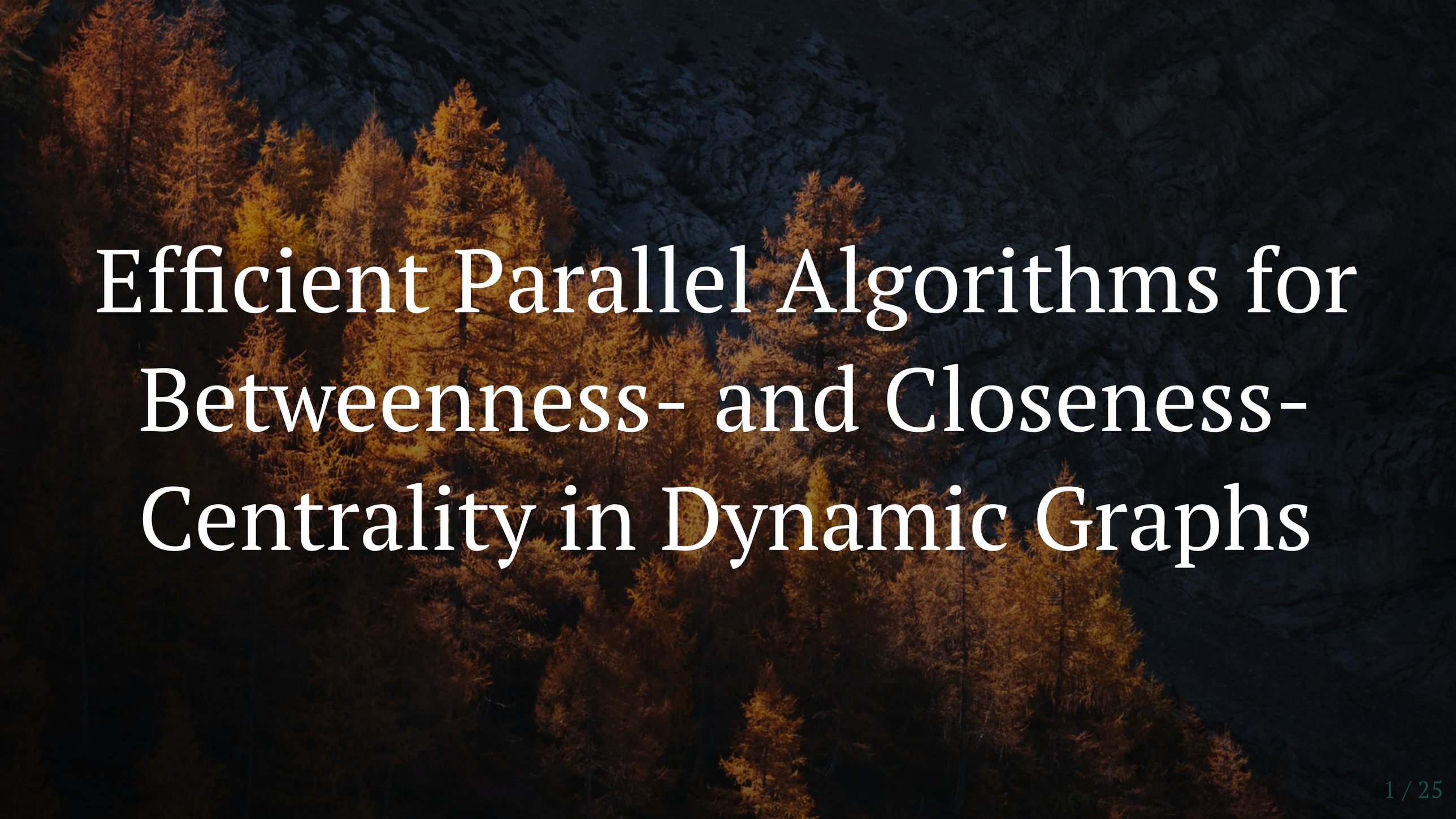


Efficient Parallel Algorithms for Betweenness- and Closeness-Centrality in Dynamic Graphs

- Introduction
- Properties on the graph
- Betweenness-centrality
- Closeness-centrality
- Conclusions

Efficient Parallel Algorithms for Betweenness- and Closeness-Centrality in Dynamic Graphs (ICS '20)

