Efficient Incremental Laplace Centrality algorithm for Dynamic Networks

Rui Portocarrero Sarmento, Mário Cordeiro, Pavel Brazdil, and João Gama

Abstract Social Network Analysis (SNA) is an important research area. It originated in sociology but has spread to other areas of research, including anthropology, biology, information science, organizational studies, political science, and computer science. This has stimulated research on how to support SNA with the development of new algorithms. One of the critical areas involves calculation of different centrality measures. The challenge is how to do this fast, as many increasingly larger datasets are available. Our contribution is an incremental version of the Laplacian Centrality measure that can be applied not only to large graphs but also to dynamically changing networks. We have conducted several tests with different types of evolving networks. We show that our incremental version can process a given large network, faster than the corresponding batch version in both incremental and full dynamic network setups.

1 Introduction

Centrality measures in SNA have been an important area of research as they help us to identify the relevant nodes. Researchers have invested a lot of effort to develop algorithms that could efficiently calculate the centrality measures of nodes in networks. With the explosion of social networks users, for example, like Twitter or Facebook, the networks have grown to the point where the use of batch algorithms cannot handle this data efficiently. Thus, to perform the analysis of large and changing networks, it is necessary to adopt streaming techniques and use of incremental algorithms. This way, researchers try to speed-up the process and use less memory whenever possible, by avoiding to process the full network in each iteration.

Rui Portocarrero Sarmento · Mário Cordeiro
PRODEI - FEUP, University of Porto, e-mail: up199702704@fe.up.pt, pro11001@fe.up.pt

Pavel Brazdil · João Gama
INESC TEC - LIAAD e-mail: pbrazdil@inescporto.pt,jgama@fep.up.pt

Our contribution, in this paper, is an efficient solution to calculate a particular centrality measure, the Laplacian centrality, in an incremental or dynamic setting. We present a solution that is accurate, faster than the corresponding batch algorithm for Laplacian centrality on large networks.

Succinctly, this document starts with an introduction to related work in section 2. We explain our incremental algorithm in section 3. Then, in section 4, we write about the results of our experiments with the developed algorithm. Finally, in section 5, we conclude our work and write about possible directions for future action regarding the area covered in this document.

2 Related Work

2.1 Laplacian Centrality

In [7], Xingqin Qi et al. introduced a novel centrality measure. The authors stress that their new measure based on the Laplacian energy of a node would outperform Betweenness and Closeness centrality regarding complexity. Therefore, the authors achieved a more efficient way of retrieving measurements of centrality in a network. In their paper, authors also compare their new measure with Betweenness centrality and Closeness centrality and prove the reliability of their measure by providing similar results per node. The definition of the Laplacian Centrality of a node u_i is based in the Laplacian Energy of the graph G obtained after removing the node u_i from the graph, and by following the expression [7]:

$$C_L(u_i, G) = \frac{(DE)_i}{E_L(G)} = \frac{E_L(G) - E_L(G_i)}{E_L(G)} \tag{1}$$

From equation 1, we can expect a Laplacian Centrality value between 0 and 1. These values increase correspondingly to the increased centrality of the node. Nonetheless, since the definition is in itself a normalization, it is a normal procedure to present the non-normalized values $(DE)_i$ for each node. This way, we can achieve a faster ranking computation without considering the total graph energy $E_L(G)$. The following algorithm (Algorithm 1) presents an implementation of the Laplacian Centrality measure as described in [7, 6] for unweighted networks. Presented pseudocode is based on the implementation of the Laplacian Centrality as made available by [10] and [9].

