Reducing pivots of approximated betweenness computation by hierarchically clustering complex networks

Angelo Furno, Nour-Eddin El Faouzi, Rajesh Sharma and Eugenio Zimeo

Abstract Betweenness centrality is a widely adopted metric to analyze the impact of *critical* nodes of graphs in several domains (social, biological, transportation, service, computer networks). Its exact computation is very demanding due to an $O(nm)$ time complexity on a graph with n nodes and m edges. This represents an obstacle to the adoption of betweenness centrality for continuous monitoring of critical nodes in very large networks. In this paper, we analyze the performance of a parametric two-level clustering algorithm for computing approximated values of betweenness with different kinds of graph. The analysis aims at evaluating how the properties of different complex networks impact the reduction of the number of single-source shortest-paths explorations of Brandes' algorithm. The results confirm that one-level clustering strongly reduces the number of pivots (and consequently the computation time) on scale-free graphs, while small-world (and random) graphs need an additional level of clustering to achieve acceptable results.

1 Introduction

Betweenness centrality (BC) is a popular graph metric to characterize node's network centrality as a measure of the extent to which a node has control over information flow between any other pair of nodes in the network.

This metric has been widely adopted to identify opinion leaders in social networks [2], critical intersections in transportation networks [9], critical nodes in biological networks [10] vulnerabilities in computer networks [7] and power grids [8], threats from terrorist networks [5]. In spite of its usefulness, high computation time limits the applicability of BC especially for real-time monitoring of very large infrastructures modeled as graphs.

Angelo Furno, Nour-Eddin El Faouzi
Univ. Lyon, IFSTTAR, ENTPE, Lyon, France. e-mail: name.surname@ifsttar.fr

Rajesh Sharma
University of Tartu, Tartu, Estonia. e-mail: rajesh.sharma@ut.ee

Eugenio Zimeo
University of Sannio, Benevento, Italy. e-mail: zimeo@unisannio.it

Computing BC requires exhaustive path traversals. The fastest algorithm for exact betweenness centrality computation has been proposed by Brandes [3]. For a graph $\boldsymbol{G(V,E)}$, it exhibits a time complexity of $\boldsymbol{O(nm)}$, where $\boldsymbol{n = |V|}$ is the number of nodes and $\boldsymbol{m = |E|}$ is the number of edges.

However, Brandes' algorithm is not suitable for real-time computations needed for monitoring, especially on very large networks. In this paper, we propose an algorithm for computing approximated values of BC that can be guided to tune its performance on the basis of a tolerable error of the computed index. The algorithm, inspired by [4] and [6], exploits some topological characteristics of graphs in order to classify nodes for their selection as pivots. To this end, it calculates the betweenness index through four phases: first, the graph is partitioned in communities by applying a clustering method; then, for each cluster, the nodes are grouped into classes according to their equivalent potential contribution for the computation of the betweenness centrality values of the other nodes of the graph. By using a representative pivot for each class, we can calculate an almost exact value of betweenness for several nodes while keeping a good approximation for the others, with a computing time that strictly depends on the number of classes identified.

Unfortunately, the number of classes is typically small for scale-free[1] graphs [16] but can be large for other kinds of graphs. To address this topology-dependent problem, we consider a *two-level clustering fast betweenness computation* aimed at reducing the number of classes by aggregating them in *super classes*, through an additional clustering inside each community (third phase). In this case, a pivot is elected for each identified super class. Finally (fourth phase), Brandes' algorithm is applied to pivot nodes, by achieving an ideal speedup $\boldsymbol{\frac{n}{k}}$, where $\boldsymbol{k}$ is the number of super classes.

By this approach, we can reduce the number of pivots as needed by the application requirements on computation time, as well as keeping the growing error under a tolerable threshold. In this paper, we analyze the accuracy of the two cluster-based algorithms on different complex networks.

The rest of the paper is organized as follows. In Sec. 2, we present related work. Sec. 3 describes the algorithm proposed for fast BC calculation. In Sec. 5, we first discuss the different kinds of complex networks used in our analysis. Then, we evaluate our approach on the considered datasets. We conclude in Sec. 6 by also discussing future directions.

2 Related Work

In spite of its significant improvement over the naive approach, especially for sparse graphs, the algorithm proposed by Brandes [3] is not adequate for real-time monitoring of very large networks due to high computation time. In this direction, several approaches for fast evaluating exact or approximated solutions have been proposed. They can be classified in three groups.

