Fast exact and approximate computation of betweenness centrality in social networks

Miriam Baglioni[1], Filippo Geraci[1], Marco Pellegrini[1] and Ernesto Lastres[2]

[1] Istituto di Informatica e Telematica del CNR, Via G. Moruzzi 1, 56100-Pisa (Italy)
[2] Sistemi Territoriali, via di Lupo Parra Sud 144 San Prospero 56023 (PI), Italy

Abstract. Social networks have demonstrated in the last few years to be a powerful and flexible concept useful to represent and analyze data emerging from social interactions and social activities. The study of these networks can thus provide a deeper understanding of many emergent global phenomena. The amount of data available in the form of social networks is growing by the day. 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. The computation of the *betweenness centrality* index (BC) is a well established method for network data analysis and it is also important as subroutine in more advanced algorithms, such as the Girvan-Newman method for graph partitioning.

In this paper we present a novel approach for the computation of the *betweenness centrality*, which speeds up considerably Brandes' algorithm (the current state of the art) in the context of social networks. Our approach exploits the natural sparsity of the data to algebraically (and efficiently) determine the betweenness of those nodes forming trees (tree-nodes) in the social network. Moreover, for the residual network, which is often of much smaller size, we modify directly the Brandes' algorithm so that we can remove the nodes already processed and perform the computation of the shortest paths only for the residual nodes. We also give a fast sampling-based algorithm that computes an approximation of the betweenness centrality values of the residual network while returns the exact value for the tree-nodes. This algorithm improves in speed and precision over current state of the art approximation methods.

Tests conducted on a sample of publicly available large networks from the Stanford repository show that, for the exact algorithm, speed improvements of a factor ranging between 2 and 5 are possible on several such graphs, when the sparsity, measured by the ratio of tree-nodes to the total number of nodes, is in a medium range (30% to 50%). For some large networks from the Stanford repository and for a sample of social networks provided by *Sistemi Territoriali* with high sparsity (80% and above) tests show that our algorithm, named SPVB (for Shortest Path Vertex Betweenness), consistently runs between one and two orders of magnitude faster than the current state of the art exact algorithm.

1 Introduction

Social networks have demonstrated in the last few years to be a powerful and flexible concept useful to represent and analyze data emerging from social interactions and social activities. The study of these networks can thus provide a deeper understanding of many emergent social global phenomena. Moreover such analytic tools and concepts have been successfully adopted in a vast range of applications including communications, marketing and bioinformatics.

According to the standard paradigm of social networks, each agent/item is associated to 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 and efficient graph algorithms play an important role in social network analysis. Among the analytic 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 more central node is likely to be involved in many processes of the network, thus its importance increases.

Depending on what we mean with the word "important", different definitions of centrality are possible [19]. For example: degree centrality highlights nodes with a higher number of connections, closeness centrality highlights nodes easily reachable from other nodes, eigenvector centrality highlights nodes connected with authoritative nodes and betweenness centrality (BC) highlights nodes which are more likely to be information hubs. A complete compendium of many centrality definitions, problems and measures can be found in [5]. *Vertex betweenness* [1, 13] is one of the most broadly used centrality indices. The (vertex) betweenness of a vertex v 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 shortest paths. Despite the fact that this assumption could be considered restrictive, betweenness finds a vast range of applications (e.g. in computing lethality for biological networks [11] and in bibliometry [21]).

A very similar concept, the *edge betweenness*, is defined in [1] where for an *edge* e, the sum is computed for each pair of nodes (s, t) of the ratio among the number of shortest paths from s to t through the edge e over the number of all the shortest paths from s to t. Edge betweenness has a prominent application as a subroutine in the algorithm of Newman and Girvan [15] for community detection of complex networks. In this paper, for sake of clarity, we discuss only the problem of computing efficiently vertex betweenness, however with minor modifications our approach applies to edge betweenness as well (see [7]). 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 based on a direct application of its definition becomes impractical, having complexity $O(n^3)$, for a graph with n nodes. Since the last decade the number

and size of social networks have been consistently increasing over time, efficient algorithms have emerged to cope with this trend.

The fastest exact algorithm to date is due to Brandes [6]. It requires $O(n+m)$ space and $O(nm)$ time where n is the number of nodes and m the number of edges in the graph. For sparse graphs, where $m = O(n)$, Brandes' method is a huge improvement over the naive direct method, however it is still quadratic in n, regardless of any other special feature the input graph may have.

In this paper we propose an evolution of the Brandes' algorithm, named SPVB (for Shortest Path Vertex Betweenness), which exploits some widespread topological characteristic of social networks to speed up the computation of the betweenness centrality index. We show that for nodes in the graph that belong to certain tree structures the beteenness value can be computed by a straightforward counting argument. The advantage of our approach is two-fold: on the one hand we do not need to count shortest paths for the subset of network nodes that have the required tree-structure, and, on the other hand, for the residual nodes we compute the shortest paths only between nodes belonging to the residual of the original graph, thus more efficiently. Our algorithm performance strictly depends on the number of nodes for which we can algebraically derive the betweenness. Therefore it works well in practice for social networks since we observed that such tree structures 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, that SPVB still reduces to the Brandes' algorithm in a strict worst case scenario.

We have tested graphs with up to 500K nodes, which is a fair size for many applications. However in some applications (e.g. web graphs, telephone calls graphs) we face much larger graphs in the regions of millions of nodes, and we might want to trade off speed and precision in computing the Betweenness Centrality (BC). In this case approximating betweenness may be the strategy of choice. Thus we combine our algebraic computation with the sampling approach in [3] so to gain the benefits of both (see Section 6), obtaining the algorithm ASPVB (for Approximate Shortest Path Vertex Betweenness).

