Two-level Clustering Fast Betweenness Centrality Computation for Requirement-driven Approximation

Angelo Furno, N. E. El Faouzi
Univ Lyon, ENTPE, IFSTTAR, LICIT UMR_T9401
Lyon, France
Email: {angelo.furno, nour-eddin.elfaouzi}@ifsttar.fr

Rajesh Sharma
University of Tartu
Tartu, Estonia
Email: rajesh.sharma@ut.ee

Eugenio Zimeo
University of Sannio
Benevento, Italy
Email: zimeo@unisannio.it

***Abstract*—Betweenness centrality is a widely adopted metric to analyze the impact of *critical* nodes of graphs in several domains (social, biological, transportation, computer networks). The computation of this centrality index is very demanding since its exact calculation exhibits $O(nm)$ time complexity for unweighted graphs (where n is the number of nodes and m is the number of edges). This relative high complexity becomes an obstacle to the adoption of betweenness centrality for continuous monitoring of critical nodes in very large networks. Several solutions have been proposed to reduce computation time, mainly via parallelism, approximation or incremental recalculation. In this paper, we propose a flexible algorithm for computing approximated values of betweenness by tuning its performance on the basis of a tolerable error of the computed index. The algorithm aims at reducing the number of single-source shortest-paths explorations via a pivot-based technique that exploits topological properties of graphs.**

The new algorithm is evaluated for studying vulnerabilities of a real-world, very-large road network. The evaluation shows that the approximation error does not significantly affect the most critical nodes, thus making the algorithm well-suited for on-line operational monitoring of road networks vulnerability. Finally a performance index is defined in order to find an optimal parametrization (in terms of execution time and error) of the algorithm according to application domain requirements.

***Keywords*-Betweenness Centrality; Big-data; Transportation Networks; Monitoring.**

I. Introduction

Betweenness centrality (BC) is a very popular graph metric to characterize nodes that are most traversed by shortest paths (flows) connecting couples of other nodes of the same graph. In other words, it is a measure of the extent to which a node has control over information flowing between other ones. This metric has been widely adopted to identify opinion leaders in social networks [1], critical intersections in transportation networks [2], biological networks [3] vulnerabilities in computer networks [4] and in power grids [5], terrorist networks [6]. In spite of its usefulness, high computation time limits the applicability of betweenness centrality especially for real-time monitoring of very large infrastructures modeled as graphs.

A naive algorithm based on its definition exhibits $O(n^3)$ complexity since it requires the computation of all the shortest paths crossing each node of a graph composed of n nodes. The fastest algorithm to compute the exact value of betweenness centrality has been proposed by Brandes [7]. For a graph $G(V, E)$, it exhibits $O(n + m)$ space and $O(nm)$ time complexities for unweighted graphs and $O(nm + n^2 log(n))$ for weighted ones, where $n = |V|$ is the number of nodes and $m = |E|$ the number of edges. To achieve this performance, Brandes adopts a single-source shortest-paths (SSSP) algorithm that explores a graph by using a breadth-first search. Each exploration is computed with a complexity $O(m)$, which is good for sparse graphs where $m \ll n^2$.

However, Brandes' algorithm is not suitable for real-time computations needed for monitoring very large networks. In this paper, we propose a flexible algorithm for computing approximated values of BC by tuning the performance on the basis of a tolerable error of the computed index. The algorithm, inspired by [8] and [9], 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 in 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 [10] but could be high for other kinds of graphs. To address this topology-dependent problem, in this paper we focus on a *two-level clustering fast betweenness computation*, which is able to reduce 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

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

(fourth phase), Brandes' algorithm is applied to only pivot nodes, by achieving an ideal speedup $\frac{n}{k}$, where k is the number of super classes.

By using this approach, we are able to reduce the number of pivots as needed by the application requirements on computation time, taking into account that the growing error should remain under a tolerable threshold.

To evaluate our algorithm, we propose a case study related to the continuous detection of vulnerabilities in a real-world, very large road network. In transportation literature, vulnerability detection and resilience quantification are considered crucial aspects to improve transportation networks robustness [2], [11]–[13]. To identify vulnerable nodes we use BC, already used in transportation to identify topological criticalities [2], [13], and traditionally preferred to other centrality metrics such as degree and closeness centrality [12]. In particular, we propose to use BC for performing *real-time M-contingency analysis* of traffic networks, focusing on the road network of Lyon, France. M-contingency analysis is a simulation-based technique that considers different qualitative assumptions over M likely events or phenomena in order to imagine different scenarios and come up with optimal responses under the considered circumstances.

Even though, in this paper, we evaluate our algorithm over transportation networks, many other kinds of networks could be analyzed. Moreover, we mainly focus on topological bottlenecks while more dynamic aspects could be also explored. For example, in our application domain, the availability of real-time information, provided by a growing number of sensors and small devices distributed across geographic areas, joined with cloud computing and big data techniques and technologies, allows for a more dynamic and quantitative analysis of robustness against possible unpredictable events.

The rest of the paper is organized as follows. In Sec. II, we present related work. Sec. III describes the algorithm proposed for fast BC calculation. In Sec. IV, we discuss the large-scale dataset used in our analysis as well as the model and the metrics used to characterize road-network vulnerability. In Sec. V, we evaluate our approach on the considered dataset. We conclude in Sec. VI by also discussing future directions.

II. Related Work

In spite of its significant improvement over the naive approach, especially for sparse graphs, the algorithm proposed by Brandes [7] is not sufficient for real-time monitoring of very large networks. In this direction, several approaches, aiming at evaluating exact or approximated solutions, have been developed to further reduce the computation time. The proposed solutions can be classified according to three main approaches: (a) exploiting and increasing parallelism, (b) estimating BC values through a partial exploration of graphs, (c) calculating BC values of single nodes in dynamically changed graphs. Of course, mixed approaches are possible: for instance, an approximated algorithm could be parallelized or applied to dynamic graphs.