[1] A scale-free graph is characterized by a power-law degree distribution, i.e., few nodes have a very large degree, while the majority of the nodes has only a few neighbors.

In the first set of approaches, parallel implementations are leveraged for computing betweenness centrality. In [1], a solution based on a fine-grained multi-level parallelism is proposed. The neighbors of a given node are traversed concurrently on a shared data structure with granular locking. The algorithm has been successively improved [12] by removing the need for locking in the dependency accumulation stage of Brandes' algorithm through the adoption of a successor list instead of a predecessor list for each node.

The second research trend aims at achieving low computation time by calculating approximated BC values. These strategies try to penalize some shortest paths, whose computation is the most expensive task in the whole process. For example, in [18], the authors only consider paths up to fixed length $\boldsymbol{k}$. Brandes and Pich [4] also proposed an approximated algorithm for faster BC calculation by selecting only $\boldsymbol{k \ll n}$ pivots as sources for the computation of single-source shortest-paths (SSSP). To this purpose, different strategies are considered, showing that random selection of pivots can achieve accuracy levels comparable to other heuristics. However, this approach overestimates the BC of unimportant nodes that are near a pivot. To overcome this problem a generalization framework for betweenness approximation has been proposed in [6]. The idea is to scale BC values in order to reduce them with reference to nodes close to pivots. In a recent work [14], another approximated BC algorithm has been proposed. However, the approach shows large fluctuations of accuracy over the top-100 nodes on a scale-free graph.

A third class of approaches (stream-based) tries to avoid recomputing the BC values of all the nodes of a graph $\boldsymbol{G' \equiv G + \Delta G}$ when they are known for a previous configuration $\boldsymbol{G}$. Recently, an efficient algorithm for incremental BC computation [11] has been proposed. The algorithm performs very well when BC has to be recalculated as a consequence of adding or removing one node. However, the high speedup achieved for the scenarios above drastically reduces when a graph configuration differs from the previous one in $\boldsymbol{M}$ nodes.

The algorithm analyzed in this paper is an extension of our previous work [16] that selects the pivots by classifying nodes with reference to border nodes of communities, which are considered most relevant for BC evaluation. The solution can be classified as an approximated algorithm based on BC estimation through a partial exploration of graphs, driven by pivot nodes. While the original version identifies the number of pivots on the basis of graph properties, the extended version is able to reduce this number through a further clustering based on K-means.

3 Background: Fast BC computation based on clustering

The algorithm analyzed in this paper reduces the number of pivots by exploiting clustering. As previously proved [16], border nodes of clusters represent the most significant nodes of a graph to compute betweenness centrality, since they are the most traversed ones. Therefore, finding the nodes inside each cluster that exhibit the same (or very similar) properties with reference to border nodes allows for reducing the number of single source shortest paths computations of Brandes' algorithm. In particular, the algorithm is hierarchical: (a) at the first level, it exploits Louvain clustering to find communities and border nodes; (b) at the second level, it adopts K-means clustering to force pivots reduction. Since the first level finds a number of pivots based on graph topological properties, the second one is necessary when the identified number of pivots is not small enough to ensure the desired computation time. However, forcing the algorithm to reduce the number of pivots may increase approximation errors.

We assume the following definitions throughout the paper. For a generic graph $G(V, E)$ a shortest path between any two nodes v_i and v_j is a path with the minimum number of hops among all the paths connecting the two nodes and is denoted as $sp(v_i, v_j)$. Distance $d(v_i, v_j) = len(sp(v_i, v_j))$ is the length of a shortest path between nodes v_i and v_j. We denote $\sigma_{v_i v_j}$ as the number of shortest paths between v_i and v_j, while $\sigma_{v_i v_j}(v_k)$ represents the number of shortest paths from v_i to v_j that cross node v_k.

