Peer-to-Peer Systems

Andrea Maggiordomo – `mggndr89[at]gmail.com`
Laurea Magistrale in Informatica, Università di Pisa

2016

1 Introduction

2 Preliminary definitions and assumptions

The task is to compute centrality indices for a given undirected graph $G = (V, E)$, which is assumed to be connected. Each node v represents an independent agent with some given computational power, and that can communicate only with its neighbors $N_v = \{u \in V : \{u, v\} \in E\}$. A path $p(s,t)$ from a source s to a destination t is a sequence of edges that connects the two endpoints. The distance $d(u, v)$ between two nodes is the length of the shortest path that connects them, while the diameter Δ of the network is the maximum distance between any pair of nodes. Note that $d(u, v) = d(v, u)$ since the network is assumed to be connected, and $d(u, u) = 0$. A node v is a predecessor of w with respect to a source s if $\{v, w\} \in E$ and $d(s, v) + 1 = d(s, w)$. The *predecessor set* $P_s(w)$ of w is the set of all predecessors of w with respect to s.

The algorithms described in this report all assume an underlying synchronous model where the computation evolves in steps: at each step all the agents perform their computations independently and autonomously, and the messages they send at step t are delivered to the destination and processed at step $t + 1$.

3 The Deccen algorithm

The specification of Deccen is based on the algorithmic scheme outlined in [4]. Initially, each node broadcasts itself on the network. Exploiting the synchronous model, after k steps any node $t \in V$ will know all the nodes $s \in V$ such that $d(s, t) = k$ and the number of shortest paths σ_{st} that links it to each of them. This information is stored locally and also reported in broadcast to allow other nodes $v \in V$ to compute the quantity $\sigma_{st}(u)$ necessary to compute the betweenness and stress centrality indices. The value $\sigma_{st}(u)$ can be determined by exploiting the following

Lemma (Bellman conditions)**.** *A node $v \in V$ lies on a shortest path from $s \in V$ to $t \in V$ if and only if $d(s,t) = d(v,s) + d(v,t)$.*

The synchronous model ensures that if a node v lies on a shortest path between s and t and receives a report for such a pair, it has already computed σ_{vs} and σ_{vt}. Then, according to the above conditions $\sigma_{st}(v) = \sigma_{vs} \cdot \sigma_{vt}$.

3.1 Message types

DISCOVERY$\langle s, u, d, \sigma_{su} \rangle$ These messages are used to track distances and the number of shortest paths from a specific origin. They contain the source $s \in V$ of the broadcast, the sender $u \in V$, the distance $d = d(s,u)$ of u from s and the number of shortest path that connect u to s.

REPORT$\langle (s,t), \sigma_{st}, d_{st} \rangle$ These messages are broadcast by t after having determined the number of shortest paths to s and the distance from it.

3.2 Node state

Each node v maintains three accumulators C_C, C_B and C_S for closeness, betweenness and stress centrality (the closeness centrality will be retrieved as $1/C_C$), a set R of node pairs for which a REPORT message has already been received, and two dictionaries D and S that associate each node $s \in V$ with the discovered distance $d(s,v)$ and the number of shortest paths σ_{sv} respectively.

3.3 Protocol initialization

Each node $v \in V$ initializes the accumulators C_C, C_B and C_S to 0, the set R to the empty set, the dictionary D so that it only contains the entry $(v, 0)$ and S so that it only contains the entry $(v, 1)$. Furthermore, it sends to all its neighbors a DISCOVERY$\langle v, v, 0, 1 \rangle$ message.

3.4 Step actions

The processing performed by each node v at each step is the following:

1. All the DISCOVERY messages having a source s for which the dictionary D contain no mapping are grouped together. (These will be the nodes "discovered" at this step).