In [14], the first parallel implementation for computing betweenness centrality is presented. It is based on a fine-grained multi-level parallelism, in which the neighbors of a given node are traversed concurrently on a shared data structure with granular locking. The algorithm has been successively improved [15] by removing the need for locking in the dependency accumulation stage of Brandes' algorithm through the adoption of a successors list instead of a predecessors 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 [16], the authors only consider paths up to fixed length k. Brandes and Pich [8] also proposed an approximated algorithm for faster BC calculation by choosing only $k \ll n$ pivots as sources for the SSSP algorithm through different strategies, showing that random selection of pivots can achieve an approximation level 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 [9]. The idea is to scale BC values in order to reduce them with reference to nodes close to pivots. In a recent work [17], another approximated BC algorithm has been proposed. However, the approach shows a 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 $G' \equiv G + \Delta G$ when they are known for a previous configuration G. Recently, an efficient algorithm for incremental BC computation [18] 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 M nodes (M-contingency analysis). M-contingency analysis considers M different perturbations of the network to simulate possible critical scenarios as consequence of large-impact events or phenomena.

In transportation field, BC has been studied for traffic flow prediction (e.g., [11], [19]–[21]) or vulnerability quantification (e.g., [2], [12], [13]). In this context, some authors have highlighted limitations of the BC metric in representing traffic dynamics [19], [21]–[23]. However, such shortcomings can be overcome by augmenting the graph representation of the network by taking into account also spatio-temporal aspects (e.g., congestion, accidents, road capacity changes, etc.) and geometric properties of the road network [11], mapping them on a weighted dynamic graph.

This additional graphs information contributes to improve the effectiveness of the analysis but does not impact performance for searching relevant nodes in very large networks, which is the main objective of this work. Moreover, weighted graphs can be transformed in unweighted ones by using techniques like *virtual nodes* [24].

The proposed algorithm is based on and extends a previous work [10] that introduces the idea of exploiting clustering to identify border nodes, 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. Differently from its previous version, the extension proposed in this paper is able to tune performance for real-time monitoring applications, taking under control the approximation error.

III. Clustered BC computation

To improve the efficiency of BC computation and support quasi real-time M-contingency analysis, we propose an approximated, cluster-based approach aimed at finding a useful trade-off between computation time and accuracy. The term *useful* depends on the specific application domain, as knowing the exact values of BC is often less important than discovering the M highest BC-ranked nodes[2].

The proposed algorithm is based on Brandes' one but also exploits an important property of *betweenness*: *an edge with a high betweenness is highly traversed by shortest paths; consequently, it can be considered as a sort of backbone between two graph areas that identify possible clusters*. The high number of shortest paths crossing the high betweenness edge also contribute to a high value of BC of the nodes connected by such link [10]. Consequently, if we are able to identify clusters inside a graph by using a more efficient algorithm than *edge betweenness*, which exhibits a $O(nm)$ time complexity, then we can focus computation mainly on border nodes of the clusters to calculate their (almost) exact BC, whereas BC of the other nodes could be approximated with an acceptable error.

The approach has been previously studied by some authors of this paper in [10]. Here, we introduce a generalization for further reducing the number of pivots identified with the cluster-based approach, which could be high in some kinds of graphs (such as non-scale-free ones).

Before illustrating the algorithm, we briefly describe the Brandes' one, which is the basis of the proposed approach.

A. Brandes' Algorithm

We assume the following definition throughout the paper: for a generic graph G(V, E), a path $p(v_i,\ v_j)$, between two nodes v_i and v_j, consists of a set of nodes and edges that connect these two nodes. If this set does not exist, the graph is disconnected in *separated components*. The length of a path between any two nodes v_i and v_j, represented by $len(p(v_i,\ v_j))$, is the sum of the weights (or hops in case of unweighted graphs) of the edges (or hops) to reach v_j from v_i. If nodes v_i and v_j are directly connected, then the path length is the weight of the link, or 1 for unweighted graphs. A shortest path between any two nodes v_i and v_j, denoted as $sp(v_i, v_j)$, is a path with the minimum number of hops among all the paths connecting the two nodes. Multiple shortest paths may exist between the same pair of nodes, i.e., all the paths having the same minimum number of hops. Distance $d(v_i, v_j) = len(sp(v_i, v_j))$ is the length of the shortest path between nodes v_i and v_j. We denote as $\sigma_{v_i v_j}$ 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.

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* $\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v)$, 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 this 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 $w \in W$ [7]. 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:

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

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.

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 starting from every node s, considered as source of the 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

[2]As required by the vulnerability assessment based on M-contingency analysis.

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.

B. *One-level clustering fast BC*

Before describing the proposed two-level clustering (*2C-Fast-BC*) algorithm, we first discuss the original one-level clustering (*1C-Fast-BC*) technique with the support of the pseudo-code in Alg. 1 (see Fig. 1).

Given a graph $G(V, E)$, we split it into a set C of clusters by using the Louvain method. This non-parallelizable heuristic exhibits a very good $O(n \cdot log(n))$ complexity. It exploits modularity [25] as a key metric for grouping nodes. Modularity is the ratio between the density of links inside communities to the one of the links among them.

The method consists of two phases that are repeated iteratively until the maximum modularity is reached. In the first phase, each node is assigned to a different community. Then, each node is moved to the community of a neighbor for which the gain of modularity is maximum. If no improvement is possible, the node remains in its current community. Thus, the first phase terminates when no other modularity improvement is possible, i.e., a local maximum has been reached. The second phase of the algorithm generates a new graph where the nodes are the clusters detected during the first phase and the weights of the edges between the new generated nodes are the sums of the weights of the edges between nodes in the corresponding different clusters [26]; the edges between nodes of the same cluster become self-loops.