3.1 Brandes' algorithm

Given a *pair-dependency* of a *source* node s on an another node v for a *destination* t of the graph, defined as:

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}},$$

the betweenness centrality of any node v can be expressed in terms of *dependency score* $\delta_{s\bullet}(v) = \sum_{t \in V, v \neq t} \delta_{st}(v)$, obtained by summing the pair-dependencies of each pair of nodes on v that has s as source node. To compute this score, Brandes' algorithm exploits a recursive relation that is motivated by the following observation: let $W = \{w : v \in P_s(w)\}$ be the set of nodes w such that v is a predecessor of w along a shortest path that starts from node s, and $P_s(w) = \{v \in V : \{v, w\} \in E, d(s, w) = d(s, v) + d(v, w)\}$ the set of direct predecessors of a generic node w in the shortest paths from the source node s to w, for unweighted graphs; then, v is a predecessor also in any other shortest path starting from s and passing through a different $w \in W$ [3]. Consequently, we have:

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

Finally, the betweenness centrality BC of node v is obtained by[2]:

$$BC(v) = \sum_{s \in V, v \neq s} \delta_{s\bullet}(v). \tag{2}$$

Conceptually, Brandes' algorithm runs in two phases. During the first phase, it performs a search on the whole graph to find all the shortest paths

[2] For scaling purpose, BC values are often normalized by dividing them by $(n-1) \cdot (n-2)/2$ for undirected graphs and by $(n-1) \cdot (n-2)$ for directed ones.

starting from every node s, considered as the source of a breadth-first exploration of the whole graph. Then, in the second phase, it performs dependency accumulation by backtracking along the discovered shortest paths. During these two phases, the algorithm maintains four data structures for each node found on the way: a predecessor list $P_s(v)$, the distance $d_s(v)$ from the source, the number of shortest paths from the source $\sigma_{st}(v)$ and the dependency accumulation when backtracking at the end of the search.

3.2 One-level clustering fast BC

In this section, we briefly describe the algorithm based on one-level clustering (*1C-Fast-BC*) with the support of the pseudo-code in Alg. 1. We remark that *1C-Fast-BC* was originally proposed in [16] by some authors of this paper. It exploits some principles also introduced in [15], such as structural properties and partitioning, by using clustering.

Algorithm 1 2C-Fast-BC algorithm
(1C-Fast-BC algorithm is the same without blue code)

```
procedure ClusteredBrandes($G, C, KFrac$)
    map $i \leftarrow 1, |C|$ do
        $bordernodes_i \leftarrow$ findBorderNodes($G, C_i$)
    end map
    map $i \leftarrow 1, |V|$ do
        $BCgraph \leftarrow$ computeLocalBC($i, C, bordernodes$)
    end map
    reduce $BCgraph.localBC_i, Bgraph.localBC_j, i = j$ do
        $BCcluster_i \leftarrow BCgraph.localBC_i + BCgraph.localBC_j$
    end reduce
    map $i \leftarrow 1, |C|$ do
        $BCgraph.superClasses_i \leftarrow$ KMeansClustering($C_i$,
                                      $BCgraph.classes_i, KFrac$)
    end map
    map $i \leftarrow 1, (|BCgraph.superClasses|)$ do
        $P_i \leftarrow$ selectPivotOf($BCgraph.superClasses_i, BCcluster$)
    end map
    map $i \leftarrow 1, (|BCgraph.superClasses|)$ do
        $\delta_i \leftarrow$ computeDependencyScoresFromPivot($P_i$)
        $\delta_i \leftarrow (\delta_i - BCgraph.localBC) \cdot |BCgraph.superClasses_i|$
    end map
    reduce $\delta_{im}, \delta_{jl}, m = l$ do
        $BC_m \leftarrow \delta_{im} + \delta_{jl}$
    end reduce
    for $i \leftarrow 1, |V|$ do
        $BC_i \leftarrow BC_i + BCcluster_i$
    end for
    return $BC$
end procedure
```

Given a graph $G(V, E)$, we split it into a set C of clusters by using the Louvain method[3], a non-parallelizable heuristic that exhibits a very good $O(n \cdot log(n))$ complexity. The main result of Louvain clustering is the iden-

[3] The Louvain method exploits modularity [13] for grouping nodes. Modularity is the ratio between the density of links inside communities to the one of the links among them.

tification of *border nodes* (an array for each cluster - line 3 of Alg. 1). A border node is a node having at least one neighbor in a different cluster.

Then, a parallel execution of Brandes' algorithm is performed inside each cluster (line 6) to compute the *local BC*. This computation generates the partial inner-cluster contribution to the BC of each node and also additional information, such as the shortest paths and the distances from a node of a cluster towards each border node of the same cluster.

