ISSN 2281-4299

DIPARTIMENTO DI INGEGNERIA INFORMATICA
AUTOMATICA E GESTIONALE ANTONIO RUBERTI

SAPIENZA
UNIVERSITÀ DI ROMA

Fully-decentralized computation of importance measures in dynamic evolving networks

Gianluca Amori
Luca Becchetti
Giuseppe Persiano
Andrea Vitaletti

Technical Report n. 6, 2013

Fully-decentralized computation of importance measures in dynamic evolving networks

Gianluca Amori*
gianluca.amori@gmail.com

Luca Becchetti*
luca.becchetti@dis.uniroma1.it

Giuseppe Persiano†
giuper@dia.unisa.it

Andrea Vitaletti*
andrea.vitaletti@dis.uniroma1.it

Abstract

With the growing diffusion of devices with wireless communication capabilities as well as the success of social networking platforms, it has become more and more interesting to study dynamic evolving networks, in which the set of connections (however defined) between the agents in the network varies over time. For instance, ad-hoc mobile networks are evolving networks in which a number of mobile hosts are free to move about a given area and capable – when close enough – of interacting with each other over wireless links without the need of an underlying backbone network. Other examples include P2P networks and in general social network contexts, in which the users dynamically establish and terminate social interactions. The topology of such networks changes over time, as edges (either directed or undirected) that represent interactions between nodes are dynamically added or removed in the network graph. Computing measures of centrality in such scenarios can be a challenging task. Classic measures of centrality and nodes' importance from graph theory and network analysis can be computed by a centralized entity on an aggregated representation of a dynamic network. However, privacy and/or scalability issues, or simply the absence of central coordination, may suggest a fully decentralized approach in which the computation is carried out by each node considering its own interactions with other nodes in the network. In this article we propose lightweight algorithms for computing some importance measures of nodes in a dynamic evolving network in a fully decentralized way, without any knowledge of the whole network structure or assumptions on its future evolution. In particular, the main part of our work regards the computation of decentralized estimations of Google's PageRank and its theoretical analysis as a problem of random walks on dynamic graphs. We also introduce algorithms for computing some classical degree centrality measures with efficient use of resources. As it turns out, while straightforward in a centralized setting, some of these measures are hard to compute in a fully decentralized

*Sapienza University of Rome, Department of Computer, Control, and Management Engineering
†Università di Salerno, Dipartimento di Informatica

way. We analyze all these algorithms in terms of hardware resources (storage space and computational power) required at each node as well as time complexity and network overhead in the transmissions, showing how they are implementable also on low-power devices such as RFID tags. We also run simulations of our algorithms on real-world dynamic evolving network data and show their performances with respect to centralized computations of analogous measures on aggregated static representations of such networks.

Keywords: Dynamic Networs, Importance Measures, PageRank

1 Introduction

In the last years there has been a great diffusion of devices with wireless communication capabilities as well as an impressive boost in the popularity of social networking platforms. A huge amount of empirical data sets about social interactions have become available, and a great number of possible applications for dynamic network systems have come to light as well as the possibility to better study the behaviour of people in dynamic contexts . Such network applications are usually formed by a large number of agents, and it can be difficult to have a centralized point of aggregation in which to keep track of the mutual interactions between the agents. Collecting this kind of information can be very interesting for studying the behaviour of people among their peers or for analyzing the spread of information within a group of social users.

In network analysis, much interest has been given to the problem of computing the relative importance of nodes within a network with respect to the structure of the network itself. Even though most of such *centrality measures* are defined on static networks, whose structure is fixed, a great number of networks actually has a dynamic nature, in that they evolve over time by means of addition and deletion of edges connecting pairs of nodes. The usual approach when dealing with such *dynamic (evolving) networks* is to consider an equivalent static representation of the original dynamic evolution process. However, this process of information aggregation can fail to fully capture the dynamic behaviour of the original network. Moreover, lack of central coordination can make it impossible to take into consideration such a centralized approach. All these motivations may suggest a fully decentralized approach in which the computation is carried out by each node considering its own interactions with other nodes in the network, without any central point of computation or information aggregation.

1.1 Evolving networks

In a dynamic evolving network the set of connections (however defined) between the agents in the network varies over time. This implies that the topology of the corresponding network graph changes over time as the connections between agents (that represent edges/arcs in the graph) are established or terminated. The problems of routing [21, 39] and distributed computation [22] in dynamic networks have been considered, but the usual approach is to make assumptions on the future evolution

of the network, either assuming that it will eventually stabilize and stop changing or imposing restrictions on the connectivity – for instance, assuming that the network always remains connected over time. However, in real-world networks it is in general impossible to make such assumptions, due to the unpredictability of the agents' actions.

Many real-world networks have been studied in the past in order to attempt to model their properties. These networks can be devided in four loose categories [32].

- Social networks, composed by a group of people with some pattern of contacts or interactions between them (friendship, business relations, family ties, etc.)
- Information networks, in which the structure defined by the interactions between nodes depends on the information stored at each node.
- Technological networks, designed typically for distribution of some commodity or resource on actual physical links.
- Biological networks, which represent biological processes.

Many of these real-world networks are actually evolving networks, since the interactions between nodes change over time.

For instance, consider the most important example of information network: the World Wide Web, which is composed by web pages containing information and linked one to another through hyper-links. Although it is often analyzed as a static graph, the structure of the web is inherently evolving since hyper-links to/from pages can be added or deleted in a dynamic fashion. Another classical information network is a file sharing peer-to-peer network, in which links between nodes (corresponding to the actions of sharing files) dynamically appear and disappear depending on file requests as well as nodes joining and leaving the network.

Social networks have also inherent dynamic patterns, since a key characteristic of social interactions is their continual change. The definition of "interaction" greatly depends on the specific application. For instance, it can represent common interests between two users, like friendship relationships and joining the same group membership, or it can represent other social interactions like phone calls or exchanges of emails. It is clear how these kinds of interactions are not static but rather change over time according to the users' actions.

Another example of evolving network are ad-hoc mobile networks, in which a number of mobile hosts capable to communicate with each other over wireless links (for instance people carrying transmission devices) are free to move in every direction. Such connections only depend on whether two hosts are sufficiently close to each other – within the so-called transmission range – without the need of an underlying backbone network, and it is straightforward how connections between nodes dynamically change because of their nomadic behaviour. The transmission range can vary depending on the technology used for the wireless transmissions (e.g. Bluetooth or RFID). Possible applications of such networks include vehicular ad-hoc networks, for communication among vehicles and traffic management, mobile social networks, military networks, conference networks, etc. As we see in Chapter 4, two of the network traces on which we run simulations are ad-hoc mobile networks derived from

from real experiments in which groups of people and their mutual interactions were tracked using RFID technology.

1.2 Motivations

Classical centrality measures find the relative importance of a node within a static network. For instance, in the context of social networks it is useful to have a measure of how a member is seen as "influential" or "popular" or how well it is connected to all the other nodes, so to have the possibility to weight the exchanges of information with a measure of the importance of their source. In general, identifying the most important elements of a network can be useful in a wide range of applications which are built upon social interactions. This is also true for the evolving case, and it is important to study the ability of uncoordinated evolving networks, like for instance people in a social event, of performing basic distributed computations.

For instance, decentralized centrality indeces in an evolving network can be very useful in the context of file sharing P2P networks. Such values of importance could represent a valid measure of nodes' level of "authority", useful when establishing sharing links in order to select the most reliable peers among all those available. Of course, in this scenario there is no central coordination and it is therefore crucial to be able to perform a fully decentralized computation at each node of the P2P network. Centrality measures in evolving networks can also have a role in routing schemes within Pocket Switched Networks [19, 18], a kind of Delay Tolerant Networks which aim to enable communications between mobile users in the absence of end-to-end connectivity. In such contexts it could be possible to design a forwarding scheme in which a node forwards messages only to nodes with a centrality value higher than its own, in order to have higher probability of the message reaching its destination without an expensive flooding approach. Also in this case a fully decentralized approach is highly advisable. Another possible application is in implementing a fully decentralized recommender platform in a multi-agent system, in which mobile users can share information about their preferences on products, personal interests, etc. In such a system, recommendations received by "important" users should be considered much more valuable, since they also indirectly include recommendations from a high number of other nodes with which there were probably no direct interactions. Central nodes can aggregate a great number of information and can thus be assigned a higher level of "trust" in their predictions. In general, this decentralized importance indeces prove themselves very useful whenever the concept of "popularity" is meaningful in an application, especially when the network is composed by a large number of nodes and it is too expensive (or even completely unfeasible) to have central coordination.

There are two main advantages of a decentralized approach over a centralized one:

- no expensive/unfeasible central coordination is needed, since the computation is carried out by the single nodes;
- there is no central point of failure; indeed, the network still survives even in the presence of failure of single nodes.

On the other hand, there are disadvantages as well. For instance there is no knowledge of the whole network structure, and nodes can only rely on their interactions with other nodes in the network to carry out the computations. As it turns out, this introduces new problems which can be hard to solve in a decentralized scenario. This also implies that it can be difficult or even impossible for an individual node to have a reasonable global view of the whole network. Luckily, for a wide range of applications it is not actually necessary to have global information to solve a task; such a *locality property* allows nodes to compute some importance indeces while interacting only with their local vicinity.

1.3 Our contribution

In this article we focus on computing some importance measures of nodes in a dynamic evolving network in a fully decentralized way, without any knowledge of the whole network structure or assumptions on its future evolution. Although straightforward in centralized systems, some degree-based centrality measures can be hard to compute in a fully decentralized way, depending on how one defines the degree of a node in an evolving network. For instance, to incrementally compute the degree of a node in an evolving graph as the number of distinct other nodes that it encountered would require to solve the distinct elements counting problem, which can be too expensive to be solved exactly. On the other hand, we see how very simple algorithms for the decentralized computation of eigenvector centrality measures can be designed that achieve good approximations of the same measures computed on equivalent aggregated representations of the dynamic networks by a central entity.

The most important part of our work is the design of two algorithms for computing fully decentralized estimations of Google's famous PageRank at each node, as well as their theoretical analysis as a problem of random walks on dynamic graphs. We also introduce algorithms for efficiently computing two measures of degree centrality with little performance requirements, so to be implementable in a wider range of real-world low-power devices. We see how the distinct elements counting problem arises and techniques to approximately solve it. Moreover, as an extension of the degree centrality problem we show an algorithm for computing at each node an estimation of the size of its local vicinity, that is the network component to which the node belongs. In combination with the previous algorithms, this can be used by a node to have an estimation of the percentage of other nodes in its vicinity it interacted with. Finally, we experimentally simulate the algorithms on three different dynamic network data sets derived from the real world, and we analyze the results with respect to analogous computations made on static representations of the same networks.

1.4 Structure of the article

This article is structured as follows.

- In Chapter 2 we present some notions useful to better understand the rest of our work, including literature on the subject of evolving networks as well as other previous works at the basis of our own contributions.

- In Chapter 3 we present our analysis on the decentralized computation of centrality measures in evolving networks, introducing our algorithms and analyzing each of them in terms of hardware resources required at each device, time complexity and network overhead in the transmissions.
- In Chapter 4 we evaluate the algorithms though extensive simulations, describing the three real network traces on which the experiments are made, the experimental setup and the performance metrics considered; we compare the results of the algorithms with respect to analogous measures computed on aggregated static representations of our sample networks.
- In Chapter 5 we briefly summarize our work and the results obtained, as well as possible future developments.

2 Evolving social networks

In this chapter we present some related work on the subjects of evolving social networks, definitions of centrality measures in an evolving contex and other concepts that will be useful for better understanding the rest of our work.

2.1 The Evolving Graph Model

The traditional representation of a static network is a graph $G = (V, E)$ in which V represents the set of nodes of the network and $E \subseteq V \times V$ the set of edges (either directed or undirected) connecting pairs of nodes. However, this kind of static representation does not take into account the time parameter and it is therefore not suited to represent a network evolving over time.

The most general representation of a dynamic (evolving) network is the Evolving Graph (EV) model [39, 4, 12], in which an evolving network is represented by a dynamic graph $\mathcal{G} = G_0, G_1, \ldots$, a sequence of static graphs over the same fixed set of nodes V. Each graph $G_t = (V, E_t)$ of the sequence represents a snapshot of the network at a given time step (or round) t. The sets of edges $E_t \subseteq V \times V$ represent the connections between the nodes in the network in time step t, either directed or undirected. In this general model, we put no restrictions on the changes of the topology over time, thus allowing any given sequence of additions and removals of edges. However, some conditions about the connectivity of the single graphs may be needed to ensure *explorability* of the network and to guarantee a finite cover time.

The duration of the time steps can be seen as a "refresh interval" of the network, indicating how frequently the network graph is updated; of course, lower time step durations lead to a more accurate discrete representation of the actual (continuous) process, raising, on the other hand, space consumption for storing the sequence of graphs. The time step duration can be arbitrarily chosen and in general depends on the nature of the network being studied. For instance, tracking the movements of people in a room or a building with the expectancy of many contacts between them would suggest it better to frequently monitor their interactions, therefore choosing low time step durations – one second, for instance. In other applications with fewer

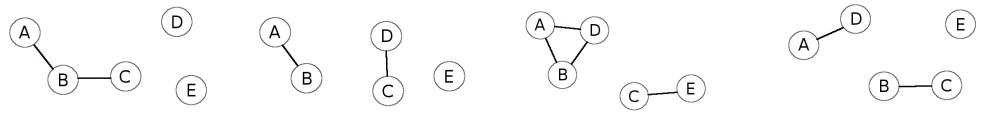
and more sparsely distributed interactions, instead, it makes sense to choose higher time step intervals.

However, sometimes it can be useful to have a static and compact structure to represent and study a dynamic network. This type of structure can be used for a standard network analysis to find "important" nodes with respect to different metrics. Given the evolutionary nature of such a network, though, it can only represent some sort of approximate aggregated information about the network in a given time period; a more accurate approach would be to consider the evolution of the network over time in both its representation and analysis.

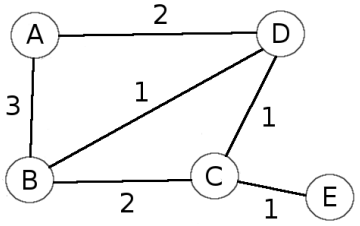
One possible static representation [26] of a dynamic network in time period $[0, T]$ is by means of a weighted graph $G^T = (V, E^T, w)$ where

- V is, as before, the fixed set of nodes in the network
- E^T is the set of edges that appeared at least once in the network in time period $[0, T]$, to which are assigned weight labels; $\forall (u, v) \in V \times V$, $(u, v) \in E^T_w$ if and only if $\exists t \in [0, T]$ such that $(u, v) \in E_t$
- a weight function $w(u, v) \to \mathcal{R}$ assigns to each edge $(u, v) \in E$ a weight.

For instance, the weight of an edge can represent a measure of its frequency in the network in time period $[0, T]$; $w(u, v) =$ number of time steps $t \in [0, T]$ such that $(u, v) \in E_t$. Figure 1 shows an example of how an undirected dynamic network can be statically represented in this way. For the directed case, an analogous example can be made.



(a) Snapshot at t_0 (b) Snapshot at t_1 (c) Snapshot at t_2 (d) Snapshot at t_3



(e) Aggregate graph

Figure 1: Example of how a dynamic network over four time steps t_0, t_1, t_2 and t_3 (a-d) can be statically represented by means of a weighted aggregate graph (e) in which each edge is labelled with the number of times it appeared.