Some keywords

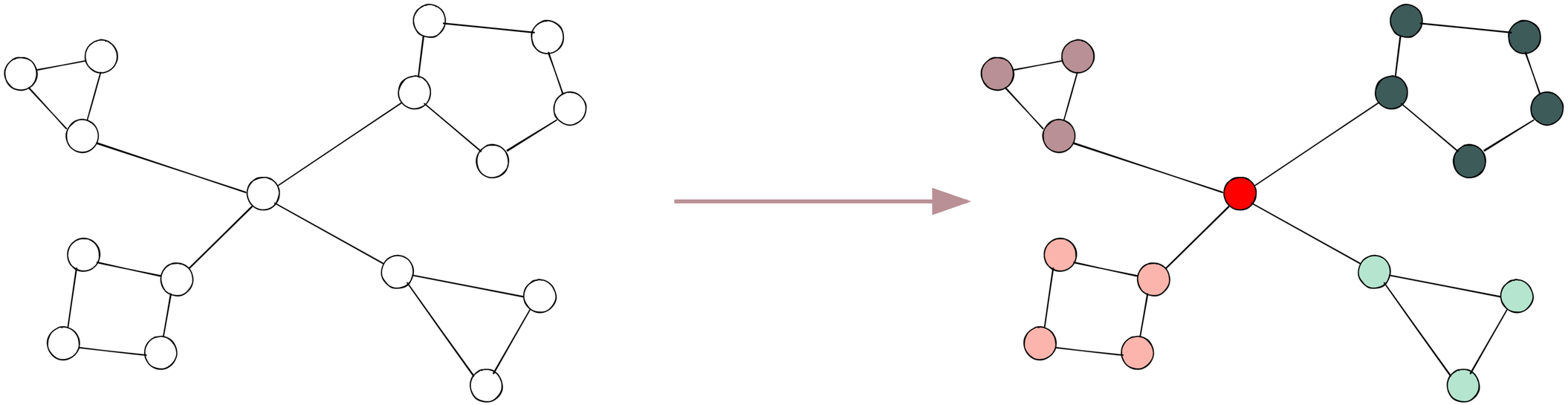
- Dynamic graph (how to calculate something on a dynamic graph)
- Parallel algorithm
- Betweenness- and closeness- centrality

Dynamic graph

- static results + transfer

Parallel algorithm

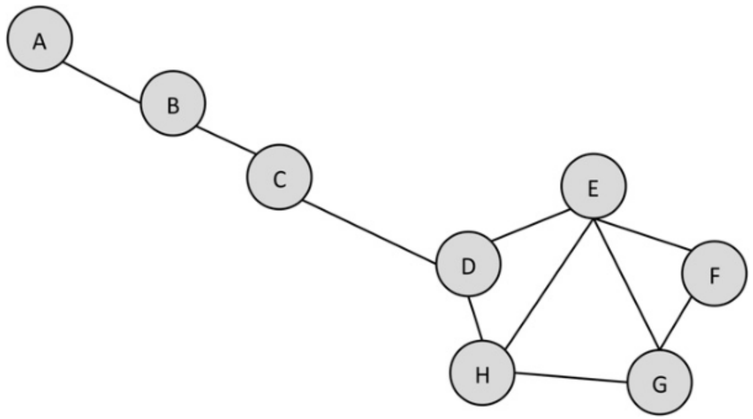
- How to be parallel on a graph?



- BCC: biconnected components (will cover this very soon)

Closeness-centrality

$$cc(v) = \frac{N-1}{\sum_u dis(u,v)}$$

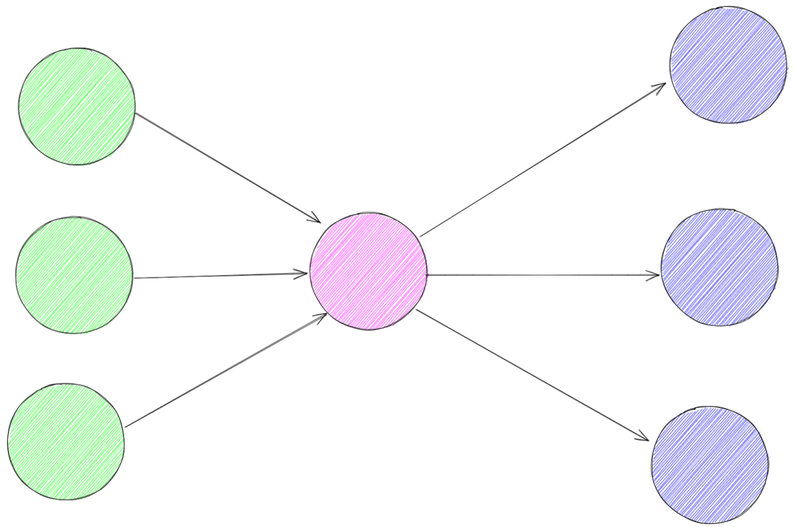


- $dis(u,v)$: shortest path length between u and v
- how close you are to others

Betweenness-centrality

$$bc(v) = \sum_{s \neq v \neq t} \frac{\delta_{st}(v)}{\delta_{st}}$$

- ratio of: shortest paths that pass through v to shortest paths number
- measures the node's influence