The information above is used to identify the nodes inside each cluster that equally contribute to the dependency score of each node of the graph. To this end, we introduce a class of equivalence ($\boldsymbol{BCgraph.classes_i}$)[4] defined according to the following rule: *two nodes $\boldsymbol{v}$ and $\boldsymbol{w}$ belong to the same class if and only if they have the same normalized distance from each border node of that cluster and the same amount of shortest paths towards the border nodes.* The *normalized distance* of node $\boldsymbol{j}$ is the length of the shortest path for reaching a border node minus the minimum distance to reach any of the border nodes. Therefore, the smallest normalized distance is always zero.

Taking into account that nodes belonging to the same class produce the same dependency score on each node of the graph, one representative node should be identified as a source node for applying Brandes' algorithm. This node is called class $\boldsymbol{pivot}$[5]. Since a pivot does not contribute to its BC, it is selected by considering the lowest BC value among the class nodes that are not border nodes (line 16).

The partial dependency score calculated for the pivot (line 19) is then multiplied by the cardinality of the pivot class (line 20). That avoids to compute the dependency score of the nodes of the same class, thus ensuring fast calculation of BC if $\boldsymbol{k} \ll \boldsymbol{n}$, where $\boldsymbol{k} = |\boldsymbol{P}|$ is the number of pivots belonging to set $\boldsymbol{P}$ of selected pivots and $\boldsymbol{n}$ is the total number of nodes.

The final value of BC is obtained for each node (line 26) by summing up all partial contributions (produced by the reduce operation) with local BC values (to compensate the partial local contribution subtracted at line 20).

4 Beyond 1C-Fast-BC: Two-level clustering fast BC

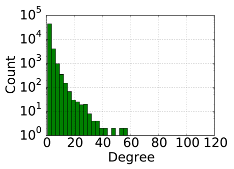
The algorithm above exhibits very good performance [16] when the overall number of classes in the graph is $\ll \boldsymbol{n}$. However, this is true only for some kinds of complex networks (such as scale-free social networks studied in our previous work [16]). When networks are more regular or exhibit degree gaussian distributions, the 1C-Fast-BC approach fails since the number of classes inside each cluster is close to the number of nodes in it. However, we observed that by relaxing the constraints on normalized distance we can conveniently exploit a weaker equivalence class, where nodes have not the same normalized distance from border nodes but "almost" the same

[4] By using only one-level clustering, classes and super classes are equivalent.

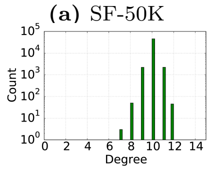
[5] The partial contribution on border nodes of the same cluster of the pivot is the same only for nodes $\boldsymbol{t} \notin \boldsymbol{C_s}$ (where $\boldsymbol{C_s}$ is the cluster of pivot $\boldsymbol{s}$).

Network Type	**Network Name**	**Diameter**	**APL**	**CC**
Scale-Free	SF-50K	20	4.526	0
Small World	SW-50K-P.0	9999	3333.799	0.6666
	SW-50K-P.0001	2728	798.898	0.6662
	SW-50K-P.001	1004	211.233	0.6623
	SW-50K-P.005	593	77.994	0.6467
	SW-50K-P.01	440	49.481	0.6263
	SW-50K-P.015	307	37.070	0.6076
	SW-50K-P.05	204	19.103	0.4871
	SW-50K-P.1	159	13.537	0.3459
	SW-50K-P1	25	7.097	0.0002

Table 1: Complex networks dataset



(a) SF-50K



(b) SW-50K-P.005

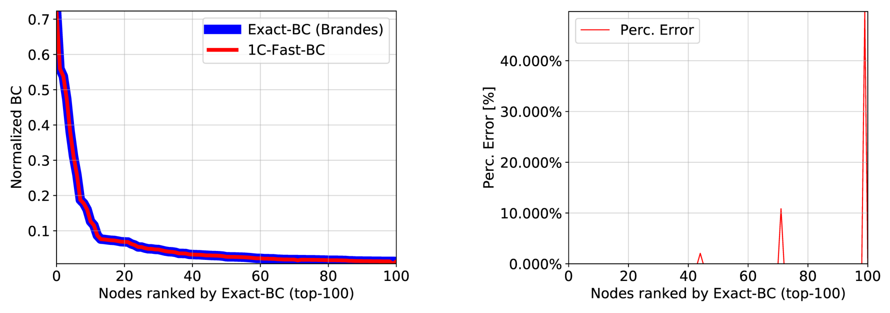
one. We call *super class* this kind of class (see Alg. 1 - blue text) and identify them by using an additional clustering (based on the euclidean distance among the vectors representing the normalized distances) inside each initial Louvain-derived cluster (see Alg. 1 - line 12). As a consequence, a super class $\boldsymbol{BCgraph.superClasses_i}$ is a group of classes belonging to the same cluster: nodes are considered equivalent even when they belong to different classes but are very close (i.e., Euclidean distance close to zero).

