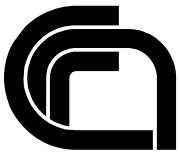


Consiglio Nazionale delle Ricerche

Fast exact computation of betweenness centrality in social networks

M. Baglioni, F. Geraci, M. Pellegrini, E. Lastres

IIT TR-10/2011

Technical report

Giugno 2011

Istituto di Informatica e Telematica

Fast exact computation of betweenness centrality in social networks

Miriam Baglioni*, Filippo Geraci*, Marco Pellegrini*and Ernesto Lastres†
*IIT - CNR, Pisa, Italy
†Sistemi Territoriali, Navacchio (Pisa), Italy

***Abstract*—Social networks have demonstrated in the last few years to be a powerful and flexible concept useful to represent and analyze data. They borrow some basic concepts from sociology in order to model how people (or data items) establish relationships with each other. The study of these relationships can provide a deeper understanding of many emergent global phenomena. The amount of data available in the form of social networks data is growing by the day, and this poses many computational challenging problems for their analysis. In fact many analysis tools suitable to analyze small to medium sized networks are inefficient for large social networks.**

In this paper we present a novel approach for the computation of the *betweenness centrality*, which speeds up considerably Brandes' algorithm, in the context of social networking. Our algorithm exploits the natural sparsity of the data to algebraically (and efficiently) determine the betweenness of those nodes organized as trees embedded in the social network. Moreover, for the residual network, which is often of much smaller size we modify the Brandes' algorithm so that we can remove the nodes already processed and perform the computation of the shortest paths only for the remaining nodes.

We tested our algorithm using a set of 18 real sparse large social networks provided by *Sistemi Territoriali* which is an Italian ICT company specialized in Business Intelligence. Our tests show that our algorithm consistently runs more than an order of magnitude faster than the Brandes' procedure on such sparse networks.

***Keywords*-Betweenness centrality, social network analisys**

I. Introduction

Social networks are a powerful instrument to represent a large set of heterogeneous objects and the relationships among them. They are successfully adopted in a vast range of applications from marketing to bioinformatics. According to the paradigm of social networking, to each object is associated a node of the network and the edges between pairs of nodes represent the relationship between them. Social networks are naturally represented as graphs, consequently graph theory plays an important role in social network analysis.

Among the analysis tools, centrality indices are often used to score (and rank) the nodes (or the edges) of the network to reflect their centrality position. The intuitive idea behind this class of indices is that a central node is likely to be involved in many processes of the network, thus its importance increases. According to what we mean with the word "important", different definitions of centrality are possible [1]. For example degree centrality highlights nodes with a higher number of connections, closeness centrality highlights nodes easily reachable from other nodes, and eigenvector centrality highlights nodes connected with authoritative nodes. A complete compendium of many centrality definitions, problems and measures can be found in [2]. Betweenness [3] is one of the most broadly used centrality indices. The betweenness of a node is defined as the sum, for each pair of nodes (s, t) in the network, of the ratio between the number of shortest (aka geodesic) paths from s to t passing through v and the total number of shortest paths from s to t. The main assumption of this index is that the information flows in the network following only shortest paths. Despite this assumption that could be considered as a restrictive condition, betweenness finds a vast range of applications such as: lethality in biological networks [4] and bibliometry [5], and is quite useful in applications.

The computation of the Betweenness centrality index is demanding because, for a given node v, all the shortest paths between each couple of nodes passing through v have to be counted (even if it is not necessary to explicitly enumerate them). This means that, in general, for fairly large networks the computation of this index becomes impractical. In the last decade a large number of social networks have been identified and their size is consistently increasing over time. For example, social media like Facebook have reached size in the order of millions of nodes, and such networks are essentially connected (or have a single giant connected component) [6]. The fastest known exact algorithm is due to Brandes [7]. It requires $O(n + m)$ space and $O(nm)$ time where n is the number of nodes and m the number of edges. Thus, the computation of such an index poses non trivial computational problems for large values of n and m even when using this efficient algorithm. For sparse graphs where $m = O(n)$ Brandes' method is still quadratic, which makes it inefficient for very large graphs.

In this paper we propose an evolution of the Brandes' algorithm which exploits some intrinsic topological characteristics of social networks to algebraically compute the centrality of the subgraphs of the network which exhibit a tree structure. The advantage of our approach is twofold: on the one hand we do not need to count shortest paths for a subset of network nodes, and, on the other hand for the other nodes we have to compute the shortest paths only between nodes belonging to a subgraph of the original graph.

Our algorithm performance strictly depends on the number of nodes for which we can algebraically derive the

betweenness, however it works well in practice for social networks since we observed that such structures (trees) are quite frequent in the context of social networks where the number of edges of the graph is of the same order of magnitude of the number of nodes. Note, however, our algorithm reduces to the Brandes' algorithm in the worst case.

We tested our algorithm on a set of 18 real social network graphs of *Sistemi Territoriali* which is an ICT company whose headquarters is in Italy, specializing in Business Intelligence applications. These graphs coming from real applications are very large and sparse, a property our algorithms exploits to gain in efficiency. Compared to Brandes' method we can gain orders of magnitudes (between one and two) in terms of computation time. At the best of our knowledge this approach is novel.

The paper is organized as follows. Section II gives a brief survey of related work, while section III gives key insights from Brandes' methods. In section IV we describe our method, we provide illustrative examples, and a formal proof of correctness. In Section VI we give the experimental results.

