFS

BFS traversals using bit operations

∀v∈V: ∀n∈neighbors(v): next[n] = visit[v] & ~seen[n]



Used to win SIGMOD 2014 programming contest

[1] Then et al., The More the Merrier: Efficient Multi-source Graph Traversal, VLDB 2015
[2] Kaufmann et al., Parallel Array-Based Single- and Multi-Source Breadth First Searches on Large Dense Graphs, EDBT 2017

Overview

Motivation: Graph Centrality

Background: MS-BFS

Centrality in **unweighted** graphs

Centrality in **weighted** graphs

Evaluation

Summary and Future Work

Unweighted Closeness Centrality

Distance-based centrality metric

- Central vertices have a low average geodesic distance to all other vertices

$$CC_v = \frac{|reachable(v)|^2}{(|V|-1) * (\sum_{u \in reachable}(v) : distance(v,u))}$$

MS-BFS from all vertices

- No need to store distances

Efficient batch incrementer

- Significantly improves the performance of counting discovered vertices

Unweighted Betweenness Centrality

Path-based centrality metric

- Central vertices are part of many shortest paths

$$BC_v = \sum_{u,w \in V,\ u \neq v \neq w} : \frac{|\{\mathscr{P} \mid \mathscr{P} \in shortest\_paths(u,w) \wedge v \in \mathscr{P}\}|}{|shortest\_paths(u,w)| * (|reachable(v)|) * (|reachable(v)| - 1)}$$

Naïve computation very costly. We use Brandes's algorithm

Forward step can leverage MS-BFS

- Batching **improves locality**
- Allows **vectorization** of numeric computations

Challenges: Backward step requires

- Reverse MS-BFS
- Vertex predecessor calculation

[3] Brandes, A Faster Algorithm for Betweenness Centrality, Journal of Mathematical Sociology, 2001

Reverse MS-BFS and Vertex Predecessors

Reverse BFS: traverse graph in inverse BFS order

- Stacks unsuited for MS-BFS

Reconstruct traversal order forward iteration frontiers

Batched vertex **predecessor computation**

$$predecessorIn(p,v) = \begin{cases} frontiers[iter-1][p] \ \& \ frontiers[iter][v], & if\ (p,v) \in E \\ \varnothing, & otherwise \end{cases}$$

Correctness proof and full batched betweenness centrality algorithm in the paper

Overview

Motivation: Graph Centrality

Background: MS-BFS

Centrality in **unweighted** graphs

Centrality in **weighted** graphs

Evaluation

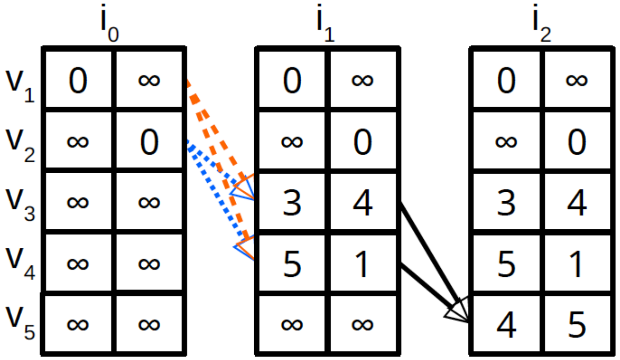
Summary and Future Work

Batched Algorithm Execution

Problem: MS-BFS cannot be used for distance computation in weighted graphs

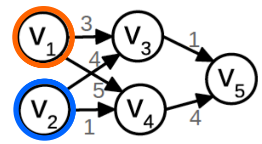
Batched Algorithm Execution

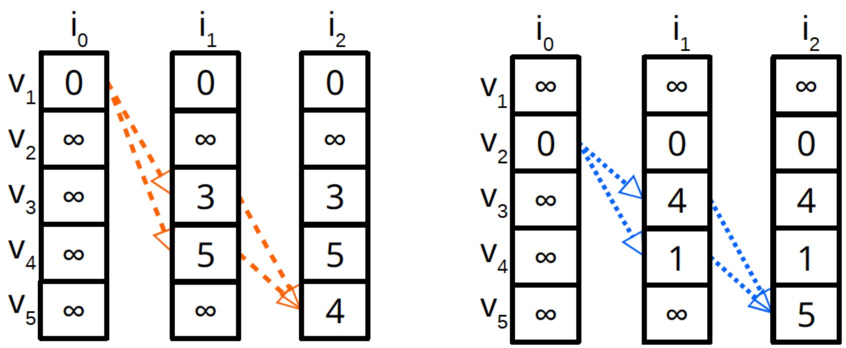
- Run algorithm from **multiple** vertices **at the same time**
- Synchronize algorithm executions
- **Share** common **computations** and **data accesses**
- Adapt memory layout