We tested our algorithm on a set of 18 social graphs of *Sistemi Territoriali* which is an ICT company with headquarters in Italy, specializing in Business Intelligence applications. These graphs coming from real applications are very large and very sparse, a property SPVB exploits to gain in efficiency. Compared to Brandes' method we can gain orders of magnitudes (between one and two) in terms of computation time. We also tested SPVB on a set of 16 social graphs from the Stanford Large Network Dataset Collection. We obtained marginal improvements on seven cases, speed ups by a factor from 2 to 6 in six cases, and speedups by orders of magnitude in two cases. At the best of our knowledge this approach is novel.

The paper is organized as follows. Section 2 gives a brief survey of related work, while section 3 gives key insights from Brandes' methods. In section 4 we describe our method in detail for exact computations. In Section 5 we give the

experimental results for exact computations. In Section 6 we give the approximation algorithm and the corresponding experimental results.

2 Related work

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. Betweenness centrality [13] measures, for a given vertex v, the fraction of all the possible shortest paths between pairs of nodes which pass through v. Formally betweenness centrality $B(v)$ is defined as:

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

The practical application of centrality indices depends also on the scalability of the algorithm designed to compute them. Early exact algorithms have a complexity in the order of $O(n^3)$ [17], where n is the number of nodes. Thus the computation of betweenness by this direct approach becomes impractical for networks with just a few thousands nodes.

In 2001 Brandes [6] developed the asymptotically fastest exact algorithm to date, that exploits a recursive formula for computing partial betweenness indices efficiently. It requires $O(n + m)$ space and $O(nm)$ time where n is the number of nodes and m the number of edges in the graph. For sparse graphs, where $m = O(n)$, Brandes' method is a huge improvement over the naive direct method, allowing to tackle graphs with tens of thousands of nodes.

Given the importance of the index, and the increasing size of networks to be analyzed, several strategies for scaling up the computation have been pursued. Algorithms for parallel models of computations have been developed (se e.g. [22], [18] and [2]).

A second strategy is to resort to approximations of the betweenness [8]. In [3] the authors describe an approximation algorithm based on adaptive sampling which reduces the number of shortest paths computations for vertices with high centrality. In [14] the authors present a framework that generalizes the Brandes' approach to approximate betweenness. In [25] the authors propose a definition of betweenness which takes into account paths up to a fixed length k.

Another important complexity reduction strategy was presented in [12] 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.

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 [23] geodesic paths are replaced with random walks. Modularity properties of social networks are used in [10] to define a notion of *Community Inbetweenness*. In experiments this measure is shown to weakly correlate with standard BC for networks of high modularity.

In graphs that change dynamically or are built incrementally (e.g. in a streaming model) algorithms have been proposed that dynamically update the betweenness by detecting efficiently those nodes whose BC is affected by the graph update (see [16, 20]).

In this paper we propose to use specific local structures abundant in many types of social graphs in order to speed up the exact computation of the betweenness index of each node by an adaptation of the exact algorithm due to Brandes.

An approach that exploits special structures in social graphs is advocated also a paper by Puzis et al. [24] that appeared just after the preliminary conference version of this work [4]. In [24] R. Puzis et al. develop two algorithms for exact BC computation. The first algorithm is advantageous when many nodes that are structurally equivalent, that is when they have the same set of neighbors. In this case equivalent nodes can be contracted into a single node and a quotient graph is generated. The original Brandes' procedure is adapted to work on the quotient graph, while computing the BC relative to the original graph. Experiments in [24] show a speed up from 2 to 3 in several Autonomous Systems (AS) graphs, and from 2 to 6 in DBLP co-authors graphs. The second algorithm generates the bi-connected components of the input graph, computes partial BC independently for each bi-connected, and then combines the results of the single components to produce the BC with respect to the original graph. Combining the two algorithms it is shown a speed from 2 to 7 in the set of AS-graphs. The edges of the tree-like structures we exploit are bi-connected components of the input graph thus, our trees are a special case of the components considered in [24], however the code we propose are much simpler than the algorithm in [24], while attaining comparable speed ups in the tested as-graphs.

3 Background

In this section we give some key features of 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 notion of *pair-dependency* of s and t on v as the fraction of all the shortest paths from s to t through v over those from s to t:

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

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 direct computation of

all the sums, Brandes introduces the *dependency* of a vertex s on v as:

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

Thus the betweenness centrality B, of node v is given by summing the dependencies from all source nodes:

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

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 [6].*

Arguing form the 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, encoded in a BFS (Breadth First Search) rooted DAG (Directed Acyclic Graph) form node s.

4 Our Algorithm: SPVB

Our algorithm algebraically computes the betweenness of nodes belonging to trees in the network obtained by iteratively removing nodes of degree 1. Afterwards we apply a modification of Brandes' algorithm [6] to compute the betweenness of the nodes in the residual graph.

A first trivial observation is that nodes with a single neighbor can be only shortest paths endpoints, thus their betweenness is equal to zero. Thus we would like to remove these nodes from the graph. However, these nodes by their presence influence the betweenness of their (unique) neighbors. In fact, such neighbor v works as a bridge to connect the node to the rest of the graph and all the shortest paths to (from) this node pass through that unique neighbor. Our procedure computes the betweenness of a node v as the sum of the contribution of all nodes for which v is their unique direct neighbor.

Following this strategy, once the contribution of the nodes with degree 1 has been considered in the computation of the betweenness of their neighbors, 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. When we iterate, however, we need also to record 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.