2. Each group of messages is processed independently. For each group, let s be the source and d be the distance of all the DISCOVERY$\langle s, u, d, \sigma_{su} \rangle$ messages in it (these will be the same for all the messages), then:

   2.1. Add the entry $(s, d+1)$ to the dictionary D, so that $D[s] = d+1$.

2.2. Let $\sigma_{sv} = \sum_u \sigma_{su}$ and add the entry (s, σ_{sv}) to S. (The number of shortest paths from s to v is the sum of the number of shortest paths from s to all the predecessors of v).

2.3. Send a REPORT$\langle (s, v), \sigma_{sv}, d + 1\rangle$ message to each neighbor node.

3. For each REPORT$\langle (s, t), \sigma_{st}, d_{st}\rangle$ message such that $(s, t) \notin R$:

   3.1. If $s = v$ then $C_C \leftarrow C_C + d_{st}$.

   3.2. If $d_{st} = D[s] + D[t]$ let $\sigma_{st}(v) = \sigma_{sv} \cdot \sigma_{tv}$, then $C_B \leftarrow \frac{\sigma_{st}(v)}{\sigma_{st}}$ and $C_S \leftarrow \sigma_{st}(v)$.

   3.3. Add the pair (s, t) to the set R, and forward the REPORT message to all the neighbors.

3.5 Cost analysis

The broadcast of a DISCOVERY or a REPORT requires $O(m)$ messages. Since each nodes starts a DISCOVERY and generates $n - 1$ reports the total number of messages exchanged is $O(nm + n^2m)$, with REPORT messages inducing the dominant factor n^2m.

In terms of memory consumption, each node will add $O(n)$ entries each of the two dictionaries and $O(n^2)$ pairs to the set R.

4 Approximation of centrality indices

The main issue with the DECCEN algorithm is its computational cost both in terms of memory consumption and number of messages exchanged, rendering the algorithm impractical for networks of reasonable size.. In order to keep track of the forwarded report messages and guarantee the termination of the protocol each node needs to maintain a data structure of size $O(n^2)$, while the number of messages exchanged is $O(n^2m)$.

However, if we are for example interested in computing centrality indices in order to mitigate network congestion a simple estimation of the values could be sufficient. In the following I propose an approximation algorithm that adapts an idea originally developed for closeness centrality in [3] and expanded for the estimation of betweenness centrality in [2]. The idea is to isolate the contribution of a single node s to the centrality values of all the other nodes in the network and compute those contributions by solving a Single-Source-Shortest-Path problem starting from s. We can then compute the centrality of a node v by adding the contributions of all the other nodes to v (that is, by solving n SSSP instances starting from all the nodes and accumulating the contributions locally). In our case, the SSSP problem is converted to a decentralized Breadth First Search.

The approximation algorithms described in [3, 2] estimate the centrality values by solving the SSSP problems for a restricted set of source nodes.

Contribution of a source to Closeness centrality

The contribution of a source s to the closeness centrality value of v is simply the distance $d(s,v)$ of v from s:

$$\gamma(s|v) = d(s,v). \tag{1}$$

Contribution of a source to Betweenness centrality

The contribution of a source s to the betweenness centrality of v is the *dependency* of s on v introduced in [1]:

$$\delta(s|v) = \sum_{t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}, \tag{2}$$

that allows to rewrite the betweenness centrality of v as

$$BC(v) = \sum_{s \in V} \delta(s|v).$$

Contribution of a source to Stress centrality

The contribution to stress centrality is analogous to the betweenness centrality:

$$\sigma(s|v) = \sum_{t \in V} \sigma_{st}(v), \tag{3}$$

and the stress centrality of v is rewritten as

$$SC(v) = \sum_{s \in V} \sigma(s|v).$$

4.1 Computing contributions from a single source

As stated previously, a decentralized Breadth First Search can be adapted to compute the contribution of a source to any of the three centrality indices considered. The closeness centrality contribution is simply the distance from the source to the node (that is, the depth of the visited node in the Breadth-First tree) and it can be computed directly during the visit.