Such a graph can be compactly represented by a *weighted-adjacency matrix*. Let's define $n = |V|$ and assign identifiers $0 \ldots n - 1$ to the nodes in V. Then,

the weighted-adjacency matrix A associated to the weighted graph G_w is the $n \times n$ matrix where

- $\forall i, j \in \{0 \dots n-1\}$, the non-diagonal entry a_{ij} is the weight of the edge from node i to node j in G_w if such edge exists, and 0 otherwise
- $\forall i \in \{0 \dots n-1\}$, the diagonal entry a_{ii} is 0

In such a matrix, each entry (i, j) represents the frequency of the edge from node i to node j. It is easy to see how in the case of undirected graphs such matrix is symmetric.

2.2 Centrality measures

In graph theory and social network analysis, a number of centrality measures have been designed for computing the relative importance of a vertex within a graph. These measure of importance of vertices are given by the topology of the graph itself.

Classical centrality measures are defined on static graphs, in which the topology is fixed and it is thus simpler to define them. However, in our work we consider the problem on evolving graphs, in which the topology changes over time. A possible approach when dealing with dynamic networks is to represent them as static graphs, obtained by aggregating the edges observed over a given period of time; however, this approach can fail to accurately compute the relative importance of nodes.

Novel centrality metrics have been introduced for dynamic network analysis that specifically address the problem of dynamic centrality. For instance, a metric has been proposed [25] which generalizes the classic path-based centrality that measures the importance of a node in a static graph as the number of paths connecting it to other nodes. This dynamic centrality measure ranks a node by the number of time-dependent paths (paths defined over time) connecting it to other nodes in the network; that is, the total amount of information sent from the node that reaches all other nodes in the network. Other works have been done on how the degree centrality evolves [9] and on computing other centrality measure [16, 33] in dynamic networks. Here we focus on measuring degree centrality and eigenvector centrality over evolving graphs.

2.3 Degree centrality

Degree centrality measures the importance of a vertex in a graph as the degree of the vertex, that is the number of ties incident upon the vertex. In other words, a vertex is considered more important when it is directly connected to a greater number of other vertices in the graph.

Given a static graph $G = (V, E)$, for each vertex $u \in V$ a degree centrality measure is defined as

$$C_D(u) = \frac{d(u)}{n-1}$$

where $d(u)$ is the degree of u and $n = |V|$ is the cardinality of the set of vertices.

When dealing with directed graphs, two different measures (in-degree and out-degree) can be defined, considering the incoming and outgoing arcs of the node, respectively. A natural extension to the case of weighted graphs can be made by considering the degree of a vertex to be the sum of the weights of the edges incident upon it.

In the case of evolving graphs, things obviously change. Here we do not have a fixed topology of the graph, but instead the degree of a vertex actually changes over time. The two most natural extensions of degree centrality of a node in the dynamic graphs case are

- to consider the total number of edges incident to the node over time
- to consider the number of distinct other nodes to which the node is connected at least once during the evolution of the network.

Considering a static representation of a dynamic graph by means of a weighted graph with labels on edges representing their frequency (i.e. how many times such edges appeared in the sequence of graphs representing the dynamic network as in the Evolving Graph Model), the first measure actually corresponds to the total weighted degree on the static aggregated graph, while the second measure corresponds to the classical degree centrality measure of counting the number of edges incident to the node.

There are both advantages and disadvantages in considering one or the other as dynamic degree centrality. The first measure gives a better idea of how much a node interacted with others, which is the basic idea of degree centrality measures; however, it can lead to extreme cases in which two nodes of the network only interact among them, boosting their degree measure when they actually cannot be considered "important". Thus, it can be preferable to consider the second measure described as the dynamic degree centrality measure. However, this measure is more difficult to be dynamically maintained at a node, since it requires to deal with duplicate elements counting. Indeed, multiple interactions with the same nodes should be counted just once; this requires to incrementally keep track of all the nodes previously encountered during the evolution of the network, so to know when the degree centrality measure should be updated This is also known as the *distinct elements counting problem*. Since it can be too expensive in terms of storage space to solve this problem exactly, data structures for efficiently approximating the number of distinct elements observed in a data streams have been proposed [15, 3, 7].

2.3.1 Distinct element counting problem

Let S be a multi-set of N objects to which it is possible to deterministically assign integer identifiers. Note that S can contain duplicate elements. Then, the *distinct element counting problem* is to find out the exact number F of distinct elements in S.

Without restrictions on space and memory usage, the problem can be pretty easily solved with $O(N)$ words of space by maintaining a list of distinct identifiers

and outputting the cardinality of the set. However, there are many applications in which such space consumption cannot be afforded. For instance, in the streaming model the continuous stream of updates observed can contain a huge amount of different elements, and it is not acceptable to store all their identifiers; moreover, all the computations are required to be performed in just one pass, and checking whether the incoming element was already observed would involve in the worst case scanning a list of $O(N)$ elements.

In such applications, suitable data structures for finding an approximate count $\tilde{F}$ with a reasonable degree of confidence must be used. Here we discuss the famous Flajolet-Martin sketches, that achieve this goal with just $O(\log N)$ space.

Flajolet-Martin Sketches The *Flajolet-Martin sketch* (or FM sketch) [15, 17] is a data structure that incrementally estimates the number of distinct elements in a multi-set with low space and memory requirements. For instance, in the streaming model FM sketches can be used to maintain an approximation of the number of distinct objects in the stream in just one pass without having to store extra information about the objects previously observed.

The basic version of a FM sketch is composed by just a hash function $h()$ and a sketch represented by a r-bits bitmap, initialized to 0. An appropriate value for the sketch size r is showed to be $O(\log_2 N)$, with N the number of distinct objects. The hash function h takes as input an object id i and returns a pseudo random integer value with a geometric distribution, that is

$$\mathbf{Pr}[h(i) = v] = 2^{-v}$$

for $v \geq 1$.

When an object of the sequence is observed, its object id i is hashed and the $h(i)$-th bit of the bitmap is set to 1. This means that multiple insertions of the same objects are only counted just once. Computing the estimated value of a sketch is very easy. The procedure finds the first bit of the bitmap that is still 0 and denotes its index as R; then, $\tilde{F} = 2^R$ is output as the FM estimation.

Intuitively, this works because of the geometric distribution of the values returned by the hash function; indeed, a generic bit i of the bitmap is expected to be set to 1 after 2^i distinct hashing operations; thus, we expect R to be roughly $\log_2 N$ if N distinct objects were observed.

However, when implementing the FM procedure it is often necessary to design more advanced schemes in order to obtain more accurate estimations. The simplest idea is to use a set of m different hash functions and maintain m different bitmaps, thus obtaining m values $R_1, R_2, \ldots R_m$. Then, the estimated number of distinct elements can be computed as

$$\tilde{F} = \frac{1}{\phi} 2^{\bar{R}}$$

where the "magic number" $\phi = 0.77351$ is a correction constant and $\bar{R}$ can be found averaging over the m individual estimations: $\bar{R} = \frac{\sum_{i=1}^{m} R_i}{m}$

Flajolet and Martin proved that using this simple procedure with m bitmaps it is possible to count the number of distinct elements in a set using $O(m)$ storage and

$O(m)$ operations per insertion with an expected relative result close to $\frac{0.78}{\sqrt{m}}$ for N at least 100 [15].

They also proposed practical improvements to this procedure. They first observed that it can be expensive to generate many different good hash functions, and that instead one can just generate a single one and then the others by composing this source with a fixed set of permutations. They called the corresponding algorithm *Multiple Probabilistic Counting* and experimentally verified that it shows no performance degradation with respect to the previous version.

Moreover, instead of a simple averaging operation they proposed to use *stochastic averaging*, that allows to have $O(1)$ processing cost (that is, independent on the number of bitmaps m) instead of $O(m)$. The resulting technique is called *Probabilistic Counting with Stochastic Averaging* (PCSA) and works as follows. A second hash function is used in order to choose only one of the m sketches when performing an update. Thus, each update is performed on only one sketch, and each sketch is actually counting only approximately $\frac{N}{m}$ distinct objects. The new estimation for the PCSA Algorithm will then be

$$\tilde{F} = \frac{m}{\phi} 2^{\frac{1}{m}} \sum_{i=1}^{m} R_i$$

PCSA and Multiple Probabilistic Counting are shown to be asymptotically equivalent in terms of expectation and standard deviation. However, the smaller processing cost leads to an increased non-linearity for smaller values of the count N.

In our work, we use the FM sketches to maintain, at each node in a dynamic network, an approximate count of the number of distinct users encountered during the evolution of the network. This works because FM sketches have three important properties when considering the context of distributed data streams [17].

Property 1. (Order insensitivity) Let's consider a sequence U of updates $u_1, u_2, \ldots u_k$ performed on a sketching algorithm. Then, the sketching algorithm is said to be *order-insensitive* if for every possible permutation of the sequence U the algorithm returns the same result.

Clearly, the FM sketch is order-insensitive because the order of the update operations does not change the resulting estimation.

Property 2. (Duplicate insensitivity) Let's consider a sequence U_1 of updates $u_1, u_2, \ldots u_k$ over distinct elements, and then another sequence U_2 with arbitrary repetitions of the updates of U_1. Then, a sketching algorithm is said to be *duplicate-insensitive* if the result returned when applying only U_1 is equal to the result returned when applying bot U_1 and U_2.

Again, a FM sketch is by definition duplicate-insensitive because the hash functions map the same object on the same bits in the bitmaps.

Property 3. (Union property) Let's consider two basic implementations of the FM sketch fm_1 and fm_2 that use the same hash function and estimate the cardinality of the multi-sets S_1 and S_2, respectively. Then, a new sketch $fm_U = fm_1 \vee fm_2$

constructed through a bitwise OR of the two sketches' bitmaps can be used to estimate the cardinality of the multi-set $S_U = S_1 \cup S_2$ given by the union of the two sets.

This property follows immediately from the update procedure of a FM sketch, and can be easily extended to implementations with $m > 1$ through bitwise OR's of pairs of bitmaps associated to the same hash function.

All these properties of the FM sketches mean that it can be possible to use this powerful tool in the context of dynamic networks, where a user can have multiple interactions with the same users, the order of such interactions must not affect the user's estimation and it can be useful to merge the information in two or more users' sketches in order to have a more accurate view of the network.

2.4 Eigenvector centrality

In eigenvector centrality, the importance of a vertex depends on how much its neighbours are important themselves. The idea is that ties with vertices with high importance values contribute to the importance value of a vertex more than ties with vertices that are not so important.

There are different eigenvector centrality measures. In general, the computation of such measures involves finding the eigenvector corresponding to the first eigenvalue of the graph adjacency matrix, taking it as a centrality measure. Unlike degree centrality, not only direct connections are taken into account but also undirected connections in the entire graph; the centrality of each vertex is proportional to the sum of the centralities of the vertices to which it is connected.

Formally, given the adjacency matrix A of a graph, the eigenvector centrality is defined as the eigenvector x such that

$$Ax = \lambda x$$

where λ is the largest eigenvalue of A.

There are two main approaches when designing eigenvector methods for exploiting the structure of a (directed) graph [23]. The first approach is to introduce the notions of

- *authorities*, which are nodes with several in-links;
- *hubs*, which are nodes with several out-links.

The first algorithm which introduced these notions is *HITS* (Hypertext Induced Topic Search) [20], which defines both an authority score x_i and a hub score y_i for each node i. The main idea behind HITS is that good authorities are pointed by good hubs and good hubs point to good authorities. Let's consider a directed graph $G = (V, E)$ with $|V| = n$ and its $n \times n$ adjacency matrix $A = \{a_{ij}\}_{1 \leq i,j \leq n}$ with $a_{ij} = 1$ if $(i, j) \in E$ and $a_{ij} = 0$ otherwise. Starting from an initial distribution $y(0)$ of positive hub scores (for instance, $y_i = 1, 1 \leq i \leq n$), the authority and hub scores distributions can be found by iteratively computing

$$x(k) = A^T y(k-1)$$
$$y(k) = Ax(k)$$

until reaching a convergence or the desired level of accuracy. The main strength of HITS is its dual centrality measures scores; it computes on the graph two separate ranked lists of nodes in terms of authority scores and hub scores respectively, and one could be more interested in one or the other depending on the applications. On the other hand, it is simple for "spamming nodes" to greatly influence their own hub scores and consequently the authority scores of nodes in their local vicinity, a weakness in some possible applications, like for instance for web search information retrieval (for which it was originally designed).

The second approach is to see links between nodes in a graph as recommendations from one node to another, and to consider just one notion of "importance" of a node. This is the approach of PageRank [34], which assigns a PageRank score defined as the stationary probability distribution of a suitably modified Markov chain defined on the graph; this allows to greatly alleviate the problem of spamming nodes. We see PageRank more in detail in Section 2.6.

There are also other algorithms which employ a mixed approach. For instance *SALSA* (Stochastic Approach For Link Structure Analysis) [24] was designed with the goal of combining the strengths of HITS and PageRank. Like HITS, it defines both an authority and a hub score for each node, and like PageRank they are compute through the use of Markov chains. The problem of spamming is reduced thanks to a less interdependence of authority and hub with respect to HITS, and with respect to PageRank it has the advantage of defining dual scores instead of just one. However, its main drawback as regards an implementation on a web search engine is its query-dependence, meaning that a neighbourhood graph has to be constructed and authority and hub scores computed on it at query-time, while PageRank scores can be computed offline since are query-independent.

As regards the computation of eigenvector centrality measures on dynamic evolving graphs, the usual approach is to consider a static representation of a dynamic network obtained by aggregating the information about the state of the network over time [11]. However, this only gives an approximation of the centrality measure without considering the dynamic behaviour of the network.

2.5 Random walks and Markov chains

Given a graph and a starting vertex, a *random walk* [27] is the mathematical formalization of a random path on the vertices of the graph. At the beginning of the walk, we select one of the neighbours of the starting vertex at random and move to this neighbour; then, we select a neighbour of this vertex at random, and so on. A random walk is simply the sequence of vertices selected by this randomized process. It is a very simple yet powerful concept. Over the years, random walks have been well studied both in the context of theoretical analysis of infinite graphs (grids) and with respect to algorithmic uses, like for instance for sampling large data sets.

Markov chains and their properties A random walk can also be seen as a finite *Markov chain* [29], that is a randomized process defined by transitions from one state to another between a finite or countable number of possible states.

Definition 2.1. A sequence of random variables X_0, X_1, $\cdots \subseteq S$ is a Markov chain with state space S if all possible states have the Markov property, that is, the state at step $t+1$ $x_{t+1} \subseteq S$ only depends on the state at step t:

$$\mathbf{Pr}[X_{t+1} = x_{t+1} | X_0 = x_0 \wedge X_1 = x_1 \wedge \dots X_t = x_t] = \mathbf{Pr}[X_{t+1} = x_{t+1} | X_t = x_t]$$

The Markov chain is called *time-homogeneous* if the state transition probabilities are independent of t. On the contrary, if they can change over time the Markov chain is called *non-homogeneous*. As the name suggests, a non-homogeneous Markov Chain does not assume a homogeneous behaviour of the stochastic process; the transition probabilities matrix P is not constant but a function of time, therefore in general there is a different matrix P_k with transition probabilities $p_{ij}^{(k)}$ at each step k. A non-homogeneous Markov chain can represent a random walk on a dynamic graph, in which the set of edges changes over time and the transition probability matrices are given by the topology of the network at each step. In Section 3.1 we show a more detailed analysis on this subject.