To identify super classes, we exploit the K-means algorithm, using a different $\boldsymbol{K}$ for each initial Louvain cluster. $\boldsymbol{K}$ is defined as a fraction of the initial number of classes belonging to each Louvain cluster. For example, by considering a fraction equals to 0.4, the algorithm adopts a 0.4 fraction of the number of classes in each Louvain cluster. By this approach (2C-Fast-BC), we are able to drive the behavior of the algorithm towards the desired computation time.

5 Algorithm analysis with complex networks

We have tested our algorithm with several real-world networks, but, in this paper, we aim at characterizing both 1C-Fast-BC and 2C-Fast-BC in a more general setting, by considering different classes of synthetic graphs. This allows for selecting the best configuration of our algorithm depending on the nature of the real networks. In particular, we consider three kinds of widespread graphs: (a) *Scale-Free* (SF); (b) *Small World* (SW); (c) *Random*, where the last one is considered as a particular case of small world graphs. To achieve the objective above, we created two different types of synthetic network datasets using the R *igraph* package, namely: 1) Scale-Free (with 50,000 nodes and 49,999 edges); 2) Small World (with 50,000 nodes and 250,000 edges).

For Small World graphs, we considered several configurations by varying the rewiring probability $\boldsymbol{p}$ in the range [0, 1]. This probability adds randomness to a regular graph; when $\boldsymbol{p}$ increases the clustering coefficient and the average path lengths decrease. With $\boldsymbol{p = 0}$ we have regular graphs whereas with $\boldsymbol{p = 1}$ random graphs are generated [17].



(a) Normalized BC with 1C-Fast-BC and Exact-BC for the top-100 nodes

(b) Percentage error of 1C-Fast-BC over the top-100 nodes

Fig. 1: Performance of 1C-Fast-BC on the Scale-Free graph

Table 1 reports some other features of the generated graphs, i.e., Diameter, Clustering Coefficient (CC) and Average Path Length (APL), together with the degree distribution for some of the networks.

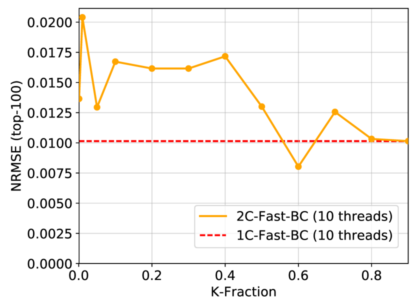
5.1 Performance on Scale-Free graphs

Firstly, we present the results of our performance evaluation of the 1C-Fast-BC algorithm on the SF graph in Tab. 1. For the evaluation, we consider three main indexes: 1) accuracy of the approximated BC values, measured by considering the *percentage error* on each node BC, the global metric of the *mean-Normalized Root-Mean-Square Error* (NRMSE)[6] and the Euclidean distance of the normalized BC vectors, as in [4,6]; 2) accuracy of nodes' ranking, measured via the inversion distance (i.e., percentage of possible inversions), as in [4,6]; 3) execution time.

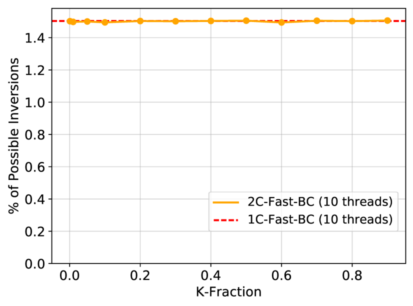
As shown in Fig. 1, the 1C-Fast-BC algorithm provides a very good approximation of the exact BC associated to the top-100 nodes. This can be appreciated as an almost perfect overlapping of the approximated and exact values in Fig. 1a, as well as the 0 percentage error associated, on average, to the nodes in Fig. 1b[7]. This behavior is also confirmed by low values of NRMSE on the top-100 and top-5% (0.01 and 0.19) nodes, which rises to 0.4 for the top-30% and 0.7 for the whole set of nodes. The Euclidean distance of the normalized BC vectors is equal to 0.04. We remark here that computation took only 29.77 seconds, by using only 277 pivots (distributed over 250 Louvain clusters) out of the 50,000 node-set.