The quality of the detected communities differs according to the node order followed during the evaluation. To reduce this effect, we perform multiple runs in parallel of the method with different initial configurations. The result with the highest modularity value is selected for the next iteration and the process is repeated until no modularity variation is observed between two iterations.

The main result of Louvain clustering is the identification of *border nodes* (an array for each cluster - line 3 of Alg. 1). A border node is a node having at least one neighbor node 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

Algorithm 1 Two-level Clustering Fast BC Algorithm

```
1: procedure ClusteredBrandes($G, C, KFrac$)
2:     map $i \leftarrow 1, |C|$ do
3:         $bordernodes_i \leftarrow$ findBorderNodes($G, C_i$)
4:     end map
5:     map $i \leftarrow 1, |V|$ do
6:         $BCgraph \leftarrow$ computeLocalBC($i, C, bordernodes$)
7:     end map
8:     reduce $BCgraph.localBC_i, Bgraph.localBC_j, i = j$ do
9:         $BCcluster_i \leftarrow BCgraph.localBC_i + BCgraph.localBC_j$
10:    end reduce
11:    map $i \leftarrow 1, |C|$ do
12:        $BCgraph.superClasses_i \leftarrow$ KMeansClustering($C_i$,
13:                                   $BCgraph.classes_i, KFrac$)
14:    end map
15:    map $i \leftarrow 1, (|BCgraph.superClasses|)$ do
16:        $P_i \leftarrow$ selectPivotOf($BCgraph.superClasses_i, BCcluster$)
17:    end map
18:    map $i \leftarrow 1, (|BCgraph.superClasses|)$ do
19:        $\delta_i \leftarrow$ computeDependencyScoresFromPivot($P_i$)
20:        $\delta_i \leftarrow (\delta_i - BCgraph.localB) \cdot |BCgraph.superClasses_i|$
21:    end map
22:    reduce $\delta_{im}, \delta_{jl}, m = l$ do
23:        $BC_m \leftarrow \delta_{im} + \delta_{jl}$
24:    end reduce
25:    for $i \leftarrow 1, |V|$ do
26:        $BC_i \leftarrow BC_i + BCcluster_i$
27:    end for
28:    return $BC$
29: end procedure
```

Figure 1. 2C-Fast-BC algorithm. 1C-Fast-BC algorithm is the same without blue code.

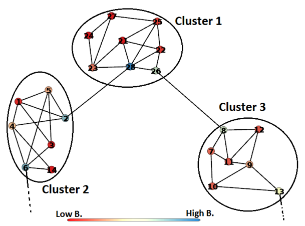
of equivalence $(BCgraph.classes_i)$[3] defined according to the following rule: *two nodes v and w belong to the same class if and only if they have the same normalized distance from each border node of that cluster and the same amount of shortest paths towards the border nodes* (see Table I). The *normalized distance* of node 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 *pivot*[4]. 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).

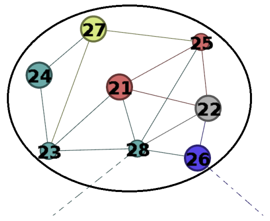
The partial dependency score calculated for the pivot (line 19) is then multiplied by the cardinality of the pivot class (line 20). This method avoids re-applying Brandes' algorithm to another node of the same class, thus ensuring fast calculation of BC if $k \ll n$, where $k = |P|$ is the number of pivots belonging to set P of selected pivots and

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

[4]The partial contribution on border nodes of the same cluster of the pivot is the same only if pair-dependencies with $t \notin C_s$ in Eq. 2 are considered (where C_s represents the pivot cluster).

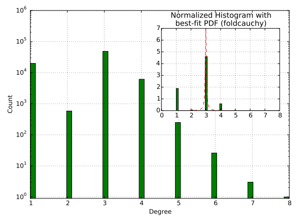


(a) Clustered simple graph

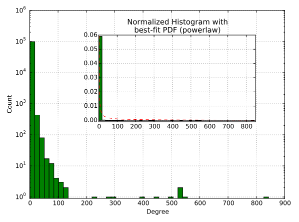


(b) Detail of cluster 1 from Fig. 2a with a different color for each class

Figure 2. Clustering example with classes detection



(a) Degree distribution for G_{Lyon}



(b) Degree distribution for $G_{\text{Scale-free}}$

Figure 3. G_{Lyon} and $G_{\text{Scale-free}}$ degree distributions and matching PDFs

Table I
EXAMPLE OF CLASSES FOUND IN THE CLUSTER IN FIG.2B.

N	N. d. to 28	N. d. to 26	S. p. to 28	S. p. to 26
21	0	1	1	2
22	0	0	1	1
23	0	1	1	1
24	0	1	1	1
25	0	1	1	2
26	1	0	1	1
27	0	1	2	3
28	0	1	1	1

n represents the number of nodes of the graph.

For example, let us consider the class of nodes $23-24-28$ in Fig. 2b: for electing a pivot, we discard border node 28. Among the remaining, we select node 24, due to lowest local betweenness centrality. The normalized distances and the number of shortest paths are reported in Tab. I; each node has a color assigned, which represents the class the node belongs to. Finally, according to Eq. 1, the partial contribution of node 24 to all other nodes of the graph is calculated and multiplied by 3 (the cardinality of the class).

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).

C. *Two-level clustering fast BC*

The algorithm above exhibits very good performance [10] when the overall number of classes in the graph is $\ll n$, being n the number of nodes. This is observable with scale-free graphs (the distribution in Fig. 3b refers to the graph considered in [10]) that contribute to generate clusters with a small amount of border nodes. Unfortunately, our case study dataset, introduced in Section IV, does not produce a scale-free graph, as demonstrated by Fig. 3a. The effect of this distribution is the generation of a number of classes that is almost $\frac{2}{3} \cdot n$. Therefore, the reduction in computation time could be marginal if compared to the initial requirements of performing almost real-time computation for efficient M-contingency analysis.

