Fast Approximated Betweenness Centrality of Directed and Weighted Graphs

Angelo Furno[1], Nour-Eddin El Faouzi[1], Rajesh Sharma[2], and Eugenio Zimeo[3]

[1] Univ. Lyon, IFSTTAR, ENTPE, LICIT UMR_T9401, Lyon, France,
angelo.furno@ifsttar.fr, nour-eddin.elfaouzi@ifsttar.fr
[2] University of Tartu, Tartu, Estonia,
rajesh.sharma@ut.ee
[3] University of Sannio, Benevento, Italy,
zimeo@unisannio.it

Abstract. Node betweenness centrality is a reference metric to identify the most critical spots of a network. However, its exact computation exhibits already high (time) complexity on unweighted, undirected graphs. In some domains such as transportation, weighted and directed graphs can provide more realistic modeling, but at the cost of an additional computation burden that limits the adoption of betweenness centrality for real-time monitoring of large networks. As largely demonstrated in previous work, approximated approaches represent a viable solution for continuous monitoring of the most critical nodes of large networks, when the knowledge of the exact values is not necessary for all the nodes.
This paper presents a fast algorithm for approximated computation of betweenness centrality for weighted and directed graphs. It is a substantial extension of our previous work which focused only on unweighted and undirected networks. Similarly to that, it is based on the identification of pivot nodes that equally contribute to betweenness centrality values of the other nodes of the network. The pivots are discovered via a cluster-based approach that permits to identify the nodes that have the same properties with reference to clusters' border nodes. The results prove that our algorithm exhibits significantly lower execution time and a bounded and tolerable approximation with respect to state-of-the-art approaches for exact computation when applied to very large, weighted and directed graphs.

Keywords: Betweenness Centrality, Directed Weighted Graphs, Fast Computation, Large Scale Networks, Real-time Monitoring

1 Introduction

Real-time monitoring of large networks for detecting and predicting critical spots is a compelling challenge due to the high complexity of computing robustness metrics. Graph models have proven to be a valid approach to study topological bottlenecks of many kinds of networks via centrality indicators such as betweenness centrality. However, while undirected and unweighted graphs represent a basic abstraction of these networks, weighted graphs can better capture edge diversity, especially in some application domains, such as transportation.

Betweenness centrality (BC) [14] is a very popular network metric to characterize nodes that are most traversed by shortest paths connecting pairs of other nodes

of a graph. It has been widely adopted to study many application domains but the high computation time limits its adoption for real-time monitoring of very large networks. The fastest algorithm to compute the exact value of betweenness centrality has been proposed by Brandes [9]. Given a graph $G(V,E)$, it exhibits $O(n+m)$ space and $O(nm)$ time complexities for unweighted graphs and $O(nm+n^2log(n))$ 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 based on breadth-first graph search or on Dijkstra algorithm for unweighted and weighted graphs, respectively. Each exploration is computed with a complexity $O(m)$ and $O(m+n\cdot log(n))$, for unweighted and weighted graphs respectively. This is typically good for sparse graphs (where $m \ll n^2$) but not sufficient for real-time monitoring of very large networks. A faster approach, useful for some kinds of applications, allows achieving lower computation time by calculating approximated BC values. These strategies try to penalize some shortest paths or to exploit topological properties to identify only $k \ll n$ *pivot nodes* as sources for the computation of SSSP. While several attempts exist for computing approximated values of betweenness centrality of unweighted graphs, a few of them focus on directed and weighted graphs, which better model several real-world networks.

In this paper, we propose an adaptable algorithm for computing approximated values of BC of directed and weighted networks. The performance of the algorithm can be tuned based on the amount of error we can tolerate on the approximation. It is an extension of the algorithm originally proposed in [15, 16] for undirected and unweighted graphs to directed and weighted ones. The algorithm, similarly to the ones proposed in [10] and [18], exploits topological characteristics of the graph in order to classify nodes for their selection as pivots. Differently from the papers above, it exploits clustering to identify reference nodes (clusters' border nodes) to perform a topological analysis. The solution can calculate an almost exact value of betweenness centrality for several nodes, i.e., the most critical ones, while keeping a good approximation for the others, with an execution time that strictly depends on the number of retained pivots.