4.1 Algorithm formalization and description

We will assume the input G to be connected, in order to simplify the argument. If G is not connected, the argument can be repeated for each connected component separately. Let F be the set of nodes in $G = (V, E)$ that can be removed by iteratively delete nodes of degree 1, and their adjacent edge. We call the nodes in F the *tree-nodes*. Let $G' = (V', E')$ be the residual graph for the residuals set of node, with $V' = V \setminus F$. The set F induces a forest in G, moreover the root of each tree T_i of the forest is adjacent to a unique vertex in V'. Each node in F is a root to a sub-tree. Let $R_G(w, F)$ be the set of nodes of trees in F having w as their root-neighbor in G'. The formula for the betweenness of node $v \in V$ involves a summation over pairs of nodes $s, t \in V$. Thus we can split this summation into sub-summations involving different types of nodes, and provide different algorithms and formulae for each case.

Tree-nodes. Let u be a node in F, and let $v_1, .., v_k$ be the children of u in the tree T_u, and let T_{v_i}, for $i = 1, ..k$, be the subtrees rooted at v_i. When s and t are in the same subtree T_{v_i}, then there is only one shortest path connecting them completely in T_{v_i} and this path does not contain u, thus the contribution to $B(u)$ is null. When s is in some tree T_{v_i}, and t is in the complement $(V \setminus \{u\}) \setminus T_{v_i}$, then each shortest path connecting them will contain u. Thus the contribution to the betweenness of u is given by the number of such pairs. We will compute such number of pairs incrementally interleaved with the computation of the set F by peeling away nodes of degree 1 from the graph. When at iteration j, we peel away node v_i we have recursively computed the value of $|T_{v_i}|$, and also for the node u the value $|R_G(u, F_j)|$ which is the sum of the sizes of trees T_{v_h}, for $h \in [1, ..k], i \neq k$ already peeled away in previous iterations. The number of new pairs to be added to $B(u)$ is:

$$|T_{v_i}| \times (|(V \setminus \{u\}) \setminus T_{v_i}| - |R_G(u, F_j)|).$$

This ensures that each pair (s,t) is counted only once. Finally observe that when both s and t are in V' no shortest path between them will contain u therefore their contribution to $B(u)$ is zero. Since the roles of s and t are symmetrical in the formula we need to multiply the final result by 2 in order to cont all pairs (s, t) correctly. The pseudocode for this procedure is shown in Section 4.2. See Figure 1 for an illustration.

Residual graph nodes. Let u be a node in V', we will see how to modify Brandes' algorithm so that executing the modified version on the residual graph G' (thus at a reduced computational cost), but actually computing the betweenness of the nodes in $u \in V'$ relative to the initial graph G. Brandes algorithm's inner loop works by computing from a fixed node s a BFS search DAG in the input graph, which is a rooted DAG (rooted at s), and by applying a structural induction from the sinks of the DAG towards the root as in formula (2).

Subcase 1. If a node $x \in V'$ has $R(x, F) \neq \emptyset$ the tree formed by $R(x, F)$ and x would be part of the BFS DAG in G having its source in V', however, since we run the algorithm on the reduced graph G', we need to account for the contribution of the trimmed trees to the structural recursive formula (2). The

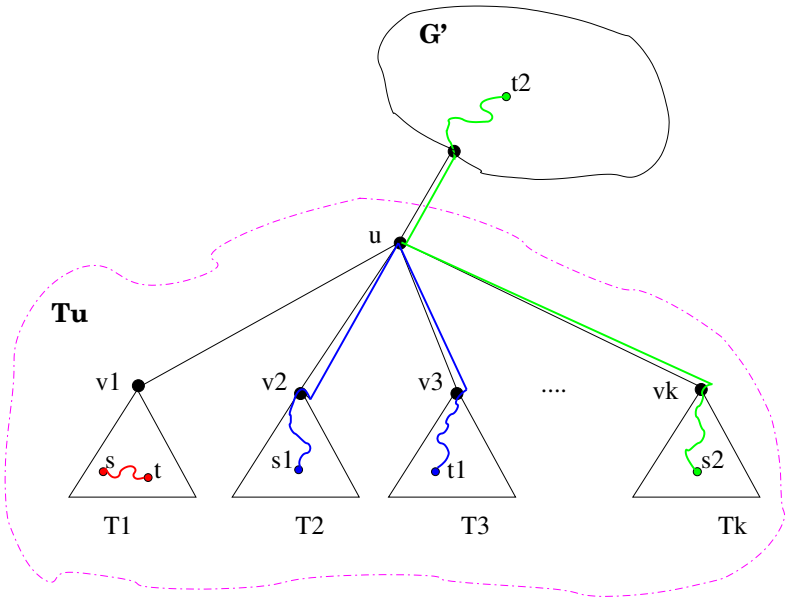

Fig. 1. Illustration of the tree-nodes structure and possible s-t paths involving tree nodes.

correction term for $\delta_{s\bullet}(x)$ is equal to $|R_G(x,F)|$ since each shortest path from s to $y \in R_G(x,F)$ must contain x. Thus we obtain the new formula:

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

Note that in the development of the above formula $R(s,F)$ does not appear. Since no shortest path from $s \in V'$ to any $t \in R(s,F)$ may include a node $u \in V'$, this subtree has zero contribution to $\delta_{s\bullet}(u)$.

Subcase 2. Consider now a node $x \in R(s,F)$ as source for the BFS. In the computation of $\delta_{x\bullet}(u)$, for $u \in V'$ each shortest path from x to $t \in R(s,F)$ cannot contain u thus gives zero contribution. For $t \in V \setminus R(s,F)$, such shortest path would contain a shortest path from s, thus we have $\delta_{x\bullet}(u) = \delta_{s\bullet}(u)$ for all $x \in R(s,F)$. In order to account for these contributions to $B(u)$ it suffices to multiply the contribution $\delta_{s\bullet}$ by $(1 + |R(s,F)|)$, obtaining:

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

4.2 Algorithm pseudo-code

In the following Algorithm 1 we show the pseudo-code for SPVB (Shortest-paths vertex betweenness) preprocessing, handling degree 1 nodes. For simplicity we

assume G to be connected. For a disconnected graph G, the algorithm should be applied to each connected component separately. For a node v of degree 1 at a certain stage of the iteration, the vector p records the number of nodes in a subtree rooted at v (excluding the root). For any other node u, vector p records the sum of the sizes of subtrees rooted at children of that node that have been deleted in previous iterations.