- note that both the two centralites relate to *shortest path*

Dependency

Dependency of a vertex s on another vertex v on a graph G is defined as:

$$\delta_s[v] = \sum_{t \in G} \delta_{st}(v)$$

- all the short paths that has to pass through v

TOC

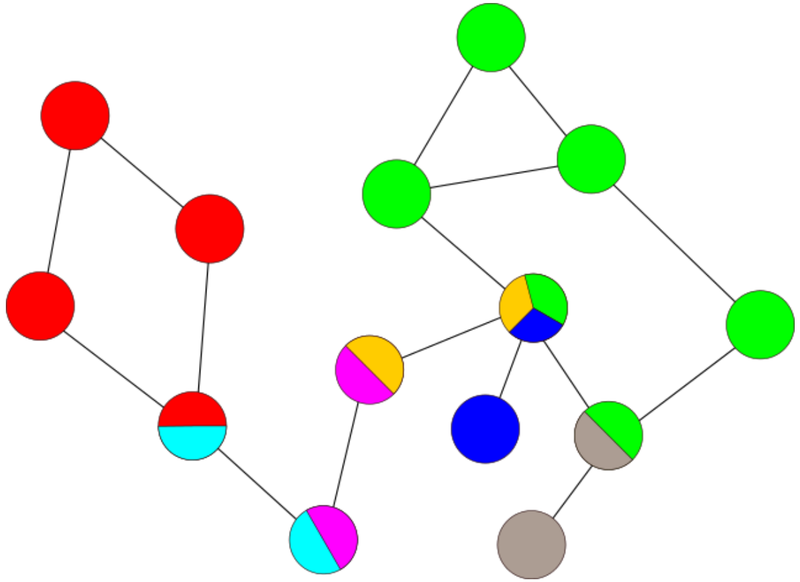
Techniques mentioned

- BCC (Biconnected component decomposition)

1. Batch update (pack a bunch of updates rather than a single one)
2. Redundant chains
3. Redundant nodes

BCC

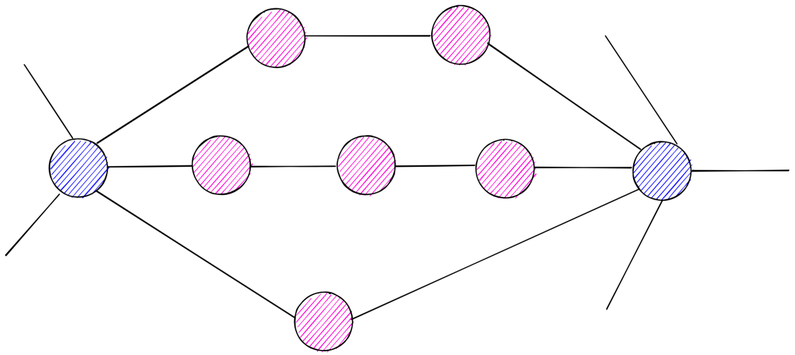
- maximal *biconnected subgrarph*
  - the graph remain connected even if remove any node



- shortest paths between intra-BCC nodes are confined to the BCC

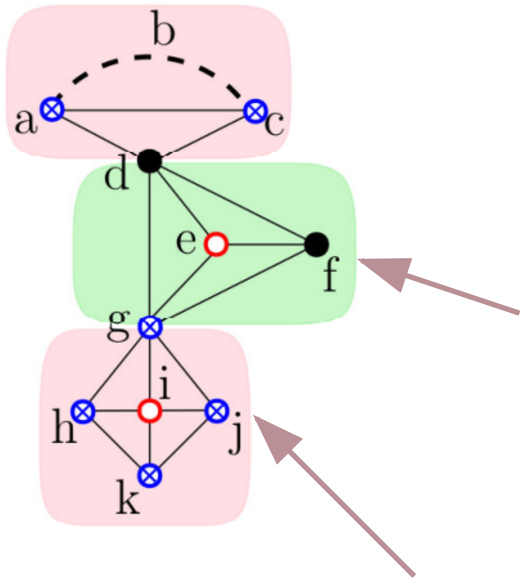
Redundant chains

- the author's definition of chain is a bit different:
  - two end points (degree != 2) and many middle nodes (degree == 2)