To further reduce the computation time, we propose an extension of *1C-Fast-BC* that significantly improves performance also for non-scale-free graphs at the price of a larger but tolerable error. The idea we propose is to extend the concept of *class* by introducing *super classes* (see Alg. 1 - blue text) through an additional clustering operation inside each initial Louvain-derived cluster (see Alg. 1 - line 12).

A super class $Bgraph.superClass_j$ is a group of classes belonging to the same cluster. This grouping is performed by using the *Euclidean distance* among the vectors generated by considering, for each node, the normalized distances from the cluster border nodes and the amount of shortest paths towards them. By this approach, nodes are considered equivalent even when they belong to different classes but are very close (Euclidean distance near zero) in the vector space built from the classes table. For example, in Tab. I, the class identified by rows 21 and 25 could be grouped with the class identified by nodes 23, 24 and 28, since the distance between the vectors is close to zero.

To perform this grouping, we exploit a parallel K-means algorithm by using a different K for each initial Louvain cluster. 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, we are able to drive the behavior of the algorithm towards the desired computation time. However, when the computation time decreases the approximation worsens, as deeply illustrated in the evaluation section.

IV. CASE STUDY

In our case study, we focus on reducing the computational effort required to compute network metrics for vulnerability analysis, in light of proposing a solution for on-line and continuous monitoring over large-scale networks.

A. *Road network representation*

We assume that road networks are in steady state and modeled as unweighted and undirected graphs. This assumption has been adopted also in other studies [2], [12], [13]. Some of them have tried to understand the effect of breakage on transportation networks in various parts of the world, such as Toronto [2], Melbourne [27], Sweden [28] and other metropolitan cities [13]. Our case study focuses on Lyon metropolitan road network, which, *differently from most other studied cases, is a non scale-free large graph.*

The related graph will be denoted as $G(V, E)$, where V is the set of nodes and $E \subseteq V \times V$ the set of edges. Each node v_i represents an intersection in the network and an edge e_{ij} between nodes v_i and v_j represents a road.

B. *Nodes removal strategies*

To analyze the impact of unpredictable events on network vulnerability, we can consider different nodes removal strategies, depending on the scenarios we intend to simulate for the analysis. In this paper, we target the worst case scenario, by removing nodes that are the most critical according to a centrality index; they represent possible vulnerabilities of a road network. As a metric to establish a criticality ranking of nodes, we consider BC [29], since it measures the number of shortest paths crossing a node. Nodes with higher BC correspond to central intersections from an infrastructural perspective, being usually selected by commuters to reach their destinations and naturally prone to breakdown compared to other nodes.

C. *Resilience metrics*

To quantify the resilience of a road network, we consider the following metrics:

1) *Global Efficiency* (GE): represents the ability to efficiently exchange information in the network [30]. It is defined as:

$$GE(G) = \frac{1}{n(n-1)} \sum_{i \neq j} \frac{1}{d(v_i, v_j)}. \tag{3}$$

2) *Vulnerability*: is the drop in information exchange due to removal of a node and all corresponding edges [31]. Formally it is defined as:

$$V_{v_i} = \frac{GE - GE_{v_i}}{GE}, \tag{4}$$

where GE is the global efficiency of the original network as from Eq. 3 and GE_{v_i} is the global efficiency after the removal of node v_i and all the edges incident on it.

D. *Road-network dataset*

For our analysis, we consider a graph corresponding to the road network of the Grand Lyon metropolitan area, France, and its surroundings, covering an area of approximately 3,000 Km^2. This dataset was created using digital maps supplied by the French National Institute of Geographic Information (IGN). The network consists of 112,567 nodes and 240,372 edges.

In the original graph, some roads are split in multiple segments only because of some property changes across the segments (e.g., street name, road slope, etc.). This means that a single long road with no real intersections along it can be actually composed of multiple nodes and edges in the graph. We decided to filter out this noise, by retaining only real intersections and actual link endpoints as nodes in our dataset.

Next, to avoid a disconnected graph, we selected only the largest connected component (termed G_{Lyon} in the rest of the paper) by filtering out very small isolated subnetworks that typically include only country roads or cut areas lying at the border of the analyzed region. Finally, due to partial available information on road driving direction and number of lanes, we treat the network as undirected. We remark here that the solutions described in the next section can be easily extended to work with directed and weighted graphs.

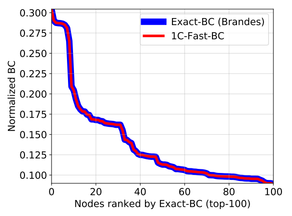
The final G_{Lyon} graph has 75,474 nodes and 96,406 edges. Nodes with the highest values of BC mostly correspond to highway interchanges (e.g., the A6 and D383) and roads that cross bridges. These represent elements of the network with limited alternative routes, and thus highly vulnerable from a topological perspective.

The graph has a density of $3.38e^{-05}$ and clustering coefficient of 0.054. Its average path length and diameter are 84.74 and 244 respectively. The average degree is 2.55. These properties indicate that G_{Lyon} is very sparse in its nature. Fig. 3a shows its degree distribution, i.e., a Gaussian-like PDF where the number of road links merging at any intersection lies between 1 and 8, being 3 the most frequently observed degree. GE is equal to 0.0148193, a fairly low value that confirms the generally low degree of connectivity of the graph. It is important to highlight the exploratory nature of our work from the graph-topology perspective, since a Gaussian-like distribution is different from those usually reported in previous studies on resilience analysis via BC that mainly considered scale-free topologies [13], [17], [19].