SPVB:
Data: undirected unweighted graph G=(V,E)
Result: the graph's node betweenness $B[v]$ for all $v \in V$
$B[v] = 0, v \in V$; $p[v] = 0, v \in V$; $i = 0$;
$G^i = G$; $deg_1 = \{v \in V^i | deg(v) = 1\}$;
repeat
  $v \leftarrow deg_1$;
  $u \in V^i.(v,u) \in E^i$;
  $B[u] = B[u] + 2(n - p[v] - p[u] - 2)(p[v] + 1)$;
  remove v from deg_1;
  $p[u] = p[u] + p[v] + 1$;
  $i++$;
  $V^i = V^{i-1} \backslash \{v\}$
  $E^i = E^{i-1} \backslash \{(v,u)\}$
  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_modified(G^i, p, B)
end

Algorithm 1: Shortest-paths vertex betweenness

The modification of Brandes' algorithm does not change its asymptotic complexity, which however must be evaluated on the residual graph with $n' = |V| - |F|$ nodes and $m' = |E| - |F|$ edges, thus with a time complexity $O(n'm')$. The complexity of the first part of SPVB is constant for each node in F, except for the operations needed to dynamically modify the graph G^i in Algorithm 1 and maintain the set of degree-1 nodes. With standard dynamic dictionary data structure we have an overhead of $O(\log n)$ for each update operation.

5 Experiments

In order to evaluate the performance of our algorithm we run a set of experiments using both a collection of 18 graphs provided by *Sistemi Territoriali (SisTer)*, which is an Italian ICT company involved in the field of data analysis for Business Intelligence and a collection of graphs downloaded from the Stanford Large Network Dataset Collection[3]. Since both SPVB and Brandes' compute the exact

[3] http://snap.stanford.edu/data/

Brandes_modified:
Data: directed graph $G = (V, E)$,
for each v:
the number of tree-nodes connected to v: $p[v]$,
the partial betweenness computed for v: $B[v]$
Result: the graph's node betweenness $B[v]$
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**
            $\delta[v] = \delta[v] + \frac{\sigma[v]}{\sigma[w]}(\delta[w] + p[w] + 1)$;
        end
        if $w \neq s$ **then**
            $B[w] = B[w] + \delta[w] \times (p[s] + 1)$
        end
    end
end

Algorithm 2: Modified Brandes' algorithm

value of betweenness, we tested the correctness of the implementation by comparing the two output vectors. Here we report only on the 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.

SisTer Collection. In Table 1 we report the graph id, the number of nodes and edges in the SisTer collection and the percentage of tree-nodes in each graph. Note that a very large percentage of the nodes can be dealt with algebraically by SPVB and the residual graph, on which we ran a modified Brandes', is quite small relative to the original size.

Graph ID	Node #	Edge #	Tree nodes (%)
G1	233,377	238,741	86 %
G2	14,991	14,990	99 %
G3	15,044	15,101	85 %
G4	16,723	16,760	84 %
G5	16,732	16,769	84 %
G6	169,059	169,080	99 %
G7	16,968	17,026	84 %
G8	3,214	3,423	95 %
G9	3,507	3,620	96 %
G10	3,507	3,620	96 %
G11	3,519	3,632	96 %
G12	44,550	46,519	77 %
G13	46,331	46,331	99 %
G14	47,784	48,461	84 %
G15	5,023	5,049	93 %
G16	52,143	53,603	85 %
G17	8,856	10,087	89 %
G18	506,900	587,529	80 %

Table 1. SisTer Collection. For each graph it is listed the number of nodes, the number of edges, and the percentage of tree-nodes. The graphs need not be connected.

Figure 2 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 2 it is possible to observe that SPVB is always more than one order of magnitude faster than the procedure of Brandes, sometimes even two orders of magnitude faster. For graph G1, with 233,377 nodes, for example, we were able to finish the computation within one hour while Brandes' needs approximately two days. For graph G6, with 169,059 nodes, we could complete in about 1 minute, compared to two days for Brandes. A notable result is shown also for graph G18 which is our biggest

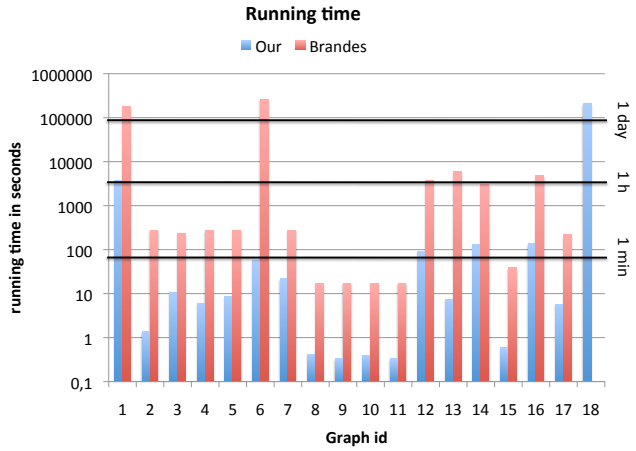

Fig. 2. A comparison of the running time of our algorithm SPVB (left) and Brandes' (right) on 18 sparse large graphs. The ordinate axis report running time in seconds and is in logarithmic scale. Data for Brandes on graph 18 is missing due to time-out

in this collection. In this case SPVB required approximately 2,4 days to finish while Brandes' could not terminate in one month (data not shown).