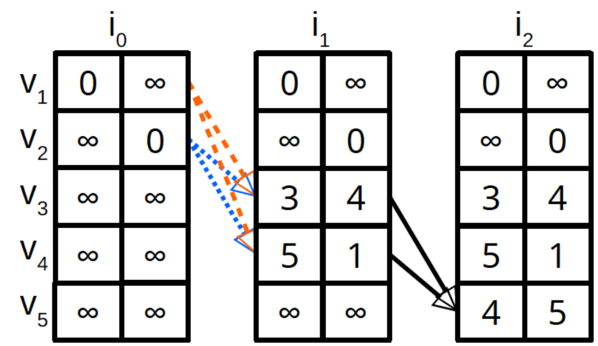
Batched Algorithm Execution: Example

Batched Bellman-Ford algorithm
Weighted all pairs shortest path





Non-batched execution



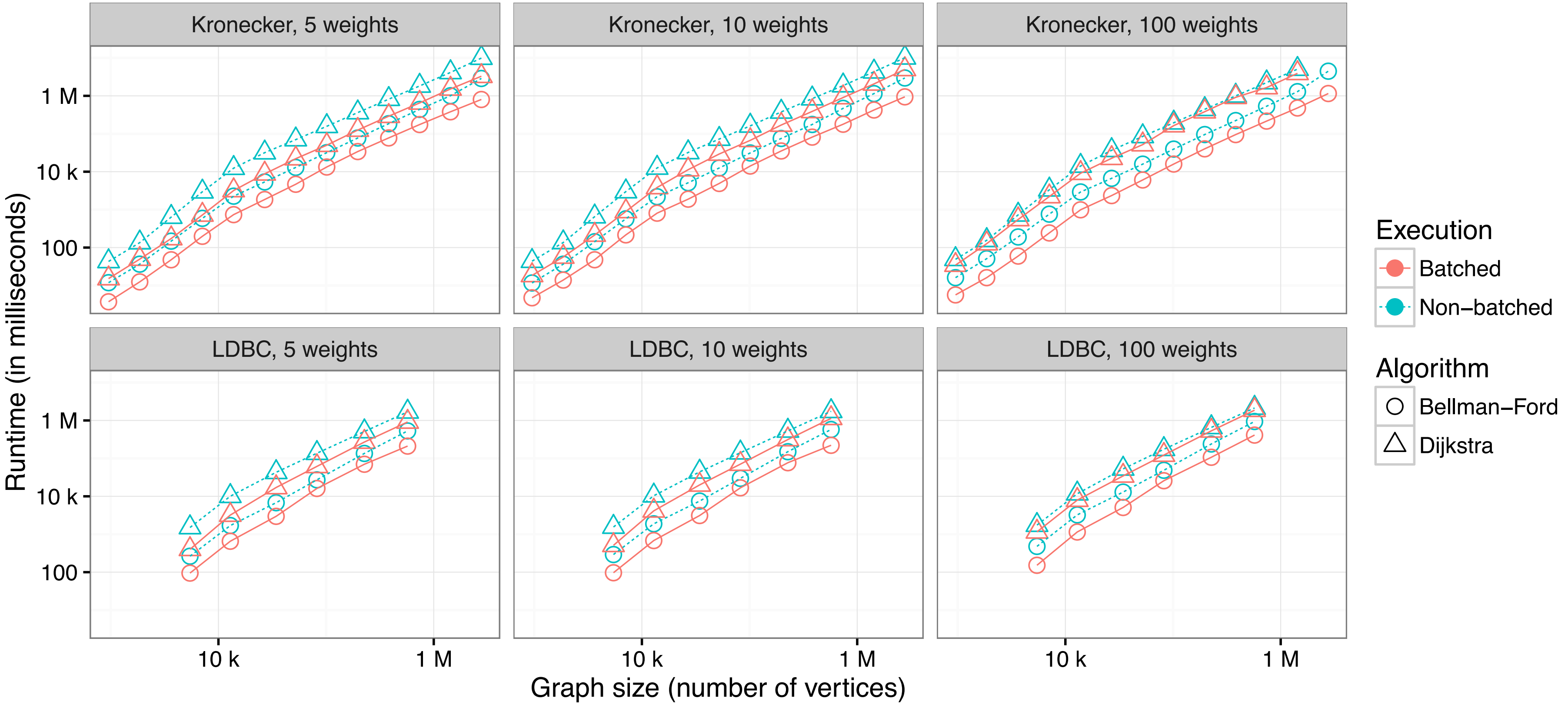
Batched execution

Batched algorithm execution

- … **improves** temporal and spatial **locality**
- … facilitates **vectorized computation**

Batched Weighted Distances

Comparison of common weighted distance algorithms:



Weighted Centralities

Closeness Centrality

- Batched execution allows vectorizing the CC computation from the distances