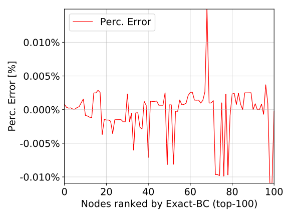
V. Evaluation

We implemented our algorithms using *Scala* programming language and the *Apache-Spark* framework, by leveraging multi-core processing for parallel execution. Apache Spark was configured to work in local mode, using 10 threads to partition the execution load of the map-reduce tasks on the available cores of an Intel Xeon E5 2640 2.4 GHz multi-core machine, equipped with 56 virtual cores and 128 GB of DDR4 RAM. We also considered a Scala implementation of Brandes' algorithm [7] (referred as *Exact-BC*), used as a benchmark in performance evaluation. *Exact-BC* was executed in the same testing environment used for *Fast-BC*, with 10 threads for parallelism.

From our experiments on the correlation between vulnerability and BC (not reported due to lack of space), we observed that nodes with higher values of betweenness centrality are the ones that affect most vulnerability of a road-traffic network, defined as in Eq. 4. Monitoring should continuously identify these nodes since, if disrupted, they may significantly reduce network connectivity. On the other hand, their inhibition may significantly alter network



(a) Values of 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

Figure 4. Accuracy of the 1C-Fast-BC algorithm

topology, forcing to recompute betweenness centrality for the whole network. To continuously identify the most M critical nodes (M-contingency analysis), a fast algorithm to compute BC is needed.

A. M-contingency analysis: 1C-Fast-BC vs Exact-BC

In this section, we compare the performance of *1C-Fast-BC* against that of *Exact-BC* on the G_{Lyon} graph. 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)[5] and the Euclidean distance of normalized BC vectors, as in [8], [9]; 2) accuracy of nodes' ranking, expressed both as the relative number of nodes not included in the exact top-N ranking by the tested algorithm and through inversion distance, as in [8], [9]; 3) execution time.

Given the specific conceptual design of the proposed algorithms and our objective of continuously monitoring only the most critical intersections of the road-traffic network, the performance analysis is mainly focused on the most-critical nodes of the network (e.g., top-100, top-5%, etc.). However, to compare our approach with the ones proposed in [8], [9], we also report on Euclidean and inversion distances of all the nodes of the graph.

Fig. 4 shows the normalized BC values computed by the two algorithms for the top-100 nodes, ranked according to their exact value of BC. Fig. 4a highlights an almost perfect overlap of the two curves, confirmed by the extremely low percentage error in Fig. 4b, bounded in the range [-0.015%, 0.015%]. Coherently, we observe an extremely low NRMSE of $2.93887e^{-5}$ and 0 missing nodes in the top-100 ranking. The NRMSE rises to $8e^{-3}$ with 341 missing nodes, when considering the top-30% (i.e., approximately 25,000) nodes, thus showing an acceptable approximation even on nodes with low BC values.

[5]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.

Regarding execution time, the algorithm takes 1,688 seconds to complete, a fairly limited improvement with respect to the 1,982 seconds of the *Exact-BC*. This is motivated by the relative high number of classes (48,960) identified by *1C-Fast-BC* , and consequently a high number of pivots compared to the total number of source nodes (75,475) used by *Exact-BC*.

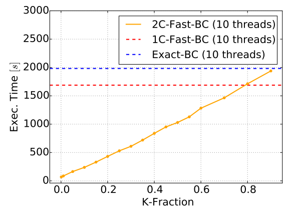
1C-Fast-BC is a powerful solution to precisely compute BC, even on nodes that exhibit very low values of BC. However, the surprising performances of this algorithm for scale-free graphs are not confirmed for road networks that exhibit a non-scale-free distribution. In these cases, the number of classes, and consequently the number of pivots, is quite high and can not be further reduced by increasing the approximation error.

B. M-contingency analysis: 2C-Fast-BC improvements

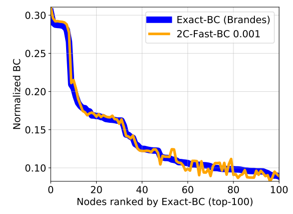
2C-Fast-BC (see Sec. III-C) aims at reducing execution time on non-scale-free graphs, by preserving acceptable levels of accuracy in BC approximation. This behavior is desirable when computation time must be very low and errors are tolerated below a specific threshold. This algorithm extends *1C-Fast-BC* by including a Map-reduce parallel version of the K-means clustering algorithm, based on the implementation provided in Apache Spark *MLlib* machine learning library.

Similar to the previous analysis, we test *2C-Fast-BC* on the G_{Lyon} graph by using different fractions (i.e., the *K-Fraction* parameter) of the number of classes as the K in the K-means. Fig. 5a shows that *2C-Fast-BC* execution time stays below *1C-Fast-BC* and *Exact-BC* ones (that do not depend on the K-Fraction parameter), up to very large values of K-Fraction (i.e., 0.8). Larger values of the parameter result in higher execution times (larger than the ones of *1C-Fast-BC* for K-Fraction > 0.8) due to the computation overhead associated to the K-means clustering. Obviously, lower K-Fractions introduce a larger approximation that negatively affects BC values even for critical nodes. As an example, Fig. 5b presents BC values for the top-100 nodes when using an extremely low K-Fraction of 0.001. For this configuration, computation time is very low (69 s) but percentage error is relatively high (i.e., in the range [-12%, 12%], not reported due to space limitation) with a NRMSE of 0.04 and 4 missing nodes in the top-100 ranking.