By observation of the `LapCent()` method, for each loop of the algorithm, it can be concluded that the centrality parameter is a function of the local degree plus the degree's of the neighbors (with different weights for each). Therefore, the metric is not a global measure. The local degree and the 1st order neighbors degree are all that is needed to calculate the metric for unweighted networks. The `Main()` method shows the pseudo-code required for performing batch Laplacian Centrality calculation on an evolving network. The algorithm will require performing a full

Algorithm 1 Batch Laplace Centrality Algorithm

```
1: V ← {u_1, u_2, .., u_v} , E ← {(i_1, j_1), (i_2, j_2), .., (i_e, j_e)}
2: procedure LapCent(G ← (V, E))
3:     C_entralities ← {}
4:     D_egrees ← G.degrees()
5:     V_s ← G.nodes()
6:     for each v in V_s do
7:         N_eighbors ← G.neighbors()
8:         Ioc ← D_egrees[v]
9:         nei ← 2. Σ_{i=1}^{N_eighbors} D_egrees[i]
10:        C_entralities[v] ← (Ioc^2 + Ioc + nei)
11:    end for
12:    return C_entralities, | V_s |
13: end procedure
14: procedure Main
15:    D_ataset ← {G_0, G_1, ..., G_n}
16:    for each snapshot in D_ataset do                ▷ calculation in the full network
17:        C_entralities, N_umC_entralities ← LapCent(snapshot)
18:    end for
19: end procedure
```

calculation for each one of the snapshots $\{G_0, G_1, ..., G_n\}$ included in the dataset $\mathcal{D}_{ataset}$. Notice that no Laplacian Centrality data nor network data is shared between snapshots. We will explore this inefficiency of the algorithm in Section 3 when proposing the incremental version of the algorithm.

2.2 Incremental Centrality Measures

Due to requirements in size or dynamics of networks, some centrality measures were already adapted to be incremental or dynamic. The authors of these algorithms argument their solutions are faster than the batch versions. Our objective is to achieve similar improvements, for the batch version of the Laplacian Centrality algorithm.

Incremental Betweenness Centrality:

The Brandes algorithm [1] (currently widely used), runs in $O(mn + n^2 logn)$ time, where $n = |V|$ and $m = |E|$. Nasre et al. [5] developed an incremental algorithm to perform Betweenness Centrality (BC) measures in dynamic networks. The BC score of all vertices in G is updated when a new edge is added to G, or the weight of an existing edge is reduced. Their incremental algorithm runs in $O(m'n + n^2)$ time, where m' is bounded by $m* = |E*|$, and $E*$ is the set of edges that lie on a shortest path in G. The authors explain that, even for a single edge update, their incremental

algorithm is the first algorithm that is faster on sparse graphs than recomputing with the well-known static Brandes algorithm. The authors also stress that their algorithm is also likely to be much faster than Brandes on dense graphs since $m*$ is often close to linear in n. The authors explain that, with preliminary experimental results for their basic edge update algorithm on random graphs, generated using the Erdós-Rényi model, they achieve 2 to 15 times speed-up over Brandes' algorithm for graphs with 256 to 2048 nodes, with the larger speed-ups on dense graphs.

Incremental Closeness Centrality:

Kas et al. [3] developed an incremental Closeness Centrality algorithm for dynamic networks. To compute the closeness values incrementally, for streaming, dynamically changing social networks, all-pairs shortest-paths algorithm proposed by Ramalingam and Reps [8] was extended, such that closeness values are incrementally updated, in line with the changing shortest path distances in the network. The addition of an edge between X and Y nodes would be processed by discovering affected sources, i.e., nodes that are on the path of the new edge, and the affected sinks, i.e., the nodes that are beyond the added connection. Finally, the authors update the Closeness Centrality values just for the affected nodes. The author's argument that, for Incremental algorithms, computation times can benefit from early pruning by updating only the affected parts. While the original algorithm for Closeness Centrality can be performed by running an all-pair shortest paths algorithm like Floyd-Warshall [2], which results in $O(n^3)$ time complexity, Kas et al. achieve several improvements to their 2-phase algorithm. They argument the time complexity of Phase-1 to be limited by $O(||A_{ffected}||_2)$ where the subscript 2 denotes the size of two hop neighborhood of all affected nodes. The complexity of Phase-2 is dominated by the complexity of priority queue, denoted by $O(||A_{ffected}||log||A_{ffected}||)$. The authors' effort results in significant speed-ups over the most commonly used method, on various synthetic and real-life Datasets, suggesting that incremental algorithm design is a fruitful research area for social network analysts. The speed-ups they achieve with this algorithm vary regarding the topology of the network, and for synthetic networks, they conclude that the incremental algorithm is, on average, six times faster than Dijkstra's algorithm.