Betweenness Centrality

- Requires **global distance ordering**
- Implicit predecessor computation
- Vectorized numeric computations

Overview

Motivation: Graph Centrality

Background: MS-BFS

Centrality in **unweighted** graphs

Centrality in **weighted** graphs

Evaluation

Summary and Future Work

Evaluation: Setup

Algorithms implemented as stand-alone programs

- C++14, GCC 5.2.1
- No framework dependencies

Synthetic datasets

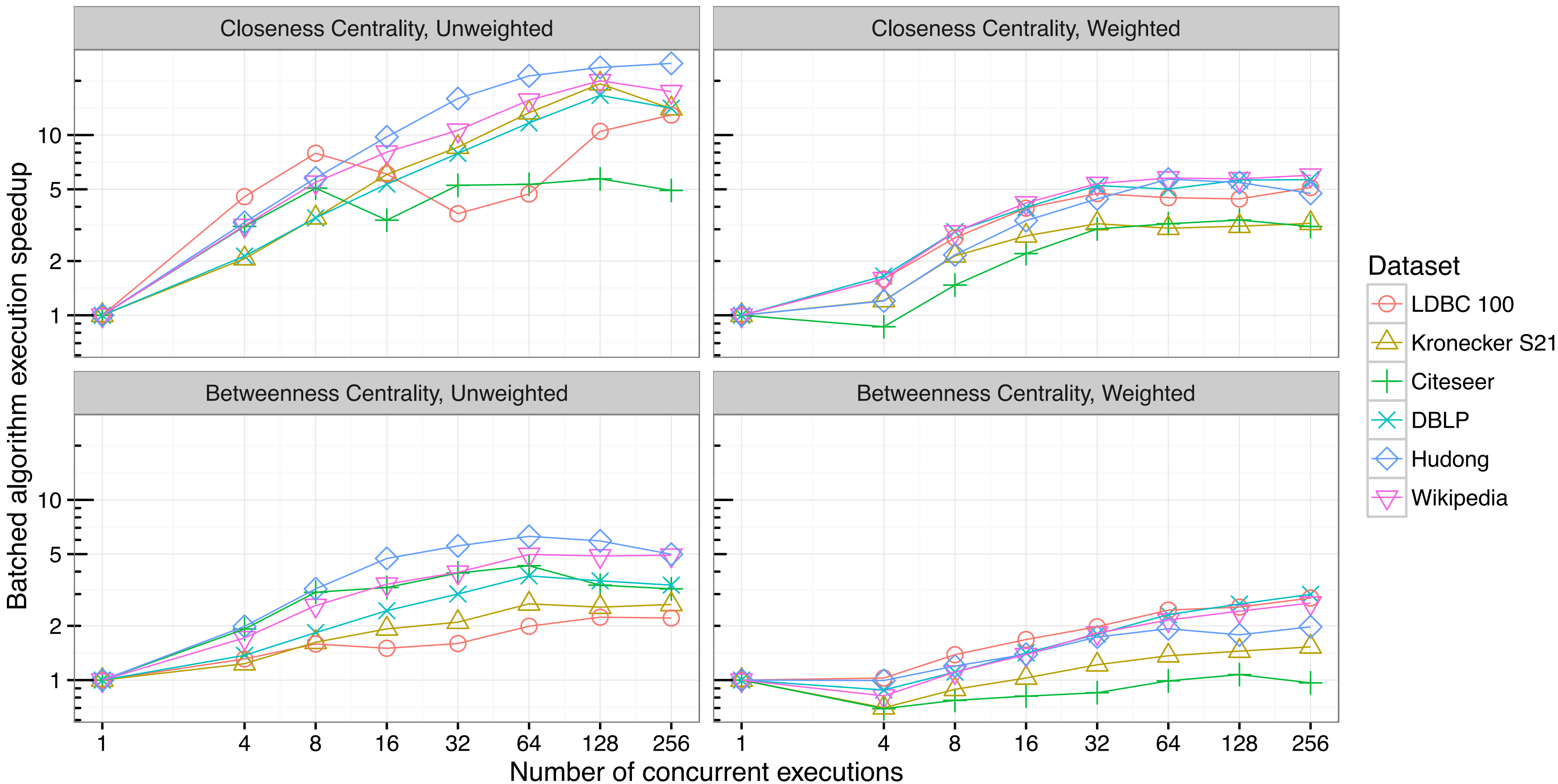
- LDBC Social Network friendships graph
- Kronecker graph, edge factor 32

