Cécile DANIEL, Angelo FURNO, Nour-Eddin EL FAOUZI

***Université de Lyon**, ENTPE, IFSTTAR,*
LICIT UMR T9401 Lyon, France
Email: angelo.furno@ifsttar.fr, nour-eddin.elfaouzi@ifsttar.fr

Eugenio ZIMEO

***University of Sannio**, Department of Engineering*
Benevento, Italy
Email: zimeo@unisannio.it

Data Science Summer School 2019 – Ecole Polytechnique

1. Context

- Large graphs are very common in different fields such as **transportation, telecommunication and social networks.**
- They are used for **analysis, exploration and prediction**: which Twitter account is an influencer? What is the impact of an accident in ring roads?
- To answer these questions and **understand topological properties**, there are many **metrics** to describe graphs. The problem with large graphs is the computation time for these metrics.
- Modelling roads and public transportation networks of modern cities leads **to large graphs**, which is challenging for **real time applications.**

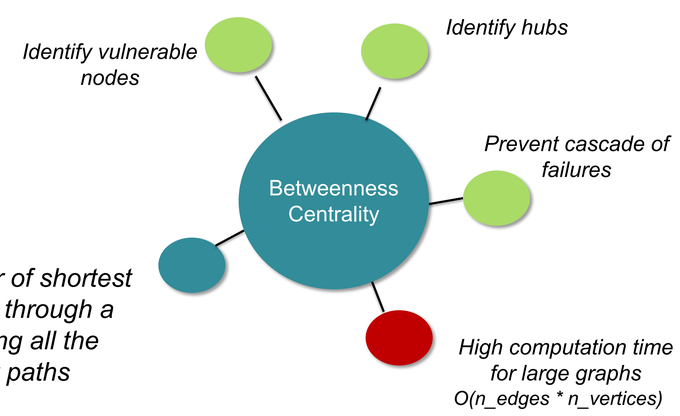
2. Objectives of the PhD Programme

***GLOBAL GOAL*: develop a platform to help the analysis of **large and dynamic transportation networks working in real time**.**

- Using proper representations of the transportation networks: **multilayer representation** (one layer per mode) as it is close to **multi-modal mobility networks**
- Implementing metrics by allowing a fast and efficient **analysis of network vulnerability**

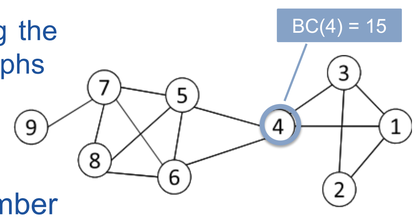
One of the most popular metrics for detection of (topological) vulnerabilities is Betweenness Centrality (BC) **[1, 2].**

- We focus on the state-of-the-art for the computation of BC and the algorithms we are working on.
- To reduce computation time, we use a **clustering-based solution** and **distributed** (map-reduce) implementations **[3, 4, 5].**



3. State of the art: Brandes algorithm

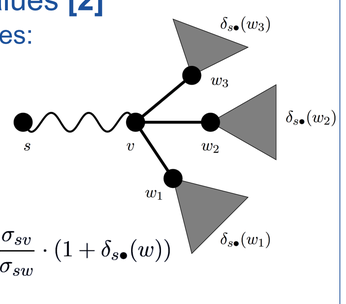
Node Betweenness centrality (BC) **[1]** allows for identifying the most vulnerable nodes on very large-scale road network graphs



- For a given graph ***G(V, E)***:

$BC(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}} = \sum_{s \neq v} \delta_{s\bullet}(v)$ with $\sigma_{st}(v)$ = number of shortest paths between node s and t that cross v and σ_{st}= number of shortest paths between node s and t.