II. Related work

As mentioned in the previous section, there is not a unique way to define centrality and related measures, but different definitions could be used that are suitable to specific applications. A survey of these definitions and related problems is beyond the purpose of this paper and can be found in [2] and [1].

An important family of centrality indices is the one based on shortest paths. Let $G = (V, E)$ be the graph associated to a social network, we denote as: σ_{st} the number of shortest paths starting from the node s and ending in t, $\sigma_{st}(v)$ the cardinality of the subset of geodesic paths from s to t passing through v, and $d_G(s, t)$ the distance between the source and destination nodes.

The first known index based on the enumeration of shortest paths is stress centrality [8]. It defines the centrality for a node v as the number of occurrences of v in a geodesic path between each couple of nodes in the graph. It was designed in the field of communication networks to measure the amount of stress a node has to sustain to allow the optimal connectivity. Formally stress centrality is defined as:

$$C_S(v) = \sum_{s \neq v \neq t \in V} \sigma_{st}(v)$$

Betweenness [3] counts, for a given vertex v, the fraction of all the possible shortest paths between pairs of nodes which pass through v. The difference between stress and betweenness centrality is that the latter sums the relative number of shortest paths. This measure can be interpreted as a quantification of how a node controls the flow of information in the network. In fact, when the number of alternative shortest paths between two nodes increases, the relative importance of a specific node reduces. In contrast, if to connect two nodes we are constrained to pass through a certain node, its importance increases. Formally betweenness centrality is defined as:

$$C_B(v) = \sum_{s \neq v \neq t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

In order to extend the use of betweenness centrality to a wider range of applications, many variants of this index were proposed in the literature. For example in [9] the betweenness definition is applied to dynamic graphs, while in [10] geodesic paths are replaced with random walks to speed up the computation.

The concept of closeness between two objects is related to the notion of distance. Two objects are close if their distance is "small". In the context of graph theory different definitions of distance between pairs of nodes are possible, one of the most accepted is the length of a geodesic path. According to this definition of distance in [11] the authors define the closeness centrality as:

$$C_C(v) = \frac{1}{\sum_{t \in V} d_G(t, v)}$$

The practical application of centrality indices depends also on the scalability of the algorithm designed to compute them. Distance based indices can be easily computed from the definition if the distance matrix is already available. Betweenness centrality has more stringent requirements, in fact it needs to know how many geodesic paths exist between two nodes, and how many of them pass through a certain node. Early exact algorithms have a complexity in the order of $0(n^3)$ [12] where n is the number of nodes. As a result the computation of betweenness becomes impractical for networks with just a few thousands of nodes. To adapt betweenness computation to real world social networks some parallel algorithms were introduced such as [13] and [14].

To reduce the computational cost, many approximated algorithms and simplified definition of betweenness were proposed lately. In [15] the authors describe an approximation algorithm based on adaptive sampling which reduces the number of shortest paths computations for vertices with high centrality. In [16] the authors present a framework that generalizes the Brandes' approach to approximate betweenness. In [17] the authors propose a definition of betweenness which take into account paths of fixed length k. Another important complexity reduction was presented in [18] where ego-networks are used to approximate betweenness. A ego-network is a graph composed by a node, called *ego*, and by all the nodes, *alters*, connected to the *ego*. Thus if two nodes are not directly connected, there is only a possible alternative path which passes through the ego node. The authors have empirically shown over random generated networks that the betweenness of a node v is strongly correlated to that

of the ego network associated to v. Anyway, they do not provide any mathematical evidence for their conclusion, so this cannot be stated for generic real networks.

III. Background

Our algorithm algebraically computes the betweenness of nodes belonging to tree nested in the network and exploits a modification of Brandes' algorithm [7] to compute the betweenness of the other nodes. In this section we give some details about Brandes' algorithm, since it gives a background to our approach. This method is based on an accumulation technique where the betweenness of a node can be computed as the sum of the contributions of all the shortest paths starting from each node of the graph taken in turns.

Given three nodes $s, t, v \in V$, Brandes introduces the *pair-dependency* of s and t on v as the fraction of all the shortest paths from s to t and those from s to t passing through v

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

The betweenness centrality of the node v is obtained as the sum of the pair-dependency of each pair of nodes on v. To eliminate the computation of all the sums, Brandes introduces the dependence of a vertex s on v as:

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v) \tag{1}$$

Observation 1. *If a node v is a predecessor of w in a shortest path starting in s, then v is a predecessor also in any other shortest path starting from s and passing through w.*

According to observation 1, equation 1 can be rewritten as a recursive formula:

$$\delta_{s\bullet}(v) = \sum_{w:v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}}(1 + \delta_{s\bullet}(w)) \tag{2}$$

where $P_s(w)$ is the set of direct predecessors of a certain node w in the shortest paths from s to w.

When the starting node s is connected to any other node only through a single shortest path, equation 2 reduces to:

$$\delta_{s\bullet}(v) = \sum_{w:v \in P_s(w)} (1 + \delta_{s\bullet}(w))$$

IV. Our Algorithm

Our method is based on the observation that nodes with a single neighbor can be only shortest paths endpoints, thus their betweenness is equal to zero. However these nodes influence the betweenness of their neighbors. In fact, the neighbor works as a bridge to connect the node to the rest of the graph and all the shortest paths to (from) the node pass through the neighbor. Our procedure computes the betweenness of a node as the sum of the contribution of all its direct neighbors. According to this strategy, once the contribution of the nodes with degree 1 has been considered to the computation of the betweenness of their neighbor, they provide no more information, and can be virtually removed from the graph. The removal of the nodes with degree 1 from the graph, can cause that the degree of some other node becomes 1. Thus the previous considerations can be repeated on a new set of degree one nodes. In this case, however, we need also to remember the number of nodes connected to each of the degree one nodes that were removed from the graph. This recursive procedure allows us to algebraically compute the betweenness of trees in the graph.