Stanford Collection. We have selected a subset of graphs from the Stanford collection, using the following criteria. First the graphs have been ranked by number of nodes and we have selected representative graphs from as many categories as possible (Social networks, Communication Networks, Citation networks, Collaboration networks, Web graphs, Internet peer-to-peer networks, and Autonomous systems graphs). We have excluded graphs that because of their size would take more than one week of computing time. In Table (2) we have listed these graphs, their size in number of nodes and edges, and the percentage of tree-nodes, which is the most important parameter influencing the performance of our method. Each input graph was considered undirected. We decided a cut-off time of seven days. In order to measure the convergence of the two methods we collected also the partial output of the two algorithms every 24 hours of execution. In Table 3 the running time, expressed in seconds, of the two methods is shown, and the speed up factor. As it is expected the speed up factor is strongly correlated to the fraction of the tree-nodes in the graph. We notice a speed-up factor ranging from 2 to almost 6 when the ratio of tree-nodes to the total number of nodes is in the range 30% to 50%.

Two large test graphs are quite noticeable. Graph *Email-EuAll* has a percentage of 80% of tree-nodes which is a value closer to those found in the SisTer collection, thus the speed up measured is at least 27 (since we stopped Brandes' after one

Graph name	Node #	Edge #	Tree nodes (%)
ca-GrQc	5,242	28,980	21%
as20000102	6,474	13,233	**36%**
ca-HepTh	9,877	51,971	20%
ca-HepPh	12,008	237,010	11%
ca-AstroPh	18,772	396,160	6%
ca-CondMat	23,133	186,936	9%
as-caida20071112	26,389	106,762	**38%**
cit-HepTh	27,770	352,807	5%
cit-HepPh	34,546	421,578	4%
p2p-Gnutella31	62,586	147,892	**46%**
soc-epinion1	75,879	508,837	**51%**
soc-sign-Slashdot090221	82,144	549,202	**36%**
soc-Slashdot0922	82,168	948,464	2%
soc-sign-epinions	131,828	841,372	**51%**
Email-EuAll	265,214	420,045	**80%**
web-NotreDame	325,729	1,497,134	**51%**

Table 2. Selected graphs from the Stanford Collection. For each graph it is listed the number of nodes, the number of edges, and the percentage of tree-nodes, which is the most important parameter affecting the time performance.

Graph name	Node #	Brandes (s)	SPVB (s)	Ratio
ca-GrQc	5,242	35 s	24 s	1.45
as20000102	6,474	141 s	54 s	**2.65**
ca-HepTh	9,877	230 s	148 s	1.55
ca-HepPh	12,008	703 s	563 s	1.24
ca-AstroPh	18,772	2,703 s	2,447 s	1.10
ca-CondMat	23,133	3,288 s	2,718 s	1.21
as-caida20071112	26,389	6,740 s	2,014 s	**3.34**
cit-HepTh	27,770	8,875 s	8,227 s	1.07
cit-HepPh	34,546	16,765 s	15,636 s	1.07
p2p-Gnutella31	62,586	74,096 s	15,573 s	**4.76**
soc-Epinion1	75,879	145,350 s	25,771 s	**5.64**
soc-sign-Slashdot090221	82,140	199,773 s	64,905 s	**3.07**
soc-Slashdot0902	82,168	199,544 s	190,536 s	1.04
soc-sign-epinions	131,828	564,343s	96,738 s	**5.83**
Email-EuAll	265,214	$>$ 7 days	22,057 s	$>$ **27**
web-NotreDame	325,729	-	$\approx$ 9 days	$\approx$ **8**

Table 3. Running time (in seconds)of the two methods over selected Stanford Collection graphs, and their ratio (speed up factor).

week). That value is between one and two orders of magnitude, consistently with those measured in the SisTer collection.
For the *web-NotreDame* graph, which is the largest graph in our sample of the Stanford collection, we estimate the convergence properties of the two algorithms as follows. SPVB has been run to completion (in about 9 days) in order to have the exact target solution vector. Also at fixed intervals each day we recorded the intermediate values of the betweenness vectors for both algorithms. For each vertex we compute the ratio of the intermediate value over the target value (setting 0/0 to value 1), and then we average over all the vertices. This measure is strongly biased by the fact that for leaves (nodes with degree 1) both Brandes and SPVB assign at initialization the correct value 0, thus in this case precision is attained by default. To avoid this bias we repeat the measurement by averaging only over those nodes with final value of betweenness greater than zero (see Figure 3). From Figure 3 we can appreciate that the average convergence rate is almost linear in both case, but the curve for SPVB has a much higher slope. After 7 days our algorithm reached about 75% of the target, against 10% of Brandes', by a linear extrapolation we can thus predict a speed up factor of about 8.

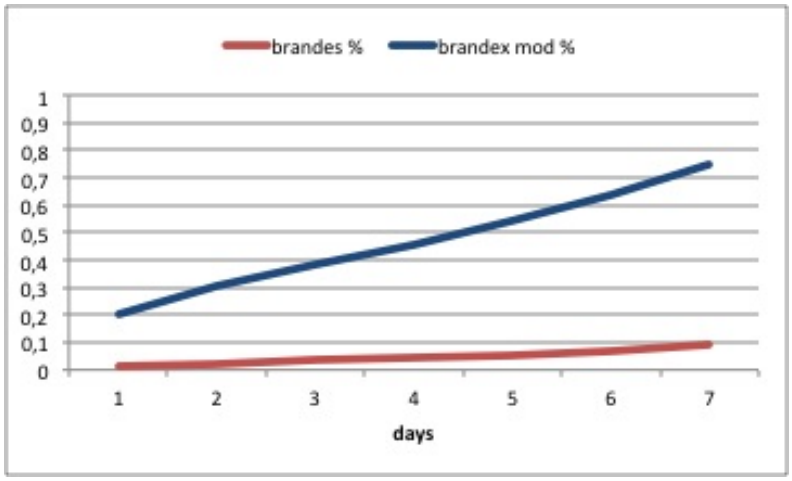

Fig. 3. Evolution in time of the average (over the vertices) ratio of the partial betweenness values over the final betweenness value. In the averaging leaves are excluded.