Given a connected graph $G = (V, E)$ with a set V of n vertices and a set E of m edges, let's define $P = (p_{ij})_{i,j \in V}$ as the $n \times n$ transition probability matrix of a time-homogeneous Markov chain on the graph. We can see the Markov chain as a random walk on the weighted graph associated with G with labels on the edges representing the probability of moving from a vertex to the other. For every pair of vertices $i, j \in V$, the generic entry p_{ij} of P will be

$$p_{ij} = \begin{cases} \frac{1}{d(i)} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

denoting with $d(i)$ the degree of vertex i.

Given a *marginal distribution* $\pi_k = \{\pi_k(1), \pi_k(2) \dots \pi_k(n)$ over states at step k, we have that

$$\pi_{k+1}^T = P\pi_k^T$$

and thus

$$\pi_k^T = P^k \pi_0^T$$

whichever the initial distribution π_0.

Definition 2.2. A probability vector πT satisfying $\pi T = P\pi T$ is called the **stationary** (or steady-state or equilibrium) **distribution** of the homogeneous Markov chain.

In general, the stationary distribution is not guaranteed to exist, to converge and to be unique. To do so, we have to ensure that the transition probability matrix P is both *stochastic* and *irreducible*.

Definition 2.3. A probability matrix is said to be **stochastic** if it has only non-negative elements and each row sums exactly to 1.

In general, P can fail to be stochastic because of the vertices $v_i \in V$ in the graph with $d(v_i) = 0$ – the rows of the matrix corresponding to these vertices are composed by all zero elements. A way to make P stochastic is simply to add self-loops to these nodes.

Definition 2.4. A probability matrix is **irreducible** if the underlying Markov chain is irreducible, that is, if every state is eventually reachable from any other:

$$\mathbf{Pr}[X_k = s_j | X_0 = s_i] > 0$$

for every pair of states in the state space $s_i, s_j \in S$.

If the transition probability matrix P of the Markov chain is stochastic and irreducible, then the Markov chain is guaranteed to have a unique positive stationary distribution.

Undirected graphs case In the case of undirected graphs, some convergence results can be shown for the stationary distribution of a random walk.

Theorem 2.1. *Given a connected, undirected graph $G = (V, E)$ with $|V| = n$ and $|E| = m$, the stationary distribution of a random walk on G is the probability distribution π such that*

$$\pi(u) = \frac{d(u)}{\sum_{v \in V} d(v)} = \frac{d(u)}{2m}$$

for every vertex $u \in V$.

Proof. First of all, we prove that π is a probability distribution. Indeed:

$$\pi = \sum_{u \in V} \pi(u) = \sum_{u \in V} \frac{d(u)}{\sum_{v \in V} d(v)} = \frac{\sum_{v \in V} d(v)}{\sum_{v \in V} d(v)} = 1$$

Then, we show how the distribution at step $k + 1$ is equal to the distribution at step k. Let's consider the $n \times n$ transition probability matrix $P = (p_{ij})_{i,j \in V}$ defined as before:

$$p_{ij} = \begin{cases} \frac{1}{d(i)} & \text{if } (i, j) \in E \\ 0 & \text{otherwise} \end{cases}$$

As we have seen, the distribution is stationary if the following condition holds:

$$P\pi = \pi$$

And for any vertex $i \in V$:

$$[P\pi]_i = \sum_{j=1}^{n} p_{ij}\pi(i) = \sum_{(i,j) \in E} \frac{1}{d(i)} \frac{d(i)}{\sum_{v \in V} d(v)} = \frac{1}{\sum_{v \in V} d(v)} \sum_{(i,j) \in E} 1 = \frac{d(i)}{\sum_{v \in V} d(v)} = \pi(i)$$

□

This means that the probability of being at any given vertex v tends to a well-defined limit independently of the starting vertex of the random walk (or Markov chain process), and this value depends by the degree of v.

This is a very nice property, and we will show an analysis of its extension in the case of random walks on directed graphs in Section 3.1.

Random Walks on weighted graphs The concept of random walk on a weighted graph is a simple extension of the one regarding a simple graph.

Let $G_w = (V, E)$ be a graph with a weight label e_{ij} associated to each edge $(i, j) \in E$. The entries of the weighted adjacency matrix $A = \{a_{ij}\}_{i,j \in V}$ associated to G_w are simply defined as

$$a_{ij} = \begin{cases} e_{ij} & \text{if } (i,j) \in E \\ 0 & \text{otherwise} \end{cases}$$

Then, the transition probability between vertices i and j – that is the probability of selecting vertex j when the random walk is at vertex i – will be proportional to a_{ij} and defined as

$$p_{ij} = \frac{a_{ij}}{\sum_{w \in V} a_{iw}}$$

Then, the transition probability matrix $P = \{p_{ij}\}_{i,j \in V}$ is analogous to the transition probability matrix of a non-weighted graph, and all the considerations made so far in this section are valid in the case of weighted graphs as well.

In particular, it can be extended the simple result about the stationary distribution of random walks on undirected graphs. That is, given an undirected weighted graph, the probability distribution π on the vertices of the graph such that

$$\pi(u) = \frac{d(u)}{\sum_{v \in V} d(v)}$$

is the stationary distribution on the graph, noting that in this case the degree of a vertex is defined as the sum of all edges incident to the vertex – that is, $d(u) = \sum_{(u,v) \in E} e_{uv}$.

2.6 PageRank

PageRank [34, 23] is a link analysis algorithm for measuring the importance of nodes in a graph with respect to the structure of the graph itself. It was first introduced by Google founders Larry Page and Sergey Brin [34, 10] and it is at the basis of Google's success as a web search engine.

Intuitively, given as input a directed graph – for instance the graph representing the web, where the websites are the nodes and links (citations) of one toward another are the edges – the PageRank algorithm assigns a numerical weight (the PageRank, or PR) to each node representing its "importance" with respect to the others. The idea behind PageRank is that a link from node u to node v carries a weight that contributes to the rank of v that is directly proportional to the PageRank of u and

inversely proportional to the number of out-links of u. In other words, the PageRank of v increases as:

- the size of the set $N_{in}(v)$ of nodes that link to v increases;
- the PageRank of the nodes in $N_{in}(v)$ increases;
- the number of nodes to which nodes in $N_{in}(v)$ link decreases.

There can be a lot of subjective definitions of the concept of "importance" of a node (or a web page). PageRank tries to model the behaviour of a user navigating the web; in this so-called *random surfer model*, the user performs a random walk on the graph representing the web by starting from a random page and, at each step, clicking on one of the links contained in the page, choosing uniformly at random. In addition, at each page she has a given probability $1 - d$ of jumping to a random page in the graph; the value $0 \leq d \leq 1$ is the so-called *damping factor*.

Given a node v and the set $N_{in}(v)$ of nodes linking to v, and defining $N_{out}(u)$ as the set of nodes to which a node u links to, the recursive definition of the PageRank of v, $PR(v)$, is

$$PR(v) = \frac{1-d}{n} + d \sum_{u \in N_{in}(v)} \frac{PR(u)}{|N_{out}(u)|}$$

where n is the total number of nodes in the graph.

2.6.1 Computation and convergence

Mathematically, PageRank is the probability distribution over the set of nodes in a graph that represents the probabilities that a Markov chain (or a random walk) on the graph is, at a given time, on each of the nodes. In other words, given a Markov chain with transition probability matrix P, the computation of PageRank involves finding the the stationary distribution of the Markov chain; that is, the probability vector π^T satisfying

$$\pi^T P = \pi^T \text{ and } \pi^T 1^T = 1$$

(π^T is the *left eigenvalue* of the probability matrix).

The transition probability matrix P can be derived from the adjacency matrix of the graph by normalizing each row such that the elements of P are

$$p_{ij} = \begin{cases} \frac{1}{N_{out}(i)} & \text{if } i \text{ links to } j \\ 0 & \text{otherwise} \end{cases}$$

However, for a stationary distribution to exist and the Markov chain to converge, the matrix P must be both stochastic (to ensure that such distribution exists) and irreducible (to ensure that the distribution converges); in general, this is not true and P must be adequately adjusted. In Section 2.5 we defined a stochastic matrix as a probability matrix with only non-negative elements and such that each row sums exactly to 1. P can fail to be stochastic because of rows summing to 0, corresponding

to nodes that don't have any outgoing links; then, given n the order of P, we can replace each zero row with e^T/n and call this new stochastic matrix P'. We also defined the irreducibility property, which guarantees that the Markov chain has a unique positive stationary distribution. P' is stochastic but can fail to be irreducible; one way to solve the problem is to introduce a $n \times n$ perturbation matrix $E = \frac{ee^T}{n}$ and define a new matrix

$$P'' = dP' + (1-d)E$$

where d as before is the damping factor that defines the probability of "teleporting" to any other node in the graph.

P'' can then be used in the Power method to iteratively compute the PageRanks of the nodes in the network. Starting from an initial uniform distribution with $PR(N_i) = \frac{1}{n}\ \forall i = 1 \dots n$, the PageRank vector $\pi^T = \{PR(N_1), PR(N_2), \dots PR(N_n)\}$ can be found by iteratively computing $\pi^T = \pi^T P''$ until π^T converges to a stationary distribution, or at least until the desired level of accuracy is reached.

2.6.2 Monte Carlo algorithms

We have seen how it is possible to compute PageRank with the deterministic power iteration method. However, depending on the problem size, this method may need very long and intensive computations before giving meaningful results. On the other hand, Monte Carlo methods for the PageRank computation have been proposed by Avrackenkov et al. [5] that present many advantages over the power iteration. For instance, they provide good PageRank results for important nodes after just one iteration; moreover, they have efficient parallel implementations and allow to update the PageRank continuously as the topology of the network changes.

Such methods are motivated by the following formula that derives from the PageRank definition:

$$\pi = \frac{1-d}{n} 1^T [I - dP]^{-1} = \frac{1-d}{n} 1^T \sum_{k=0}^{\infty} d^k P^k$$

given a damping factor d and a transition probability matrix P.

From this formula they derived simple methods for sampling from the PageRank distribution. Indeed, considering a random walk $\{X_t\}_{t \geq 0}$ that starts from a randomly chosen node, terminates at each step with probability $1 - d$ and makes instead a transition according to the matrix P with probability d, it follows from the formula that the end-point of such random walk has a distribution π.

Hence, algorithms can be designed to sample the PageRank distribution by simulating a number of random walks on the graph. Although the basic idea behind these algorithms is always the same, different versions can be implemented varying the number of random walks to be simulated, the starting points of these walks (either cyclic or random) and how the walks are used to sample the PageRank distribution. Here we report just two of the algorithms proposed in [5], which are the most significant with respect to our work.

Algorithm: MC end-point with cyclic start *Simulate $N = mn$ runs of the random walk $\{X_t\}_{t\geq 0}$ initiated at each page exactly m times. Evaluate $\pi(j)$ as the fraction of the N random walks which end at node $j = 1 \dots n$.*

The expected value of the estimation given by the algorithm for each node j is exactly the PageRank value of j $PR(j)$. The number of required samples to achieve good accuracy with high probability is linear in n, meaning that we only would need one sample per node to have good estimations of the PageRank vector.

Another version of this algorithm is the *MC end-point with random start*, in which the N runs of the random walk are simulated starting from a random node. Again, the PageRank of each node is evaluated as the fraction of the N walks that end at it. This version of the algorithm is actually outperformed by the one with cyclic start. Indeed, the variance of this version is showed to be higher, and moreover it introduces the problem of picking nodes at random from the node set, that can be potentially very huge.

Algorithm: MC complete path *Simulate the random walk $\{X_t\}_{t\geq 0}$ starting exactly m times from each page. For any node j, evaluate $\pi(j)$ as the total number of visits to node j multiplied by $\frac{1-d}{nm}$.*

This is one of the algorithms that are part of the "complete path" Monte Carlo family; in these algorithms, the whole path of a random walk is considered rather than just its end-point. This can be done because it is equivalent to consider the average number of times that the random walk visits a node j. This actually gives a lower variance with respect to the case in which we only consider the end-points of the random walks, with the same results in terms of the number of walks necessary to achieve good estimations with high probability. Therefore, it is preferable to use complete paths algorithms.

A similar algorithm is the *MC complete path stopping at dangling nodes*. In this version, a random walk $\{Y_t\}_{t\geq 0}$ can be terminated at each step either with probability $1 - d$ or when it reaches a dangling node – that is, we stop at dangling nodes rather than jump with equal probabilities to any other node at random. Then, for any node j, $\pi(j)$ is evaluated as the total number of visits to node j divided by the total number of visited page.

The third and final version of complete path Monte Carlo methods is the *MC complete path with random start*, in which the N samples of the random walk $\{Y_t\}_{t\geq 0}$ start at a random page and the PageRank value $\pi(j)$ of node j is evaluated as the total number of visits to node j divided by the total number of visited page (as in the version stopping at dangling nodes). However, as in the end-point algorithms, the version of the complete path algorithms with random start provides an estimator with a higher variance than the one with cyclic start. It is therefore preferred to use one of the other two complete path Monte Carlo methods, especially the version in which the samples fo the random walk stop at dangling nodes.

Experimental simulations confirmed how these Monte Carlo methods give very accurate estimation for important nodes after just one iteration. In particular, it is showed how the complete path method stopping at dangling nodes gives results with a relative error of only 7% with 95% confidence for nodes with high PageRank values. As regards the comparison between the different algorithms, it is showed how

MC complete path stopping at dangling nodes has the best performance, followed by MC end-point with cyclic start and MC complete path with random start, thus confirming the better performances of complete path algorithms with cyclic start over end-point algorithms with random start.

As we said, these Monte Carlo methods have natural parallel implementation and allow to perform continuous update of the PageRank vector as the network topology changes. Indeed, in a multi-processor system each available processor can run an independent Monte Carlo simulation. It is reported that the iterative power method performed by Google over the graph representing the web takes a few weeks of intense computations [36]. The PageRank vector changes significantly during such period of time, so Google prefers to recompute the PageRank vector starting from the uniform distribution rather than to use the previous version of the PageRank vector as the initial distribution for the new computation. It could be possible to update the PageRank vector using linear algebra methods, but one would need first to detect all the topology changes, that is all the nodes and edges added/removed to/from the previous network representation. However, this is not necessary when using Monte Carlo algorithms, since it could be possible to run them continuously while the database is updated with the changes in the topology, so to have an up-to-date estimation of the PageRank for relatively important nodes with high accuracy in real-time.

Monte Carlo algorithms for efficient incremental computation of the PageRank distribution have also been proposed [6]. They allow to incrementally update the approximations of the PageRanks as the underlying graph (like for instance the Web graph) changes, whereas the power method will always be out-of-date, failing to keep up with these changes while its computations take place. However, this methods are still centralized, because a central server is needed to simulate the random walks of which the algorithms make use, and the whole structure of the graph as well as its changes over time have to be known.

3 Centrality measures in evolving networks

In this chapter we discuss the challenges of a decentralized approach to an evolving network, and propose algorithms for the computation of some importance measures. We analyze the problems both from a theoretical point of view and considering the possibility of actual implementations.