3 Incremental Laplace Centrality algorithm:

The original Laplace Centrality algorithm proposed by Xingqin Qi et al. [7] was primarily designed for static networks (i.e., networks that do not evolve). Nevertheless, being a static algorithm, it can be used to calculate centralities in changing networks with the penalty of performing a full computation of the centralities in each one of the network snapshots. With our proposal, the same Xingqin Qi et al. [7] principles were adapted for an incremental algorithm. The proposed incremental algorithm

presents better computational efficiency, by performing selective Laplace Centrality calculations only for the nodes affected by the addition and/or removal of edges in each one of the snapshots (i.e., it reuses information of the previous snapshot to perform the Laplace Centrality calculations on the current snapshot). Although the algorithm is called incremental, because it avoids full calculations, in each one of the snapshots, it is prepared for the addition and removal of edges in each one of the increments (i.e.: *full dynamic* algorithm).

3.1 Locality of the Laplacian Centrality

(a) Batch Algorithm (b) Incremental Algorithm

Fig. 1. Calculated node centralities with edge {(4, 6)} added. Dark grey nodes affected by addition of edges. Light grey nodes centralities need to be calculated due to their neighbourhood with affected nodes.

Table 1. Example of Centrality calculation for the network presented in Fig. 1.

	Batch				**Incremental**			
Node	**step=1**		**step=2**		**step=1**		**step=2**	
	Centrality	Calculated	Centrality	Calculated	Centrality	Calculated	Centrality	Calculated
1	6	yes	6	yes	6	yes	6	no
2	12	yes	12	yes	12	yes	12	no
3	18	yes	18	yes	18	yes	18	no
4	18	yes	28	yes	18	yes	28	yes
5	34	yes	38	yes	34	yes	38	yes (neighbour)
6	10	yes	20	yes	10	yes	20	yes
7	18	yes	20	yes	18	yes	20	yes (neighbour)
Total:	7 out of 7 nodes		7 out of 7 nodes		7 out of 7 nodes		4 out of 7 nodes	
	14 centrality calculations				11 centrality calculations			

As already stated before, the Laplacian Centrality metric is not a global measure, i.e., is a function of the local degree plus the degree's of the neighbors (with different weights for each). Xingqin Qi et al. [7, 6], and the pseudo-code presented in Algorithm 1, shown that local degree and the 1st order neighbors degree is all that is needed to calculate the metric for unweighted networks. We will use the toy network example in Fig. 1 to show the locality of the Laplacian Centrality. In this toy network we consider two distinct snapshots $G_0 = \{(1,2),(2,3),(3,5),(5,6),(5,4),(4,7),(5,7)\}$, in the second snapshot G_1 a new edge $(4,6)$ is added to the network, so $G_1 = G_0 \cup \{(4,6)\}$. In the fist snapshot (G_0), it will be needed to calculate centralities for all the nodes. In the second snapshot (G_1), once that only local degree and 1st order neighbors degree will affect the centrality values for nodes 4 and 6, The algorithm will just require to calculate the centralities for nodes 4 and 6 (i.e.: affected nodes) and nodes 5 and 7 (1st order

neighbors). Due to the determinism of the algorithm, the node centrality results for the incremental Laplace algorithm are equal to the results of the batch version of the same algorithm. This is explained by the fact that we are not dealing with probabilistic phenomena nor any randomness in the initialization. The results of both versions of the algorithm have the same values for each node in the dynamic graph of Fig. 1, as we can see in Table 1. The obtained values for centrality are the same for both algorithms (expected), but for step 2 in the incremental algorithm, we only calculated the centralities of the affected nodes plus their neighbors. In total, the Batch algorithm performed 14 node centrality calculations (7 for step 1 and 7 for step 2) while the incremental achieved the same result using only 11 node centrality calculations (7 for step 1 and 4 for step 2).