6 Approximating Betweenness Centrality

In this section we show how we can combine our algebraic approach to computing BC with the approximation scheme in [3], which is based on adaptive sampling.

First of all we notice that it is not possible in general to choose a random sample size for each data set that ensures a uniform relative error ϵ at each node. In [3] it is shown that with high probability, we can approximate the betweeenness $B[v]$ of a node v in a graph of n nodes, up to a factor $1/\epsilon$, with a number s of randomly chosen source nodes (from here referred as *pivots*), where $s = s(B[v], n, \epsilon)$. Since s depends also on the value $B[v]$ we cannot hope to have a uniform approximation factor bound over all the nodes in the graph. For this reason, we select an uniform sample size function having as input only the number of nodes and we *measure* the resulting mean relative error in each experiment. Thus we select a fixed value $s = \sqrt{n}$, and we measure empirically the mean relative error against the exact value[4]. The BC value of tree-nodes is known exactly and their contribution to the BC value of other nodes can be attributed to the root of the tree, therefore we restrict the sampling on the nodes in the residual graph. Also the shortest path computations are done in the residual graph. Note however that the expansion factor used to estimate the BC is referred to the size of the original graph. The pseudocode of the combined approach is shown in Algorithms 3, 4, and 5.

6.1 Approximate Algorithm Pseudo Code

In the following Algorithm 3 we show the pseudo-code for ASPVB (Approximate Shortest-paths vertex betweenness) preprocessing. For the sake of clarity we consider G to be connected. For disconnected graphs the same procedure should be applied to each component.

The algorithm takes as input an undirected graph $G = (V, E)$ and returns the approximate betweenness value for each node of the graph. Since the algebraic computation is the same of the exact algorithm, for nodes whose betweenness is algebraically computed the returned value is exact.

In Algorithm 4 we show our approximate algorithm for the residual graph. We compute the betweenness of the nodes in each path starting from a generic node s as if we were considering the path in the whole graph (see lines 1 and 2 in Algorithm 4). This is because we need to consider the contribution of the node within the whole graph when computing its approximate value. We maintain update an auxiliary structure (see line 3 in Algorithm 4) with the weight of each node in the shortest path from s for all the nodes connected to the residual graph through s. This value will be used in case of exact computation (see line 1 in Algorithm 5) to return the exact value of each node. As in [3], the computation of the approximate betweenness is the sum of the contributions due to the pivots times $\sqrt{n}$.

6.2 Experimental results on approximating betweenness

In Tables 4 and 5 we report quality (measured by the mean relative error) vs. time measurements over ten runs of our approximation algorithm and the

[4] For nodes whose BC exact value is zero, the partial BC contribution for any source is also zero, thus the sampling procedure will estimate the correct value, zero.

ASPVB:
Data: unweighted graph G=(V,E)
Result: the graph's node approximate betweenness $c_B[v]$ for all $v \in V$
$c_B[v] = 0, v \in V$;
$c_0[v] = 0, v \in V$; `/* ` $c_0[v]$ `stores the algebraic computation of the degree one nodes */`
$p[v] = 0, v \in V$; $i = 0$;
$G^i = G$;
$deg_1 = \{v \in V^i | deg(v) = 1\}$;
repeat
  $v \leftarrow deg_1$;
  $u \in V^i.(v,u) \in E^i$;
  $c_0[u] = c_0[u] + 2(n - p[v] - p[u] - 2)(p[v] + 1)$;
  remove v from deg_1;
  $p[u] = p[u] + p[v] + 1$;
  $i++$;
  $V^i = V^{i-1} \backslash \{v\}$
  $E^i = E^{i-1} \backslash \{(v,u)\}$
  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**
  ComputeApproximateBetweenness($G^i, p, c_B[v], c_0[v], |V|$)
end
else
  $c_B[v] = c_0[v]$
end

Algorithm 3: Approximate shortest-paths vertex betweenness

ComputeApproximateBetweenness:
Data: directed graph $G = (V, E)$,
for each v:
the number of tree-nodes connected to v: $p[v]$,
the accumulator for the approximate betweenness$v$: $AB[v]$,
the betweenness algebraically computed so far v: $c_0[v]$,
the number of nodes in the original graph n
Result: the graph's node approximate betweenness $AB[v]$
pivot_number = 0;
$AB_s[v] = 0, v \in V$
max = sqrt(n)
if $max > |V|$ **then**
  $max = |V|$
end
while *pivot_number* $<$ *max* **do**
  pivot_number + +
  pivot = choose($n \in V$)
  s = pivot
  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;
1   $\delta[w] = \delta[w] + p[w]$
    for $v \in$ *P[w]* **do**
2     $\delta[v] = \delta[v] + \frac{\sigma[v]}{\sigma[w]}(\delta[w] + 1)$
      if $w \neq s$ **then**
        $AB[w] = AB[w] + \delta[w]$
3       $AB_s[w] = AB[w] + (\delta[w] * p[w])$
      end
    end
  end
end
ApproximateValue($AB, AB_s, c_0, n, max, |V|$)

Algorithm 4: Modified Brandes' algorithm

ApproximateValue:
Data: for each v:
Approximate betweenness AB,
the betweenness value depending on nodes not in the residual graph AB_s,
the algebraic betweenness computation c_0,
the number of nodes in the original graph, n
the number of pivot, max
the number of nodes in the residual graph, nr
Result: the graph's node approximate betweenness $AB[v]$
i=0;

```
1 if max = nr then
    for i < n do
      AB[i] = AB[i] + AB_s[i] + C_0[i]
    end
  else
    for i < n do
      if AB[i] ≠ 0 then
        AB[i] = AB[i] * n/max
      else
        AB[i] = c_0[i]
      end
    end
  end
```

Algorithm 5: Rescaling of the results.