The most interesting and challenging part of our work is designing algorithms for the decentralized computation of a PageRank-like measure on each node of the network, and analyzing from a theoretical point of view why the approach that we used can actually give good approximations. This involves studying the problem of random walks on evolving graphs with particular properties. We also attack the problem of computing distributed measures of degree centrality, showing how to use well-known aggregating data structures for designing algorithms and techniques.

3.1 Random walks on dynamic graphs

As we saw in Chapter 2, random walks on static graphs involve a simple discrete step process in which, at each step, the next node of the walk is chosen uniformly at random from the (fixed) set of nodes adjacent to the one at which the walk is currently on. This process is modelled by a *homogeneous* Markov Chain, that is a Markov Chain where the transition probabilities remain the same at all times.

Random walks on dynamic graphs [1, 4, 14] are somewhat more tricky, for the set of edges changes over time and so do the transition probabilities between nodes. The adjacency between nodes u and v in time step t_k does not imply that they will be adjacent also in time step t_{k+1}. That is, in general for each time step t_k, $1 \leq k \leq n$, we will have a different adjacency matrix A_k and consequently a different transition probability matrix P_k – obtained by normalizing the rows of A_k such that each one sums to 1 – that defines the transition probabilities of the random walk in step k. The resulting Markov chain is non-homogeneous.

Let's define X_k as the state of the Markov chain in time step k – that is, the node at which the random walk is at time k – and $p_{k,k+\tau}$ (u, v) the transition probability between two nodes u and v over a number $\tau \geq 1$ of time steps. Then, we define

$$p_{k,k+\tau}(u,v) = Pr[X_{k+\tau} = v | X_k = u]$$

and thus the transition probability between u and v over a single time step can be defined as

$$p_{k,k+1}(u,v) = \begin{cases} \frac{1}{N_k(u)} & \text{if } u \text{ and } v \text{ are adjacent at time } k \\ 0 & \text{otherwise} \end{cases}$$

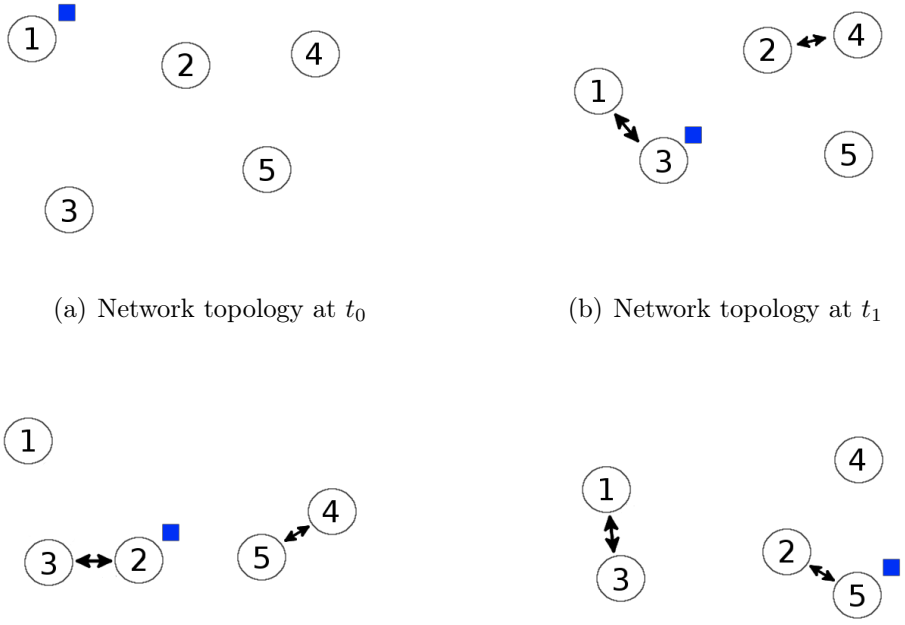
given $N_k(u)$ the set of nodes adjacent to u at time k.

Each of these transition probability matrices P_k has the same properties of the fixed transition probability matrix of a homogeneous Markov chain:

- all transition probabilities are non-negative; $p_k(u,v) \geq 0, \forall u, v \in V$ and $\forall k \geq 0$
- all rows of P_k are normalized so to sum to 1; $\sum_{v \in V} p_k(u,v) = 1$, $\forall u \in V$ and $\forall k \geq 0$
- in P_k, a node has a self-loop if and only if it has no neighbours in step k; $p_k(i,i) = 1$ if $N_k(u) = \emptyset$ and 0 otherwise, $\forall k \geq 0$

The density of such graphs can in general be very low, so that in most of the time steps source-destination pairs would not be connected by complete paths. In such intermittent networks, end-to-end connectivity between nodes can be achieved taking into consideration interactions in the network during a given number of steps and using a *store-forward-carry paradigm*, i.e. nodes can store information and forward them in later time steps. Figure 2 shows an example of a contact network dynamically evolving over time. Although node 1 is never directly connected to, say, node 5, still there exists an end-to-end path between them in the time period considered. In step t_1, node 1 sends a packet to its neighbour 3; however, none of its current neighbours is able to forward the packet to 5, so it stores it until t_2, when

it forwards it to 2; finally, in t_3, 2 forwards the packet to 5. It has to be noted, however, that it can be impossible to have such end-to-end connections between all possible pairs of nodes.



(a) Network topology at t_0

(b) Network topology at t_1

(c) Network topology at t_2

(d) Network topology at t_3

Figure 2: Example of end-to-end connectivity during the evolution of a network over time

The problem of evaluating the cover time for dynamic networks has been approached [4, 13], but of course the network considered must satisfy some properties in order to even have a finite cover time, not to mention computing bounds on it. Instead, in real-world dynamic networks is not always possible to have guarantees on how the network will evolve over time, meaning that it is likely to have a clustered structure, with a number of disjoint network components that do not interact with each other. If this is the case, each node will be able to have only a partial view of the network, representing the component it belongs to. However, even if it is not possible to guarantee the possibility of having measures or information that globally take into account the whole network, it still makes sense to have local importance measures of a node with respect to its locally near peers, because in real applications of such systems what is important in the end is each node understanding how important it is with respect to others that it encounters during the evolution of the network.

3.2 Estimating the PageRank distribution

The standard way to compute the PageRank vector on a static graph involves the computation of the eigenvector of the adjacency matrix representing links between

nodes (users). In the context of a fully decentralized evolving network, however, three kinds of problems arise when trying to compute a PageRank-like importance measure.

The first problem is that there is no general knowledge of the overall structure of the network, and such adjacency matrix is not known; the computation is carried on by the single users and can only rely on exchanges of information among them, without any centralized computation. This also means that without a central authority collecting the data computed by all users in the network, it will not be possible (or at least it will not be a trivial task) to have a PageRank-like probability distribution over all users, i.e. a list of probabilities summing to 1. In other words, each user will not have an *absolute* measure of her own importance in the network, bur rather a *relative* value representing her degree of importance with respect to others. It will then be possible for two users that meet at any given time to share their relative values and establish not only which one of them is more important, but also *how much* more important. Indeed, it is simple to see that the ratio between the two relative values is equal to the ratio of the two probabilities taken from the PageRank-like probability distribution obtained normalizing by the relative values of all users. Defining r_u as the relative importance value of user u and $\pi_u = \frac{r_u}{\sum_{\forall \text{ user } w} r_w}$ as the corresponding PageRank-like value of u in the probability distribution over all users, we have

$$\frac{r_u}{r_v} = \frac{\pi_u}{\pi_v}$$

meaning that two users can compute their relative importance with respect to each other in the estimated PageRank-like probability distribution without knowing their actual probability values in the distribution.

The second problem is related to the computational capacity of the mobile devices involved – in the case of ad-hoc mobile networks – or, in general, of the computing elements. They perform all the computations, so it is crucial to carefully consider how much they can do and how much information they can store. In the case of mobile devices, new smartphones with better and better hardware resources are being produced every day, so using for instance this kind of device would give the possibility to have both reasonably high computational power and available storage space for carrying out our computation. On the other hand, if we were to use simpler devices, like RFID tags, things would greatly change; indeed, in general these kinds of devices can have very limited computational power and hardware space for storing information. A possible approach in designing algorithms and techniques for fully decentralized computations could be to make assumptions on the hardware resources necessary to run them, in fact limiting their possible applications to contexts in which suitably powerful devices can be deployed. However, a better approach (although certainly more challenging) would be to work on such algorithms and techniques in order to simplify them and reduce as much as possible the computational power and space consumption needed. Of course it is possible that going down this road we would need to lower the level of the goals that we aimed to achieve, since our initial targets could be (or seem to be) infeasible with only a limited amount of hardware resources. On the other hand, the results that we would reach would be

highly portable and very efficient, and it could be possible to deploy such techniques in a greater number of applications. Therefore, it is important to carefully keep in mind the resources demands when attacking this kind of problems. As we will show, in the design of our algorithms we care very much about this issue and make sure that the complexity of the operations performed as well as the space consumption on each node are minimal.

The third problem is inherently related to the dynamism of the network. Links are formed and destroyed over time, so the measure that we compute must keep up with the topology changes in real time; moreover, while in a network representation we can model the asynchronous interactions between users as instantaneous, in reality the duration of such interactions can vary in some applications –for instance, in ad-hoc mobile networks. To cope with this problem, it is important to reduce the amount of information that need to be exchanged between two users; the ideal situation would be to have a single packet of information sent, so to avoid inconsistencies when the duration of the interaction is too small to allow the exchange of multiple packets. Again, we take particular interest in designing techniques as generally implementable as possible, that is we try to require our protocols to have a minimal amount of information exchanges. We will see that our algorithms are very lightweight and need a very small information overhead to work.

An interesting thing to note is that while certainly more challenging to deal with, a fully distributed approach has the important advantage of better capturing the dynamic behaviour of an evolving network than a centralized one. A static centralized computation of users' importance can take a lot of time, depending on the network size, and the values computed are necessarily outdated since in the time needed for the computation the network has continued to evolve and change. On the other hand, with a decentralized approach our estimated measures are updated in real time and fully represent the history of a user up to a given point in time.

Here we propose two distributed algorithms for computing at each node an estimation of its PageRank value while simulating the dynamic behaviour of the network. The basic ideas behind these algorithms come from the probabilistic algorithms for PageRank computation proposed by Avrachenkov et al. [5] for static networks and described in Section 2.6.2. Such Monte Carlo algorithms sample from the PageRank distribution by simulating random walks on the graph underlying the network and counting the fraction of such walks that pass through (or end at) each node. The number of walks that need to be performed varies with the desired level of accuracy. Although a good accuracy with high probability requires a very high estimated number of walks ($O(n^2)$), it is showed that those algorithms give a reasonable evaluation of the PageRank values for important nodes with just a linear number of random walks. The idea we exploit is similar, but extended to a dynamic environment with some expedients necessary for dealing with random walks in a dynamic network. In fact, the density of the network can in general be so low that source-destination pairs are not connected by complete paths most of the time (if not ever). For instance, in the real network traces that we consider only a very small fraction of all possible links in the network is actually present at any given time, and moreover there are pairs of users that cannot be connected by a path, even considering all the links in a given time period (that is, these networks

are not entirely connected). In such intermittently connected networks, end-to-end connectivity is achieved over time through the *store-carry-forward paradigm*: considering a random walk as the action of users forwarding a token (the random walk is at user u if u has the token), if at any given time user u has the token but no contacts with others, she can store and carry it until she meets one or more other users, upon which she can forward the token according to the classic rules of simple random walks. This is how we model the concept of random walk on an evolving network in our algorithms. Additionally, we model the "teleportation property" of the PageRank algorithm by introducing probabilities (fixed and equal for all users) of generating and dropping tokens. The result of this randomized process is that in both algorithms we introduce, while the network evolves tokens are continuously generated, exchanged and dropped; the main difference between the two is in how users update their importance estimations upon interacting with the tokens.

3.2.1 CWP Algorithm: Counting Walks Passing through a node

Let's consider a dynamic evolving network with a set of nodes V represented by the Evolving Graph model, in which time is seen as a series of discrete time steps with a suitable duration. For simplicity, we model the system as synchronous even though it can be showed that it works also in asynchronous cases.

Initially, at time step t_0, each node $u \in V$ initializes a counter R_{CWP} to 0 and is provided with the same *generate probability* α_g and *forward probability* α_f, with $0 < \alpha_g < 1$ and $0 < \alpha_f < 1$. Let's also define N_t as the set of nodes that have at least one neighbour in the network at time step t. Then, for each time step t:

- every node generates a token with probability α_g;
- for every node in N_t: for every token it has, with probability α_f it forwards it to one of its current neighbours picked at random (each token independently from the others), dropping it instead with probability $1 - \alpha_f$;
- when a node receives a token, it increments its own count value by one unit (i.e. it counts how many random walks pass through itself);
- at any given time, the count value R_{CWP} of a node represents a measure of its importance.

Note that nodes that do not have any neighbours in a time step are "frozen", in that they do not forward nor drop any token as in the store-carry-forward paradigm.

In this way we somehow simulate a number of random walks on the dynamic graph. Then, as in [5] we use the number of walks passing through a node to compute its PageRank-like value. The combination of the probabilities α_g and α_f of, respectively, generating and forwarding/dropping a token tries to model the *random surfer model* of the original PageRank algorithm, for which at each step of the random walk there is a given probability of jumping to a random node.

Collecting the count values of all nodes, it is straightforward to get a PageRank-like distribution by normalizing by the sum of all count values:

Algorithm 1 CWP (Counting Walks Passing through a node)

```
UPON EVENT < init >:
    R_CWP = 0                                   ▷ relative importance value of the node
    tokens = 0                                  ▷ number of tokens at the node
    for all time steps t do
        generate a random value r_g ∈ [0, 1]
        if r_g ≤ α_g then
            tokens ← tokens + 1
        end if
        let N_t(u) be the set of links (u, v) that exist in the network at time step t
        if N_t(u) ≠ ∅ then
            while tokens > 0 do
                generate a random value r_f ∈ [0, 1]
                if r_f ≤ α_f then
                    pick user v such that (u, v) ∈ N_t uniformly at random
                    send token to v
                end if
                tokens ← tokens − 1
            end while
        end if
    end for

UPON EVENT < receive token >:
    tokens ← tokens + 1
    R_CWP ← R_CWP + 1

UPON EVENT < query >:
    return R_CWP
```

Figure 3: Pseudo-code of the CWP Algorithm at generic node u

$$\pi_{CWP}(u) = \frac{R_{CWP}(u)}{\sum_{\forall v \in V} R_{CWP}(v)}$$

In Chapter 4, this is the distribution that we compare to the real PageRank distribution on the data sets chosen for the simulations; this way, we evaluate the goodness of the results returned by the algorithm.