3.2 Algorithm

Based on the assumptions devised in the Section 3.1 and having by reference the batch version of the Laplacian centrality shown in Algorithm 1, a new incremental algorithm is here presented (Algorithm 2). This incremental algorithm achieves better efficiency than the batch version by performing Laplace centrality calculations only in the nodes affected by addition or removal of edges and their 1st order neighbors (full dynamic incremental algorithm). In Algorithm 2, the incremental method `LapCentAddRemove()` receives as parameters a full graph (network of the previous snapshot), the lists of edges that will be added and removed from the graph in the current iteration (A_{dd} and R_{emove} respectively), and the previously calculated centralities to be updated in the current iteration ($C_{entralities}$). Previously to the beginning of the `for each` cycle, the algorithm will calculate the set of nodes affected by the addition and removal of nodes ($\mathcal{V}_s$) and the list of 1st order neighbours of affected nodes ($\mathcal{V}_f$). Laplace centrality will only be calculated for this two sets of nodes using the same centrality calculation function employed in the batch algorithm. Function `LapCentAddRemove()` returns the updated graph G, the updated centrality list $C_{centralities}$, and $\mid \mathcal{V}_f \mid$ the number of nodes for which new centralities were calculated for the current iteration. In the main function, initially, in the first iteration, the centralities are calculated for the full network, and this information if reused in the following iterations. In each of the incremental steps, the function `LapCentAddRemove()` only receives the list of edges that changed from the previous iteration. By analysis of the proposed algorithm, we can conclude that obtained efficiency can be related to the number of edges that change in each of the snapshots, and also the degree of its nodes. Higher degrees will require performing more computations of 1st order neighbours.

The complexity of the original algorithm [7] is $O(n*D^2)$, where n is the number of vertices and D as the maximum degree ($D = max_{v \in V(G)} d_v$). Thus, the total complexity for computing Laplacian centrality for network G with n vertices, m edges and maximum degree D would be no more than $O(n*D^2)$. Nonetheless, according to [10] it should be something like $O(n*a)$, where a is the average number of neigh-

bours for the entire graph, and n are the number of nodes. In the worst case, a is the maximum number of neighbours any node has in the graph. Our improvement of this algorithm brings the use of locality features of the original algorithm to lower the complexity to $O(n' * a)$, where n' are the affected nodes in each snapshot of the evolving changes of the networks. The affected nodes by addition or removal of m' edges is given by $n' = (2 * m' + 2 * m' * a)$, i.e.: the sum of 2 nodes per modified edge $(2 * m')$ and their respective 1st order neighbours $(2 * m' * D)$ or $(2 * m' * a)$ with [10] assumption. Final time complexity is $O(2 * m' * a + 2 * m' * a^2)$.