- **Brandes' algorithm**: fastest solution for computing exact BC values **[2]**
  - For each node *s* (called **source node**) the algorithm executes two phases:
    - *Exploration:*
      - *Perform a Breadth-First Search starting from node s (SSSP)*
      - *Count the number of shortest paths from s to any other node*
    - *Back Propagation*
      - *Determine the dependency scores from s to all other nodes ($\delta_{s\bullet}(v)$)*
      - *Sum the dependency scores obtained for each source*



Brandes' recursive relationship to compute dependency scores

$$\delta_{s\bullet}(v) = \sum_{w\,:\,v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w))$$

- **Brandes' algorithm *time complexity*: $O(n\,m)$, $n = |V|$ and $m = |E|$**
  - **Not suited for real-time computation over large network, even if parallelized**
  - **Solution: reducing the number of sources through *pivots* identified by *clustering***
  - **Side effect: approximation.**

Predecessors set: $P_s(v) = \{u \in V : \{u, v\} \in E,\ d_G(s, v) = d_G(s, u) + \omega(u, v)\}$

4. Our solutions: One and Two level Clustering

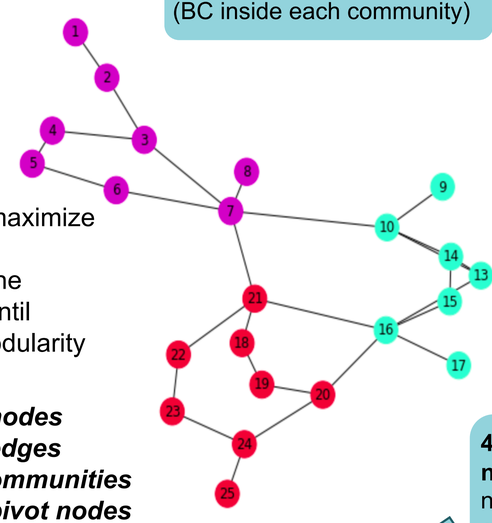
1. Cluster the graph into communities based on **topological properties** (Louvain method).

Louvain Clustering: maximize the modularity.
At each step, change the community of a node until maximization of the modularity

2. Computation of Local BC (BC inside each community)

3. Determine **classes of equivalence*** for nodes inside each community and select a pivot (node with lowest local BC) in each class.

**Class of equivalence: nodes that have the same distance and number of shortest paths from the different border nodes of their community*



25 nodes
29 edges
3 communities
13 pivot nodes

3. BIS **Two Level clustering** using **Kmeans** when number of pivot ~ n. (non scale free graphs)
Solution: create classes of classes.

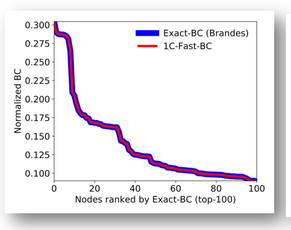
4. Execute **SSSP only from pivot nodes.** Multiply the contributions of the nodes by the cardinality of the class of equivalence

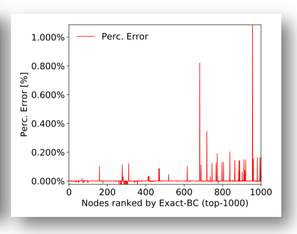
5. Sum local BC (source and destination are in the same community) **and global BC** (source and destination in different communities)

1C-Fast-BC (without step 3. BIS)
2C-Fast-BC (with step 3. BIS)

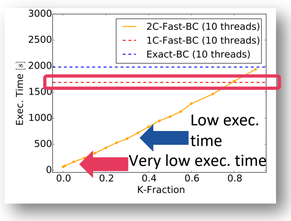
5. 1C-Fast-BC and 2C-Fast-BC Results

Very good accuracy with 1C Fast-BC

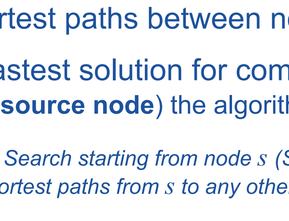


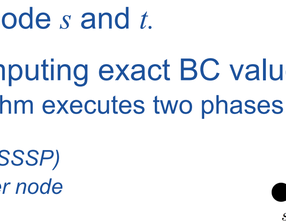


…but **1C-Fast-BC exhibits high execution time due to the presence of 48,960 classes out of 75k nodes (non scale-free graph)**