3.2.2 CWE Algorithm: Counting Walks Ending on a node

The evolution process of the CWE Algorithm is the same as CWP. Time is seen as a sequence of discrete time steps in which users generate and forward (or keep, if they are isolated) tokens; the exchange of tokens represents the execution of a number of random walks on the dynamic graph representing the network, and nodes compute

Algorithm 2 CWE (Counting Walks Ending on a node)

```
UPON EVENT < init >:
    R_CWE = 0                                   ▷ relative importance value of the node
    tokens = 0                                  ▷ number of tokens at the node
    for all timesteps t_i do
        generate a random value r_g ∈ [0, 1]
        if r_g ≤ α_g then
            tokens ← tokens + 1
        end if
        let N_t(u) be the set of links (u, v) that exist in the network at time step t
        if N_{t_i}(u) ≠ ∅ then
            while tokens > 0 do
                generate a random value r_f ∈ [0, 1]
                if r_f ≤ α_f then
                    pick user v such that (u, v) ∈ N_{t_i} uniformly at random
                    send token to v
                else
                    R_CWE ← R_CWE + 1
                end if
                tokens ← tokens − 1
            end while
        end if
    end for

UPON EVENT < receive token >:
    tokens ← tokens + 1

UPON EVENT < query >:
    return R_CWE
```

Figure 4: Pseudo-code of the CWE Algorithm at generic node u

their relative importance values with respect to these walks. Here, however, a node increments its counter R_{CWE} by one unit when it *drops* a token, that is when a random walk ends at the node.

This way each node counts how many walks end on itself, rather than how many walks it is part of. Again, at any time a PageRank-like distribution over the nodes can be found simply normalizing all relative count values by the total sum of counts (i.e. the total number of random walks with length greater than 1) up to that time:

$$\pi_{CWE}(u) = \frac{R_{CWE}(u)}{\sum_{\forall v \in V} R_{CWE}(v)}$$

Intuitively, it can be noticed how these count values will be considerably smaller than the ones given by the CWP Algorithm; indeed, the number of random walks

ending on a node can only be a fraction of the total number of random walks passing through it. A formal proof is shown in Section 3.2.3.

3.2.3 Algorithms analysis

These two algorithms work on static graphs. Indeed, if the graph topology does not change they can simply be seen as decentralized versions of the Monte Carlo algorithms presented in Section 2.6 [5], with the nodes generating tokens at a fixed rate and the random walks simulated through the exchanges of tokens used for sampling from the real PageRank distribution. In the remainder of this section, we show that, for a static network, i) the CWE algorithm essentially computes the standard Pagerank vector, up to a constant and ii) the CWP algorithm in expectation computes the same vector as the CWE algorithm, again up to a constant.

The CWE on static networks A nice property of the CWE algorithm is that it essentially computes the Pagerank vector on static networks [8].

In particular, assume for simplicity the system is synchronous and consider T consecutive time steps. It is possible to characterize $\mathbf{E_A}[\mathbf{C}_i]$, i.e., the expected number of tokens that die at i during this interval according to the CWE algorithm. We have:

Theorem 3.1.

$$\mathbf{E_A}[\mathbf{C}_i] = rTn(1-\theta)\boldsymbol{\pi}_i,$$

Here, the expectation is taken with respect to the distribution A on the set of possible random walks generated by Algorithm CWE. In turn, θ is the probability that a random walk generated according to the process whose distribution defines the PageRank, defined in Subsection 2.6.2.

It should be noted that θ is a quantity that only depends on the network. This implies that the orderings defined by $\mathbf{E_A}[\mathbf{C}]$ and $\boldsymbol{\pi}$ are the same.

CWE vs CWP The two algorithms are based on a simple idea and are easily implementable; the main tool that each node must possess is a good random number generator (RNG) for computing independent random numbers. The computation required at each node in a time step is minimal, reducing in the worst case to the generation of two random numbers and the update of two counters. The random number generation can be implemented as to take constant time, so the algorithm in each time step runs in constant time as well.

The space requirements are also negligible, reducing to two integer counters (one for the estimated count value and the other for the number of tokens possessed), plus the space to store the two probabilities α_g and α_f, leading to $O(1)$ storage space.

As regards the communication overhead, the algorithm requires the exchange of tokens between nodes. This means that the transmission of only a small amount of information is needed; indeed, the transmission of a "token" can be implemented with a small special packet or a 1-bit field in a packet containing other information, so the exchange overhead is minimal.

In ad-hoc mobile networks, all this makes it possible to implement the algorithms on devices with low computational capacity and small storage space – such as RFID tags – giving the possibility of reducing the costs for the deployment of more powerful devices.

As we already noted, the CWE Algorithm gives as results smaller count values than the CWP Algorithm.

Theorem 3.2. *For each node, the expected importance value over a given time period returned by the CWE Algorithm is less or equal to the expected importance value returned over the same time period by the CWP Algorithm. Moreover, the ratio between the two is equal to the probability* $1 - \alpha_f$ *of dropping a token:*

$$\mathbb{E}[R_{CWE}] = (1 - \alpha_f)\mathbb{E}[R_{CWP}]$$

Proof. Let's define $X(u)$ as the number of tokens a node u receives from others during the network evolution in the given time period and $X_d(u)$ as the number of such tokens that u drops (in both cases we do not consider tokens generated by u which did not pass through at least one other node).

Trivially, we have

$$\mathbb{E}[X_d(u)] \leq \mathbb{E}[X(u)]$$

because, from the definition, the number of tokens dropped $X_d(u)$ can only be a fraction of the total incoming tokens $X(u)$. Moreover, noting that each token is dropped or forwarded independently from the others, we can rewrite $X_d(u)$ as

$$X_d(u) = \sum_{i=1}^{X(u)} \mathbf{Pr}[\text{the node drops token i}] = \sum_{i=1}^{X(u)} (1 - \alpha_f) = (1 - \alpha_f)X(u)$$

and then, for the linearity property of the expectation of a random variable,

$$\mathbb{E}[X_d(u)] = (1 - \alpha_f)\mathbb{E}[X(u)]$$

But it follows from the definition that the two random variables $X(u)$ and $X_d(u)$ represent exactly the importance values over the time period returned by the CWP Algorithm and the CWE Algorithm, respectively. Therefore

$$\mathbb{E}[R_{CWE}] = (1 - \alpha_f)\mathbb{E}[R_{CWP}]$$

□

This means that choosing, for instance, a forward probability $\alpha_f = 0.85$, the expected count value for each node returned by the CWE Algorithm will be about 15% of the expected count value of the same node returned by the CWP Algorithm.

Working with smaller values has two consequences. On one hand, it allows to save a little computational power for updating the counter, since a much smaller number of operations are needed on a node. On the other hand, smaller values mean a greater convergence time to reach "stable" values, meaning that it would take more time at the beginning of the evolution process to reach a reasonable level of accuracy as well as more time to respond to the changes in the network. This is especially true for very sparse or slow-evolving networks in which there are only few links in each time step.

In general, however, as we will also experimentally show in Chapter 4, the two algorithms basically provide the same results for undirected networks. In the case of directed networks, however, things can greatly change. This is because it would be possible to have nodes with high in-degree and low (or zero) out-degree that will generate and receive a high number of tokens with very few (or none) occasions of forwarding/dropping them. For these "sink" nodes, the CWP Algorithm works as well as usual, while the results of CWE are corrupted by their inability of dropping the tokens they receive, so we expect CWP to outperform CWE when dealing with directed networks.

3.3 Dynamic degree centrality

Another important measure of relative importance of a node within a graph is degree centrality. As we said in Section 2.2, the degree centrality of a node u in a static graph is simply defined as the degree of u, that is the number of links (edges) incident to u in the graph. Instead, in a weighted graph the degree centrality of node u can be defined as the sum of the weights of all edges incident to u. In the case of a dynamic graph, there are two possible definitions of the degree of a node.

A natural approach would be to define the degree of a node as its total number of contacts with other nodes, that is the total number of edges incident upon itself in the evolution of the dynamic graph. This is the same as considering the weighted degrees in the aggregated weighted representation of the dynamic graph, in which each edge is labelled with the number of times it appeared during the evolution of the graph. This definition of degree centrality, that we call *volume degree*, has a major drawback in the context of dynamic evolving networks. Suppose we define

the importance of a user as the total number of contacts with other users she had, without taking into account any other information on the diversity of such contacts, and consider a dynamic network composed by a number of people that move about a room interacting with each other over the period of an evening. Now consider two lovers that spend all evening in a corner of the room with each other, without interacting with the other users. At the end of the evening, they will both have a high degree centrality value, because they had many interactions between themselves. But, although they are certainly very important one for the other, it seems quite wrong to consider them "important" with respect to the entire group of people. It would be more natural to consider as "important" the guy who moved a lot about the room, meeting and becoming friends with many different people.

The second possible way of defining degree centrality measure in a dynamic graph is counting the number of nodes with which at least an interaction occurred – we call it *absolute degree*. In the example of before, the guy that met many people in the room would gain importance with respect to the previous measure, while the two lovers' importance measure would be greatly downgraded. This is not a perfect measure either, since the total number of contacts can still be important to consider, but it seems overall a better way of trying to understand which users we consider the more "active".

Computing the volume degree at each user is trivial: all that is needed is a counter to be incremented every time a new interaction with another user occurs. Computing the absolute degree is also an easy problem, if there are no restrictions on the hardware resources: if all users are assigned a unique identifier at the beginning, each user could simply maintain a counter and a list of the identifiers of the users encountered so far, and upon each interaction consult such list for deciding whether to increment the counter or not. However, this approach does not scale well as the number of users in the network increases, since both the space required for storing the identifiers and the time necessary to scan the list would increase as well. Moreover, a low-power device could not have enough space to store the identifiers of all the (potentially thousands) users in the network.

This challenging aspect of efficient implementation and the considerations made before led us to consider absolute degree as the main degree centrality measure to focus on in our work, but we present a very simple procedure for computing volume degree as well. In our analysis we take into account into account both the directed and undirected network cases, and show how the computations are affected.

3.3.1 Absolute degree and Distinct Users Count Algorithm

Generally speaking, the agnets that form evolving networks cannot be assumed to be able to store the identifiers of potentially all their peers, that is the total number of users in the network. This is a problem when counting the number of distinct other nodes encountered, since it could be too expensive to store all their identifiers. To solve this problem it is needed a data structure that works with a small amount of resources and can give a reasonable approximation of the real value we would like to compute. The algorithm that we propose uses the famous or FM sketch [15] data structure for counting the number of distinct values in a sequence in just one pass.

As we already described in Section 2.3.1, it is a probabilistic counting technique based on the use of a hash function and a simple bitmap; when an element of the sequence arrives, its identifier is hashed and a corresponding bit of the bitmap is set to 1; the bitmap is then used to estimate the number of distinct elements observed. Several advanced variants have been proposed over the years,but whichever of them one chooses the important property guaranteed by this data structure is the *duplicate insensitivity*: if an element has already been inserted in the sketch, multiple further insertions of the same element do not affect any more the distinct count estimation. In the context of dynamic networks, this means that if a user maintains an FM sketch to estimate the number of distinct users she has encountered so far, multiple interactions with the same user will not affect such estimation.

A special consideration has to be made for directed networks. In this case, links are not symmetric any more, meaning that node A can "see" node B and not vice-versa; nodes will then have two degree measures (in-degree and out-degree) representing the number of incoming and outgoing links, respectively. This introduces the question of how a node should count the number of distinct users encountered, i.e. if a node should consider the in-degree, the out-degree or both for selecting the users to add to its count.

The solution to this problem comes from the nature of a particular type of evolving social networks that we analyze: ad-hoc mobile networks. In such systems there are mobile hosts that, when in proximity, communicate with each other through wireless communication links. Generally speaking, when a node detects the presence of another one it does not know its identity, and they can only identify each other by sharing informations. In this context and with directed communication links (due for example to characteristics of the specific wireless technology), a node A can only know the identity of node B when B transmits to A its own identifier. Therefore, for the distinct user counting problem a node can update its estimation only upon receiving transmissions from others, that is we are actually counting the absolute in-degree of the node. On the contrary, when detecting the presence of another node, it can transmit to it its own identifier but it is not able to update the estimation of the number of distinct users because it does not know the identifier of the other one.

This is a reasonable assumption; in directed networks a the distinction can be made between "popular" users (that receive many links from others, leading to high in-degree) and "active" users (that link to many others, leading to high out-degree), and in the context of measuring importance the first ones are much more relevant to us than the others.

DUC Algorithm: Distinct Users Count Let's again consider a dynamic evolving network with node set V represented by the Evolving Graph model, in which time is seen as a series of discrete time steps with a suitable duration. At time step t_0, each node of the network initializes a FM sketch; to guarantee the correct functioning of the algorithm, it is essential that all nodes implement not only the same version of the data structure, but also that they all use the same hash function(s) to insert elements in the sketch. At t_0 each node is also provided with an identifier that is guaranteed to be unique in the network. Then, throughout the evolution of

the network, whenever a node interacts with others in a time step, it sends its own id to them (and consequently receives theirs). Upon receiving an id, it simply uses it as key to update its FM sketch. At any given moment, it can then estimate the number of distinct nodes it has encountered by querying the FM sketch to get the estimated result.

Algorithm 3 DUC (Distinct Users Count)

```
UPON EVENT < init >:
    initialize a FM sketch sketch
    for all time steps t do                                  ▷ algorithm step
        let N_t(u) be the set of links (u, v) that exist in the network at time step t
        for all v ∈ N_t(u) do
            send packet to v containing id id_u
        end for
    end for

UPON EVENT < receive packet containing id id_w >:
    add id_w to sketch

UPON EVENT < query >:
    return estimated count of sketch
```

Figure 5: Pseudo-code of the DUC Algorithm at generic node u

A couple of observations can be made for this algorithm. The computations at each node consist essentially of the update operation of a FM sketch; such updates involve a number m of hashing operations and bit flips. The value of m depends on the implementation of the FM sketches; the more accurate one wants the estimations to be, the more hash functions and bitmaps are to be used in order to get better results. Then, for each FM sketch update $O(m)$ operations are needed; such operations can be efficiently implemented in parallel. As regards the query operation for estimating the number of distinct users encountered, it also depends on the implementation – each version different from the simple original one will need some sort of averaging on multiple estimations that are used to improve accuracy. The query operation is not expected to be performed very often anyway, so it does not represent a concern for the overall performances.

An analogous trade-off goes for the space needed at each node to store the FM sketch, that is given by the space to store the hash functions and the bitmaps representing the sketch. Good hash functions have been proposed [31] that only need two words of memory to be stored. The optimal bitmap size is logarithmic with respect to the total number of users in the network ($|V|$), so the algorithm overall requires $O(m \log |V|)$ bits of storage space per node.

On the other hand, the communication overhead between nodes does not depend on the implementation of the FM sketches, i.e. it does not depend on m. When communicating, the nodes transmit to each other just a small packet of information

containing the numerical identifier of the sender, so the algorithm introduces a constant communication overhead and the number of packets exchanged in the network goes with the number of interactions between the nodes.

Evaluating the performance of the algorithm on real data sets (see Chapter 4), we see how it is possible to get good approximations of the actual values using just a small value of m (~ 4). This means that it is feasible both from the point of view of computational power and storage space to implement the algorithm on low-power devices such as RFID tags with minimal overhead.

3.3.2 Volume degree and Total Interactions Count Algorithm

We defined volume degree of a node as the sum of the weights of all edges incident upon the node in the aggregated graph representing a network. In the case of our dynamic evolving network, the weights on the edges represent the frequency of the corresponding links between nodes, that is how many times such links appeared in the real network traces. As we said, this is a simple measure to compute at each node, since it is only needed to count how many interactions the node has with others. We also pointed out how this kind of degree centrality can fail to measure the importance of a node, since it can be exploited by "spammer users" that boost their relative importance values. However, the total number of interactions of a node can still be interesting to know in addition to other importance measures to understand how the node socially behaves among the others.