		MRE Approx			Time Approx		
Graph name	Node #	ASPVB	Brandes	Ratio	ASPVB (s)	Brandes (s)	Ratio
ca-GrQc	5,242	**0.260**	0.374	**1.43**	5.359	11.475	**2.12**
as20000102	6,474	**0.394**	0.427	1.08	5.709	8.058	1.41
ca-HepTh	9,877	**0.329**	0.457	**1.38**	13.479	23.322	**1.73**
ca-HepPh	12,008	**0.353**	0.472	**1.34**	29.881	48.448	1.62
ca-AstroPh	18,772	**0.413**	0.548	**1.32**	83.516	100.566	1.20
ca-CondMat	23,133	**0.341**	0.458	**1.34**	89.589	90.286	1.01
as-caida20071112	26,389	**0.435**	0.454	1.04	74.025	126.258	**1.70**
cit-HepTh	27,770	**0.729**	0.742	1.01	209.085	211.766	1.01
cit-HepPh	34,546	0.724	**0.246**	0.34	330.874	347.646	1.05
p2p-Gnutella31	62,586	**0.362**	0.537	**1.48**	392.815	892.982	**2.27**
soc-Epinion1	75,879	**0.398**	0.466	1.17	650.055	1,586.527	**2.44**
soc-sign-Slashdot090221	82,140	**0.566**	0.595	1.05	1,154.123	2,111.585	**1.82**
soc-Slashdot0902	82,168	0.616	**0.604**	0.98	2,003.166	2,081.609	1.03
soc-sign-epinions	131,828	**0.566**	0.595	1.05	1,154.123	2,111.585	**1.83**
Email-EuAll	265,214	0.072	**0.067**	0.93	868.456	25,704.993	**29.59**
web-NotreDame	325,729	0.671	**0.539**	0.80	14,364.103	51,372.872	**3.57**

Table 4. Running time (in seconds) of the two approximate methods over selected Stanford Collection graphs, their mean relative error and their ratio (speed up factor). Each value is the mean of 10 runs with different random samples.

		MRE Approx			Time Approx		
Graph name	Node #	ASPVB	Brandes	Ratio	ASPVB (s)	Brandes (s)	Ratio
G8	3,214	**0.166**	0.272	**1.63**	0.669	2.073	**3.09**
G9	3,507	**0.222**	0.251	1.13	0.715	2.260	**3.16**
G10	3,507	0.250	**0.236**	0.94	0.687	2.161	**3.14**
G11	3,519	0.271	**0.236**	0.87	0.690	2.033	**2.94**
G15	5,023	**0.075**	0.347	**4.63**	0.912	3.750	**4.11**
G17	8,856	**0.168**	0.402	**2.39**	2.802	9.517	**3.39**
G2	14,991	**0.000**	0.023	-	-	13.988	**-**
G3	15,044	**0.022**	0.229	**10.4**	4.151	12.863	**3.09**
G4	16,723	**0.017**	0.159	**9.30**	3.607	14.440	**4.00**
G5	16,732	**0.019**	0.159	**8.36**	3.704	14.554	**3.92**
G7	16,968	**0.028**	0.158	**5.64**	5.104	14.736	**2.88**
G12	44,550	**0.050**	0.323	**6.46**	17.007	99.715	**5.86**
G13	46,331	0.070	**0.016**	0.22	5.377	130.774	**24.32**
G14	47,784	**0.028**	0.231	**8.25**	20.658	108.105	**5.23**
G16	52,143	**0.035**	0.235	**6.71**	22.431	131.889	**5.87**
G6	169,059	0.120	**0.001**	**120.00**	57.238	2,156.538	**37.67**
G1	233,377	**0.049**	0.264	**5.38**	338.383	2,461.949	**7.27**
G18	506,900	**0.166**	0.366	**2.20**	4,849.750	160,623.840	**33.12**

Table 5. Running time (in seconds) of the two approximate methods over selected Sister Collection graphs, their mean relative error and their ratio (speed up factor). Each value is the mean of 10 runs with different random samples. For G2 the sample size is the residual graph size, thus the computation is exact.

original scheme in [3], where both algorithms are executed with the same number of samples.

We notice that almost always on the graphs from the Stanford repository our combined approximations scheme gains against [3] in quality (reducing the mean relative error), even with a low percentage of tree-nodes. We also gain in speed by a factor between 3.5 and 1.7 for graphs with a large percentage of tree-nodes. The speedup factor is not as high as in the exact case since the uniform sampling size (same number of sources) eliminates one of the gain factors we have in the exact case. For the Sister Collection, due to the very high sparsity we gain substantially in speed (by a factor 3 or larger), and the error is reduced (often by an order of magnitude) in 14 tests over 18. In two cases, $G6$ and $G13$, the speed up effect is large, but the quality measure is worse. This is due to the fact that the sample size is smaller but close to the the residual graph size, thus the final scaling factor introduces a small bias. However in such cases the exact algorithm of Section 4, should be run, as there is no time gain in resorting to the approximated version.