For betweenness centrality, [1] shows that dependencies of a source obey a recursive relation expressed in terms of predecessors set:

Theorem (Brandes, 2001)**.** *The dependency of $s \in V$ on any $v \in V$ obeys*

$$\delta(s|v) = \sum_{w : v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta(s|w)). \tag{4}$$

For stress centrality contributions a similar relation holds (the proof is reported in the appendix):

Theorem. *The stress centrality contribution of $s \in V$ on any $v \in V$ obeys*

$$\sigma(s|v) = \sum_{w:v \in P_s(w)} \sigma_{sv} \cdot \left(1 + \frac{\sigma(s|w)}{\sigma_{sw}}\right). \tag{5}$$

The predecessor sets of all the nodes can be easily discovered by adjusting the "descent" phase of the BFS algorithm, while contributions are computed during a backward walk from the frontier of the BF-Tree back to the source.

4.2 Random sampling of source nodes

To let the algorithm operate in a decentralized way, each node independently initiates a visit with a given probability p, which reflects the fraction of the network sampled. Even if the number of samples is not known beforehand, eventually all the nodes will become aware of the number of sources that initiated a visit (let it be k).

The contributions of sample v_i to the closeness, stress and betweenness centrality of any node u can be modeled with the following random variables

$$X_i(u) = n \cdot d(v_i, u), \quad Y_i(u) = n \cdot \delta(v_i|u), \quad Z_i(u) = n \cdot \sigma(v_i|u),$$

which are used to estimate the centrality indices as:

$$\widehat{C}_C(u) = \sum_{i=1}^{k} \frac{1}{X_i(u)/k}, \quad \widehat{C}_B(u) = \sum_{i=1}^{k} Y_i(u)/k, \quad \widehat{C}_S(u) = \sum_{i=1}^{k} Z_i(u)/k.$$

The derivations for the following theorem – which ensures the estimates are correct – are reported in the appendix.

Theorem. *The expected value of the centrality estimators are the centrality values, that is*

$$\mathbf{E}[\widehat{C}_C(u)] = C_C(u), \quad \mathbf{E}[\widehat{C}_B(u)] = C_B(u), \quad \mathbf{E}[\widehat{C}_S(u)] = C_S(u)$$

4.3 Algorithm specification

In this section the algorithm is detailed. The only parameter of the algorithm is the probability p with which each independent node decides to begin a visit of the network. The network size n is assumed to be known to all the agents in the network. Initially, every node sets to zero the estimation of each centrality index.

4.3.1 Message types

DISCOVERY$\langle s, u, d, \sigma_{su}\rangle$ Messages of this type are used during the descent from a source to build the predecessors sets at each node. Relevant fields are the source s of the visit, the sender u, the distance d of the sender to the source (that is, $d = d(s, u)$) and the number of shortest path from the source to the sender σ_{su}.

REPORT$\langle s, v, \delta(s|v), \sigma(s|v), \sigma_{sv}\rangle$ These messages are sent by a node v as part of the backtracking phase to inform its predecessors of the computed contributions and to allow them to compute their own by applying the recursive relations introduced in section 4.1.

4.3.2 Visit states

Visit states are parametric with respect to the discovery from a source $s \in V$. A node v is in state:

WAITING(s) if it has not yet received any DISCOVERY having s as source.

ACTIVE(s) if it has received one or more DISCOVERY messages with source s and has not yet computed the contributions of s to its centrality indices.

COMPLETED(s) if it has computed the contributions of s to its centrality indices, updated them accordingly, and reported the contributions to each predecessor in $P_s(v)$ by sending the appropriate REPORT messages.

4.3.3 Node state

Each node v maintains three centrality accumulators C_C, C_B and C_S like in DECCEN, and a counter k to track the number of sample nodes involved in the protocol.

Furthermore, while a node is in state ACTIVE(s) when dealing with a visit from s it will need to partition the set of neighbors N_v in three subsets: the set of predecessors $P_s(v)$, the set of siblings $S_s(v)$ and the set of children $C_s(v)$, as well as track contributions of s to its centrality scores with three parametric accumulators $C_C(s)$, $C_B(s)$ and $C_S(s)$.