[6] The NRMSE is defined as: $\frac{1}{\bar{\sigma}}\sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$, with $\bar{\sigma}$ denoting the mean of the exact BC values and e_i representing the difference between exact and approximated values of BC for node i. NRMSE is generally preferred to the percentage error when 0-values are present among the expected ones. Values close to 0 are representative of good approximation, while values close to 1 or even larger tend to indicate a bad approximation.

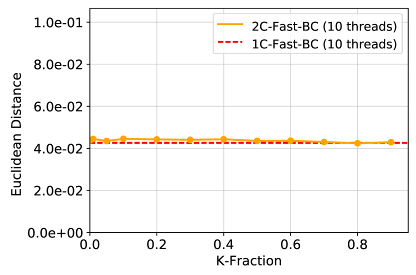
[7] Higher percentage errors are related to nodes that are far from the border ones. Typically these nodes have low values of BC but in a few cases they are highly crossed by intra-cluster paths and consequently exhibit high BC values.



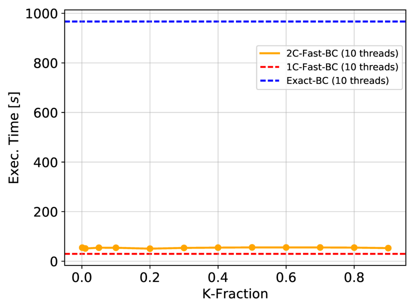
(a) NRMSE of 1C-Fast-BC, 2C-Fast-BC and Exact over the top-100 nodes



(b) Percentage of inversions for 1C-Fast-BC, 2C-Fast-BC and Exact



(c) Euclidean distance of approximated BC vectors from exact ones

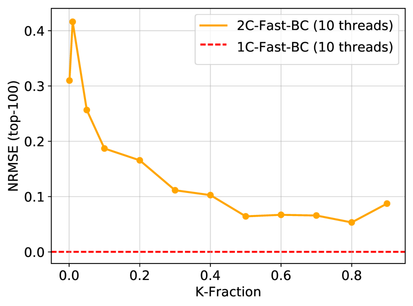


(d) Execution time for 1C-Fast-BC, 2C-Fast-BC and Exact

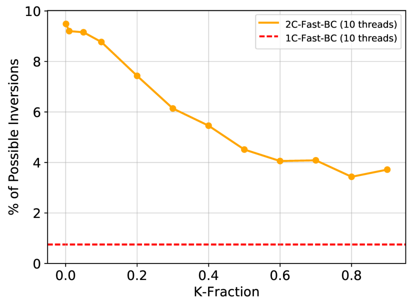
Fig. 2: Performance of 2C-Fast-BC, 1C-Fast-BC and Exact on the Scale-Free graph

Secondly, we evaluated the 2C-Fast-BC algorithm on the same SF network (Fig. 2). As expected, the 2C-Fast-BC reduces the quality of the approximation. Also, as clearly shown in Figs. 2a, 2b and 2c, no relevant improvement is achieved by increasing the value of the K-Fraction parameter. This can be easily explained by considering the #pivots to #clusters ratio (very close to 1) in the 1C-Fast-BC, which leaves no room for further reduction of the number of pivots. Furthermore, the overhead due to the K-means clustering causes the 2C-Fast-BC algorithm to be slightly slower than the 1C-Fast-BC one, as in Fig. 2d (i.e., an average of 55 seconds against the 29.77 seconds of 1C-Fast-BC and the 966.93 seconds of the Exact).

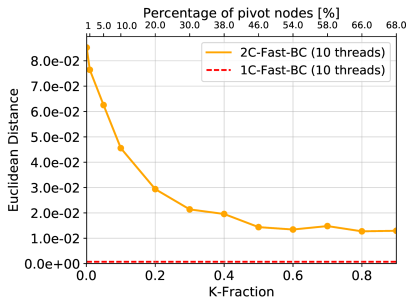
As previously verified for social networks [16], we confirm here that the 1C-Fast-BC algorithm represents a very good technique for a fast and accurate computation of approximated BC values of Scale-Free graphs, especially for the most critical nodes (i.e., those with the highest BC). Therefore, with this kind of graph, the performance of 1C-Fast-BC represents a sort of upper bound that 2C-Fast-BC does not overcome (i.e., computation times with 2C-Fast-BC is higher for any fraction k).