Real-world datasets

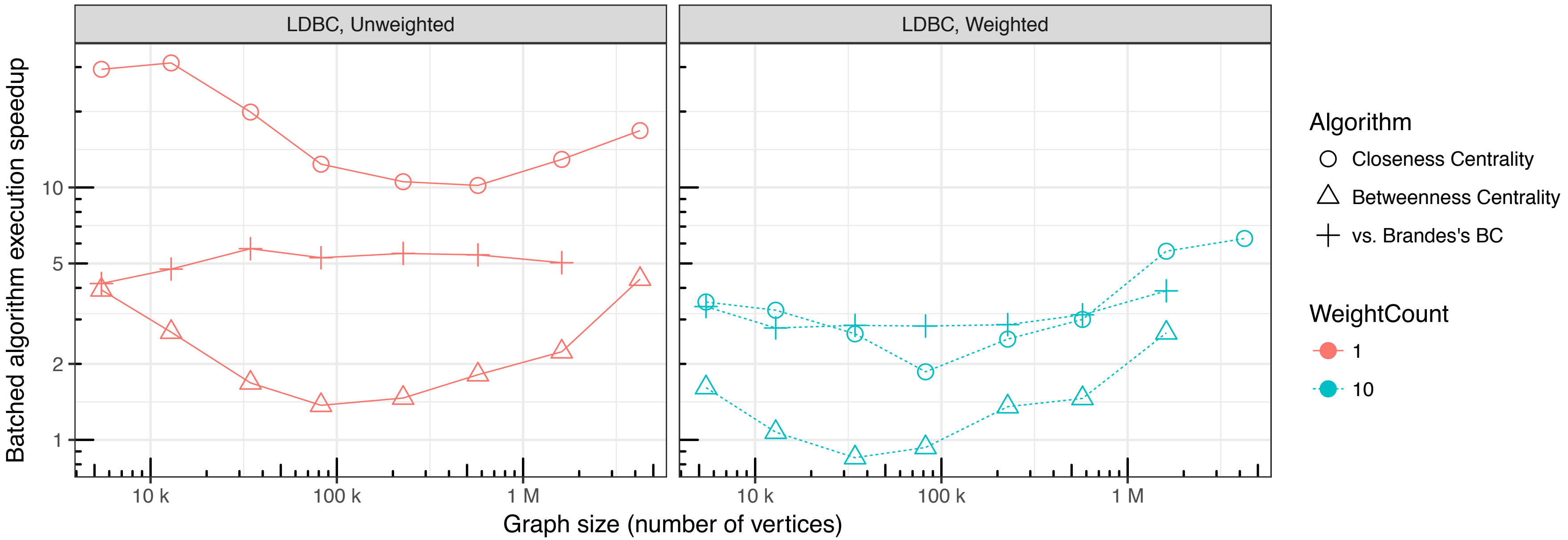
- Citeseer (384k verts), DBLP (1.3M verts), Wikipedia (1.9M verts), and Hudong (3M verts)
- KONECT repository

