Fast Computation of Betweenness Centrality to Locate Vulnerabilities in Very Large Road Networks

Submission Date: July, 31st 2017

Angelo Furno (Corresponding Author)[1],
[1] Univ. Lyon, IFSTTAR, ENTPE, LICIT UMR_T9401, F-69675, Lyon, France
angelo.furno@ifsttar.fr

Nour-Eddin El Faouzi[1,2]
[1] Univ. Lyon, IFSTTAR, ENTPE, LICIT UMR_T9401, F-69675, Lyon, France
[2] Queensland University of Technology, STRC, Gardens Point Campus, 2 George Street, G.P.O. Box 2434, Brisbane, Queensland 4001, Australia.
nour-eddin.elfaouzi@ifsttar.fr

Rajesh Sharma[3]
[3] University of Tartu, Institute of Computer Science, Juhan Liivi 2, Tartu, Estonia
rajesh.sharma@ut.ee

Eugenio Zimeo[4,5]
[4] University of Sannio, Department of Engineering, 82100, Benevento, Italy
[5] CINI - Smart Cities and Communities lab, Italy
eugenio.zimeo@unisannio.it

Submitted to the 97th Annual Meeting of the Transportation Research Board for publication and presentation

---

Word Count:

Number of words: 5962
Number of figures: 6 (250 words each)
Number of tables: 0 (250 words each)
Total: 7462

Abstract

The ability to detect critical spots in transportation networks is fundamental to improve traffic operations and road-network resilience. Real-time monitoring of these networks, especially in very large metropolitan areas, is a compelling challenge due to the complexity of computing robustness metrics.

This paper presents a study of vulnerability in a real-world, very-large road network by adopting graph-based modeling and analysis, and big-data techniques for processing the related datasets. We first analyze the correlation between global efficiency and betweenness centrality, proving that nodes with higher betweenness centrality influence network vulnerability the most. Then, we present an algorithm for fast computation of approximated betweenness centrality that significantly reduces execution time. 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.

1. INTRODUCTION

Nowadays, cities are facing unprecedented challenges due to urbanization, climate change, and fast technology advances. They will be forced to cope with growing populations and connected (digital and physical) infrastructures whose robustness and resilience will become an essential property for their continuous operation.

Resilience describes the ability of a given system to provide fundamental services to people without discontinuity, even in presence of adverse or catastrophic events. As a consequence, it will be a key property for achieving smartness in cities of the near future. Resilience concerns several infrastructures used for transporting goods or people: energy, water, information and road networks are a few examples. An unforeseen event that originates a breakage of one of these infrastructures may cause incalculable damages with serious socio-economic consequences.

Transportation networks are bound to controllable (e.g., riots, technical errors, etc.) and uncontrollable events (e.g., earthquakes, floods, etc.) that can easily cause disruptions in the smooth operation of their services. In transportation literature, resilience is widely considered crucial to understand and quantify network robustness with respect to different kinds of failures and other unpredictable events [1, 2, 3, 4].

While resilient transportation networks can help in reducing the impact of such failures on traffic flow, human safety and urban economy, *vulnerability* can be viewed as the flip-side of resilience: a vulnerable system has limited ability to absorb and react to the strains caused by adverse events.

The availability of real-time information provided by a growing number of sensors and small devices distributed across geographic areas makes it possible today to use novel approaches to study robustness and adaptation ability of cities against possible unpredictable events. Moreover, the adoption of cloud computing and big-data techniques to effectively analyze this large amount of data opens the way to more quantitative studies to characterize city infrastructures. City key indicators derived from quantitative complex analyses will help planners and mobility operators to clearly identify vulnerabilities and adopt relevant strategies to mitigate risks. This is particularly important for physical transportation networks where crashes, weather conditions and social events create everyday risky situations for their correct operation.

In this paper, we follow a network-based approach to characterize the impact of breakage on road networks. We aim at identifying significant correlations between network properties and structural features of road networks. To this end, we start by considering a steady state of the transportation network with correctly operating links and free-flow traffic conditions.

We study vulnerability using *Global Efficiency* (GE) [5] which has been widely used in varied types of networks, such as power grids [6], the Internet [7], biological networks [8] and transportation [1]. To identify vulnerable nodes we use *betweenness centrality* (BC), a fundamental metric of centrality already used in transportation to identify topological criticalities [1, 4], traditionally preferred to other centrality metrics such as degree and closeness centrality [3]. BC has been also adopted for analyzing and predicting traffic flows in transportation networks [9, 10]. However, studying the impact of network breakage on road traffic remains a very complex task, difficult to be treated with traditional desktop hardware and state-of-the-art algorithms, especially in large metropolitan areas with a tremendous amount of links and intersections.