Algorithm 2 Incremental Laplace Centrality Algorithm

$V \leftarrow \{u_1, u_2, .., u_v\}$, $E \leftarrow \{(i_1, j_1), (i_2, j_2), .., (i_e, j_e)\}$
$A_{dd} \leftarrow array\{(i_1, j_1), .., (i_n, j_n)\}$, $R_{emove} \leftarrow array\{(i_1, j_1), .., (i_n, j_n)\}$
procedure LAPCENTADDREMOVE($G \leftarrow (V, E)$, A_{dd}, R_{emove}, $C_{entralities}$)
    $V_s \leftarrow \{\}$
    for each *edge* **in** A_{dd} **do**
        $V_s \leftarrow V_s \cup edge.source() \cup edge.destination()$
        $G.add\_edge(edge)$
    end for
    for each *edge* **in** R_{emove} **do**
        $V_s \leftarrow V_s \cup edge.source() \cup edge.destination()$
    end for
    $V_f \leftarrow \{\}$
    for each *node* **in** V_s **do**
        $V_f \leftarrow V_f \cup G.neighbors(node)$
    end for
    for each *edge* **in** R_{emove} **do**
        $G.remove\_edge(edge)$
    end for
    $D_{egrees} \leftarrow G.degrees()$
    for each v **in** V_f **do**
        $N_{eighbors} \leftarrow G.neighbors()$
        $Ioc \leftarrow D_{egrees}[v]$
        $nei \leftarrow 2. \sum_{i=1}^{N_{eighbors}} D_{egrees}[i]$
        $C_{entralities}[v] \leftarrow (Ioc^2 + Ioc + nei)$
    end for
    return $C_{entralities}$, $\mid V_f \mid$, G
end procedure
procedure MAIN
    $D_{ataset} \leftarrow \{G_0, G_1, ..., G_n\}$, $A_{dd} \leftarrow \{A_0, A_1, ..., A_n\}$, $R_{emove} \leftarrow \{R_0, R_1, ..., R_n\}$
    $C_{entralities}$, $N_{um}C_{entralities} \leftarrow$ LAPCENT(G_0) ▷ initial step in the full network
    $G \leftarrow G_0$, $i \leftarrow 1$
    while $(i \leq |D_{ataset}|)$ **do** ▷ calculate centralities for the increments
        $C_{entralities}$, $N_{um}C_{entralities}$, $G \leftarrow$ LAPCENTADDREMOVE($G, A_{dd}[i], R_{emove}[i], C_{entralities}$)
    end while
end procedure

4 Results

The Dynamic Laplace Centrality algorithm was evaluated in incremental and dynamic network setups, Section 4.1 and Section 4.2 respectively. Results for incremental networks were obtained for the High-energy physics theory citation network [4], using the original Laplace Centrality (now on called Batch) and the proposed Dynamic Laplace Centrality (now on called Dynamic) in an incremental setting configuration. The original Laplace Centrality served as a baseline. In this setup, we considered the 136 snapshots of the dataset, with snapshots built by aggregating timestamps of citations in a monthly basis. In the Batch algorithm, for every snapshot, the centralities were calculated having the full network as input. For the Dynamic algorithm, in the first snapshot the full network is passed as input, and in the following snapshots, the algorithm only receives the set of edges added to the network in that snapshot (incremental). For evaluating the Dynamic Laplace Centrality algorithm in a dynamic network setup (addition and removal of edges between snapshots), two datasets were used. The Autonomous systems AS-733 dataset [4] with 733 snapshots from November 8 1997 to January 2 2000 and CAIDA AS Relationships Datasets [4] with a total of 122 snapshots from January 2004 to November 2007. The performed empirical evaluation consisted mainly in comparing run times of each increment (duration of each increment and cumulative execution time). Additionally, the size of the network (number of nodes and edges), the number of added/removed edges in each snapshot (negative values mean more edges were removed than added), and the total number of calculated centralities for each snap shot was also registered. Remarks that the batch algorithm will always need to calculate centralities for all nodes in the snapshot while it is expected that the Dynamic algorithm only performs centrality calculations for the affected nodes. In the end, an analysis of the total speed-up ratio obtained in each of the steps is added. For all the experimentation and development, we used an Intel (R) Core (TM) i7-4702MQ processor computer with 8 GBytes and SSD HDD. Three runs per algorithm/ dataset were performed with values presented in graphs as the average values of each one of those three runs.