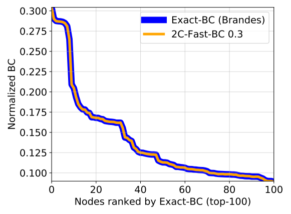
Conversely, by choosing slightly larger K-Fractions (e.g., a K-Fraction of 0.3), the algorithm quickly converges towards very good levels of accuracy, as highlighted by the perfect overlap with the *Exact-BC* values in Fig. 5c. The much better quality of the approximation is confirmed by a percentage error in the range [-0.2%, 0.3%] (see Fig. 5d), a NRMSE of 0.001 and 0 missing nodes in the top-100 ranking. Also the Euclidean distance (Fig. 5e) and the percentage of inversions against the maximum number of possible inversions (Fig. 5f) are good if compared with the



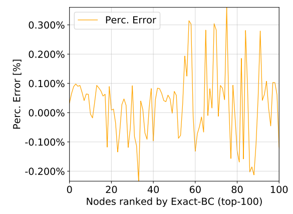
(a) Execution time of 2C-Fast-BC with different K-Fractions



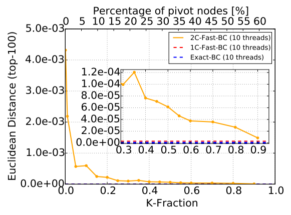
(b) BC values with K-Fraction = 0.001 for the top-100 nodes



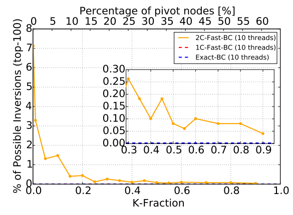
(c) BC values with K-Fraction = 0.3 for the top-100 nodes



(d) Percentage error with K-Fraction = 0.3 over the top-100 nodes



(e) Euclidean distance of BC values with different values of K-Fraction (top-100 nodes)



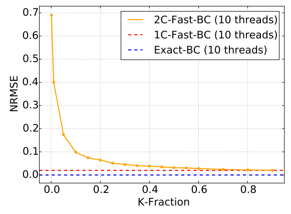
(f) % of possible inversions with different values of K-Fraction (top-100 nodes)

Figure 5. Accuracy of the 2C-Fast-BC algorithm for the top-100 BC values

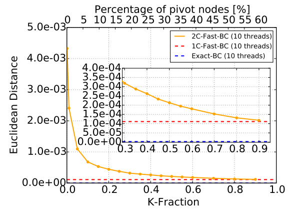
results of similar graphs in [8], [9]. Most importantly, the *2C-Fast-BC* with a K-Fraction of 0.3 executed in only 609 seconds, corresponding to a 277% and a 325% decrease in execution time with respect to the *1C-Fast-BC* and *Exact-BC*, respectively.

Figures 6a, 6b and 6c present the full trend for different K-Fractions of the global NRMSE, the Euclidean distance and the percentage of inversions, respectively. Fig. 6d reports on the percentage of missing elements over the top-5% nodes. The results show that by choosing a K-Fraction equals to 0.3 (18% of pivots), the global Euclidean distance is $3.2e^{-4}$ whereas the percentage of inversions is about 2.3%[6], whereas for the top-100 ranked nodes the same metrics exhibit the following values: $1e^{-4}$ and 0.25%, respectively. The difference between the two scenarios confirms that our approach privileges top-ranked nodes; however, the precision of the algorithm is high even considering a limited number

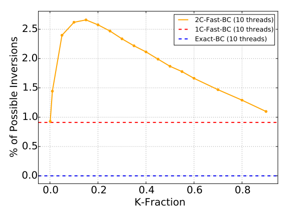
[6]The low value of % inversions in Fig. 6c for small values of K-Fractions depends on the presence of the high number of very small BC values where the ranking error is limited.



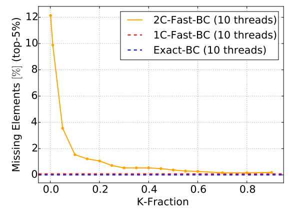
(a) Global NRMSE with different values of K-Fraction



(b) Euclidean distance of BC values with different values of K-Fraction



(c) % of possible inversions with different values of K-Fraction



(d) Percentage of missing nodes in top-5% node-ranking

Figure 6. Accuracy of the 2-Fast-BC algorithm for the whole graph and top 5% nodes

(18% of the whole set) of pivots.

By properly selecting the value of the K-Fraction, 2C-Fast-BC can produce high-accurate approximation of BC values while significantly reducing execution time, when compared to both the Exact-BC and the 1C-Fast-BC algorithms. The algorithm is very useful when the application domain tolerates small errors but requires quick responses for real-time M-contingency analysis.

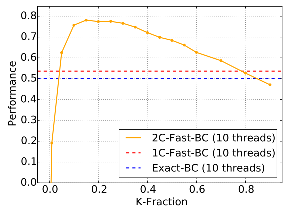
C. A synthetic index of performance

Since our algorithm generates approximated BC values in a time depending on the error we can tolerate, an aggregated index for evaluating both performance and accuracy is useful. To this end, we propose the following index:

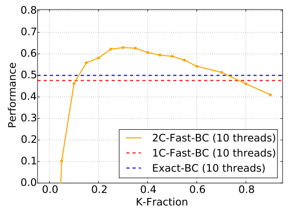
$$P(x) = 0.5\left(1 - \frac{t_x - t_{ideal}}{t_{Max} - t_{ideal}}\right) + 0.5\left(1 - \frac{e_x}{e_{Max}}\right) \quad (5)$$

The equation depends on two variables that are computed after executing 2C-Fast-BC in configuration x: 1) the time taken (t_x) for BC calculation; 2) the NRMSE error (e_x) of the approximated BC values. The equation includes three parameters t_{ideal}, t_{Max} and e_{Max} that can be fixed according to the specific domain requirements. Specifically, t_{Max} is the maximum tolerated execution time. Similarly, e_x represents the maximum allowed NRMSE. Finally, t_{ideal} constitutes the minimum amount of time allowed to the execution of the algorithm.