The best algorithm for computing exact BC has been proposed in [11], however real-time monitoring needs faster algorithms joined with big data computation techniques and scalable hardware. In this work, we propose a fast algorithm to compute approximate BC for quickly locating the nodes of a transportation network that severely impact its vulnerability. The paper overcomes state-of-the-art limitations by building the foundation for *real-time M-contingency analysis* of dynamic traffic networks (including information on traffic flow, traffic demand, accidents, etc.), 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 circumstances.

The rest of the paper is organized as follows. In Sec. 2, we present related work. In Sec. 3, we describe our model and the metrics used to characterize road-network vulnerability. In Sec. 4, we discuss the large-scale dataset used in our analysis, while in Sec. 5 we present the algorithm proposed for fast BC calculation. In Sec. 6, we evaluate our approach on the considered dataset. We conclude in Sec. 7 by also discussing future directions.

2. RELATED WORK

Several graph-based metrics have been proposed for the analysis of transportation networks vulnerability. To simulate the process of network breakage and evaluate network robustness, nodes can be removed by using various strategies such as randomly or by selecting relevant nodes with respect to a given metric. It is intuitive that nodes at very central position of the network, i.e., crossed by multiple shortest paths, could be highly critical spots. Thus, breakage of such nodes may have a higher impact on the network compared to non-central ones. In this regard, one of the most used metric to calculate nodes' centrality is BC [12], especially in transportation network settings [3].

A large stream of transportation studies have exploited BC also for traffic flow prediction (e.g., [13, 14, 2, 15]) or vulnerability quantification (e.g., [1, 3, 4]). In this context, some authors have highlighted limitations of the BC metric in representing traffic dynamics [16, 13, 15, 17]. However, such shortcomings can be overcome by augmenting the graph representation of the networks by taking into account also spatio-temporal aspects (e.g., congestion, accidents, road capacity changes, etc.) and geometric properties of road network [2], 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.

For the reasons above, 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 [1, 3, 4]. Some of them have tried to understand the effect of breakage, especially of nodes selected via BC, on transportation network in various parts of the world, such as Toronto [1], Melbourne [18], Sweden [19] and other metropolitan cities [4]. In comparison, no other study has been performed on Lyon metropolitan road network, which, *differently from most other studied cases, is a non-scale-free graph*[1]. Moreover, *the datasets being used in previous studies are relatively small*, that is in the order of thousands of nodes. A larger size has an important impact on performance as the computation time for calculating BC is very high.

Very recently, some researchers have proposed different (exact or approximated) solutions to reduce the computation time of BC [11, 20, 21, 22]. In [23], an efficient algorithm for calculating BC has been proposed for incremental BC computation. However, the high speedup, characterizing the algorithm when one single node is inhibited, drastically reduces when analyzing the impact of M nodes. These incremental algorithms are very fast and useful when network changes are limited and continuous, but *M-contingency analysis is a much more complex and challenging problem as it should consider M different perturbations of the network to simulate possible critical scenarios as consequence of catastrophic events or phenomena.* In [24], authors have analyzed large scale-free graphs by proposing an approximated BC algorithm. However, in spite of a modest improvement of performance, BC accuracy largely fluctuates over the top-100 nodes, exhibiting an important error even on critical nodes.

[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.

In conclusion, a proper contingency analysis for vulnerability estimation in large-scale transportation networks is missing from existing studies. In our 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. To the best of our knowledge, this is the first paper tackling this perspective with respect to road-traffic networks, by *advancing the state of the art of M-contingency analysis.*

3. NETWORK MODELING AND METRICS

In this section, we introduce the model used for road networks representation and the metrics to measure their degree of vulnerability.

3.1. Network representation

We model transportation networks as undirected graphs $G(V, E)$, where V denotes nodes and $E \subseteq V \times V$ 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. $N = |V|$ denotes the number of nodes in the graph.

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 number of 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 1. 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. 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.

3.2. 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:

1. *Possible case*: the nodes are removed uniformly at random. This particular strategy captures real scenarios where an event can happen at any road or intersection causing traffic disruptions;

2. *Worst case*: removed nodes are the most critical according to a centrality index; they represent possible vulnerabilities of a road network.

In this paper, we target the worst case scenario. As a metric to establish a criticality ranking of nodes, we consider BC [12], 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. The BC of a node v is defined as:

$$BC(v) = \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{1}$$

where σ_{st} is the total number of shortest paths from node s to node t and $\sigma_{st}(v)$ is the number of those paths that cross v. For scaling purpose, values are often normalized by dividing them by $N \cdot (N-1)$.