- redundant chain: only one common-end-point chain will be used when calculating shortest path
  - we can save computation

Redundant nodes



- R_3: three neighbors of v are connected to each other
- R_4: four neighbors of v are connected to each other
- shortest path does not pass through v
- we can save computation
- the changes are recorded, so no need to scan the whole graph every time

TOC

Betweenness centrality

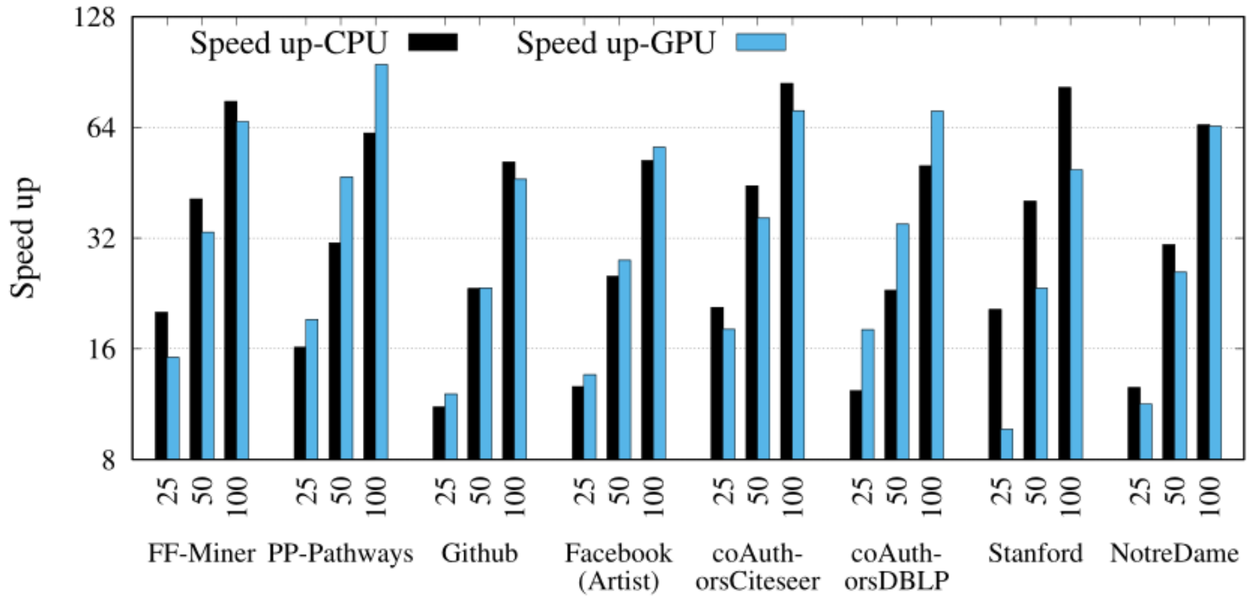
Algorithm 1: BatchBCUpdate (G, bc_G, B)

```
1  L = findAffectedNodes(G, B)
2  bc_G'[v] = bc_G[v] for all v ∈ V
3  G' = G ∪ B
4  for each affected BCC S ∈ G do in parallel
5      Subtract dependencies δ_r[ ] for all r such that r ∈ S and
         r ∈ R_3 ∩ L
6      Add dependencies δ_r[ ] for all r such that r ∈ S' and
         r ∈ R_3 ∩ L
7  G' = G' \ (R_3 ∩ L) // G' is now the reduced graph
8  for each affected BCC S in G do
9      for each affected chain Q in S do in parallel
10         Subtract dependencies δ_c[ ] for c ∈ Q
11     for each node s ∈ S do in parallel
12         AtomicICentral(s, −1)
13 for each affected BCC S' in G' do
14     for each affected chain Q in S' do in parallel
15         Add dependencies δ_c[ ] for c ∈ Q
16     for each node s ∈ S' do in parallel
17         AtomicICentral(s, +1)
```