4.1 Incremental Networks

In this subsection, we introduce the reader to the results obtained by the incremental setup of the algorithm. The results are presented in the following Fig. 2 and are related to the High-energy physics theory citation network [4] in an incremental network setting where no edges are removed from previous snapshots. Fig. 2 – Network Size, shows the variation of the network over the 136 snapshots regarding the number of nodes and number of edges. Fig. 2 – # Added Edges, shows the number of added edges in each snapshot. Notice that the total number of edges in this dataset increases over the time. In the first snapshots a few edges are added to the network, but at the final snapshots, more than 6000 edges are added in each snap-

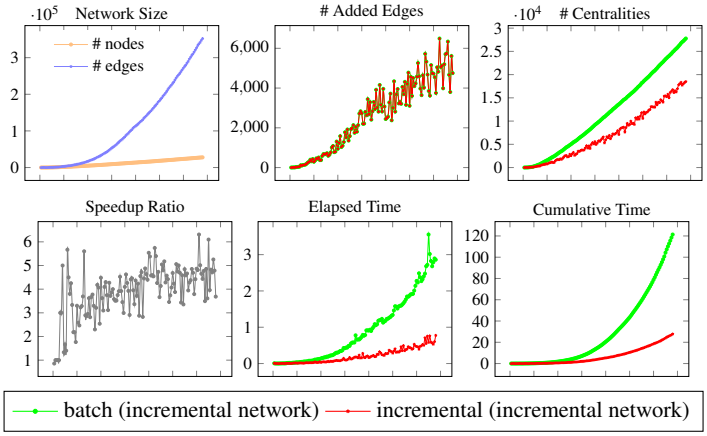

Fig. 2. Results for the Cit-HepTh network (incremental only)

shot. Fig. 2 – Speedup Ratio, shows that the number of calculated centralities in the incremental version is much lower than in the batch version. The batch version requires to compute centralities for all nodes in the snapshot as can be seen on Fig. 2 – # Centralities. Fig. 2 – Elapsed Time, shows the time required to perform the centralities measurements in each increment. This figure also shows that the incremental version is not only more efficient but also deals better with both increases in the size of network and increase in the number of added edges. This is confirmed by the total time required for processing the whole network: 27,806 seconds of the incremental version compared to the 121,345 seconds of the batch. It is clear that, as the number of added edges increases, the processing elapsed time of the batch version of the algorithm grows much faster than the incremental algorithm. Thus, regarding speed-up ratio, we achieve a speed-up of up to 6 times the processing time of the batch version with an incremental network (Fig. 2 – Speedup Ratio).

4.2 Full Dynamic Networks

Regarding the evaluation of full dynamic networks, the Autonomous systems AS-733 dataset with 733 snapshots was used. Fig. 3 – Network Size, shows the evolution of the network concerning number of nodes and edges over the time. Please note that this setting contains removal of edges and therefore, Fig. 3 – # Added Edges, presents negative values (more removed edges that added for some snapshots). Fig. 3 – # Centralities, again shows that the batch version calculated centralities for all nodes in each snapshot, while the incremental version did that only for affected nodes. It can be observed by this figure that the network has many edges and nodes variation causing significant change on the Elapsed Time required to compute centralities in each of the snapshots. Although this variation is also seen in the batch version (Fig. 3 – Elapsed Time), the incremental version keeps elapsed times per

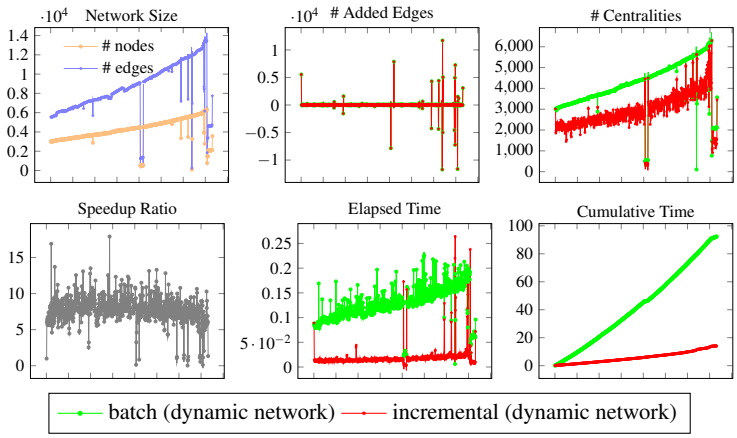

Fig. 3. Results for the as-733 network (full dynamic)