3.3. Resilience metrics

Network partitioning and network ability to move vehicles among nodes with a short number of hops might be useful metrics to evaluate the impact of network breakages on transportation networks. Therefore, 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 [5]. Let N be the total nodes in the network, and $len(sp(v_i, v_j))$ the length of the shortest path between any two node v_i and v_j, the GE of the network G is defined as:

$$GE(G) = \frac{1}{N(N-1)} \sum_{i \neq j} \frac{1}{len(sp(v_i, v_j))}. \tag{2}$$

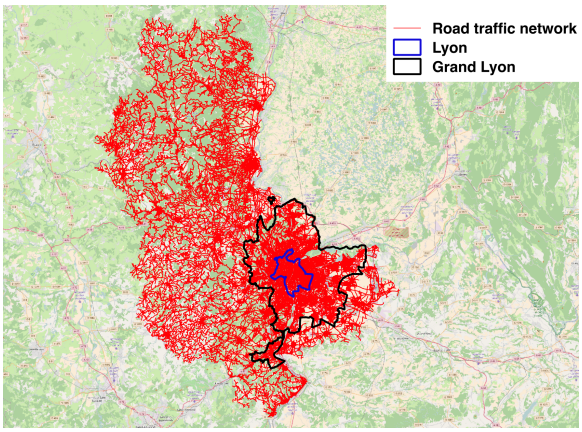
2. **Vulnerability**: is defined as the drop in information exchange due to removal of a node and all corresponding edges [25]. Formally it is defined as:

$$V_{v_i} = \frac{GE - GE_{v_i}}{GE}, \tag{3}$$

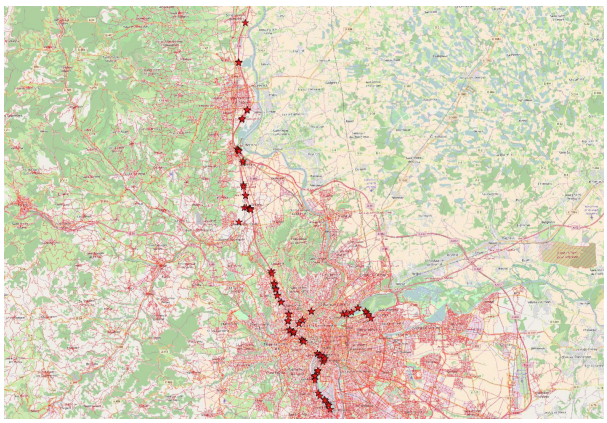
where GE is the global efficiency of the original network as from Eq. 2 and GE_{v_i} is the global efficiency after the removal of node v_i and all the edges incident on it.

4. 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 (see Figure 1a). 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.



(a) Map (in red) of the analyzed road-traffic network and its geographical extent



(b) Critical nodes from the BC metric (mostly on the A6 and D383 highways)

Figure 1: The Grand Lyon road traffic network.

In the original graph, some roads are split in multiple segments only because of 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 following) 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 graph.

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, as reported in Fig. 1b. These represents 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 [4, 24, 13].

5. FAST 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 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 path crossing the high betweenness edge also contribute to a high value of BC of the nodes connected by such link [22].

Consequently, if we are able to identify clusters inside a graph by using a more efficient algorithm than *edge betweenness*, which exhibits a $O(|V||E|)$ 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 [22]. Here, we introduce a variation to address the problems arisen from the application of the original algorithm to the specific G_{Lyon} dataset.

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

5.1. 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:

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

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

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 $P_s(w)$ 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 $w \in W$ [11]. Consequently, we have:

$$\delta_{s\bullet}(v) = \sum_{w:v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_{s\bullet}(w)), \tag{4}$$

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

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

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 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.

5.2. One-level clustering fast BC

In this section, we describe the proposed two-level clustering (*2C-Fast-BC*) algorithm by first discussing the original one-level clustering (*1C-Fast-BC*) technique with the support of the pseudo-code in Alg. 1 (see Fig. 2).

Initially, given a graph $G(V, E)$, we split it into a set C of clusters by using the Louvain method. This non-parallelizable algorithm exhibits a very good $O(N \cdot log(N))$ complexity. It is a heuristic method for community (cluster) detection that exploits modularity [26] 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 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 of equivalence ($BCgraph.classes_i$)[3] defined according to the following rule: *two nodes v and w belong to the same*

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

Algorithm 1 Two-level Clustering Fast BC Algorithm

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

Figure 2: 1C-Fast-BC algorithm with 2C-Fast-BC extensions in blue.