- $B = \{e_1, e_2, \ldots, e_n\}$, the batch-updated edges

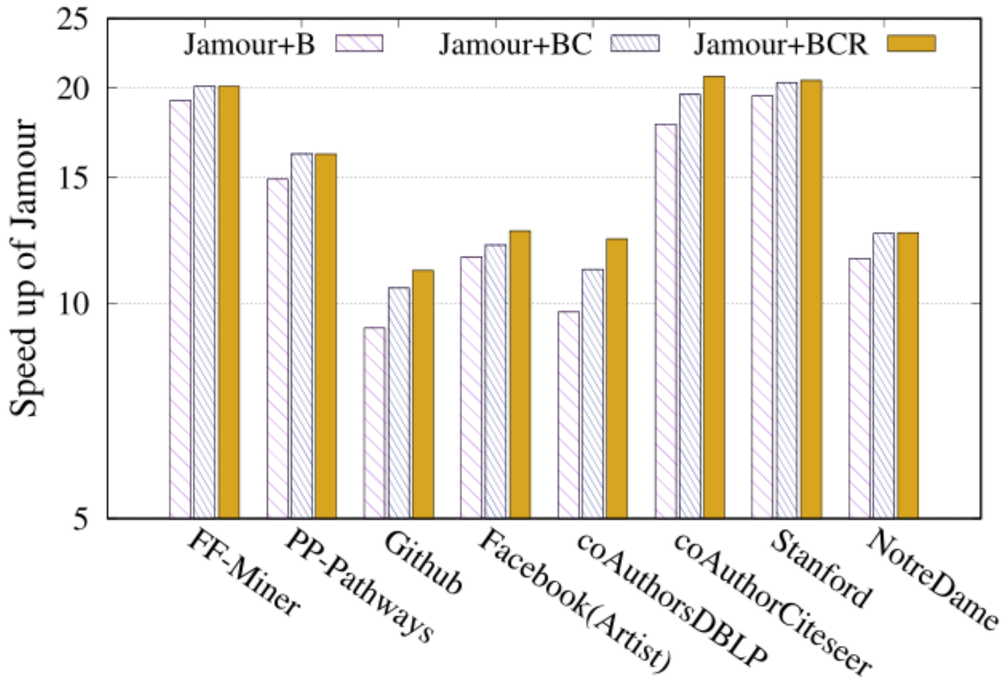
Batch update

Results: speed up



- all datasets are preprocessed, made undirected, unweigted. Self-loops, multiple edges are removed
- the reuslts are very good

Results: BCR



- B: batch update
- C: redundant chains
- R: redundant nodes
- Jamour + BCR = algorithm the author used

TOC

Closeness centrality

Results

TOC

Conclusions

What learned:

- Several centrality measurements [4]
- Ideas on calculating some properties on graph, especially in parallel

Upsides:

- Clear theoretical derivations

Downsides:

- The algorithm now only works on unweighted and undirected graphs (mentioned in future works)
- depends on previous people's works a lot (you need to read at least another 3 papers to understand the author's work)
  - some modifications to improve efficiency

References

- [1] F. Jamour, S. Skiadopoulos, and P. Kalnis, “Parallel Algorithm for Incremental Betweenness Centrality on Large Graphs,” IEEE Trans. Parallel Distrib. Syst., vol. 29, no. 3, pp. 659–672, Mar. 2018, doi: 10.1109/TPDS.2017.2763951.
- [2] U. Brandes, “A faster algorithm for betweenness centrality*,” The Journal of Mathematical Sociology, vol. 25, no. 2, pp. 163–177, Jun. 2001, doi: 10.1080/0022250X.2001.9990249.
- [3] A. E. Sariyuce, E. Saule, K. Kaya, and U. V. Catalyurek, “STREAMER: A distributed framework for incremental closeness centrality computation,” in 2013 IEEE International Conference on Cluster Computing (CLUSTER), Indianapolis, IN, USA, Sep. 2013, pp. 1–8. doi: 10.1109/CLUSTER.2013.6702680.
- [4] “Centrality - Neo4j Graph Data Science,” Neo4j Graph Database Platform. https://neo4j.com/docs/graph-data-science/1.7/algorithms/centrality/ (accessed Nov. 05, 2021).

Code

Use code snippets and get the highlighting directly![1]

```
interface User {
  id: number
  firstName: string
  lastName: string
  role: string
}

function updateUser(id: number, update: User) {
  const user = getUser(id)
  const newUser = {...user, ...update}
  saveUser(id, newUser)
}
```

1. Learn More