The contribution of a node v to the betweenness of a neighbor w depends on the number of nodes constrained to pass through v to reach w. In the following we say that a node w *is connected* to the graph through v if it exists a node u such that the removal of v makes the graph disconnected and causes that u and w belong to different components

Consider, for example, the case in which v is a node with only one neighbor w, and let n be the number of nodes of the graph. The betweenness of v is zero, and the betweenness of w is at least $2*(n-2)$. In fact, w connects v with all the other nodes of the graph and vice-versa. If v had connected other nodes removed at a previous step, we would have considered this information when computing the contribution of v to the betweenness of w. Observe that each neighbor with degree 1 contributes to the betweenness of w and each contribution depends on the number of nodes that both the neighbor and w have already connected to the graph. The contribution of a node to its neighbor betweenness is called δ of the neighbor with respect to the node and denoted as $\delta_v(w)$ where v is the node with degree one and w is its neighbor.

As mentioned above, the removal of some nodes from the original graph G can cause that the degree of other nodes reduces to 1. According to this observation our algorithm iteratively produces a sequence of subgraphs of G such that $G^{i+1} \subseteq G^i$. Graph G^{i+1} is obtained from G^i by removing all the nodes with degree 1. The procedure stops when the graph G^i does not contain nodes with degree 1. The value of the betweenness of a node w in the original graph G is at least equals to the sum of all the contributions of its neighbors with degree one for each subgraph of G.

A. Algorithm formalization and description

Let $G = (V, E)$ be a connected, indirect, unweighted graph, and let $T_i(G)$ be the subset of nodes of G with degree i. Starting from $G = G^0$ we define a sequence of graphs $G^n = (V^n, E^n)$ such that $V^n = V^{n-1} \backslash T_1(G^{n-1})$ and $E^n \subseteq E^{n-1}$ where $E^n = \{(u, v) \in E^{n-1} | u, v \in V^n\}$. Note that G^n is a subgraph of G^{n-1} and, if $T_1(G^k) = \emptyset$, then $G^h = G^k$ for every $h \geq k$.

Our algorithm maintains a vector B containing a partial (under)estimation of the betweenness of each node of the graph G. At each iteration B is updated. We use the notation

B^n to indicate the vector B after n iterations. At iteration n, $B^n(x)$ is obtained adding the contribute $b^n(x)$ (computed over G^n as the sum of $\delta_\bullet(x)$) to $B^{n-1}(x)$. Under certain conditions $b^n(x)$ can be algebraically estimated with no need to compute all the shortest paths passing through x. As a consequence the computation of the betweenness in this case is much faster. For example, given the graph G^n, for each node $x \in T_1(G^n)$, we have $b^n(x) = 0$. In simple words, if a node in a graph has only one neighbor, it can not be in the middle of any shortest path, thus its betweenness is equal to 0. More in general, the contribution $b^n(x)$ to the betweenness of the node x during iteration n can be computed when it exists a subset of nodes in G which are constrained to cross x to reach all the other nodes of G^n.

In order to compute $b^n(u)$ we define the following sets. Let $v \in T_1(G^n)$ and $(u, v) \in E^n$, we define the set

$$N^-_{G^n}(u, v) = \{w \in V | v \in P_G(u, w)\} \quad (3)$$

where $P_G(u, w)$ is a shortest path in G that connects u to w. $N^-_{G^n}(u, v)$ is the set of nodes in G connected to u through v. Hence it identifies the set the nodes removed from G that are constrained to pass through v to be connected to the rest of the graph[1]. We define also the set

$$N^+_{G^n}(u, v) = (V \backslash N^-_G(u, v)) \backslash \{u\} \quad (4)$$

as the complement of $N^-_{G^n}(u, v)$ in which we removed also the node u.

Our algorithm mantains a set F in which at each stage we keep the nodes of G whose contribution to the betweenness computation has already been considered. We define also the set $R_G(u, F) = \{w \in F | \forall z \in P_G(u, w) \backslash \{u\}, z \in F\}$ as the subset of F that contains all the nodes of G that have already been connected through u.

We are now ready to define $\delta_v(u)$ as

$$\delta_v(u) = 2 * |N^-_{G^n}(u, v)| * (|N^+_{G^n}(u, v)| - |R_G(u, F)|) \quad (5)$$

Every time we compute a new contribution we update the set F accordingly (by adding the node v).

At each iteration our procedure checks whether the corresponding G^n has nodes with only one neighbor ($|T_1(G^n)| > 0$) or not. In the former case we can repeat the above procedure over such nodes. In the latter case our algorithm verifies if $|V^n| = 1$. In this case the procedure terminates and the vector B contains the betweenness of all the nodes of G, otherwise (when $|V^n| > 1$) the remaining graph does not have any node with degree 1. Thus, to compute (or update) the betweenness of the nodes in G^n we apply a slightly modified version of the Brandes' algorithm.