As we did with absolute degree, it must be discussed also here how the degree measure is defined in the case of directed networks. In this case it is possible to take into consideration both in-degree and out-degree; indeed, a node can increment its count of interactions upon detecting nodes that it sees (out-degree) and upon receiving transmissions from nodes it does not see (in-degree). Counting both, we have a sort of aggregated total degree analogous to the one in the undirected case. However, implementing the algorithm in this way, counting both in-links and out-links of a node, we would have a discrepancy when dealing with undirected network; in such cases, in-degree and out-degree of a node are always the same and should not be counted both, so the total number of interactions of a node returned by the algorithm would be twice the actual value. To avoid a loss of generality and the need to specify two different implementations for the undirected and directed network cases, as we did before we again considered only the volume in-degree, counting for each node only the number of incoming transmissions. Again, this is a reasonable assumption because in this way we are somehow computing the degree of popularity of a node among its peers.

TIC Algorithm: Total Interactions Count The TIC algorithm is very simple and easily implementable. Each node in the network just increments a counter by 1 every time another one links to it in a time step, that is whenever it receives a packet of information from another node. These packets are transmitted over the wireless medium whenever a node detects another one. At any given time, the count value represents the number of interactions of the node with others.

As we said, there is a different definition of "interaction" in undirected and directed networks. The algorithm only counts the in-degree of a node, so that no updates on the counter are made when dealing with out-links. This way, in the undirected case it counts the number of symmetric contacts with others while in the directed case it counts the number of incoming links.

Algorithm 4 TIC (Total Interactions Count)

```
UPON EVENT < init >:
    count = 0
    for all time steps t do                                  ▷ algorithm step
        let N_t(u) be the set of links (u, v) that exist in the network at time step t
        for all v ∈ N_t(u) do
            send generic packet to v
        end for
    end for

UPON EVENT < receive generic packet >:
    count ← count + 1

UPON EVENT < query >:
    return count
```

Figure 6: Pseudo-code of the TIC Algorithm at generic node u

This algorithm can be trivially implemented with negligible overhead in terms of computational power, storage space and communication overhead. In fact, rather than to deploy it in a standalone implementation, it makes more sense as a low cost "bonus feature" on top of the implementation of other decentralized importance measure algorithms like the ones described early in this chapter.

3.3.3 A simple extension: estimating the network size

In a fully decentralized approach to a dynamic network, each user is kind of left alone to herself, without any knowledge of the actual extension of the network she is part of. It is a natural consequence that a user would want to know how big is the outside world and try to understand how much of it she explored, and to have a measure of the percentage of people she interacted with. This is why it is interesting to analyze a way to compute at each node the approximate size of the network.

The obvious problem is that it is not possible to guarantee that each node will be able to estimate the size of the *whole* network. Since the only way of information propagation is through the communication between pairs of nodes, each user will only be able to have a reasonable idea of her local vicinity; but if, for instance, the network is composed by a number of disjoint connected components without any point of contact, two users belonging to different components will never be able to

know the existence of each other, let alone to have an idea of how big the network really is. Without assumptions on the network topology it is not possible to make this computation with a reasonable level of accuracy.

Nonetheless, this is not a big problem under the right perspective. Let's recall that what we are actually interested in is a relative measure of the importance of a user with respect to her neighbours rather than an absolute value. From this point of view, the fact that a user is only able to estimate the size of the sub-network she is part of is sufficient to our purposes, because her *relative* importance measure will refer to that sub-network. It is a simple matter of perspectives – if there is a group of people which only interact with each other ignoring the existence of other groups in the network, as regards them that group *is* the network, and a local estimation of the group size will suffice.

What can be done with an estimation of the (sub)network size? The most interesting use is in combination with the distinct users count (see Section 3.3.1) giving the possibility for a user to estimate the fraction (and percentage) of distinct users in the (sub)network with which she interacted. Then, the higher is this fraction, the more important the user will be considered, being seen as influential.

The problem of how this result can be achieved has something in common with the problem of counting the number of distinct users. To compute a distributed estimation of the network size, each node has, first of all, to count the distinct number of peers she interacted with; then, all these information at the individual users must be shared and somehow aggregated so to give the final result. Again, without a central computational point the users can only rely on the information exchanged with their neighbours.

The solution is in using again the FM sketches. As in the distinct count problem, each user maintains a FM sketch; this time, however, the sketch represents the estimated number of users in the connected component of the network to which the user belongs. The key idea is that whenever two users interact, they not only share with each other their identifiers but also their respective sketches; this way, they merge their individual knowledge and get both a clearer view of the environment they are in. In the case of directed networks the problem is somehow more difficult, since the communications are unidirectional and only one of the two users involved in an interaction will get an improvement on her estimation, leading to less information propagation and accuracy problems.

In terms of algorithmic steps it is a very simple procedure, but a careful analysis on the transmission overheads has to be made.

ENS Algorithm: Estimating Network's Size As in the other algorithms we presented, we model time as a sequence of discrete time steps. Each node in the dynamic graph maintains a FM sketch and upon detecting a link in a time step it transmits to its neighbour the bitmap stored in it, that represents the estimation given by the sketch. Upon receiving such information from another node, an update is made on the FM sketch in terms of bitwise OR between the two bitmaps; indeed, from the definition of the FM sketch it follows that this is exactly how the union operation on two sketches can be performed.

The analysis of the algorithm is similar to the one of the DUC algorithm. In

Algorithm 5 ENS (Estimating Network's Size)

```
UPON EVENT < init >:
    initialize a FM sketch sketch
    add id_u to sketch
    for all time steps t do                                   ▷ algorithm step
        let N_t(u) be the set of links (u, v) that exist in the network at time step t
        for all v ∈ N_t(u) do
            send packet to v containing the FM sketch sketch
        end for
    end for

UPON EVENT < receive packet containing a FM sketch sketch_w >:
    sketch ← sketch ∪ sketch_w                                ▷ bitwise OR

UPON EVENT < query >:
    return estimated count of sketch
```

Figure 7: Pseudo-code of the ENS Algorithm at generic node u

general, for the FM sketch implementation to give reasonable results it will need to use a number m of independent estimations, that is m hash functions and m bitmaps. The space needed to store all of them is again $O(m \log |V|)$ bits of memory, since the optimal size of a bitmap is logarithmic with respect to the total number of nodes in V. The computations needed at each node now involve m bitwise OR between pairs of bitmaps; the complexity of the bitwise operation depends on the length of the input bitmaps, so the algorithm has $O(m \log |V|)$ complexity also for the computational time. Since $\log |V|$ grows slowly as the network's size grows and since, as we will see in Chapter 4, we can choose a reasonably low value of m and still get good approximations, the hardware resources required by the algorithm are quite reasonable, and this makes it possible to implement the algorithm on low-power devices.

The main drawback of the algorithm is the communication overhead needed to work. Each time that couples of nodes interact, they have to transmit to each other the whole FM sketch, leading to $O(m \log |V|)$ information overhead. As we previously noted, it is important to keep the size of the packets exchanged between nodes as small as possible, in order to avoid transmission problems and inconsistencies; therefore, in real implementations the value of m has to be carefully tuned as to deal with this trade-off.

4 Experimental analysis

In this chapter, the performances of the algorithms described in Chapter 3 are evaluated through comprehensive simulations.

All the three network traces on which the experiments are performed come from real-world data in a way or another, and both the directed and undirected case are covered; for each network trace we show how it was obtained and its topological characteristics. Then we briefly describe the experimental setup for the simulations processes, including the input data and a summarization of the procedures used. We also describe the different performance metrics used for the evaluation of the different algorithms, motivating our choises case by case. Finally, we provide the simulation results over the dynamic network traces for all the importance measures introduced, and analyze how the corresponding algorithms succeed to give a good approximation of the "real" importance values computed on aggregated static representations of the same networks.

4.1 Network traces

Here we present the real-world network traces on which we simulate the algorithms proposed in Chapter 3 in order to evaluate them. Two of these networks are undirected and are the results of two projects in which the movements of real people who agreed to participate to the experiments were tracked. On the contrary, the third network is directed and was artificially built from the data made available by the MemeTracker project [30], considering blog articles/web pages referencing each other over time as nodes interacting in a dynamic network.

4.1.1 SocialDIS project

In the SocialDIS project [37], the movements of a group of volunteer students were (anonymously) tracked at the first floor of the Sapienza university Department of computer and systems science in Rome for a period of 5 days. Taking proximity as measure of connection between two students (nodes), a dynamic network was constructed taking into account the behaviour of people and their connections during the period of time considered.

20 RFID reader devices were placed in the rooms and corridors on the first floor of the building; these readers are simple wireless access points able to receive packets of information sent by tags and forward them to a central collector server via a LAN (Local Area Network). All the about 120 students that participated to the experiment were asked for generic information about themselves (age, sex and academic status), not enough however to make possible to identify the owner of a specific tag. They were also given an active RFID tag in order to collect information on both their position and proximity between two people. Indeed, each RFID tag periodically broadcasts packets of information about its approximate position and its social contacts (i.e. other tags which are near it). All proximity packets sent by the tags (containing the *timestamp* of the interaction, the *identifier* of the source tag and the identifiers of all the other tags involved) were then collected by the readers and

sent to a central server for offline processing. From the about 250 megabytes of data logs collected, it was possible to obtain a dynamic contact network considering the evolving graph of relationships for each timestamp of the time period considered; that is, for every second of the experiment we can build a graph representing a snapshot of the network at that moment, containing all the nodes and only a subset of edges between nodes in proximity of one another in that timestamp. Notice that each of these graphs is inherently undirected, as the proximity measure that we study is a two-way relationship.

Figure 8 shows the aggregated graph obtained from the data of the SocialDIS experiment. The size and colour of nodes indicate the number of neighbours (i.e. number of distinct nodes with which it had a contact), while the thickness of the edges indicates the number of interactions between two nodes. It appears clear from the figure how the network has a quite large connected component, with only few isolated nodes that did not interact at all with others. The representation of the network also highlights the most important nodes in terms of number of neighbours and contacts.

4.1.2 MACRO project

Similarly to the SocialDIS project, in the MACRO experiment [28] the movements of a group of volunteers were tracked for a whole evening in one of the rooms of the MACRO (Museo d'Arte Contemporanea di Roma) museum[1], site in Rome.

During the NEON exhibition opening on the 20th of June 2012, about 120 visitors agreed to submit some personal information like age sex, nationality, etc. (again, not enough to identify the owner of a specific tag) and to wear an RFID in the Enel Room of the museum in order to gather information about their approximate location and their social interactions. Information sent by the tags were collected by RFID readers located near the main artworks of the exhibition, that forwarded all the data to a central collector server for offline processing. Again, as in the SocialDIS experiment, a dynamic contact network was then constructed taking into consideration the proximity between people over time, corresponding to dynamic edges in the network.

A representation of the aggregated data of the experiment is shown in Figure 9. The size and colour darkness of circles represents the amount of social contacts of a visitor, while the thickness of the edges represents the total time of interaction between two visitors.

4.1.3 MemeTracker data set

MemeTracker [30] is a project that aims to track news phrases over the web. It scans every day news stories from a great number of websites – including both professional mass media and personal blogs – tracking the phrases and quotes that appear more frequently and how the corresponding stories propagate of fade away among the several news sources over time.

[1] http://www.museomacro.org/

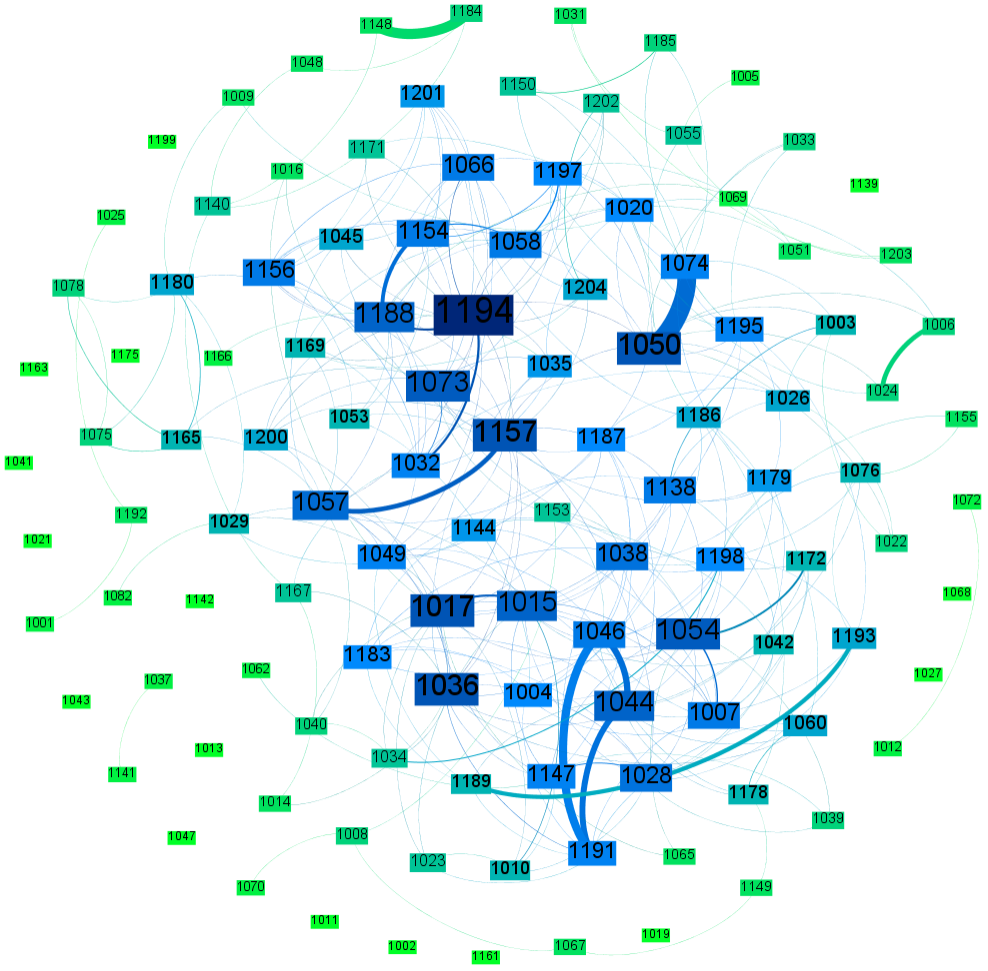


Figure 8: Aggregated graph of the SocialDIS experiment. Big and dark nodes have many direct interactions with others, while thick and dark edges appear many times during the evolution of the newtwork

We artificially build a dynamic evolving network from the data logs available for free download on the MemeTracker website. There are two kinds of data available:

- *Phrase cluster data*, that contain phrase clusters (sets of correlated phrases); for each phrase cluster there are all the phrases in the cluster and a list of URLs where the phrases appeared.
- *Raw phrases data*, that contain phrases and hyper-links extracted from each article/blog post specifying for each article the timestamp of its creation, the phrases contained in it and hyper-links pointing to other web pages.

The first type of data is not interesting for our purposes, since it does not seem possible to extract information regarding entities that interact among them over time. On the contrary, the second type of data proves very useful to us.