Our algorithm is validated by using real-world transportation networks. By assuming that edge weights represent the traveled distance or the free-flow-travel time, betweenness centrality provides indications about redistributions of the traffic flow (or potential congestion), if we make the assumption that people prefer the shortest (fastest) paths to reach their destinations. Our analysis is based on a real large-scale road-network and Global Positioning System (GPS) taxi traces. We leverage geo-referenced, time-stamped taxi trips to reconstruct per-link median travel time/speed with a hourly frequency. Such traffic indicator is used as the edge weight in the modeling graph. However, it is not possible to estimate such traffic indicator for all network edges: some portions of the road networks are never traversed by taxis in the observed time period. Therefore, for such regions of the network, we used an interpolation technique based on supervised machine learning. The performance analysis shows that the proposed algorithm is a valid solution for real-time monitoring of large-scale graphs.

The rest of the paper is organized as follows. Sec. 2 presents related work, while Sec. 3 describes our fast BC algorithm. In Sec. 4, we evaluate our approach on a large-scale transportation dataset. Sec. 5 concludes the work and highlights future directions.

2 Related Work

Betweenness centrality, originally proposed in [13] for undirected graphs was extended to directed graph in [31]. Brandes in [9] proposed a faster algorithm which also works for weighted networks. The idea was based on the adoption of SSSP algorithm based on breadth-first graph search or on Dijkstra algorithm, for unweighted and weighted graphs respectively. Several approaches, aiming to evaluate exact or approximated solutions, have been developed to further reduce the computation time. The proposed solutions can be classified according to three main categories: *(i)* exploiting and increasing parallelism, *(ii)* estimating BC values through a partial exploration of the graph, *(iii)* calculating BC values of fraction of nodes in dynamically evolving graphs.

Parallel approaches: In [4], the first parallel implementation for computing betweenness centrality is presented, which handles weighted graphs as well. 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 [23] by removing the need for locking in the dependency accumulation stage of Brandes' algorithm through the adoption of a successor list instead of a predecessor list for each node.

Incremental approaches: A different set 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. For example, in [21], researchers proposed an efficient approach that reduces the search space by focusing only on the vertices whose betweenness centralities get changed as a consequence of an update in the graph. Similarly, in [19], by using the hypergraph sketch data structure, i.e., a weighted hyper-edge representation of shortest paths, computation time was reduced. Based on sampling based techniques [28], Bergamini et. al. first proposed a semi-dynamic [6] and, later, a fully dynamic approach [5] for dynamic networks (both weighted and unweighted), capable of performing in-memory computation with millions of edges. Recently, an efficient algorithm for incremental BC computation [20] has been proposed. The algorithm exhibits good performance when the graph changes for a very limited number of nodes. Conversely, the high speedup drastically reduces when the graph is subject to significant changes of its topology.

Approximated approaches: The third research trend focuses on 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 [32], the authors only consider paths up to fixed length k. Brandes and Pich [10] 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 accuracy levels comparable to other heuristics. However, this approach overestimates the BC of unimportant nodes that are near a pivot. To overcome this problem, various solutions have been proposed, e.g., a generalization framework for betweenness approximation has been proposed in [18]. The idea is to scale BC values in order to reduce them with reference to nodes close to pivots. In another paper, a solution to reduce the pivots for nodes with high centrality is proposed via adaptive sampling techniques [3]. A recent work [27] based on approximation shows large fluctuations of accuracy over the top-100 nodes

on a scale-free graph. A random, shortest path based [26] approximation approach was presented in [28].

For directed and unweighted networks an approach is presented in [8], where, similarly to [11], authors precompute reachable vertices for all the graph nodes. However, at the best of our knowledge, there is a scarcity of contributions focusing on both weighted and directed networks. BC has been proven to be a proxy for traffic volume in [2], but this paper does not address the problem of performing fast computation of BC in large dynamic networks, which is the main objective and contribution of this paper.