The modification of the Brandes' algorithm applied to G^n is necessary to take into account the contribution of the nodes in $R_G(s, F)$ to the computation of δ in the original graph G. Recall $P_s(w)$ is the set of predecessors of w in a geodesic path starting from s and σ_{su} is the number of shortest paths from s to u. For each starting node $s \in G^n$ our modified formula is:

$$B(u) = B(u) + \delta_{s\bullet}(u) * (1 + R_G(s, F))$$

where

$$\delta_{s\bullet}(u) = \sum_{w:u \in P_s(w)} \frac{\sigma_{su}}{\sigma_{sw}} (1 + \delta_{s\bullet}(w) + |R_G(w, F)|))$$

.

B. Proof

Given a node $s \in G^n$ of degree 1, the contribution to the betweenness of its neighbor v can be computed by the following general formula:

$$\delta_s(v) = 2((n - c) - m - 1)m \quad (6)$$

where m is the number of nodes in $N^-_{G^n}(v, s)$, c is the number of nodes already connected by v, and n is the number of nodes of G. According to Brandes' algorithm we know that the betweenness of a node is obtained as subsequent sums of δs:

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

We claim that the betweenness of the nodes belonging to a subgraph of G which shows a tree structure can be directly computed by subsequent sums of the formula 6. We show also how to modify the Brandes' algorithm to run on the rest of the graph and compute correctly the betweenness taking into account the contribution of all the nodes.

We show here how the Brandes' formula for the computation of δ of the neighbor v of a node s with degree 1 reduces to equation 6.

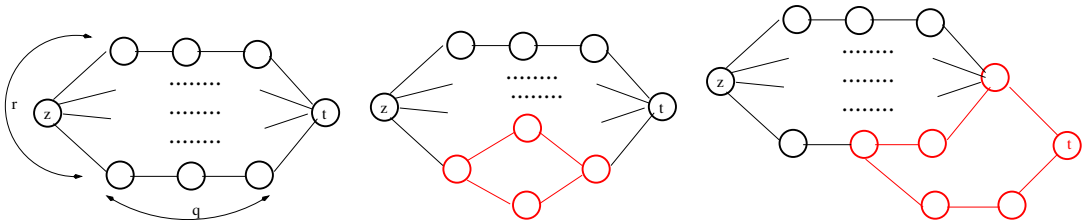
Let G' be a subgraph of G in which we removed the nodes of $N^-_{G^n}(v, s)$ except s. We have two cases: v was not the neighbor of any node with degree 1 in a previous iteration ($c = 0$), or it was ($c > 0$). In the first case G' contains $n' = n - m + 1$ nodes. If it exists only one shortest path from s to any other node of the graph G' then for each pair of consecutive nodes w, w' in the path, the following relation holds: $\frac{\sigma_{sw}}{\sigma_{sw'}} = 1$. Thus for each node in the path we add 1 to the value of δ. As a result it holds the following:

$$\delta_{s\bullet}(v) = \sum_{w:v \in P_s(w)} (1 + \delta_{s\bullet}(w)) = n' - 2 = n - m - 1 \quad (7)$$

Equation 7 is also valid for all the nodes in $N^-_{G^n}(v, s)$ since, to reach nodes in G' all the shortest paths are constrained to pass through v. Hence:

$$\forall w \in N^-_{G^n}(v, s),\ \delta_{s\bullet}(v) = \delta_{w\bullet}(v)$$

[1]The definition holds since the nodes are connected to u through a shortest path that contains v. This means they are nodes belonging to the set $T_1(G^i)$ with $i < n$



(a) Alternative disjoint shortest paths

(b) Alternative forking shortest paths

(c) External shortest paths

Figure 1. Some cases of multiple shortest paths.

The computation of $\delta(v)$ is not completed yet. In fact, all the shortest paths from a node $G^{'}$ to a node $w \in N^{-}_{G^n}(v,s)$ are constrained to pass through v. Furthermore we know, by construction, that v is not contained in any shortest path between two nodes of $N^{-}_{G^n}(v,s)$. As a consequence $\forall w^{'} \in G^{'}.\delta_{w^{'}\bullet}(v) = m$. This holds because: Brandes computes the value of δ from the farthest node back to the considered node, v is the only connection between the two parts of the graph and $m = N^{-}_{G^n}(v,s)$. The value m is added to the partial betweenness of v a number of times equals to the number of nodes in $G^{'}$ (i.e $n-m-1$). Hence, we can conclude that: $\delta_s(v) = 2(n-m-1)m$.

Suppose now that it exists more than one shortest path from s to some other node in $G^{'}$ (see figure 1(a)). This means that for at least one node there are at least 2 shortest paths that reach it. Let z be the fork starting node, t be the fork ending node (that is the node reached by more that one shortest path), q be the number of nodes of every alternative path in the fork, and r be the number of the different shortest paths. Let us concentrate only over the part of the graph that contains the fork, then if $\delta_{s\bullet}(t) = X \ \forall w \in P_s(t).\delta_{st}(w) = \frac{1}{r}(1+X)$.

Supposing that each node is present only in one path we have that the contribution of each path of the fork to the $\delta_{s\bullet}(z)$ is:

$$q + \delta_{s\bullet}(w) = q + \frac{1}{r}(1+X) \tag{8}$$

Hence

$$\delta_{s\bullet}(z) = rq + (1+X) \tag{9}$$

As a result, each node of the alternative paths in the fork contributes with 1 to the partial value of δ, and all the previous considerations still hold. Once again $\delta_s(v) = 2(n-m-1)m$.