7 Conclusions

Brandes' 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 possible additional locally sparse features of the input graph to speed up further the computation on large graphs. In this work we show that combining exact algebraic determination of betweenness centrality for some tree-like sub-graphs of the input graph, with a modified Brands' procedure on the residual graph we can gain orders of magnitudes (between one and two) in terms of computation time for very sparse graphs, and a good factor from 2 to 5, in moderately sparse graphs. Also in the approximate setting combining the algebraic technique with an adaptive sampling our experiments show gains in speed and/or precision over state of the art approximate algorithms. At the best of our knowledge this approach is novel. Among the graphs tested in this paper, we did not find a significant number of tree-nodes only in author collaboration graphs and citation graphs, while for the other categories we found a significant number of tree-nodes. We thus conjecture that this feature is common enough in a range of social networks so to make the application of our method an interesting option when exact or approximate betweenness is to be computed.

As future work 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. Moreover the impact of our approach combined with other approximation schemes will be investigated.

8 Acknowledgments

This research is partially supported by the project *BINet* "Nuova Piattaforma di Business Intelligence Basata sulle Reti Sociali" funded by Regione Toscana POR CReO 2007-2013 Programme.

References

1. J. M. Anthonisse. The rush in a directed graph. Technical Report BN 9/71, Stichting Mathematisch Centrum, 2e Boerhaavestraat 49 Amsterdam, October 1971.
2. D.A. Bader and K. Madduri. Parallel algorithms for evaluating centrality indices in real-world networks. In *Parallel Processing, 2006. ICPP 2006. International Conference on*, pages 539 –550, aug. 2006.
3. David Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. In Anthony Bonato and Fan Chung, editors, *Algorithms and Models for the Web-Graph*, volume 4863 of *Lecture Notes in Computer Science*, pages 124–137. Springer Berlin / Heidelberg, 2007.
4. M. Baglioni, F. Geraci, M. Pellegrini, and E. Lastres. Fast exact computation of betweenness centrality in social networks. In *Proceedings of the 2012 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, Istambul, Turkey, 26-29 August 2012.
5. Stephen P. Borgatti. Centrality and network flow. *Social Networks*, 27(1):55 – 71, 2005.
6. Ulrik Brandes. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology*, 25(2):163–177, 2001.
7. Ulrik Brandes. On variants of shortest-path betweenness centrality and their generic computation. *SOCIAL NETWORKS*, 30(2), 2008.
8. Ulrik Brandes and Christian Pich. Centrality estimation in large networks. *I. J. Bifurcation and Chaos*, 17(7):2303–2318, 2007.
9. Tami Carpenter, George Karakosta, and David Shallcross. Practical issues and algorithms for analyzing terrorist networks, 2002. Invited paper at WMC 2002.
10. Shu Yan Chan, Ian X. Y. Leung, and Pietro Liò. Fast centrality approximation in modular networks. In *CIKM-CNIKM*, pages 31–38, 2009.
11. Antonio Del Sol, Hirotomo Fujihashi, and Paul O'Meara. Topology of small-world networks of protein–protein complex structures. *Bioinformatics*, 21:1311–1315, April 2005.
12. Martin Everett and Stephen P. Borgatti. Ego network betweenness. *Social Networks*, 27(1):31 – 38, 2005.
13. Linton C. Freeman. A Set of Measures of Centrality Based on Betweenness. *Sociometry*, 40(1):35–41, March 1977.
14. Robert Geisberger, Peter Sanders, and Dominik Schultes. Better approximation of betweenness centrality. In *ALENEX*, pages 90–100, 2008.
15. M. Girvan and M. E. J. Newman. Community structure in social and biological networks. *Proc. Natl. Acad. Sci. USA*, pages 7821–7826, 2002.
16. O. Green, R. McColl, and D.A. Bader. Fast algorithm for incremental betweenness centrality. In *Proceeding of SE/IEEE International Conference on Social Computing (SocialCom)*, September 3-5 2012.

17. Riko Jacob, Dirk Koschtzki, Katharina Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. Algorithms for centrality indices. In Ulrik Brandes and Thomas Erlebach, editors, *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*, pages 62–82. Springer Berlin / Heidelberg, 2005.
18. Shiva Kintali. Betweenness centrality : Algorithms and lower bounds. *CoRR*, abs/0809.1906, 2008.
19. Dirk Koschatzki, Katharina Lehmann, Leon Peeters, Stefan Richter, Dagmar Tenfelde-Podehl, and Oliver Zlotowski. Centrality indices. In Ulrik Brandes and Thomas Erlebach, editors, *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*, pages 16–61. Springer Berlin / Heidelberg, 2005.
20. Min-Joong Lee, Jungmin Lee, Jaimie Yejean Park, Ryan Hyun Choi, and Chin-Wan Chung. Qube: a quick algorithm for updating betweenness centrality. In *Proceedings of the 21st international conference on World Wide Web*, WWW '12, pages 351–360, New York, NY, USA, 2012. ACM.
21. Loet Leydesdorff. Betweenness centrality as an indicator of the interdisciplinarity of scientific journals. *Journal of the American Society for Information Science and Technology*, 58:1303–1309, July 2007.
22. Kamesh Madduri, David Ediger, Karl Jiang, David A. Bader, and Daniel Chavarria-Miranda. A faster parallel algorithm and efficient multithreaded implementations for evaluating betweenness centrality on massive datasets. *Parallel and Distributed Processing Symposium, International*, 0:1–8, 2009.
23. M.E. J. Newman. A measure of betweenness centrality based on random walks. *Social Networks*, 27(1):39 – 54, 2005.
24. Rami Puzis, Polina Zilberman, Yuval Elovici, Shlomi Dolev, and Ulrik Brandes. Heuristics for speeding up betweenness centrality computation. In *Proceeding of SE/IEEE International Conference on Social Computing (SocialCom)*, September 3-5 2012.
25. Scott White and Padhraic Smyth. Algorithms for estimating relative importance in networks. In *Proceedings of the ninth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '03, pages 266–275, New York, NY, USA, 2003. ACM.