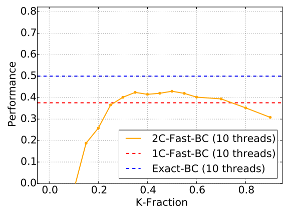
In the first setting of the P metric, represented in Fig. 7a, we defined t_{ideal} equal to the execution time of the fastest 2C-Fast-BC configuration, i.e., 60 seconds with a K-Fraction



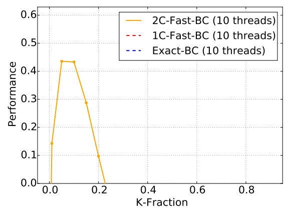
(a) e_{Max} = 0.25, t_{Max} = 1982s



(b) e_{Max} = 0.1, t_{Max} = 1982s

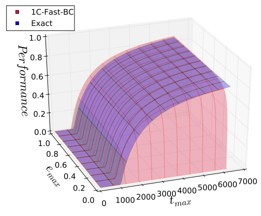


(c) e_{Max} = 0.05, t_{Max} = 1982s

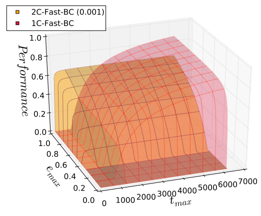


(d) e_{Max} = 0.25, t_{Max} = 600s

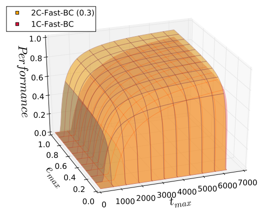
Figure 7. Performance



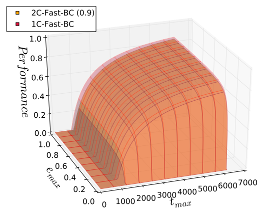
(a) Exact-BC vs 1C-Fast-BC



(b) 2C-Fast-BC (K-Fraction 0.001) vs 1C-Fast-BC



(c) 2C-Fast-BC (K-Fraction 0.3) vs 1C-Fast-BC



(d) 2C-Fast-BC (K-Fraction 0.9) vs 1C-Fast-BC

Figure 8. 3-d Performance curves with different values of t_{max} and e_{max} (t_{ideal} = 0s)

of 0.001. We set t_{Max} to the execution time of the Exact-BC algorithm (i.e., 1,982 seconds) and e_{Max} to 0.25. This setting rewards computing times below the one of Exact-BC and errors below 25% NRMSE. Fig. 7a clearly shows the values of K-Fraction satisfying these performance requirements, i.e., K-Fraction should be selected within the range [0.05, 0.7], with the most performing configuration being associated to a K-Fraction of 0.15. A comparison with the performance of the Exact-BC and the 1C-Fast-BC is also visually reported in the figure.

Different settings of e_{Max} and t_{Max} are depicted in Figs. 7b:7d. Such settings correspond to more stringent requirements on the tolerated NRMSE (Figs. 7b and Fig. 7c) and execution time (Fig. 7d), respectively. Interestingly, a configuration with an extremely low tolerated execution time of 600 seconds and maximum NRMSE of 0.25 (Fig.. 7d) can be satisfied only by using the 2C-Fast-BC with a K-Fraction between 0.001 and 0.25.

In Fig. 8, we exploit 3D plots to represent the surfaces generated by varying the two parameters t_{Max} and e_{Max} in Eq. 5, for both 1C-Fast-BC and 2C-Fast-BC. Fig. 8a compares the performance of 1C-Fast-BC to that of Exact-BC. The two surfaces are mostly overlapping, thus showing very similar performance on the G_{Lyon} graph. In particular, performance is slightly better with 1C-Fast-BC when a larger maximum error (i.e., larger than 0.4) can be tolerated. Exact-BC should instead be preferred whenever the maximum tolerated time is large enough (i.e., larger than 1600 seconds) for the algorithm to complete and the maximum tolerated error is smaller than 0.2. Different configurations of 2C-Fast-BC are compared to 1C-Fast-BC in Figs. 8b:8d. Specifically, Fig. 8b is related to the lowest values of K-Fractions (i.e., 0.001) considered in our evaluation. Good performance can be achieved with this configuration when the maximum tolerated execution time is significantly low (e.g., lower than 100 seconds) and a fairly large error on BC values can be tolerated (e.g., larger than 0.5-0.6). Configurations with larger K-Fractions (e.g., 0.3 in Figs 8c) appear to be generally preferable when requirements are not particularly stringent both on BC error and computation time. Finally, configurations with larger K-Fractions tend to exhibit performance that are very close to that of 1C-Fast-BC (see Fig. 8d).

VI. Conclusion

In this paper, we proposed a new algorithm for fast computation of approximated betweenness centrality values. The algorithm reduces the computation time by identifying k pivots for single-source shortest-paths explorations, so achieving an ideal speedup $\frac{n}{k}$. Differently from other similar approaches based on pivot selection, the proposed approach focuses on the border nodes of clusters and selects pivots by considering topological properties of graphs, taking into account the formula for evaluating dependency score.

We applied the proposed algorithms for vulnerability analysis of a large, real-world road network. The encouraging results suggest the adoption of our algorithms for M-contingency analysis over both scale-free and non-scale-free graphs, by rapidly locating the most M critical nodes of a snapshot of a dynamically evolving road network.

In order to generalize the approach to dynamic networks, two main extensions are planned: 1) adapting 1C/2C-Fast-BC algorithms to weighted and directed graphs; 2) enriching network modeling by taking into account additional structural properties of road networks.

In this perspective, our work constitutes the foundation for building a framework aimed at vulnerability continuous monitoring. Such framework should provide the necessary software infrastructure to: 1) collect and analyze large-scale, real-time, heterogeneous data; 2) handle dynamic, weighted, directed graphs; 3) adaptively change the settings of the 2C-Fast-BC algorithm according to dynamically evolving domain requirements (e.g., the framework could run in an *emergency mode*, requiring low computation time and larger tolerated error, or alternatively in a *maintenance mode*, demanding smaller error but allowing for more execution time).