The same consideration as before is valid also when the some node of the fork is present in many shortest paths. We have two possible cases: there is an internal fork within one of the alternative shortest paths as shown in figure 1(b) (in this case we can recursively consider the contribution of the internal fork until we reduce to the previous case with nodes belonging to a single path), or at least a node is involved in shortest paths outside the fork (see figure 1(c)). In this latter case, the shared nodes contribute also with their partial value of δ computed on the alternative shortest paths.

Suppose now $c > 0$. In this case some nodes were already connected to the rest of the graph through v and their contribution to $\delta(v)$ has already been considered. As a consequence, to the contribution that s gives to the betweenness of v, the part obtained by connecting nodes already considered has to be removed. Hence

$$\delta_s(v) = 2(n-m-1)m - \sum_{w \in R_G(v,F).(w,v) \in E} 2 * m * N^{-}_{G^n}(v,w)$$

but it holds that:

$$\sum_{w \in R_G(v,F).(w,v) \in E} N^{-}_{G^n}(v,w) = c$$

thus:

$$\begin{aligned} \delta_s(v) &= 2(n-m-1)m - 2*m*c = \\ &= 2((n-c)-m-1)m \end{aligned}$$

This concludes the proof of correctness for the procedure that assign the betweenness values to the nodes that belong to trees.

We demonstrate next the correctness of our modification of the Brandes' algorithm applied to the residual graph $G^{'}$ to derive the betweenness on G. Let $w \in G^{'}$ be the neighbor of at least one node $v \in F$ (i.e w is the root of a tree containing the nodes in $R_G(w,F)$ that were removed from G).

During the computation of the δs within $G^{'}$, w can belong to a shortest path, or it can be the starting node of shortest paths. In the first case we have to consider also the contribution of the nodes in $R_G(w,F)$ to the δ of a direct predecessor u of w in $G^{'}$. As we have already seen, this holds 1 for each node in $R_G(w,F)$. Thus:

$$\delta_{s\bullet}(u) = \sum_{w:u \in P_s(w)} \frac{\sigma_{su}}{\sigma_{sw}}(1 + \delta_{s\bullet}(w) + |R_G(w,F)|))$$

.

We observe that, if w is a starting node of shortest paths in $G^{'}$, each node in $R_G(w, F)$ could be a starting node in G of a shortest path constrained to pass through w. Thus to the δ of each generic node in these shortest paths we have also to add the contribution of each node in $R_G(w, F)$ (which is always the same). As a result we have:

$$B(u) = B(u) + \delta_{s\bullet}(u) * (1 + R_G(s, F))$$

□

C. Fast betweenness computation on sparse graphs. Some examples.

For sake of clarity, in this section we provide an example of our algorithm's execution on the simple graph in figure 2.

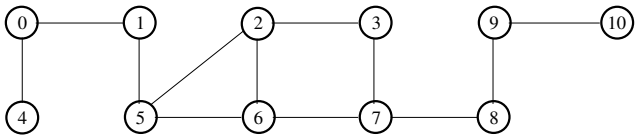


Figure 2. The example graph G.

At the first iteration our algorithm finds two nodes with degree 1, namely 4 and 10. Since they do not inherit a partial contribution from other nodes, their betweenness is 0 and nodes 0 and 9 have at least a betweenness equal to $2*9 = 18$. This is because node 0 connects node 4 to all the other nodes of the graph and vice versa. The same consideration holds for node 9 with respect to node 10. Once the partial betweenness of nodes 0 and 9 has been determined, nodes 4 and 10 can be virtually deleted from the graph, and we can iterate the mechanism on the remaining of the graph G^1 shown in figure 3.

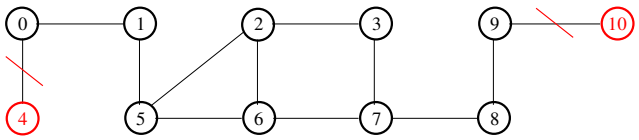


Figure 3. In black the graph G^1, in red the nodes whose betweennes has already been computed.

Consider now the graph G^1. In this graph nodes 0 and 9 have only one neighbor and their betweenness in G^1 is 0. Since the overall betweenness of a node in the original graph is obtained as the sum of all the contribution of each iteration, their betweenness is 18. As in the previous step the nodes with degree 1 influence the partial betweenness of their neighbors (nodes 1 and 8) and their contribution can be algebraically computed. In this case, however, we have to remember that nodes 1 and 8 do not connect only nodes 0 and 9 to the rest of the graph G, but also nodes 4 and 10. This means that all the paths for nodes 4 and 0 must pass through 1, and all the paths for nodes 9 and 10 must pass through 8.

Consider now, for example, node 8. This node breaks G into two components: one with 2 nodes and one having 8 nodes. Thus the betweenness of node 8 has to be at least equal to 2 (bi directionality of the connections) * 2 (# nodes of the first component) * 8 (# nodes of the second component) = 32. The same considerations holds for node 1. At this point we can virtually remove from G^1 nodes 9 and 0, obtaining the graph G^2 whose endpoints are the nodes 1 and 8. Hence the partial betweenness for these nodes computed in the previous step become the total one. Moreover, these nodes contribute to the partial betweenness of their neighbors (respectively nodes 5 and 7). According to our formula the contribution to nodes 5 and 7 is equal to 42 (=2*3*7).

Again nodes 1 and 8 can be virtually erased from the graph obtaining the graph G^3 shown in Figure 4.