3 Fast BC computation of weighted and directed graphs

In this section, we present the *W2C-FastBC* algorithm, the weighted and directed version of the one previously proposed in [15,16]. It allows reducing BC computation time of weighted and directed graphs in a parametric way, i.e., by acting on the accuracy of BC values. The algorithm is based on the Brandes' one for weighted graphs and on the heuristic proposed in [30] for identifying graph pivots. As in our previous work, we exploit a fast clustering algorithm based on modularity for identifying graph communities and their related border nodes. Specifically, we used a distributed implementation [29] of Louvain method for weighted and directed graphs [24]. In the following subsections, we briefly introduce the adopted notation, the Brandes' algorithm and discuss modularity for weighted graphs.

3.1 Notation

We assume the following definition throughout the paper. Let $G(V, E, T, W, f(E, T))$ be a dynamic, weighted, directed graph, where V denotes the set of nodes and $E \subseteq V \times V$ the one of edges. $N = |V|$ denotes the number of nodes in the graph. W represents the set of weights and T the set of time units. For instance, for very large networks, T may represent hours of the day. We highlight that the length of the considered time unit (e.g., 1 hour) represents the period of observations before a new computation of BC is launched, and translates therefore into a time constraint for computing betweenness centrality. The function $f : E \times T \rightarrow W$ maps each edge $e_{ij} \in E$ at time slot $t \in T$ to a weight $w \in W$. We denote as $\hat{G}(V, E, \hat{W})$ a directed and weighted instance of the dynamic graph G related to a specific time slot $\hat{t}$ and therefore associated to a subset of weights $\hat{W} \subseteq W$. The algorithms reported in the following are related to a specific instance $\hat{G}$ of the dynamic graph G, i.e., BC computation is iteratively performed (in a quasi-real time fashion) at the beginning of time slot $\hat{t}+1$ on the instance of the dynamic graph related to time slot $\hat{t}$.

A path $p(v_i, v_j)$, between two nodes v_i and v_j of $\hat{G}$, consists of a set of nodes and edges that connect these two nodes. 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 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 length, among all the paths connecting the two nodes. Multiple shortest paths may exist between the same pair of nodes, i.e., multiple paths having the same length. The 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 $\sigma_{v_i v_j}$ as the number of shortest paths between v_i and v_j, while $\sigma_{v_i v_j}(v_k)$ represents the number of shortest paths from v_i to v_j that cross node v_k.

3.2 Brandes' algorithm

Given a graph $\hat{G}$, the *pair-dependency* of a *source* node s on an another node v for a *destination* t of the graph is 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 $R = \{w : v \in P_s(w)\}$ 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; then, v is a predecessor also in any other shortest path starting from s and passing through a different $w \in R$ [9]. 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 as:

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

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. In the second phase, it performs dependency accumulation by backtracking along the discovered shortest paths. During these two phases, the algorithm maintains four data structures for each node found on the way: a predecessor list $P_s(v)$, the distance $d_s(v)$ from the source, the number of shortest paths from the source $\sigma_{st}(v)$ and the dependency accumulation when backtracking at the end of the search.

3.3 Weighted modularity and Louvain method

Modularity is a metric, defined as a value between -1 and 1, to measure how tightly the nodes are attached with each other in the network. It was introduced to identify communities in undirected and unweighted (or weighted) networks [25]. Given a graph $\hat{G}$, partitioned into a set of communities $C = \{c_1, c_2, ..., c_D\}$, formally, modularity [22] of graph $\hat{G}$ is defined as follows:

$$Q = \frac{1}{m} \sum_{i,j \in V} \left[A_{ij} - \frac{k_i^{in} k_j^{out}}{m} \right] \delta(c_i, c_j) \tag{3}$$

where δ is 1 if $c_i = c_j$ (nodes i and j belong to the same community) or 0 otherwise, m is the sum of all of the edge weights in the graph, k_i^{in}, k_j^{out} are the sum of the weights of the edges entering node i and the edges exiting node j, respectively; A_{ij} is 0 if nodes n_i and n_j are not connected. In case the nodes n_i and n_j are connected then A_{ij} is $w_{i,j}$, that is the weight of the edge connecting nodes i and j.