Evaluated on dual Intel Xeon E5-2660 v2, 20x 2.2GHz, 256GB

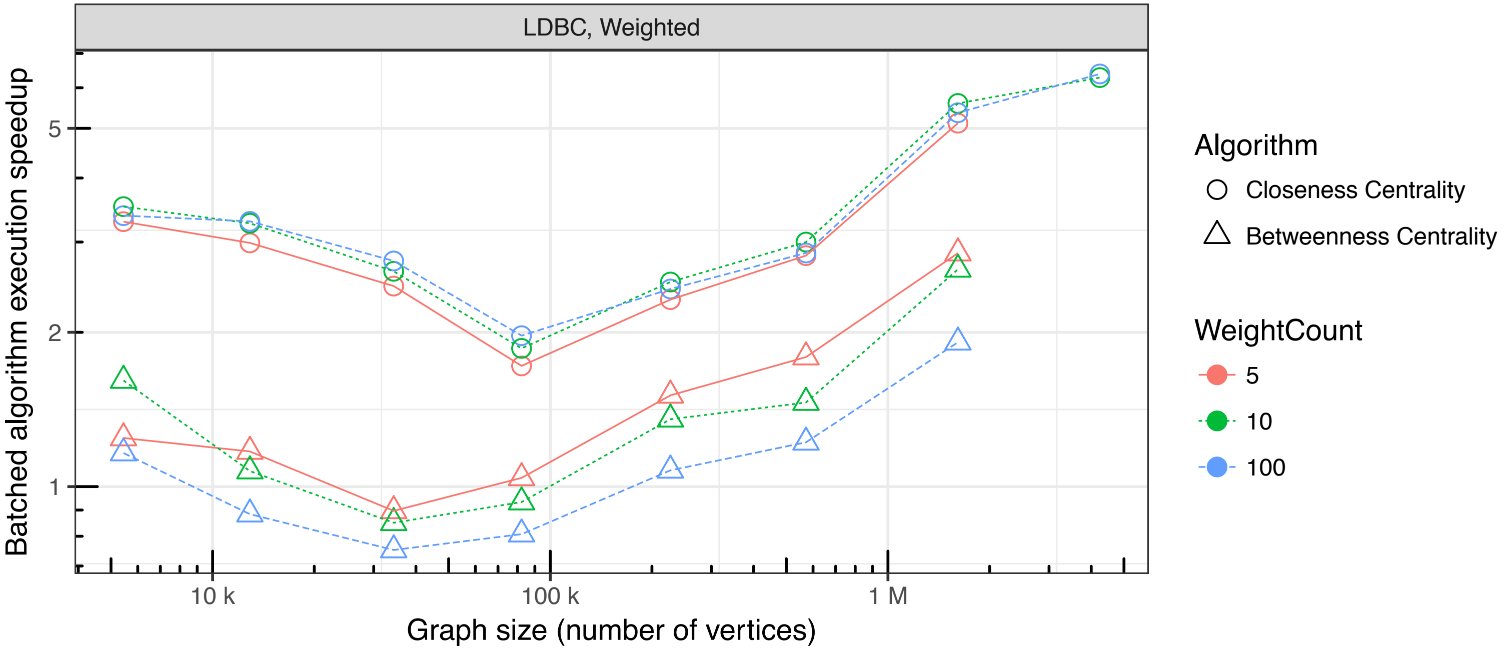
Evaluation: Number of Concurrent Executions



Evaluation: Graph Size Scalability



Evaluation: Number of Edge Weights



Summary

Batched algorithm execution

- Shares common data accesses,
- Avoids/vectorizes computations, and
- Significantly reduces graph algorithm execution times

Improved centrality computation performance

- Unweighted by up to 20x (closeness) and 6x (betweenness)
- Weighted by up to 7x (closeness) and 3x (betweenness)

Details and all algorithms are listed in the paper

Future work:

Apply batched execution to further classes of algorithms