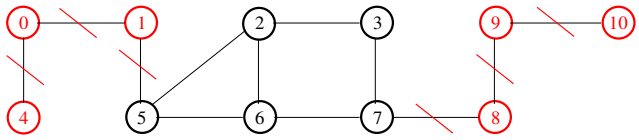


Figure 4. In black the graph G^3, in red the nodes which betweennes has already been computed.

In this case there are no nodes with degree 1 left, thus we need to apply on the remaining subgraph our slightly modified version of the Brandes' algorithm. Since we are interested in the computation of the betweenness on G, we need to take into account also the number of nodes of G connected through each node of the remaining graph. In our example nodes 5 and 7 connect three nodes to the rest of the graph, whereas node 2, 3, and 6 do not connect any node to the rest of the graph.

Let us now consider the computation of δ in the original graph. According to Brandes' algorithm node 7 should have a value for $\delta = 3$. This comes from the fact that 7 is the root of a tree nested in the reduced graph and the formula of δ applied to a node of a tree reduces to the count of all its successors. Notice that three is the number of nodes of G that node 7 connects to the reduced graph.

So consider for example the contribution to the δ of node 3 given by the path from node 2 to node 7. In this case we have:

$$\delta(3) = \frac{1}{2} * (1 + \delta(7) + 3) = 2$$

where $\delta(7)$ is computed on the reduced graph (and so it is equal to 0), and the factor 3 in the formula is the number of nodes connected through node 7. Consider now 7 as the paths starting node. Consider now a certain node x in a path

from 7, and let $\delta(x)$ be the value of δ computed for x in the reduced graph, then we modify the betweenness of x as:

$$B[x] = B[x] + \delta(x)(1+3)$$

The same holds for node 5.

Consider now the graph in Figure 5, and consider a running example of the method.

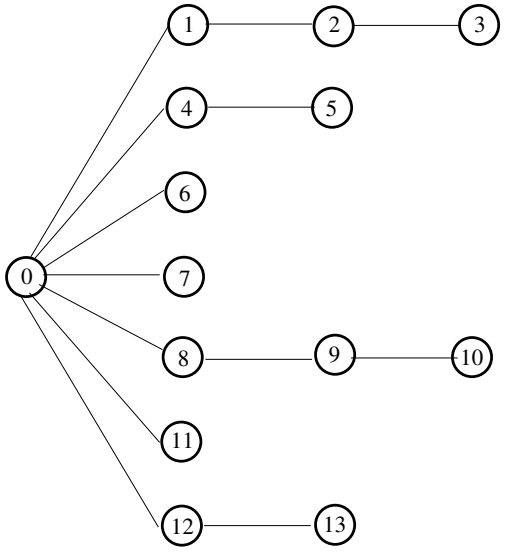


Figure 5. Example graph.

First step: identify the set of nodes with only one neighbor. This set is composed by {3,5,6,7,10,11,13} and the betweenness of each node in the set is zero. Now we compute the betweenness of the neighbors of 3, 5, 10, and 13 that is:

$$B(2) = B(4) = B(9) = B(12) = 2 * 12 = 24$$

Whereas the partial betweenness of node 0 is equal to 66 and not to 72. This is because $\delta_6(0) = 24$, $\delta_7(0) = 22$ (it holds when knowing that node 6 has already been considered) , and finally $\delta_1 1(0) = 20$ (knowing that nodes 6 and 7 have already been considered). For each node connected through node 0, the value of the betweenness of node 0 is different. Consider $\delta_6(0)$ For node 6 holds the formula seen beforeConsider now node 7. The connection between node 7 and node 6 has been already considered when computing $\delta_6(0)$ Hence for node $\delta_7(0)$ we have to consider all the connection but those with 6 For node 11, holds the same that for node 7: we have to consider only the connection obtained through 0 that have not already considered, so those with nodes 6 and 7 are discarded.

After the first step we get the graph in Figure 6 Let us concentrate only on the betweenness of 0. We get

$$B^i(0) = B^{i-1}(0) + \delta_4(0) + \delta_1 2(0) =$$
$$66 + 2 * 2 * (11-3) + 2 * 2 * (11-5)$$

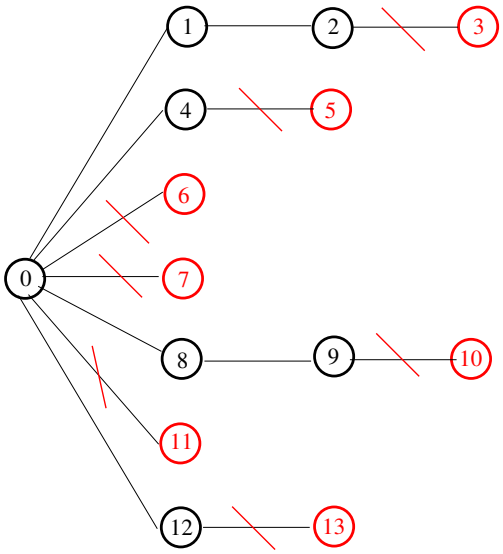


Figure 6. Example graph.

We repeat the process in this way until we remain with a graph composed of only node 0, and the execution ends.

V. Algorithm pseudo-code

In the following Algorithm 1 we show the algorithm pseudo-code for SPVB (Shortest-paths vertex betweenness) preprocessing.