snapshot lower resulting in more efficient final execution times: 14,163 seconds for the incremental version with the batch took 92,362 seconds. With this setup, we can see that the removal and addition of edges provokes an even more pronounced increase of speed-up ratio between the batch version and the dynamic version. The speed-up ratio achieves values of more than 15 times in some operations. In the second dynamic network setup, the evolution of the CAIDA AS Relationships Datasets over its 122 snapshots is shown in Fig. 4 – Network Size. This network also has negative values for # added edges, but it might be seen as a more stable network than the AS-733 dataset. Both number of affected centralities (Fig. 4 – # Centralities), and elapsed time per iteration (Fig. 4 – Elapsed Time). It can be observed that the removal and addition of edges, in this setup, provokes a pronounced increase of speed-up ratio between the batch version and the dynamic version. The speed-up ratio achieves values of up to 5 times in some operations with this network.

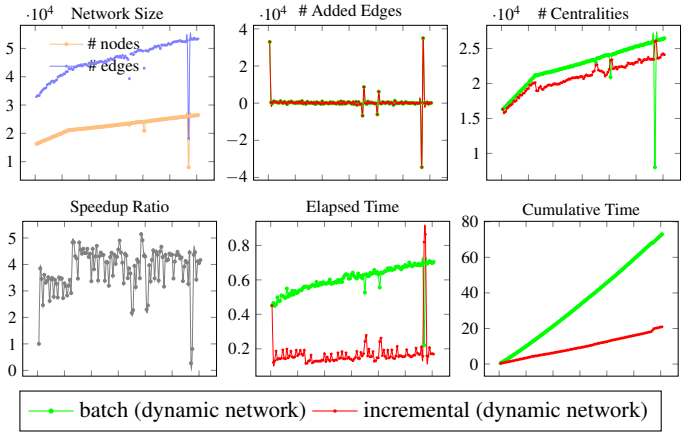

Fig. 4. Results for the as-Caida network (full dynamic)

4.3 Discussion

Table 2 show that maximum values of speed-up were obtained for step 122 in the High-energy physics (6,310 times), step 286 for AS 733 (17,909 times) and step 77 for AS-Caida (5,141 times). It can be seen that bigger speedups are obtained for small changes in the network where the number of calculated centralities decreases significantly between snapshots. Regarding the initialization of the algorithm (step 1), depending on the initial network size, the incremental algorithm achieves the same speed-up for networks above 3k nodes / 5,5k edges, but for smaller networks, it can take more time than the batch algorithm in the first iteration. The degree of the affected nodes by addition or removal of nodes could also change the speed-up ratio, higher average degrees or dense networks reduce the speed-up ratio. Networks with high variability of addition and removal of nodes can also reduce the speed-up ratio. Finally, large or very large networks with smaller network changes between snapshots achieve the maximum speedup values.

Table 2. Overview of the speed-up values achieved for the 3 datasets. Table shows the values for the initial step (full network for both algorithms), minimum, maximum and average. Apart from the speedup values, the conditions in terms of network size and number of added/removed edges per increment and the respective number of centralities calculated are also presented.

		step	speedup	network size		centralities		added edges	
				# nodes	# edges	batch	dynamic	batch	dynamic
High-energy physics	init:	1	0,857	4	2	4	4	2	2
	min:	2	1,000	9	6	9	7	4	4
	max:	**122**	**6,310**	**24023**	**280041**	**24023**	**15023**	**3751**	**3749**
	average:		3,910	11784	110842	11784	7233	2594	2593
AS 733	init:	1	0,989	3015	5539	3015	3015	5539	5539
	min:	639	0,027	103	248	103	5629	-11719	-11719
	max:	**286**	**17,909**	**4020**	**8030**	**4020**	**2268**	**-1**	**-1**
	average:		7,755	4180	8533	4180	2919	11	11
AS Caida	init:	1	1,000	16301	32955	16301	16301	32955	32955
	min:	114	0,269	8020	18203	8020	25997	-34488	-34488
	max:	**77**	**5,141**	**24013**	**49332**	**24013**	**21057**	**243**	**243**
	average:		3,818	22518	45775	22518	20973	441	441