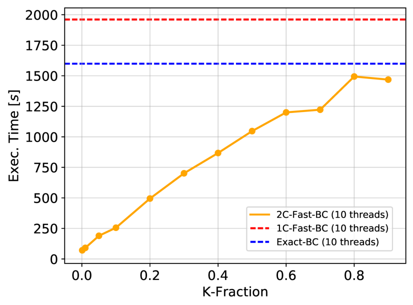
(a) NRMSE of 1C-Fast-BC, 2C-Fast-BC over the top-100 nodes



(b) Percentage of inversions for 1C-Fast-BC, 2C-Fast-BC and Exact



(c) Euclidean distance of approximated BC vectors from exact ones



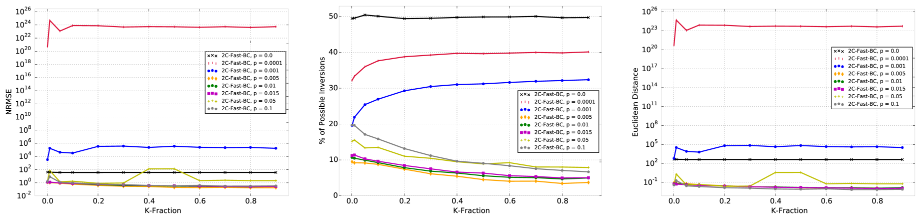
(d) Execution time for 1C-Fast-BC, 2C-Fast-BC and Exact

Fig. 3: Performance of 1C-Fast-BC and 2C-Fast-BC on the SW graph with p=0.005

5.2 Performance on Small-World graphs

To analyze the performance of the 1C-Fast-BC and 2C-Fast-BC on Small-World networks, we considered several configurations (Tab. 1), corresponding to different values of the p parameter. Fig. 3 details the performance of both 1C-Fast-BC and 2C-Fast-BC on the SW-50K-P.005 network from Tab. 1, and is the one with the best performance in terms of inversion distance and NRMSE when using the 2C-Fast-BC algorithm (see Figs. 4a and 4b).

Interestingly, the number of pivots used by the 1C-Fast-BC algorithm on the SW-50K-P.005 network is equal to the total number of nodes (i.e., 50,000 nodes are used as pivots over 138 Louvain cluster). In other words, when using only one-level of clustering, all nodes are considered pivots as in Brandes' algorithm in order to produce a high-quality approximation of BC. Obviously, as shown in Fig. 3d, this translates to a significant execution time (1961 seconds), even higher than the one required by the Exact algorithm (1598 seconds), due to the overhead of performing Louvain clustering. By reducing the number of pivots, the 2C-Fast-BC can drastically reduce the execution time up to a 10 factor, by also keeping good levels of accuracy (see



(a) NRMSE of 1C-Fast-BC, 2C-Fast-BC (top-100 nodes) **(b)** Percentage of inversion of 2C-Fast-BC **(c)** Euclidean distance of approx. and exact BC vectors

Fig. 4: Performance of 2C-Fast-BC on Small-World graphs with different values of p

NRMSE for the top-100 nodes, the inversion and the Euclidean distances in Figs. 4a, 4b and 4c, respectively).

The results of the previous analysis are confirmed on the whole set of Small-World networks reported in Tab. 1. Fig. 4 clearly shows that our algorithm suffers with random and very regular networks ($\boldsymbol{p}$ close to 1 and 0, respectively) while in other small world configurations the accuracy depends on the clustering coefficient. When the latter is high (close to the maximum one) and the average path length is low, the 2C-Fast-BC algorithm exhibits the best accuracy (this is obtained with $\boldsymbol{p = 0.005}$ in our case), while computation time can be reduced by choosing a proper value of K.

6 Conclusion

We proposed a new algorithm for fast computation of approximated betweenness centrality. The algorithm reduces the computation time by identifying $\boldsymbol{k}$ pivots for single-source shortest-paths explorations, thus achieving an ideal speedup of $\boldsymbol{\frac{n}{k}}$ with respect to Brandes' algorithm. Differently from other similar approaches based on pivot selection, the proposed solution exploits clustering to identify topological properties of graphs.