We exploit modularity for clustering weighted directed graphs with the Louvain method [7, 12]. The algorithm initially searches for *small* communities. Then, it creates a new graph whose nodes are the communities identified in the previous step. These two steps are iteratively run until there is no modularity gain derived by aggregating clusters in larger communities. In our implementation, the weights used to compute weighted modularity are assumed as in the notion of closeness (nodes are tighter if they have lower distance or travel time), i.e. "smaller is tighter". This choice is motivated by the fact that we want to reduce the number of border nodes for each cluster. Therefore, we generate communities whose nodes are highly locally inter-connected with short (or fast to travel) local paths. Conversely, when computing shortest paths in SSSPs, weights are assumed as in the notion of length (or travel time), i.e., "higher is farther". We use a distributed variant of the Louvain algorithm for weighted and directed graphs [24, 29]: all vertices select a new community simultaneously, updating the local view of the graph after each change. Even though some choices will not maximize modularity, after multiple iterations, communities will typically converge thus producing a final result relatively close to the sequential version of the algorithm.

3.4 W2C-FastBC

Given a graph $\hat{G}$, we split it into a set of clusters (i.e., $\mathbf{C}$) by using the Louvain (Alg. 1, line 2) method for weighted graphs [29][1]. The main result of clustering is the identification of *border nodes* (an array for each cluster). A border node is a node having at least one neighbor node in a different cluster (line 4). Then, a parallel execution of Brandes' algorithm (based on Dijkstra[2]) is performed inside each cluster to compute the *local BC* (lines 6-11). This computation generates the partial inner-cluster contribution to the BC of each node and additional information, such as the weighted 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 (equivalence class, see [15, 30] for more details). 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 Dijkstra's algorithm (line 19). This node is called class *pivot*. The partial dependency score calculated for the pivot is then multiplied by the cardinality of the pivot class (line 20). This method avoids re-applying Dijkstra' 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.

[1] The implementation leverages a Scala parallel solution partially based on the Distributed Graph Analytics (DGA) by Sotera: https://github.com/Sotera/distributed-graph-analytics.

[2] The adoption of Dijkstra algorithm instead of breadth first search represents a main variant of our FastBC algorithm proposed in this paper.

Algorithm 1 W2C-Fast-BC: pseudo-code of the main function

```
1: function W2C-FastBC(Ĝ, C, kFrac)
2:   C ← weightedLouvainClustering(Ĝ)
3:   map i ← 1, |C| do
4:     bordernodes_i ← findBorderNodes(Ĝ, C_i)
5:   end map
6:   map i ← 1, |V| do
7:     localδ_i ← computeLocalδ(i, C, bordernodes)
8:   end map
9:   reduce i ← (1, |V|), localδ_s, localδ_z, i = j do
10:    localBC_i ← localδ_s(i) + localδ_z(j)
11:  end reduce
12:  map i ← 1, |C| do
13:    superClasses_i ← WkMeansClustering(C_i, classes_i, kFrac)
14:  end map
15:  map i ← 1, |superClasses| do
16:    P_i ← selectPivotOf(superClasses_i, localBC)
17:  end map
18:  map i ← 1, |P| do
19:    δ_i ← computeδFrom(P_i)
20:    δ_i ← (δ_i − localBC) · |superClasses_i|
21:  end map
22:  reduce i ← (1, |V|), δ_s, δ_z, i = j do
23:    BC_i ← δ_s(i) + δ_z(j)
24:  end reduce
25:  for i ← 1, |V| do
26:    BC_i ← BC_i + localBC_i
27:  end for
28:  return BC
29: end function
```

The final value of BC is obtained for each node by summing up all partial contributions (produced by the reduce operation, lines 22:24) with local BC values (lines 25:27). To further reduce the computation time, we extended the concept of *class* by introducing *super classes* through an additional clustering operation inside each initial Louvain-derived cluster (line 13). A super class is a group of classes, belonging to the same Louvain cluster and obtained by clustering (via K-means) the vectors generated by considering, for each node, the normalized distances from the Louvain cluster's bordernodes and the amount of shortest paths towards them. To perform class grouping, we exploit a parallel *K*-means algorithm by using a different *K* for each initial Louvain cluster. *K* is defined as a fraction (*K-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 our previous work [15, 16, 30].