4.3.4 Protocol initialization

Upon initialization, a node v clears its centrality accumulators and the counter k, and enters state WAITING(s) for all $s \in V$. Then, with probability p initiates a visit by entering state ACTIVE(v), sending a DISCOVERY$\langle v, v, 0, 1\rangle$ to every neighbor and letting $P_v(v) = \emptyset$.

4.3.5 Step actions

The actions performed by a node v at each step are the following:

1. The messages received at the current step are divided by type and then grouped together by source.
2. For each group of DISCOVERY$\langle s, u, d, \sigma_{su}\rangle$ messages having source s and distance d with v in state WAITING(s):
    - 2.1. Change state to ACTIVE(s), let $C_C(s) \leftarrow d+1$, $C_B(s) \leftarrow 0$, $C_S(s) \leftarrow 0$, collect all the senders u in the predecessor set $P_s(v)$. Let $\sigma_{sv} = \sum_{u \in P_s(v)} \sigma_{su}$.
    - 2.2. If $P_s(v) = N_v$ send a REPORT$\langle s, v, 0, 0, \sigma_{sv}\rangle$ message to all the predecessors $u \in P_s(v)$ and change state to COMPLETED(s).
    - 2.3. Otherwise, send to all $w \in N_v \setminus P_s(v)$ a DISCOVERY$\langle s, v, d+1, \sigma_{sv}\rangle$ message and change state to ACTIVE(s).
3. For each group of DISCOVERY$\langle s, u, d, \sigma_{su}\rangle$ messages having source s and distance d with v in state ACTIVE(s):
    - 3.1. Under the synchronous model assumption v can only receive such messages from nodes u such that $d(s,u) = d(s,v)$. Collect these nodes in the set $S_s(v)$ of the siblings of v with respect to s. (Note that these messages are received exactly one step after v has been contacted by its predecessors).
4. For each REPORT$\langle s, w, \delta(s|w), \sigma(s|w), \sigma_{sw}\rangle$ message received with v in state ACTIVE(s):
    - 4.1. Add w to the children set $C_s(v)$.
    - 4.2. Update $C_B(s) \leftarrow C_B(s) + \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta(s|w))$.
    - 4.3. Update $C_B(s) \leftarrow C_B(s) + \sigma_{sw} \cdot (1 + \sigma(s|w)/\sigma_{sw})$.
5. For any $s \in V$ such that v is in state ACTIVE(s), if $P_s(v) \cup S_s(v) \cup C_s(v) = N_v$ then:
    - 5.1. If $s \neq v$ update the accumulators:
        - $C_C \leftarrow C_C + C_C(s)$
        - $C_B \leftarrow C_B + C_B(s)$
        - $C_S \leftarrow C_S + C_S(s)$

      and send a REPORT$\langle s, v, \delta(s|v), \sigma(s|v), \sigma_{sv}\rangle$ message to all the predecessors contained in $P_s(v)$.
    - 5.2. Increment the counter k and change state to COMPLETED(s).

A node v can obtain the value of the estimators in the following way:

$$\widehat{C}_C(v) = \left(\frac{n}{k}\, C_C\right)^{-1}, \quad \widehat{C}_B(v) = \frac{n}{k}\, C_B, \quad \widehat{C}_S(v) = \frac{n}{k}\, C_S.$$

Note that if the algorithm is executed with $p = 1$ this is basically the decentralized version of the algorithm described in [1], and the estimators are actually the exact centrality indices.

References

[1] Brandes U. (2001). A faster algorithm for betweenness centrality. *Journal of mathematical sociology*, **25**(2), 163–177.

[2] Brandes U.; Pich C. (2007). Centrality estimation in large networks. *International Journal of Bifurcation and Chaos*, **17**(07), 2303–2318.

[3] Eppstein D.; Wang J. (2004). Fast approximation of centrality. *J. Graph Algorithms Appl.*, **8**(1), 39–45.

[4] Lehmann K. A.; Kaufmann M. (2003). Decentralized algorithms for evaluating centrality in complex networks.