SPVB:
Data: directed unweighted graph G=(V,E)
Result: the graph's node betweenness $c_B[v]$ for all $v \in V$
$c_B[v] = 0, v \in V$; $p[v] = 1, v \in V$;
$c[v] = 0, v \in V$; $i = 0$;
$G^i = G$; $deg_1 = \{v \in V^i | deg(v) = 1\}$;
repeat
  $v \leftarrow deg_1$;
  $u \in V^i.(v,u) \in E^i$;
  $C_B[u] = C_B[u] + 2(n - p[v] - c[u] - 1)p[v]$;
  remove v from deg_1;
  $p[u] = p[u] + 1$;
  $c[u] = c[u] + 1$;
  $i++$;
  $V^i = V^{i-1} \backslash \{v\}$
  $E^i = E^{i-1} \backslash \{(v,u)\}$
  if $deg(u) = 1$ **then** $u \rightarrow deg_1$; /* $deg(u)$ is computed on the new graph G^i */
until $deg_1 = \emptyset$;
if $|V^i| > 1$ **then**
  Brandes($G^i, p, c_B[v]$)
end

Algorithm 1: Shortest-paths vertex betweenness

The algorithm takes as input a graph $G = (V, E)$ and returns the betweenness value for each node of the graph. It starts by selecting all the nodes with degree 1 from G. It selects one of these nodes, say v, and computes the betweenness for its neighbor, say u, by exploiting the formula shown before. Then v is removed from G, and the cardinality of the sets, N^- and R associated to u are updated. If the degree of u in the new graph is 1, then u is added to the set deg_1. This process is repeated until the set deg_1 is empty. Then, if there is only one node left in the graph, the process ends, and the betweenness of all the nodes is returned. Otherwise, the modified version of the Brandes algorithm is applied on the subgraph G^i obtained from G by removing all the nodes, and consequently all the arc connecting these nodes, that belonged to deg_1.

In Algorithm 2 we show the modification to Brandes' algorithms.

A. Complexity

Given $G = (V, E)$ such that

$$\sum_{i=0}^{k} T_1(G^i) = n^{'} \tag{10}$$

where k is such that $T_1(G^{k+1}) = 0$ and $T_1(G^k) > 0$.
Let $n = |V|$ and $m^{'} = |\{(u, v)|u, v \in V^{k+1}\}|$ then the algorithm complexity is $= O(n^{'} + (n - n^{'}) \times m^{'})$ because Brandes' algorithm is applied only over the subgraph.
Hence if $n = n^{'}$ the algorithm complexity is $O(n)$, whereas in the worst case, where $n^{'} = 0$, it is $= (n \times m)$ where $m = |E|$

VI. Experiments

In order to evaluate the performance of our algorithm we run a set of experiments using a collection of 18 real graphs provided by *Sistemi Territoriali* which is an Italian ICT company involved in the field of data analysis for Business intelligence.

Since our algorithm computes the exact value of betweenness (as the Brandes' algorithm does) we compare the running time of the two algorithms. For our experiments we used a standard PC endowed with a 2.5 GHz Intel Core 2, 8Gb of RAM and Linux 2.6.37 operating system. The two algorithms were implemented in Java. In order to avoid possible biases in the running time evaluation due to the particular CPU architecture, we decided to implement the algorithm as a mono-processor sequential program.

In table I we report the graph id, the number of nodes of the initial graph, the number of nodes of the subgraph to which we applied the Brandes' modified algorithm, and lastly, the number of subgraphs needed to get a subgraph without degree one nodes. note that a very large percentage of the nodes can be dealt with algebraically and the residual graph, on which we ran a modified Brandes', is quite small, compared to the input size.

Brandes:
Data: directed graph $G = (V, E)$,
the number of nodex connected by each node $p[v]$,
the partial betweenness computed so far $c_B[v]$
Result: the graph's node betweenness $c_B[v]$
for $s \in V$ **do**
  S = empty stack;
  P[w]= empty list,$w \in V$;
  $\sigma[t] = 0$, $t \in V$;$\sigma[s] = 1$;
  $d[t] = -1$, $t \in V^i$; $d[s]$ =0;
  Q= empty queue;
  enqueue $s \rightarrow Q$;
  while *Q not empty* **do**
    dequeue $v \leftarrow Q$;
    push $v \rightarrow S$;
    forall *neighbor w of v* **do**
      // w found for the first time?
      if $d[w] < 0$ **then**
        enqueue $w \rightarrow$ Q;
        d[w]=d[v] + 1;
      end
      // shortest path to w via v?
      if *d[w] = d[v] + 1* **then**
        $\sigma[w] = \sigma[w] + \sigma[v]$;
        append $v \rightarrow P[w]$;
      end
    end
  end
  $\delta[v] = 0$, $v \in V$;
  // S returns vertices in order of non-increasing distance from s
  while *S not empty* **do**
    pop w$\leftarrow$ S;
    for $v \in$ *P[w]* **do**
      $\delta[v] = \delta[v] + \frac{\sigma[v]}{\sigma[w]}(\delta[w] + p[w])$;
    end
    if $w \neq s$ **then**
      $C_B[w] = C_B[w] + (\delta[w] \times p[s])$
    end
  end
end

Algorithm 2: Modified Brandes' algorithm

Figure 7 compares the running time of our and Brandes' algorithms. On the x-axis we report the graph id, while on the y-axis we report in logarithmic scale the running time expressed in seconds. From figure 7 it is possible to observe that our algorithm is always more than on order of magnitude faster than the procedure of Brandes, sometimes even two orders of magnitude faster.