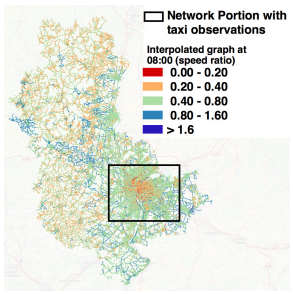
4 Evaluation: dynamic analysis of a real-world road network

We implemented our algorithms using *Scala* with the *Apache-Spark* framework. Spark was configured to work in the standalone cluster mode on two Intel Xeon E5 2640 2.4 GHz multi-core machines, each equipped with 56 virtual cores and 128 GB of DDR4 RAM. All algorithms for BC computation leverage 10 cores by spawning the map-reduce tasks on two Spark workers, each equipped with 5 executors.

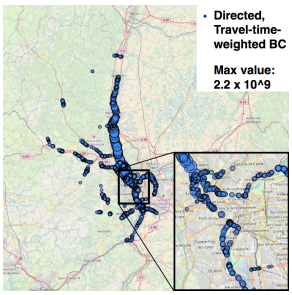
To evaluate our W2C-Fast-BC algorithm, we leverage a large-scale transportation graph, namely **Rhone-ROADS**, corresponding to the entire road network of the Rhone-Alpes region, France [17]. The graph includes the agglomeration of Lyon and its surroundings and has a geographical extent of approximately 3,300 Km^2. The network is directed and unweighted, with 117,605 nodes and 248,337 edges. We transformed the Rhone-ROADS graph into a dynamic weighted graph by relying on an additional dataset, namely **Rhone-TAXIS**, which reports on anonymized GPS traces of taxis active in the Rhone-Alpes region. Rhone-TAXIS has been collected by the French operator Radio Taxi via a fleet of approximately 400 taxis during 2011-2012. Geo-referenced taxi trips are collected according to a variable sampling interval (between 10 and 60 seconds), with a global average of 800,000 measurements per day. The generic sample of the Rhone-TAXIS dataset, i.e., an *elementary taxi trip*, includes the time-stamped start and arrival GPS positions of a small segment traveled by the associated taxi identifier. These measures permit to roughly estimate the traveled distance and the instant speed of the taxi moving along a given road segment. In order to improve the quality of the Rhone-TAXIS dataset and properly compute the edge weights, we have filtered out elementary trips with unrealistic speeds (i.e., higher than 130 Km/h).

We map-matched all the elementary trips of the Rhone-TAXIS dataset and computed hourly median speeds for the edges of the Rhone-ROADS graph, thus generating a (discrete) dynamic weighted graph. More specifically, we considered 24 hourly time slots for a typical day, and computed the weighted graph instance corresponding to each time slot t. To that purpose, we retain only the edges with non-null value of the median speed during t, as estimated from the map-matched taxi trips related to the same edge and time slot t. Thus, we calculate the weight at time slot t of each edge (which corresponds to a road link with known length) as the estimated travel time to cross the corresponding road segment, i.e., the ratio of the length of the link to the median speed estimated on that link. By iterating this process for the different time slots, we obtain the final dynamic weighted graph (i.e., our graph G).

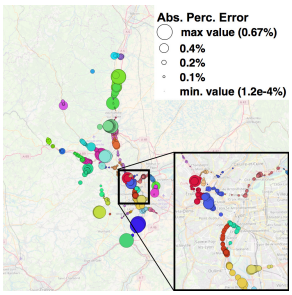
The W2C-Fast-BC and Brandes algorithms are applied iteratively to each hourly instance of the dynamic graph (i.e., a snapshot $\hat{G}$). This is conceptually equivalent to an on-line operational situation, where the graph naturally emerges from sensor-collected data used to continuously compute up-to-date traffic information on each edge with hourly periodicity. It is worth noting that, given the relatively small size of the observed taxi fleet and the circadian rhythm characterizing human mobility, snapshots of the dynamic graph related to rush hours (e.g., 7:00-09:00 and 17:00-19:00) have a much smaller size with respect to the original Rhone-ROADS graph, i.e., approximately 30,000 nodes and 60,000 edges (see the framed portion of the graph in Fig. 1a). Such size further reduces for graph snapshots related to non-rush hours. Indeed, most of the observed elementary trips are condensed within the city center of Lyon, with only few