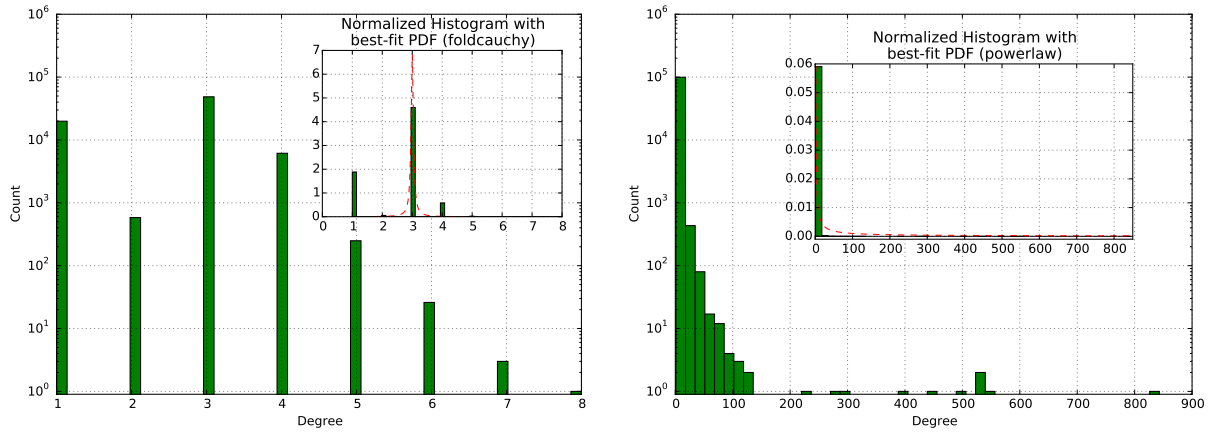
class if and only if they have the same normalized distance from each border node of that cluster and the same amount of shortest paths towards the border nodes. The *normalized distance* of node j is the length of the shortest path for reaching a border node minus the minimum distance among the ones of the shortest paths to reach each border node. 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 15).

The partial dependency score calculated for the pivot (line 18) is then multiplied by the cardinality of the pivot class (line 19). This method avoids re-applying Brandes' algorithm to another node of the same class, thus ensuring fast calculation of BC if $P \ll N$, where P represents the set of pivots selected and N represents the number of nodes of the graph.

The final value of BC is obtained for each node (line 25) by summing up all partial contribu-

[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. 5.1 are considered (where C_s represents the pivot cluster).



(a) Degree distribution for G_{Lyon}

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

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

tions (produced by the reduce operation) with local BC values (to compensate the partial local contribution subtracted at line 16).

