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
\[\forall v \in V: \forall n \in \text{neighbors}(v): \text{next}[n] = \text{visit}[v] \& \sim \text{seen}[n]\]

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

\[
CC_v = \frac{|\text{reachable}(v)|^2}{(|V| - 1) \times \left(\sum_{u \in \text{reachable}(v)} \text{distance}(v, u)\right)}
\]

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

\[ BC_v = \sum_{u,w \in V, u \neq v \neq w} \frac{|\{ P \mid P \in \text{shortest_paths}(u, w) \land v \in P\}|}{|\text{shortest_paths}(u, w)| \ast (|\text{reachable}(v)| \ast (|\text{reachable}(v)| - 1))} \]

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

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

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

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

Batched algorithm execution
• ... improves temporal and spatial locality
• ... facilitates vectorized computation
Batched Weighted Distances

Comparison of common weighted distance algorithms:

- Kronecker, 5 weights
- Kronecker, 10 weights
- Kronecker, 100 weights
- LDBC, 5 weights
- LDBC, 10 weights
- LDBC, 100 weights

Graph size (number of vertices)

Runtime (in milliseconds)

Execution
- Batched
- Non-batched

Algorithm
- Bellman–Ford
- Dijkstra
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

- **Closeness Centrality, Unweighted**
- **Closeness Centrality, Weighted**
- **Betweenness Centrality, Unweighted**
- **Betweenness Centrality, Weighted**

**Dataset**
- LDBC 100
- Kronecker S21
- Citeseer
- DBLP
- Hudong
- Wikipedia

**Evaluation:** Number of Concurrent Executions
Evaluation: Graph Size Scalability

LDBC, Unweighted

Batched algorithm execution speedup

Graph size (number of vertices)

Algorithm
○ Closeness Centrality
△ Betweenness Centrality
➕ vs. Brandes’s BC

WeightCount
• 1
• 10

LDBC, Weighted
Evaluation: Number of Edge Weights

LDBC, Weighted

Graph size (number of vertices)

Batched algorithm execution speedup

Algorithm
- Closeness Centrality
- Betweenness Centrality

WeightCount
- 5
- 10
- 100
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