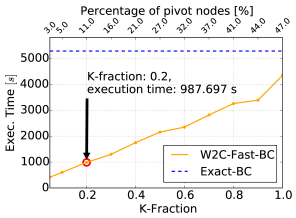
(a) KNR-interpolated graph at 08:00



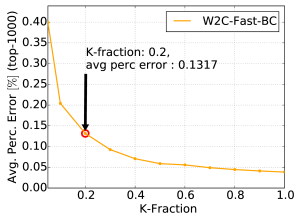
(b) Top-1000 nodes' BC values at 08:00



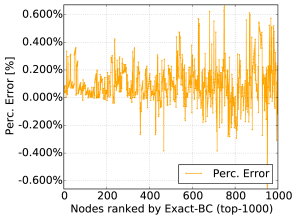
(c) Spatial distribution of the error at 08:00 with K-fraction = 0.2 (top-1000 BC)



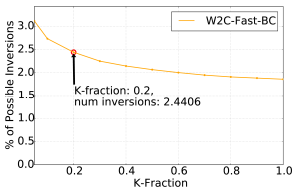
(d) Execution time of W2C-Fast-BC vs Brandes-BC at 08:00



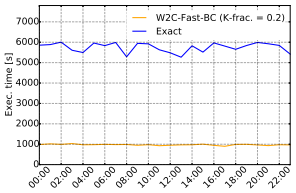
(e) W2C-Fast-BC average perc. error (top 1,000 nodes) at 08:00



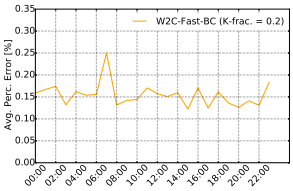
(f) Top-1000 BC perc. error at 08:00



(g) Percentage of possible inversions (all nodes)



(h) Dynamic Graph: execution time



(i) Dynamic Graph: average percentage error (top-1000)

Fig. 1. The interpolated dynamic taxi graph: median-speed-to-max-speed ratios, top-1000 BC nodes and performance evaluation of W2C-Fast-BC at different hours of the day.

observations recorded in the outskirts and within rural areas as well as during night time. However, since the goal of the paper is to prove the efficiency of our solutions with respect to very-large scale weighted networks, we have decided to increase the scale of the dynamic graph by means of a spatial interpolation technique.

To obtain a dynamic, realistic, weighted network, larger than the one directly observed from taxi trips, we leveraged an interpolation technique that we call *KNR-interpolation*[3]. The technique allows estimating the hourly value of median speed (and thus the median

[3]KNR-interpolation is based on K-nearest-neighbor regression [1], a non-parametric supervised machine-learning technique. Each edge is modeled as a data point with multiple topological features. The median speed at time slot t, available for some edges (labeled instances) and missing for other ones (unlabeled instances), represents the target interpolated feature.

travel-time weight) for those edges of the original Rhone-ROADS network with no available observation from taxi trips at time slot t. Fig. 1a graphically shows the KNR-interpolated snapshot at 08:00 of a typical working day. Fig. 1a also presents speed-ratios (i.e., median speed divided by road speed limit) either estimated via taxi traces (for the framed portion of the graph) or via the KNR interpolation technique. Red and orange colors indicate highly-congested situations on the edge, i.e., lower values of the speed ratio, while greens and blues indicate a smooth, non-congested situation at time t. The resulting graph has approximately the same size of the Rhone-ROADS network.

The values of BC for the top-1000 nodes are reported in Fig. 1b (nodes with larger circles have higher BC) for the snapshot related to 08:00. As it can be observed from Fig. 1d, the exact algorithm for computing BC on the weighted graph requires a computation time of more than one hour, therefore unable to complete within the duration of the time slot. Remarkably, our W2C-Fast-BC computes in only 987 seconds (i.e., approximately 15 minutes) when using a K-fraction equal to 0.2, showing an average percentage error of 0.13% (Fig. 1e) and a maximum percentage error of $\pm 0.7\%$ (Fig. 1f) over the top-1000 BC nodes. The number of clusters obtained via the Louvain method is 127, while the number of classes, as retrieved by the K-means algorithm, corresponds to 12494 (which also represents the number of pivot) out of 117605 nodes. The spatial distribution of the absolute percentage error is reported in Fig. 1c, with a node size proportional to the error, and a different color to represent the cluster each node belongs to. The figure highlights a scattered distribution of the percentage error over the graph. Finally, we also report in Fig. 1g the percentage of inversions against the maximum number of possible inversions [16], which clearly highlights the capability of our solution to preserve a good ranking (i.e., low percentage of inversions) of all the network nodes in terms of their BC values, even when using low values of the K-fraction parameter.