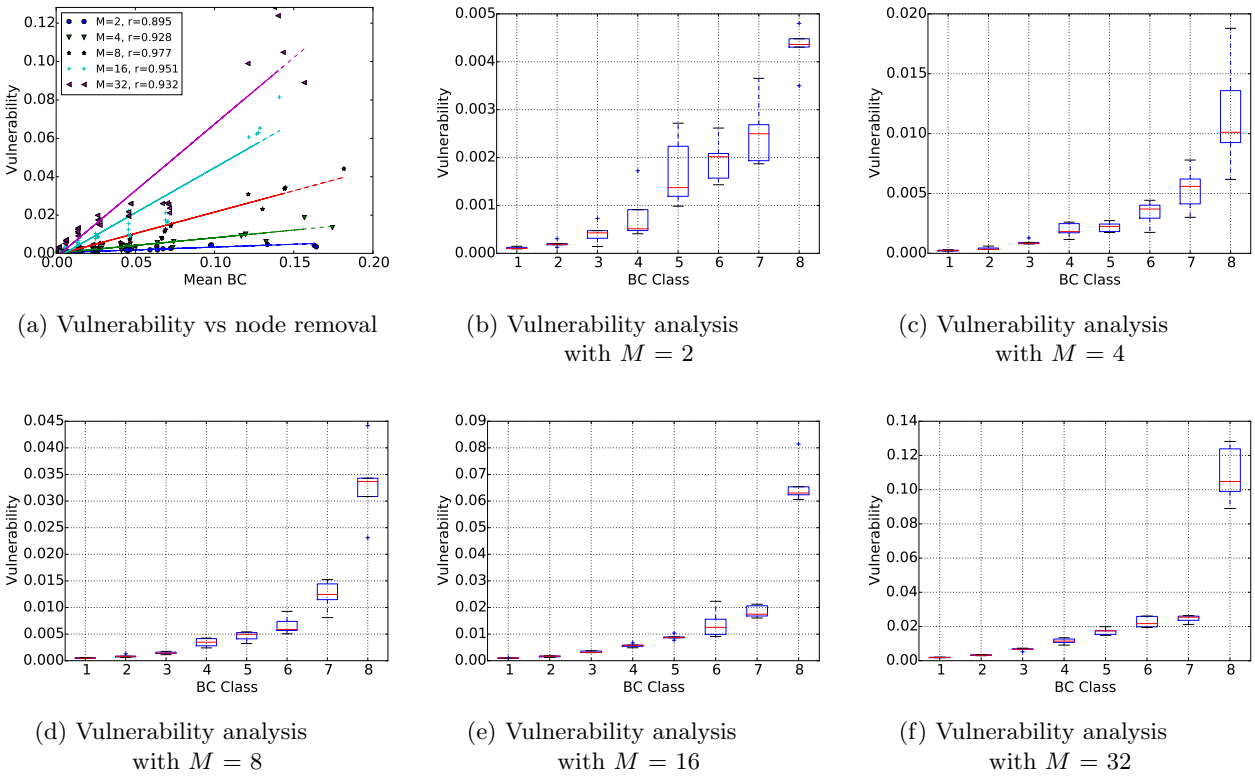
5.3. Two-level clustering fast BC

The algorithm above exhibits very good performance [22] 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 [22]) that contribute to generate clusters with a small amount of border nodes. Unfortunately, our dataset related to the Lyon road network 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 improvement 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.

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.



(a) Vulnerability vs node removal

(b) Vulnerability analysis with $M = 2$

(c) Vulnerability analysis with $M = 4$

(d) Vulnerability analysis with $M = 8$

(e) Vulnerability analysis with $M = 16$

(f) Vulnerability analysis with $M = 32$

Figure 4: Impact of M nodes removal from different BC-ranges on network vulnerability

6. 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 [11] (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.

In this section, we first present the correlation between the most critical nodes, identified by the highest BC values, and vulnerability on the G_{Lyon}. In the second part, we evaluate the performance of our proposed algorithm for fast computation of BC, when executed in different settings on the G_{Lyon} graph.

6.1. Betweenness centrality for vulnerability assessment

In this section, we verify and quantify the presence of correlation between *vulnerability* and node failures on the G_{Lyon} road traffic network.

We simulate network failures by removing M nodes from the graph and we measure the new values of the vulnerability metric (see Sec. 3) after nodes removal. As a criterion to select nodes for removal, we consider the descending order of BC (i.e., worst-case contingency). Since G_{Lyon} is characterized by many nodes with very close values of BC, to better verify the correlation between vulnerability and BC, we group the nodes of the graph in multiple ranges of BC, via Jenks natural breaks classification method [27].

The number of ranges considered in our analysis is equal to eight[5]. Hence, we perform five tests by removing M (M= 2, 4, 8, 16 and 32) nodes, selected uniformly at random from each of the ranges. Nodes are put back in the graph after each test with a specific BC range. We repeat each test several times to collect statistical information. The overall results, in terms of correlation between BC and vulnerability, are reported in Fig. 4.

Fig. 4a shows the scatter plots of the impact of M-nodes removal from the eight classes (mean BC of the M removed nodes on the X-axis) on vulnerability (Y-axis), where M is represented by different colors and markers, and a linear regression shows the relative trend. As expected, vulnerability increases sharply as mores nodes (i.e., larger M) are removed from each class. This is especially evident in those classes with higher BC values. The linear regression for each of the class shows the linear behavior with different values of slope.

Figures 4b:4f plot vulnerability for M = 2, 4, 8, 16 and 32, respectively. In each one of these micro-analyses, we observe a similar pattern by using box plots. The analysis clearly shows that removing nodes with BC in higher classes (on right side of X-axis) increases vulnerability more than removing nodes in lower BC classes.

Nodes with higher values of betweenness centrality are the ones that affect most vulnerability of a road-traffic network. 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 topology, forcing to recompute betweenness centrality for the whole network.