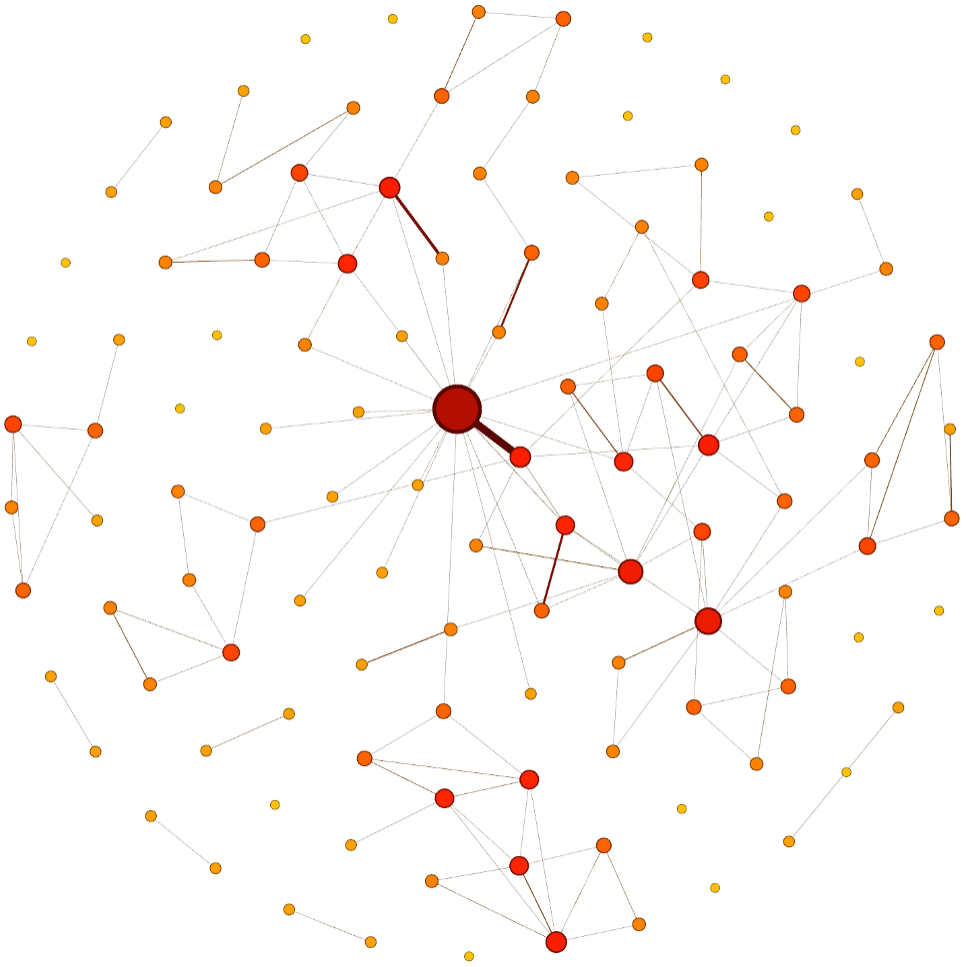


Figure 9: Aggregated graph of the MACRO experiment. Big and dark nodes have many direct interactions with others, while thick and dark edges appear many times during the evolution of the newtwork

In particular, we are able to build a dynamic network with:

- host URLs representing the nodes in the networks – multiple articles/posts on a website/blog are all considered as one source;
- hyper-links contained in an article representing out-edges toward other articles;
- the timestamp associated with each article representing the time with which each edge associated to a hyper-link in the article must be labelled.

With this network description in mind, we parse the raw phrases data downloaded from the MemeTracker website. We use the data log containing phrases data referring to the month of August 2008; in particular, given the huge amount of data

available, we consider the data tracked in just one day, from the 31st of July to the 1st of August 2008. Even if we only consider this relatively short period of time, the MemeTracker network consists of about five thousands nodes and about three times as many total edges during its evolution.

Parsing the data, we collect a list of article entries with time of creation and hyper-links pointing to others; to keep the computations as efficient as possible, at this time we also remove all the entries referring to host websites whose articles neither contain hyper-links nor are ever cited by any others – that is, host websites whose corresponding node would be isolated in the network.

Finally, we construct a JSON (JavaScript Object Notation) file whose entries represent the state of the network in each time step. Beginning from the starting timestamp and applying incrementally the time step duration, each entry is constructed by adding the source-target information of all edges present in the time step interval (that is, all hyper-links contained in articles created in the time step interval).

Figure 10 shows a representation of the network. Red and big nodes have higher in-degree values, while colour and thickness of the arcs represent their frequency in the network (that is how many times a particular arc appeared in the network evolution). For visualizational purposes, nodes with degree (summing in plus out) fewer than 3 are excluded.

Network	**Type**	**Nodes**	**Unique edges/arcs**	**Time steps**
SocialDIS	undirected	116	592	283611
MACRO	undirected	114	264	8634
MemeTracker	directed	4829	9718	86377

Table 1: Statistics about the topology of the networks. The number of edges (arcs) refers to the aggregated static representation of a network, that is it indicates the number of distinct edges (arcs) that appeared at least once during its evolution

4.2 Experimental setup

To validate the performance of the algorithms introduced in Chapter 3, we setup an environment for simulating the evolution of a network over time starting from the real data of the network traces.

The input network traces are represented by JSON data logs containing a list of entries representing the state of the network over discrete time steps. Each entry in the log contains:

- an entry identifier;
- a timestamp representing the time to which the state of the network is referred;

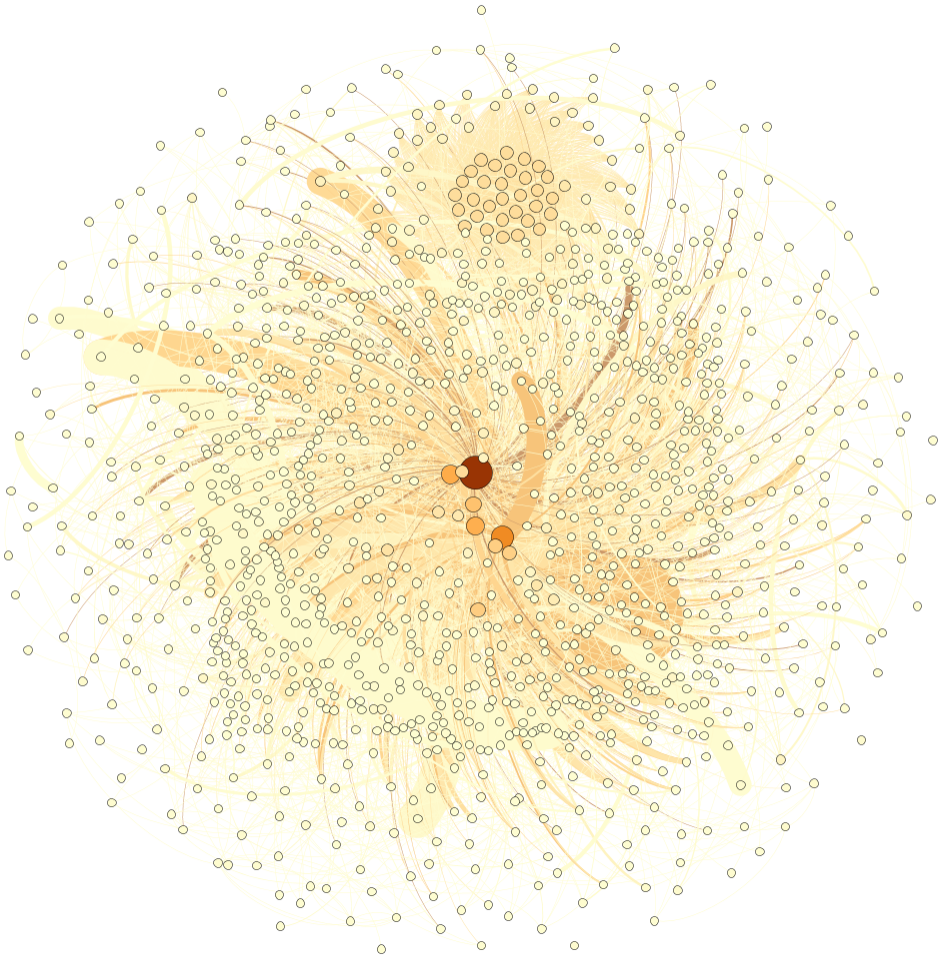


Figure 10: Aggregated graph of the MemeTracker data set. Big nodes have many in-links from others, while thick and dark arcs appear many times during the evolution of the newtwork

- a list of the nodes that are "active" in the time step (i.e. that have at least one interaction);
- a list of pairs <node ID, node ID>, one for each connection between nodes in the time step.

For each node it is indicated at least its identifier, and optionally other information about it (for instance, some general information about the person wearing a RFID tag in order to analyze the behaviour of different categories of people).

To run the simulations, we implemented a Java program that takes as input a JSON file of the kind described above. The Java class *Node* models a node, including its identifier, the memory storage and the data structures needed to run the desired algorithm.

At the beginning of the simulation, an instance of *Node* is created for each node in the network. Then, the JSON log file is scanned and, for each entry read, a procedure is called that executes for each individual node all the specific tasks required by the algorithm that is being evaluated (updates of counters, generation of random values, etc.) as well as all the information exchanges in the network (i.e. the exchange of packets between nodes). Such a procedure was implemented for every algorithm whose performance had to be evaluated. At the end of the simulation, the results of the algorithm computed at each instance of *Node* are then output in a suitable format in order to be analyzed.

Apart from the simple algorithm that counts the number of interactions of a node with others, all the algorithms involve some sort of randomized behaviour. Indeed, the two algorithms for computing a PageRank-like importance measure generate and forward tokens by computing random values, and the algorithms for computing degree centrality measure make use of FM sketches whose hash functions are generated uniformly at random at the beginning of the evolving process of the network.

To take into account the randomness of the procedures, when analyzing the results of each algorithm we executed 10 independent runs of the simulation (with the same input parameters), reporting the average values.

4.3 Performance metrics

We adopt different metrics for evaluating the performance of our algorithms.

In the case of the CWP and CWE Algorithms, we are estimating a PageRank-like value for each node. But rather than to achieve the exact PageRank values computed in the static representation of the network – which we believe would be really hard to achieve in a fully decentralized context – our algorithms aim to give good results considering the ranking of the nodes with respect to the exact values and our estimations, meaning that we expect to find on top of the ranked lists returned by the algorithms nodes that are also on top of the ranked list of "real" PageRank values. That is, instead of evaluating the error that we introduce in our estimations it makes more sense to evaluate the degree to which the algorithms capture the ranking of the real PageRank values and their ability to estimate which nodes are the most important. Thus, we evaluate the goodness of CWP and CWE by comparing the lists of estimated values returned by the experiments with the list of exact PageRank values computed on the static aggregated graph. The evaluation metric is the *precision* between the topmost k results of each pair of lists, with ranging values of k.

Definition 4.1. Given H the ranked set of estimated results and T the ranked set of exact results, **precision at k** ($P@k$) is defined as the fraction of the top-k elements of H that are also top-k elements of T.

$$P@k = \frac{|H_k \cap T_k|}{|H_k|}$$

given H_k and T_k the subsets of the top-k elements of H and T, respectively.

Ideally, the precision should be 1. The closer the precision to 1, the better the top-k elements in the estimated set correspond to the real top-k elements.

On the other hand, in the case of the algorithms estimating the degree centrality measures we aim to evaluate how they succeed to precisely compute the exact actual measures. Therefore, in this case we choose to evaluate the results in terms of average relative error and variance.

Definition 4.2. Given $\tilde{F}$ the estimated value computed by the algorithm and F the real value, the **relative error** is defined as the ratio of the absolute error to the correct value.

$$\eta = \frac{|\tilde{F} - F|}{F}$$

When the relative error is 0, the estimated measure is exactly equal to the real value. As the error increases, the accuracy of the estimation decreases; for instance, a relative error of 1 means that the estimated results is off by 100% the real value, either underestimating or overestimating – a very high error indeed!

Definition 4.3. In descriptive statistics, the **variance** of a set of N numerical data samples $x_1, x_2, \ldots x_N$ is a measure of spread that indicates the average of the squared deviation of each data sample from the mean value μ of the N samples.

$$\sigma^2 = \frac{\sum_{i=1}^{N}(x_i - \mu)^2}{N}$$

Thus, the variance is a measure of spread that takes into account both the deviations of the samples from the mean and how frequently these deviations occur. Lower values of variance are usually preferred, meaning that the samples of the set are more concentrated around the mean. On the contrary, high values of variance indicate very large deviations of the samples from the mean, so that the mean is not actually very representative of the sample set.

4.4 Experimental results

Here we show the results of the experiments and analyze them with respect to the performance metrics previously addressed. Since the nature of the computations varies from algorithm to algorithm, in each case we evaluat our results with different performance metrics.

4.4.1 CWP and CWE Algorithms

The CWP and CWE Algorithms aim to compute a PageRank-like distribution over the nodes of an evolving network; that is, they try to model in a fully decentralized way the centralized PageRank algorithm, which is defined over static networks. As we said, we are not so much interested in the very challenging goal of computing for each node a close approximation to its real PageRank score, but rather to reflect the relative importance of the nodes. Then, rather than to measure how much the individual computed values are close to the exact values, we are more interested in evaluating how the approximate distribution succeeds to preserve the ranking of the nodes in the real distribution computed over a static representation of the network.

Thus, we evaluate the performance of these algorithms by comparing the ranked list of nodes returned by them with the ranked list of exact PageRank values computed on the aggregated graph with weighted edges proportional to their frequency. The comparison between the two list is in terms of *precision at k*, for different values of k.

The exact PageRank distributions on the aggregated graphs of the three networks are computed through the classic iterative power method (see Section 2.6). Starting from the $n \times n$ weighted-adjacency matrix representing the static graph associated with our evolving network, we proceed as follows:

- we first normalize non-zero rows of the weighted-adjacency matrix dividing each element of such rows by the row sum, getting a transition probability matrix P;
- we replace all the elements of zero rows with $\frac{1}{n}$, getting a stochastic matrix P';
- we computed an irreducible matrix

  $$P'' = \alpha P' + (1 - \alpha)E$$

  with E the $n \times n$ square matrix with all elements $\frac{1}{n}$ and $\alpha = 0.85$;
- finally, starting from the initial distribution $\pi(0)$ such that $\pi_i(0) = \frac{1}{n}\ \forall i = 1 \dots n$, we iteratively compute

  $$\pi(k+1) = \pi(k)P''$$

  until reaching the convergence to a vector π, outputting it as exact PageRank vector.

The estimated PageRank vector, on the other hand, is given by running the two algorithms in the experimental context described in Section 4.2. As regards the parameters of the simulation, we choose $\alpha_g = 0.15$ and $\alpha_f = 0.85$. The average values of 10 independent simulation runs are considered.

Figure 11 shows the results for the CWP Algorithm. For each of the input networks, we plott the values of the precision at k of the algorithm's estimations, for every fifth value of k ranging from 1 to 100.