For graph 1, with 233377 nodes for example, we were able to finish the computation within one hour while Brandes' needs approximately two days. For graph 6, with 169059 nodes, we could complete in about 1 minute, compared to

Graph id	# nodes	# nodes resolved with Brandes' algorithm	# subgraphs of G
1	233377	31973	24
2	14991	86	4
3	15044	2228	15
4	16723	2592	13
5	16732	2597	17
6	169059	303	47
7	16968	2615	13
8	3214	147	5
9	3507	123	6
10	3507	124	4
11	3519	123	6
12	44550	9995	20
13	46331	120	6
14	47784	7559	17
15	5023	351	12
16	52143	7399	20
17	8856	944	11
18	506900	99448	44

Table I
DATASET DESCRIPTION

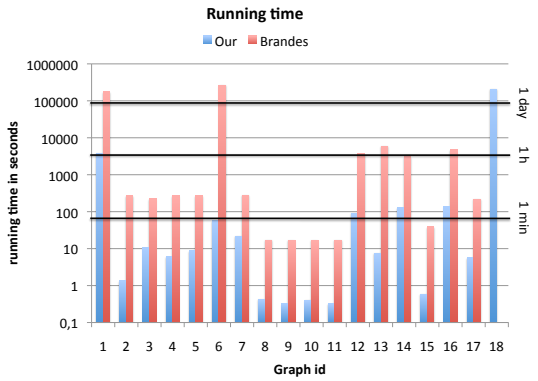


Figure 7. A comparison of the running time of our algorithm (left) and Brandes' (right) on 18 sparse large graphs. The ordinate axis report running time in seconds and is in logarithmic scale. Data for Brandes on graph 18 is missing due to time-out

two days for Brandes. A notable result is shown also for graph 18 which is our biggest and densest input graph. In this case our algorithm required approximately 2,4 days to finish while Brandes' could not complete in one month.

VII. CONCLUSIONS

Brandes's algorithm for computing betweenness centrality in a graph is a key breakthrough beyond the naive cubic method that computes explicitly the shortest paths in a graph. However, it is not able to exploit fully the sparsity of the input graph to speed up the computation on large graphs.

In this work we show that combining exact algebraic determination of betweenness centrality for the tree-like portion of the input graph, with a modified Brands' procedure on the residual (non-sparse) graph we can gain orders of magnitudes (between one and two) in terms of computation time. At the best of our knowledge this approach is novel.

Future work will include testing our method on a larger family of social networks. Moreover we plan to explore further this approach by determining other classes of subgraphs (besides trees) in which we can gain by the direct algebraic determination of the betweenness. Also the role of articulation points in block tree decomposition of sparse social graphs will be investigated.

REFERENCES

[1] D. Koschtzki, K. Lehmann, L. Peeters, S. Richter, D. Tenfelde-Podehl, and O. Zlotowski, "Centrality indices," in *Network Analysis*, ser. Lecture Notes in Computer Science, U. Brandes and T. Erlebach, Eds. Springer Berlin / Heidelberg, 2005, vol. 3418, pp. 16–61.

[2] S. P. Borgatti, "Centrality and network flow," *Social Networks*, vol. 27, no. 1, pp. 55 – 71, 2005.

[3] L. C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, Mar. 1977.

[4] A. Del Sol, H. Fujihashi, and P. O'Meara, "Topology of small-world networks of protein–protein complex structures," *Bioinformatics*, vol. 21, pp. 1311–1315, April 2005.

[5] L. Leydesdorff, "Betweenness centrality as an indicator of the interdisciplinarity of scientific journals," *Journal of the American Society for Information Science and Technology*, vol. 58, pp. 1303–1309, July 2007.

[6] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *Link Mining: Models, Algorithms, and Applications*, P. S. S. Yu, J. Han, and C. Faloutsos, Eds. Springer New York, 2010, pp. 337–357.

[7] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, 2001.

[8] A. Shimbel, "Structural parameters of communication networks," *Bulletin of Mathematical Biology*, vol. 15, pp. 501–507, 1953, 10.1007/BF02476438. [Online]. Available: http://dx.doi.org/10.1007/BF02476438

[9] T. Carpenter, G. Karakosta, and D. Shallcross, "Practical issues and algorithms for analyzing terrorist networks," 2002, invited paper at WMC 2002.

[10] M. J. Newman, "A measure of betweenness centrality based on random walks," *Social Networks*, vol. 27, no. 1, pp. 39 – 54, 2005.

[11] G. Sabidussi, "The centrality index of a graph," *Psychometrika*, vol. 31, pp. 581–603, 1966.

[12] R. Jacob, D. Koschtzki, K. Lehmann, L. Peeters, and D. Tenfelde-Podehl, "Algorithms for centrality indices," in *Network Analysis*, ser. Lecture Notes in Computer Science, U. Brandes and T. Erlebach, Eds. Springer Berlin / Heidelberg, 2005, vol. 3418, pp. 62–82.

[13] 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," *Parallel and Distributed Processing Symposium, International*, vol. 0, pp. 1–8, 2009.

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

[15] D. Bader, S. Kintali, K. Madduri, and M. Mihail, "Approximating betweenness centrality," in *Algorithms and Models for the Web-Graph*, ser. Lecture Notes in Computer Science, A. Bonato and F. Chung, Eds. Springer Berlin / Heidelberg, 2007, vol. 4863, pp. 124–137.

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

[17] 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*, ser. KDD '03. New York, NY, USA: ACM, 2003, pp. 266–275.

[18] M. Everett and S. P. Borgatti, "Ego network betweenness," *Social Networks*, vol. 27, no. 1, pp. 31 – 38, 2005.