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

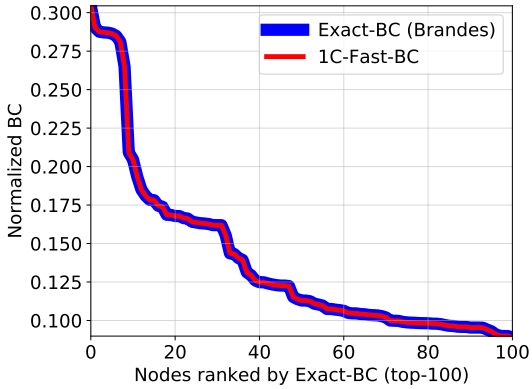
To continuously identify the most M critical nodes (M-contingency analysis), a fast algorithm to compute BC is needed. In this section, we compare the performance of *1C-Fast-BC* against that of *Exact-BC* on the G_{Lyon} graph. For the evaluation, we consider three main indexes: 1) accuracy of the approximated BC values, measured by considering both the *percentage error* on each node BC and by using the global metric of the *mean-Normalized Root-Mean-Square Error* (NRMSE)[6]; 2) accuracy of nodes' ranking, expressed as the relative number of nodes not included in the exact top-N ranking by the tested algorithm; 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 reported performance analysis is focused on the most-critical nodes of the network (e.g., top-100, top-5%, etc.), as reported in the figures.

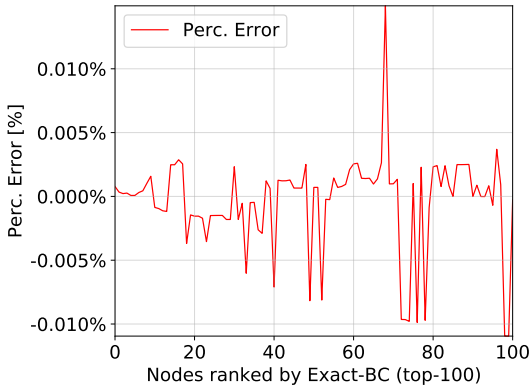
Fig. 5 shows the normalized BC values computed by the two algorithms for the top-100 nodes, ranked according to their exact value of BC. Fig. 5a highlights an almost perfect overlap of the two curves, confirmed by the extremely low percentage error in Fig. 5b, bounded in the range [-0.015%, 0.015%]. 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. In other words, the algorithm produces the exact same results (i.e., identification of the most critical nodes) reported in Fig. 1b for *Exact-BC* on the G_{Lyon} graph. However, 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 considering that *1C-Fast-BC* requires 48,960 classes (and thus pivot nodes), a relatively high number compared to the total

[5]Other numbers of ranges have been tested with similar results.

[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.



(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 5: Accuracy of the 1C-Fast-BC algorithm

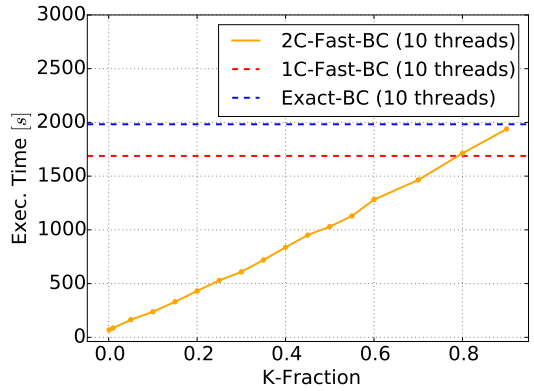
number of source nodes (75,475) used by *Exact-BC*.

1C-Fast-BC is a powerful solution to precisely compute BC, even on nodes that exhibit very low values of BC. 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 for this reason the execution time of 1C-Fast-BC is only slightly lower than the one obtained with Exact-BC.

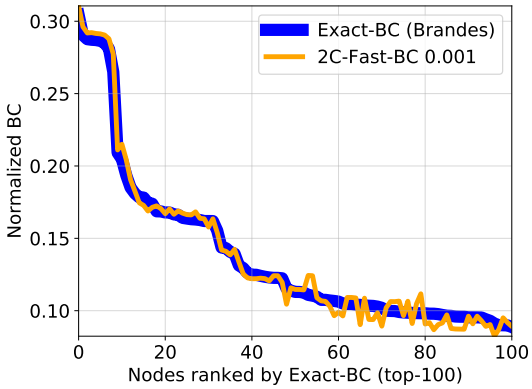
6.3. *M*-contingency analysis: 2C-Fast-BC improvements

2C-Fast-BC (see Sec. 5.3) 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.