References

[1] S. P. Borgatti, A. Mehra, D. J. Brass, and G. Labianca, "Network analysis in the social sciences," *Science*, vol. 323, no. 5916, pp. 892–895, 2009.

[2] D. King and A. Shalaby, "Performance metrics and analysis of transit network resilience in Toronto," *Transportation Research Record*, 2016.

[3] D. Koschützki and F. Schreiber, "Centrality analysis methods for biological networks and their application to gene regulatory networks," *Gene regulation and systems biology*, vol. 2, pp. 193–201, 2008.

[4] P. Holme, B. J. Kim, C. N. Yoon, and S. K. Han, "Attack vulnerability of complex networks," *Physical Review E*, vol. 65, no. 5, p. 056109, 2002.

[5] S. Jin, Z. Huang, Y. Chen, D. Chavarra-Miranda, J. Feo, and P. C. Wong, "A novel application of parallel betweenness centrality to power grid contingency analysis," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, 2010, pp. 1–7.

[6] T. Carpenter, G. Karakostas, and D. Shallcross, "Practical issues and algorithms for analyzing terrorist networks," *Proceedings of the Western Simulation MultiConference*, 2002.

[7] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, no. 163, 2001.

[8] U. Brandes and C. Pich, "Centrality estimation in large networks," *International Journal of Bifurcation and Chaos*, vol. 17, no. 07, pp. 2303–2318, 2007.

[9] R. Geisberger, P. Sanders, and D. Schultes, "Better approximation of betweenness centrality." in *ALENEX*. SIAM, 2008, pp. 90–100.

[10] P. Suppa and E. Zimeo, "A clustered approach for fast computation of betweenness centrality in social networks," in *2015 IEEE International Congress on Big Data*, June 2015, pp. 47–54.

[11] S. Zhao, P. Zhao, and Y. Cui, "A network centrality measure framework for analyzing urban traffic flow: A case study of Wuhan, China," *Physica A: Statistical Mechanics and its Applications*, vol. 478, pp. 143 – 157, 2017.

[12] Y. Zhang, X. Wang, P. Zeng, and X. Chen, "Centrality characteristics of road network patterns of traffic analysis zones," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2256, pp. 16–24, 2011.

[13] B. Berche, C. von Ferber, T. Holovatch, and Y. Holovatch, "Resilience of public transport networks against attacks," *The European Physical Journal B*, vol. 71, no. 1, pp. 125–137, 2009.

[14] D. A. Bader and K. Madduri, "Parallel algorithms for evaluating centrality indices in real-world networks," in *Parallel Processing, 2006. ICPP 2006. International Conference on*. IEEE, 2006, pp. 539–550.

[15] K. Madduri, D. Ediger, K. Jiang, D. A. Bader, and D. Chavarria-Miranda, "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*. IEEE, 2009, pp. 1–8.

[16] S. White and P. Smyth, "Algorithms for estimating relative importance in networks," in *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2003, pp. 266–275.

[17] K. Ohara, K. Saito, M. Kimura, and H. Motoda, *Accelerating Computation of Distance Based Centrality Measures for Spatial Networks*.

[18] N. Kourtellis, G. D. F. Morales, and F. Bonchi, "Scalable online betweenness centrality in evolving graphs," *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, vol. 00, pp. 1580–1581, 2016.

[19] P. Holme, "Congestion and centrality in traffic flow on complex networks," *Advances in Complex Systems (ACS)*, vol. 06, no. 02, pp. 163–176, 2003.

[20] Y. Altshuler, R. Puzis, Y. Elovici, S. Bekhor, and A. Baglion-iPentland, "Augmented betweenness centrality for mobility prediction in transportation networks," *Finding Patterns of Human Behaviors in Network and Mobility Data (NEMO)*, 2011.

[21] S. Gao, Y. Wang, Y. Gao, and Y. Liu, "Understanding urban traffic-flow characteristics: a rethinking of betweenness centrality," *Environment and Planning B: Planning and Design*, vol. 40, no. 1, pp. 135–153, 2013.

[22] A. Kazerani and S. Winter, "Can betweenness centrality explain traffic flow?" *AGILE*, 2009.

[23] A. Jayasinghe, K. Sano, and H. Nishiuchi, "Explaining traffic flow patterns using centrality measures," *International Journal for Traffic & Transport Engineering*, vol. 5, no. 2, pp. 134–149, 2015.

[24] J. Yang and Y. Chen, "Fast computing betweenness centrality with virtual nodes on large sparse networks," *PLoS One*, 2011.

[25] M. E. Newman, "Analysis of weighted networks," *Physical Review E*, vol. 70, no. 5, p. 056131, 2004.

[26] A. Arenas, J. Duch, A. Fernández, and S. Gómez, "Size reduction of complex networks preserving modularity," *New Journal of Physics*, vol. 9, no. 6, p. 176, 2007.

[27] G. Leu, H. Abbass, and N. Curtis, "Resilience of ground transportation networks: a case study on melbourne," *33rd Australasian Transport Research Forum Conference*, 2010.

[28] E. Jenelius and L.-G. Mattsson, "Road network vulnerability analysis of area-covering disruptions: A grid-based approach with case study," *Transportation Research Part A: Policy and Practice*, vol. 46, no. 5, pp. 746–760, 2012.

[29] L. C. Freeman, "A set of measures of centrality based on betweenness," *Sociometry*, vol. 25, pp. 35–41, 1997.

[30] V. Latora and M. Marchiori, "Efficient behavior of small-world networks," *Phys. Rev. Lett.*, vol. 87, no. 19, Oct. 2001.

[31] L. Costa, F. Rodrigues, G. Travieso, and P. Boas, "Characterization of complex networks: A survey of measurements," *Advances in Physics*, vol. 56, no. 1, pp. 167–242, 2007.