Similar results have been observed over the whole dynamic graph (i.e., the 24 hourly time slots, as reported in Fig. 1h and Fig. 1i), thus proving the adequacy of our solution for quasi real-time monitoring of dynamic, directed, weighted road-networks.

5 Conclusion

We presented an approximated betweenness centrality computation method for weighted and directed graphs. By exploiting representative pivot nodes, identified through the definition of a class of equivalence, the proposed algorithm is able to find the most critical nodes in a directed and weighted graph with a significant speedup and a negligible error if compared to the exact Brandes' algorithm. The algorithm has been evaluated on a real transportation network dataset, where dynamic weights were derived from the analysis of GPS-taxi data. The results reported in the paper show that directed and weighted graphs provide further information useful for a more realistic interpretation of high BC values. Moreover, the dynamic analysis exhibits continuously changing values of BC that are useful for predicting possible critical spots. Therefore, the algorithm represents an important milestone towards the objective of defining a complete framework for monitoring road networks and predicting traffic flows.

In the future, we aim to leverage additional solutions to further reduce computation time and error by exploiting, for instance, the hierarchical information produced

by the Louvain community detection algorithm. Moreover, in relation to the transportation case study, we will consider additional information (e.g., dynamic traffic volumes) in order to exploit dynamically computed BC values as effective predictors of network congestion. Finally, we aim to generalize the study of the performance of the algorithm, by means of a thorough evaluation with respect to other approximated solutions, by using different kinds of network (e.g., small-world, random networks) related to different application domains (e.g., social networks, brain networks, etc.).

Acknowledgment

This work has been supported by the French research project PROMENADE (grant number ANR-18-CE22-0008), the H2020 framework project SoBigData, grant number 654024, and the GAUSS project (MIUR, PRIN 2015, Contract 2015KWREMX).

References

1. Altman, N.S.: An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician **46**(3), 175–185 (1992)
2. Altshuler, Y., Puzis, R., Elovici, Y., Bekhor, S., Pentland, A.S.: Augmented betweenness centrality for mobility prediction in transportation networks. In: International Workshop on Finding Patterns of Human Behaviors in Network and Mobility Data (NEMO), (2011).
3. Bader, D.A., Kintali, S., Madduri, K., Mihail, M.: Approximating betweenness centrality. In: Proceedings of the 5th International Conference on Algorithms and Models for the Web-graph, WAW'07, pp. 124–137. Springer-Verlag, Berlin, Heidelberg (2007)
4. Bader, D.A., Madduri, K.: Parallel algorithms for evaluating centrality indices in real-world networks. In: Parallel Processing, 2006. ICPP 2006. International Conference on, pp. 539–550. IEEE (2006)
5. Bergamini, E., Meyerhenke, H.: Approximating betweenness centrality in fully dynamic networks. Internet Mathematics **12**(5), pp. 281–314 (2016).
6. Bergamini, E., Meyerhenke, H., Staudt, C.L.: Approximating betweenness centrality in large evolving networks. In: 17th Workshop on Algorithm Engineering & Experiments, ALENEX '15, pp. 133–146. SIAM, Philadelphia, PA, USA (2015)
7. Blondel, V.D., Guillaume, J.L., Lambiotte, R., Lefebvre, E.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment **P10008**, (2008)
8. Borassi, M., Natale, E.: KADABRA is an adaptive algorithm for betweenness via random approximation. In: ESA, *LIPIcs*, vol. 57, pp. 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2016)
9. Brandes, U.: A faster algorithm for betweenness centrality. Journal of Mathematical Sociology **25**(163) (2001)
10. Brandes, U., Pich, C.: Centrality estimation in large networks. International Journal of Bifurcation and Chaos **17**(07), pp. 2303–2318 (2007)
11. Chehreghani, M.H., Bifet, A., Abdessalem, T.: Efficient exact and approximate algorithms for computing betweenness centrality in directed graphs. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. Springer, Cham, p. 752–764 (2018).
12. Dugué, N., Perez., A.: Directed louvain: maximizing modularity in directed networks. In: PhD Thesis, Université d'Orléans (2015)
13. Freeman, L.C.: Centrality in social networks conceptual clarification. Social Networks **1**(3), pp. 215–239 (1978)