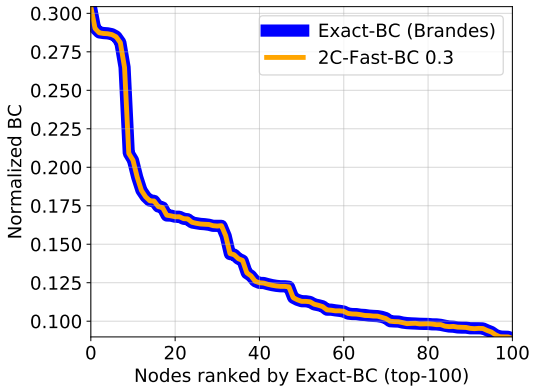
Fig. 6a shows that *2C-Fast-BC* execution time stays below *1C-Fast-BC* and *Exact-BC* ones (that do not depend on the K-fraction parameter), up to very large values of K-Fraction (i.e., 0.8). Larger values of the parameter result in higher execution times (larger than the ones of *1C-Fast-BC* for K-Fraction > 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. 6b 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. 6c. The much better quality of the approximation is confirmed by a percentage error in the range [-0.2%, 0.3%] (see Fig. 6d), a NRMSE of 0.001 and 0 missing nodes in the top-100 ranking. As in the previous case of *1C-Fast-BC*, the most critical nodes reported in Fig. 1b were correctly identified by the *2C-Fast-BC* on the G_{Lyon} graph. Most importantly, the *2C-Fast-BC* with a K-



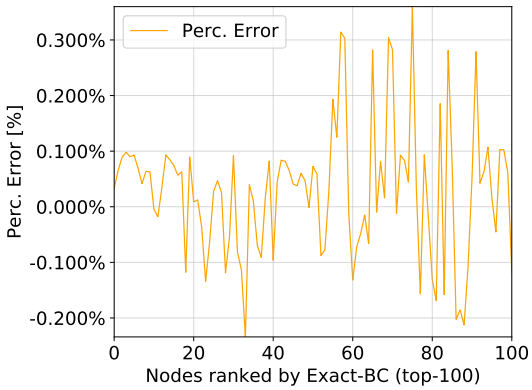
(a) Execution time of 2C-Fast-BC with different K-Fractions



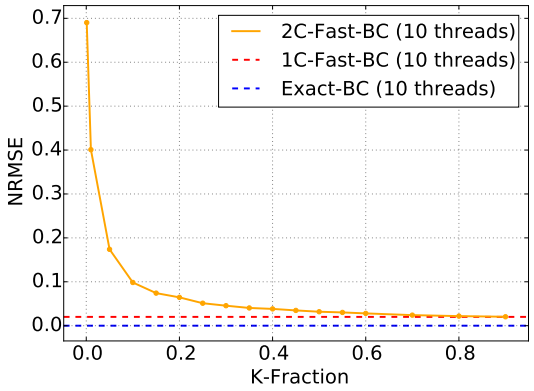
(b) BC values with K-Fraction = 0.001 for the top-100 nodes



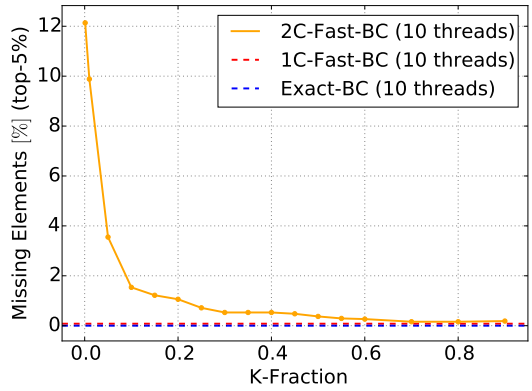
(c) BC values with K-Fraction = 0.3 for the top-100 nodes



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



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



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

Figure 6: Performance Evaluation of the 2C-Fast-BC algorithm

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 6e and 6f present the full trend for different K-Fractions of the global NRMSE and the percentage of missing elements over the top-5% nodes, respectively.

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 peculiar property of driving the computation time via the K-Fraction parameter makes this algorithm very interesting when the application domain tolerates small errors but requires quick responses for real-time analysis. 2C-Fast-BC can be exploited for any type of graph, but its improvements are relevant with non-scale-free graphs.

7. CONCLUSION

In this paper, we used betweenness centrality to perform efficient M-contingency analysis. The work has a twofold contribution. First, we confirm the importance of BC for analyzing road networks vulnerability and propose a new algorithm for fast computing BC in very-large networks. Second, we applied our algorithms for vulnerability analysis of a large, real-world road network.

The encouraging results suggest the adoption of our algorithms for implementing 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. Even though our current assumptions on network modeling may limit the effectiveness of the approach for highly dynamic networks, they do not invalidate the proposed algorithms and methods for continuous M-Contingency analysis.

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 (e.g., road length and capacity, number of lanes, etc.), as well as dynamic traffic data (e.g., demand, accidents, travel times, etc.), which represent crucial factors for determining road-network vulnerability.

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] David King and Amer Shalaby. Performance metrics and analysis of transit network resilience in Toronto. *Transportation Research Record*, 2016.