Our analysis shows that selecting pivots by means of clustering significantly reduces the time for computing the approximated values of BC on scale-free graphs, which exhibit an ideal topology for the proposed algorithm. Unfortunately, the number of pivots is too high with small world and random graphs. In these cases, 2C-Fast-BC forces an artificial reduction that further decreases the number of pivots and computing time at a cost of a lower accuracy. However, by taking the errors under control (in terms of inversions, euclidean distance, NRMSE, etc. depending on the application requirements), our hierarchical two-level clustering algorithm represents a valid alternative for approximated BC computation.

To quantify the advantages of our solution compared to [4, 6], we have planned a more exhaustive analysis on additional kinds of graphs and a direct comparison with their approaches. We are also working on an improvement of our 1C-Fast-BC strategy to reduce the error on nodes with lower BC values.

7 Acknowledgements

This work has been partially supported by the GAUSS national research project (MIUR, PRIN 2015, Contract 2015KWREMX).

References

1. Bader, D.A., Madduri, K.: Parallel algorithms for evaluating centrality indices in real-world networks. In: Parallel Processing, 2006. ICPP 2006. International Conference on, pp. 539–550. IEEE (2006)
2. Borgatti, S.P., Mehra, A., Brass, D.J., Labianca, G.: Network analysis in the social sciences. Science **323**(5916), 892–895 (2009)
3. Brandes, U.: A faster algorithm for betweenness centrality. Journal of Mathematical Sociology **25**(163) (2001)
4. Brandes, U., Pich, C.: Centrality estimation in large networks. International Journal of Bifurcation and Chaos **17**(07), 2303–2318 (2007)
5. Carpenter, T., Karakostas, G., Shallcross, D.: Practical issues and algorithms for analyzing terrorist networks. Proceedings of the Western Simulation MultiConference (2002)
6. Geisberger, R., Sanders, P., Schultes, D.: Better approximation of betweenness centrality. In: ALENEX, pp. 90–100. SIAM (2008)
7. Holme, P., Kim, B.J., Yoon, C.N., Han, S.K.: Attack vulnerability of complex networks. Physical Review E **65**(5), 056,109 (2002)
8. Jin, S., Huang, Z., Chen, Y., Chavarra-Miranda, D., Feo, J., Wong, P.C.: A novel application of parallel betweenness centrality to power grid contingency analysis. In: 2010 IEEE International Symposium on Parallel Distributed Processing, pp. 1–7 (2010)
9. King, D., Shalaby, A.: Performance metrics and analysis of transit network resilience in Toronto. Transportation Research Record (2016)
10. Koschützki, D., Schreiber, F.: Centrality analysis methods for biological networks and their application to gene regulatory networks. Gene regulation and systems biology **2**, 193–201 (2008)
11. Kourtellis, N., Morales, G.D.F., Bonchi, F.: Scalable online betweenness centrality in evolving graphs. 2016 IEEE 32nd International Conference on Data Engineering (ICDE) **00**, 1580–1581 (2016). DOI doi.ieeecomputersociety.org/10.1109/ICDE.2016.7498421
12. Madduri, K., Ediger, D., Jiang, K., Bader, D.A., Chavarria-Miranda, D.: A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets. In: Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on, pp. 1–8. IEEE (2009)
13. Newman, M.E.: Analysis of weighted networks. Physical Review E **70**(5), 056,131 (2004)
14. Ohara, K., Saito, K., Kimura, M., Motoda, H.: Accelerating Computation of Distance Based Centrality Measures for Spatial Networks
15. Puzis, R., Zilberman, P., Elovici, Y., Dolev, S., Brandes, U.: Heuristics for speeding up betweenness centrality computation pp. 302–311 (2012)
16. Suppa, P., Zimeo, E.: A clustered approach for fast computation of betweenness centrality in social networks. In: 2015 IEEE International Congress on Big Data, pp. 47–54 (2015). DOI 10.1109/BigDataCongress.2015.17
17. Watts, D.J., Strogatz, S.H.: Collective dynamics of'small-world'networks. Nature **393**(6684), 409–10 (1998)
18. White, S., Smyth, P.: Algorithms for estimating relative importance in networks. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 266–275. ACM (2003)