5 Conclusions and Future Work

In this paper, we presented an incremental and dynamic setup of the Laplacian Centrality algorithm. Through empiric experiments, we have shown that, the incremental and the dynamic setup of the algorithm are faster than the batch version. Both settings have shown improvements over the batch version of the algorithm, with both types of evolving networks. Additionally, the dynamic algorithm can achieve a speed-up of more than 15 times when compared to the batch algorithm. In the future, we plan to extend our research, by comparing the incremental setup of the algorithm with other incremental centrality measures like, for example, Betweenness

Centrality and Closeness Centrality. For this purpose, we expect to improve results by achieving further optimization, either for weighted or unweighted networks.

Acknowledgements

This work is financed by the ERDF – European Regional Development Fund through the Operational Programme for Competitiveness and Internationalisation - COMPETE 2020 Programme within project «POCI-01-0145-FEDER-006961», and by National Funds through the FCT – Fundação para a Ciência e a Tecnologia (Portuguese Foundation for Science and Technology) as part of project UID/EEA/50014/2013. Rui Portocarrero Sarmento also gratefully acknowledges funding from FCT (Portuguese Foundation for Science and Technology) through a PhD grant (SFRH/BD/119108/2016)

References

1. Brandes, U.: A faster algorithm for betweenness centrality. Journal of Mathematical Sociology **25**, 163–177 (2001)
2. Floyd, R.W.: Algorithm 97: Shortest path. Commun. ACM **5**(6), 345– (1962). DOI 10.1145/367766.368168. URL http://doi.acm.org/10.1145/367766.368168
3. Kas, M., Carley, K.M., Carley, L.R.: Incremental closeness centrality for dynamically changing social networks. In: Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM '13, pp. 1250–1258. ACM, New York, NY, USA (2013). DOI 10.1145/2492517.2500270. URL http://doi.acm.org/10.1145/2492517.2500270
4. Leskovec, J., Kleinberg, J., Faloutsos, C.: Graphs over time: densification laws, shrinking diameters and possible explanations. In: Proceeding of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining - KDD '05, p. 177. ACM Press, New York, New York, USA (2005). DOI 10.1145/1081870.1081893
5. Nasre, M., Pontecorvi, M., Ramachandran, V.: Betweenness centrality - incremental and faster. CoRR **abs/1311.2147** (2013)
6. Qi, X., Duval, R.D., Christensen, K., Fuller, E., Spahiu, A., Wu, Q., Wu, Y., Tang, W., Zhang, C.: Terrorist Networks, Network Energy and Node Removal: A New Measure of Centrality Based on Laplacian Energy. Social Networking **02**(01), 19–31 (2013). DOI 10.4236/sn.2013.21003
7. Qi, X., Fuller, E., Wu, Q., Wu, Y., Zhang, C.Q.: Laplacian centrality: A new centrality measure for weighted networks. Inf. Sci. **194**, 240–253 (2012). DOI 10.1016/j.ins.2011.12.027. URL http://dx.doi.org/10.1016/j.ins.2011.12.027
8. Ramalingam, G., Reps, T.: An incremental algorithm for a generalization of the shortest-path problem. J. Algorithms **21**(2), 267–305 (1996). DOI 10.1006/jagm.1996.0046. URL http://dx.doi.org/10.1006/jagm.1996.0046
9. igraph core team, T.: igraph - python recipes. http://igraph.wikidot.com/python-recipes#toc4 (2014). [Online; accessed July-2017]
10. Wheeler, A.P.: Laplacian centrality in networkx (python). https://andrewpwheeler.wordpress.com/2015/07/29/laplacian-centrality-in-networkx-python/ (2015). [Online; accessed April-2017]