[2] Shuangming Zhao, Pengxiang Zhao, and Yunfan Cui. A network centrality measure framework for analyzing urban traffic flow: A case study of wuhan, china. *Physica A: Statistical Mechanics and its Applications*, 478:143 – 157, 2017.

[3] Yuanyuan Zhang, Xuesong Wang, Peng Zeng, and Xiaohong Chen. Centrality characteristics of road network patterns of traffic analysis zones. *Transportation Research Record: Journal of the Transportation Research Board*, 2256:16–24, 2011.

[4] Bertrand Berche, Christian von Ferber, Taras Holovatch, and Yurij Holovatch. Resilience of public transport networks against attacks. *THE EUROPEAN PHYSICAL JOURNAL B*, 71(1):125–137, 2009.

[5] Vito Latora and Massimo Marchiori. Efficient behavior of small-world networks. *Phys. Rev. Lett.*, 87(19), October 2001.

[6] Ryan Kinney, Paolo Crucitti , Reka Albert, and Vito Latora. Modeling cascading failures in the north american power grid. *The European Physical Journal B - Condensed Matter and Complex Systems*, 46(1):101–107, 2005.

[7] Christopher R. Palmer, Georgos Siganos, Michalis Faloutsos, Christos Faloutsos, and Phillip B. Gibbons. The connectivity and fault tolerance of the internet topology. *ACM SIGMOD/PODS Workshop*, 2001.

[8] Sophie Achard and Ed Bullmore. Efficiency and cost of economical brain functional networks. *PLoS Computational Biology*, 03(02), 2007.

[9] Aisan Kazerani and Stephan Winter. Can betweenness centrality explain traffic flow? *International Conference on Geographic Information Science*, 2009.

[10] Amila Jayasinghe, Kazushi Sano, and Hiroaki Nishiuchi. Explaining traffic flow patterns using centrality measures. *International Journal for Traffic & Transport Engineering*, 05(02):134–149, 2015.

[11] Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(163), 2001.

[12] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 25:35–41, 1997.

[13] Petter Holme. Congestion and centrality in traffic flow on complex networks. *Advances in Complex Systems (ACS)*, 06(02):163–176, 2003.

[14] Yaniv Altshuler, Rami Puzis, Yuval Elovici, Shlomo Bekhor, and Alex BaglioniPentland. Augmented betweenness centrality for mobility prediction in transportation networks. *Finding Patterns of Human Behaviors in Network and Mobility Data (NEMO)*, 2011.

[15] Song Gao, Yaoli Wang, Yong Gao, and Yu Liu. Understanding urban traffic-flow characteristics: a rethinking of betweenness centrality. *Environment and Planning B: Planning and Design*, 40(1):135–153, 2013.

[16] Aisan Kazerani and Stephan Winter. Can betweenness centrality explain traffic flow? *AGILE*, 2009.

[17] Amila Jayasinghe, Kazushi Sano, and Hiroaki Nishiuchi. Explaining traffic flow patterns using centrality measures. *International Journal for Traffic & Transport Engineering*, 5(2):134–149, 2015.

[18] George Leu, Hussein Abbass, and Neville Curtis. Resilience of ground transportation networks: a case study on melbourne. *33rd Australasian Transport Research Forum Conference*, 2010.

[19] Erik Jenelius and Lars-Göran Mattsson. Road network vulnerability analysis of area-covering disruptions: A grid-based approach with case study. *Transportation Research Part A: Policy and Practice*, 46(5):746–760, 2012.

[20] Miriam Baglioni, Filippo Geraci, Marco Pellegrini, and Ernesto Lastres. Fast exact computation of betweenness centrality in social networks. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, pages 450–456, 2012.

[21] Jing Yang and Yingwu Chen. Fast computing betweenness centrality with virtual nodes on large sparse networks. *PLoS One*, 2011.

[22] 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*, pages 47–54, June 2015.

[23] Nicolas Kourtellis, Gianmarco De Francisci Morales, and Francesco Bonchi. Scalable online betweenness centrality in evolving graphs. *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, 00:1580–1581, 2016.

[24] Kouzou Ohara, Kazumi Saito, Masahiro Kimura, and Hiroshi Motoda. *Accelerating Computation of Distance Based Centrality Measures for Spatial Networks.*

[25] L.F. Costa, F.A. Rodrigues, G. Travieso, and P.R.V. Boas. Characterization of complex networks: A survey of measurements. *Advances in Physics*, 56(1):167–242, 2007.

[26] Mark EJ Newman. Analysis of weighted networks. *Physical Review E*, 70(5):056131, 2004.

[27] George F. Jenks. The data model concept in statistical mapping. *International Yearbook of Cartography*, 7:186–190, 1967.