14. Freeman, L.C.: A set of measures of centrality based on betweenness. Sociometry **25**, pp. 35–41 (1997)
15. Furno, A., El Faouzi, N.E., Sharma, R., Zimeo, E.: Reducing pivots of approximated betweenness computation by hierarchically clustering complex networks. In: International Conference on Complex Networks and their Applications, pp. 65–77. Springer (2017)
16. Furno, A., El Faouzi, N.E., Sharma, R., Zimeo, E.: Two-level clustering fast betweenness centrality computation for requirement-driven approximation. In: 2017 IEEE International Conference on Big Data (Big Data), pp. 1289–1294. IEEE (2017)
17. Furno, A., El Faouzi, N.E., Sharma, R., Cammarota, V., Zimeo, E.: A Graph-Based Framework for Real-Time Vulnerability Assessment of Road Networks. In: 2018 IEEE International Conference on Smart Computing (SMARTCOMP), pp. 234–241. IEEE (2018)
18. Geisberger, R., Sanders, P., Schultes, D.: Better approximation of betweenness centrality. In: Proceedings of the Meeting on Algorithm Engineering & Expermiments, pp. 90–100. Society for Industrial and Applied Mathematics (2008)
19. Hayashi, T., Akiba, T., Yoshida, Y.: Fully dynamic betweenness centrality maintenance on massive networks. Proc. VLDB Endow. **9**(2), pp. 48–59 (2015).
20. Kourtellis, N., Morales, G.D.F., Bonchi, F.: Scalable online betweenness centrality in evolving graphs. In 2016 IEEE 32nd International Conference on Data Engineering (ICDE) pp. 1580–1581 (2016)
21. Lee, M.J., Lee, J., Park, J.Y., Choi, R.H., Chung, C.W.: Qube: A quick algorithm for updating betweenness centrality. In: Proceedings of the 21st International Conference on World Wide Web, WWW '12, pp. 351–360. ACM, New York, NY, USA (2012)
22. Leicht, E.A., Newman, M.E.J.: Community Structure in Directed Networks. Physical Review Letters **100**(11), 118703 (2008).
23. Madduri, K., Ediger, D., Jiang, K., Bader, D.A., Chavarria-Miranda, D.: A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets. In: 2009 IEEE Parallel & Distributed Processing International Symposium on, pp. 1–8. IEEE (2009)
24. Newman, M.E.: Analysis of weighted networks. Physical Review E **70**(5), 056131 (2004)
25. Newman, M.E.J., Girvan, M.: Finding and evaluating community structure in networks. Phys. Rev. E **69**, 026113 (2004)
26. Newman, M.J.: A measure of betweenness centrality based on random walks. Social Networks **27**(1), pp. 39 – 54 (2005).
27. Ohara, K., Saito, K., Kimura, M., Motoda, H.: Accelerating Computation of Distance Based Centrality Measures for Spatial Networks. In: International Conference on Discovery Science. Springer, Cham, (2016).
28. Riondato, M., Kornaropoulos, E.M.: Fast approximation of betweenness centrality through sampling. Data Mining and Knowledge Discovery **30**(2), pp. 438–475 (2016)
29. Sotera: dga-graphx: Graphx algorithms. online URL https://github.com/Sotera/spark-distributed-louvain-modularity. Last checked on Oct. 2018.
30. Suppa, P., Zimeo, E.: A clustered approach for fast computation of betweenness centrality in social networks. In: 2015 IEEE International Congress on Big Data, pp. 47–54 (2015)
31. White, D.R., Borgatti, S.P.: Betweenness centrality measures for directed graphs. Social Networks **16**(4), pp. 335 – 346 (1994)
32. White, S., Smyth, P.: Algorithms for estimating relative importance in networks. In: Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 266–275. ACM (2003)