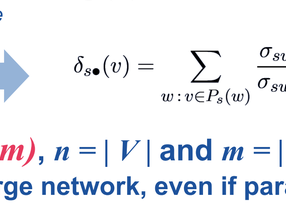


… **2C Fast-BC allows for reducing execution time at the price of a tolerable error** by properly tuning the **Kfraction parameter** according to the domain requirements





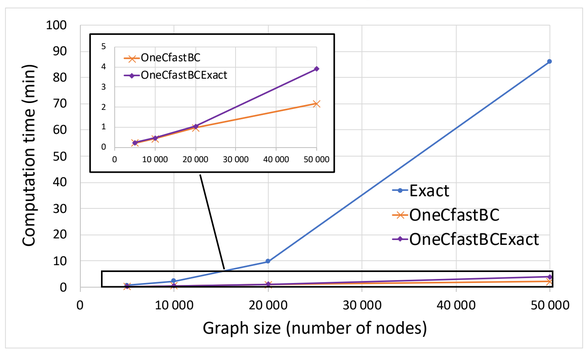
- The algorithm has been implemented in Scala on top of the Spark framework
- Execution with 10 Threads on 10 cores
- Hardware: dual Intel Xeon E5 2640 2.4 GHz multi-core machine, equipped with 128 GB RAM DDR4



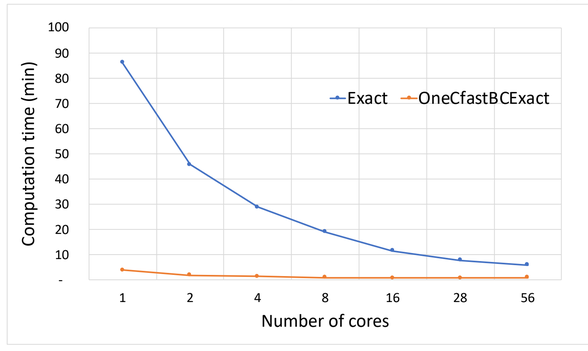
6. 1C-FastBCExact - Preliminary Results

***Two main sources** of error with one-level clustering algorithm:*
→ local BC computation
→ global BC computation

To remove the two sources of error we are implementing a new version of 1C-Fast-BC for undirected scale-free graphs (OneC-FastBC-Exact).
We have preliminary promising results with **no errors for BC,** for undirected graphs (soon published).



BC computation on scale-free graphs with different sizes (number of nodes), sequential execution



BC computation on scale-free graph with n=50k and m=100k with different levels of parallelism (number of cores)

The computation time is very low compared to Brandes algorithm, with different levels of parallelism and different sizes of the graph
The computation time is slightly higher than the previous OneCFastBC, but the results are exact
→ Our new algorithm succeeds in **computing the exact metric in a short time.**

7. What's next?

- Adapting our algorithm to **weighted and directed graphs**
- **Evaluating scalability** and further **optimizing the distributed implementation**
- Using **metric learning** to improve the approximated techniques
- **Exploring** effective metrics for vulnerability of **multi-layer networks**
- Implementing and integrating efficient metrics for **real time vulnerability monitoring** of critical nodes in an on-line framework.

References

[1]. **L. C. Freeman**, *"A set of measures of centrality based on betweenness", Sociometry, 25:35-41,19 1997.*
[2]. **U. Brandes**, *"A Faster Algorithm for Betweenness Centrality", Journal of Mathematical Sociology, 25(163), 2001.*
[3] **Suppa, P. and Zimeo, E.,** *A clustered approach for fast computation of betweenness centrality in social networks. In IEEE International Congress on Big Data (pp. 47-54), 2015.*
[4] **Furno A., El Faouzi N.E., Sharma, R. and Zimeo E.**, *Two-level clustering fast betweenness centrality computation for requirement-driven approximation. In IEEE International Conference on Big Data (Big Data) (pp. 1289-1294), 2017.*
[5] **Furno A., El Faouzi N.E., Sharma R. and Zimeo E.,** *"Fast approximated betweenness centrality of directed and weighted graphs. In International Conference on Complex Networks and their Applications (pp. 52-65). Springer, Cham, 2018.*

DATA-DRIVEN RESILIENT MOBILITY