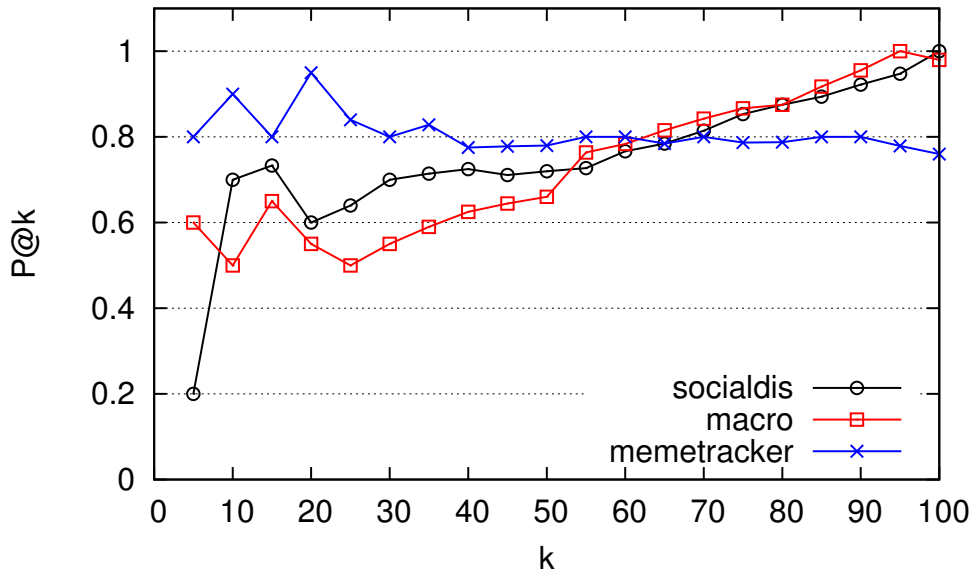


Figure 11: Precision values of the CWP Algorithm. The precision of the top-k nodes in the result list with respect to the top-k nodes in the list of real PageRank is considered, for ranging values of k

It can be noted from the chart how the algorithm achieves good precision values, meaning that it succeeds to overall reflect the ranking of the real distribution. As k (i.e. the size of the set considered) increases, the precision plots show an increasing trend as well. This is especially true for the two undirected networks (SocialDIS and MACRO), since they are composed by less than 150 nodes and in the extreme case of considering the whole sets the precision would trivially be 1. On the contrary, the MemeTracker network is composed by thousands of nodes, so the constant precision values that it shows as k increases are even more significant for proving the goodness of the algorithm than the (higher) precision values of the other two networks as k increases.

Another peculiarity of the MemeTracker network with respect to the other two is that it is much more sparse, thus having in proportion a greater number of nodes with few interactions with others. Such nodes will have very small PageRank values as well as very small (or zero) estimations in our algorithms; but in this case it is impossible to keep track of the small differences between them in our estimations, thus having a "flatter" distribution that it would not make sense to use for comparing ranks. This is why we only considered low values of k also for this network, when in theory we could have analyzed values of k up to the total number of nodes (about five thousands).

Figure 12 shows the precision values of the results returned by the CWE Algorithm. It can be noticed how the precision trend is very similar to the one of CWP as regards the undirected networks (SocialDIS and MACRO), while in the case of the directed network MemeTracker it gives very poor results in terms of precision values. We anticipated this problem in Section 3.2.3, while analyzing the two algorithms. This performance drop is due to the "sink nodes anomaly" of directed networks, in

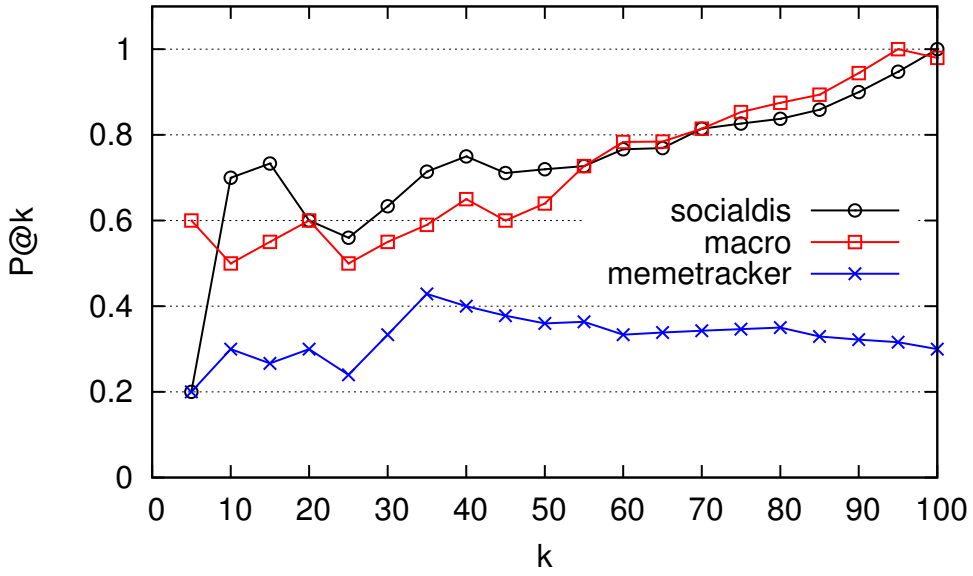


Figure 12: Precision values of the CWE Algorithm. The precision of the top-k nodes in the result list with respect to the top-k nodes in the list of real PageRank is considered, for ranging values of k

which nodes with only incoming links accumulate tokens without the possibility of "dropping" them, as the CWE algorithm requires as condition to increment their importance counter. On the contrary, CWP updates the counters as the tokens arrive at the node, so it does not suffer from this problem. The difference in the performance of the two algorithms, that can be better appreciated in Figure 13, is therefore given by the loss of accuracy in estimating the PageRanks of sink nodes, or nodes with otherwise very few out-links.

4.4.2 DUC Algorithm

At each node of an evolving network, the DUC Algorithm computes an estimation of the number of distinct nodes that interacted with it. As we said in Section 2.3.1, differently from the computation of a PageRank-like measure the distinct counting problem can be exactly solved in distributed systems, given that there are no restrictions on the storage space on the nodes. This is why when implementing an algorithm for computing approximations of the results with bounded space requirements, we are interested in measuring how the approximate values are close to the real ones; that is, the error that the approximations introduce.

We evaluate the performance of the DUC Algorithm results in terms of *relative error* and *variance* as follows. At each given time step during the simulated evolution of an evolving network, each node stores both the approximate value computed by the algorithm and a no-duplicate list of the distinct nodes encountered up to that time step (the size of the list then represents the number of distinct nodes encountered). At any given time, the node is then able to compute the relative error

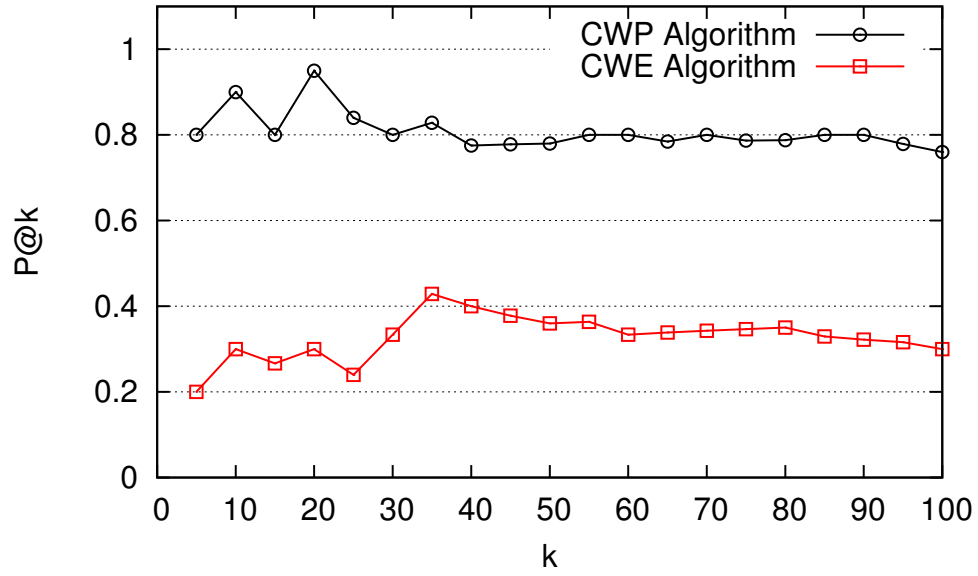


Figure 13: Precision values of the CWP and CWE Algorithms for the MemeTracker (directed) network; the loss of performance of the latter is due to the "sink nodes anomaly"

introduced by the algorithm. We divide each of our sample networks in 50 intervals of equal length (with the number of time steps in an interval depending on the total number of time steps that compose each simulated network) and study the state of the network at the end of each interval.

Defining V as the set of nodes of the network with cardinality $|V| = N$, at each of these 50 checkpoints we compute:

- the average relative error over all nodes $u \in V$:

$$\eta_{avg} = \frac{\sum_{u \in V} \eta_u}{N}$$

- the variance of the set of relative error values of all nodes:

$$\sigma^2 = \frac{\sum_{u \in V} (\eta_u - \eta_{avg})^2}{N}$$

In other words, during the evolution of the network we evaluate at regular points in time the average relative error introduced by the algorithm's approximations up to that certain point in time and the variance associated with these values.

There is also another important factor to consider in the evaluation. We recall that the algorithm makes use of FM sketches, and that this data structure can be implemented with a different number m of bitmaps and associated hash functions, leading to a trade-off between the accuracy of the estimations and the space and computational power required. To fully evaluate the algorithm, then, we run comprehensive simulations with 6 values of m ranging from 1 to 32, and analyze how

the performance of the algorithm changes accordingly. The graphical representations of plotting the simulations results are showed in Figures 14, 15 and 16 for the SocialDIS, MACRO and MemeTracker network respectively.

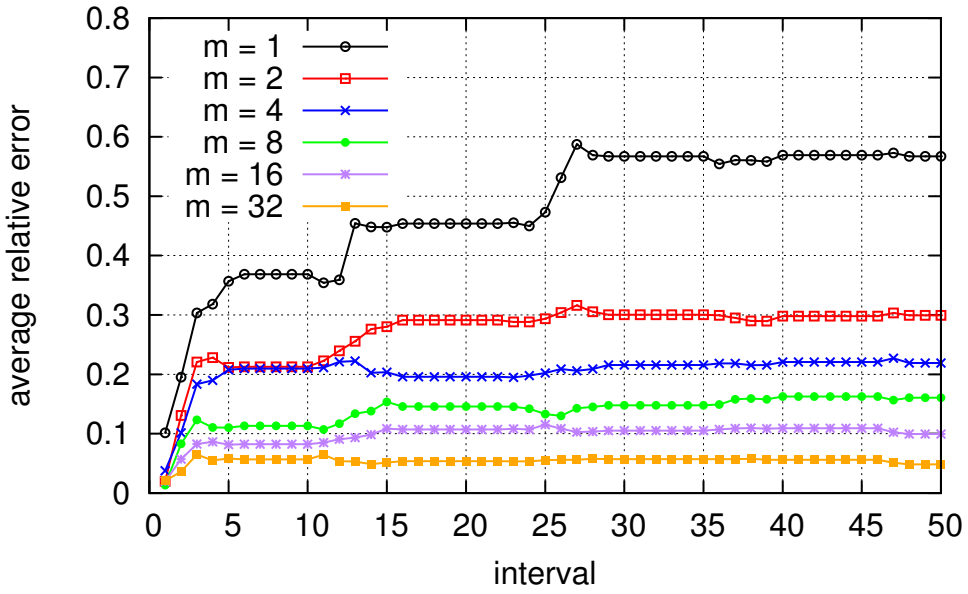
The evolution of the average relative error and corresponding variance for the SocialDIS network is showed in Figure 14. The first consideration is a trivial one: choosing $m = 1$ in the FM sketch implementation, we get high average errors in our estimations (up to almost 60% of the real average values, according to the chart) and with high variance, meaning that the estimation for an individual node will probably have an even poorer level of accuracy.

On the other hand, the charts show that, as we could expect, when choosing higher values of m both the average error and the variance decrease and become more "stable", leading to more accurate results both in average and if considering individual nodes. For instance, consider the case with $m = 4$. The chart plotting the errors shows how in that case the average error is more or less constantly around 0.2, meaning that in average the estimation at each node will be off by 20% of the real number of distinct nodes encountered, and the low variance values guarantee that the error at an individual node will not exceed that percentage too much. This is a reasonable trade-off between the space needed to store the m bitmaps and hash functions at each node and the results obtained. Of course, higher values of m guarantee more accurate results at the cost of more storage space needed on nodes, and an analysis of which value of m to choose must be made on a case by case basis.

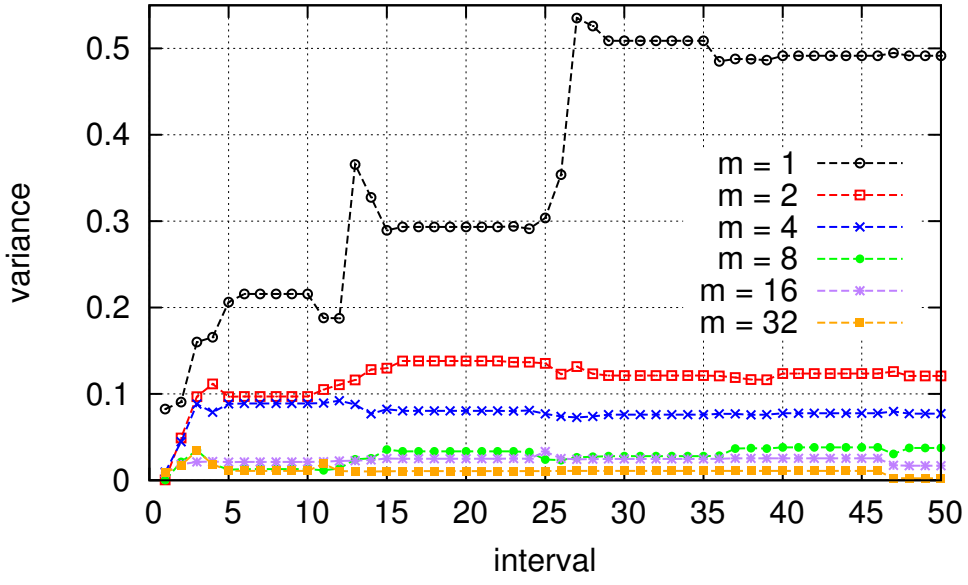
It is interesting to note how the error curves seem to eventually converge around some average error values after an initial adjusting phase. This is probably due to the nature of the evolving network, in the sense that most of the distinct user interactions happen in the early phases of the network evolution.

Figure 15 shows the same results for the simulations on the MACRO network. There are some similarities as well as some differences with respect to the charts regarding the SocialDIS network. As before, higher values of m correctly give us more accurate estimations of the distinct counting problem at each node. However, instead of having relatively "flat" curves as before, here we can notice a less stable behaviour of the error and variance trends during the evolution of the network. This can be simply explained by the different "life time" of the two networks – i.e. the number of time steps in which the evolution of the network is observed. As we said in Section 4.1, the SocialDIS project lasted for about one week, while in the MACRO project the experiment only took place over a span of one evening. The low life time of this last one is probably the reason of the inconstant behaviour of the error curves, since a certain amount of time is needed at the beginning of the process for the nodes to "adjust" their estimations. We believe that if the MACRO experiment had lasted longer, we would have observed an analogous trend to "stable" error values as in the results obtained for the SocialDIS network.

As regards the MemeTracker network, some considerations have to be made in order to better analyze the performance of the algorithm. As we pointed out while describing how this network is artificially built, it contains thousands of nodes, the majority of which has only very few interactions with others. Moreover, in the case of directed networks the DUC Algorithm only counts the incoming connections with distinct nodes, and a great number of such very poorly connected nodes has

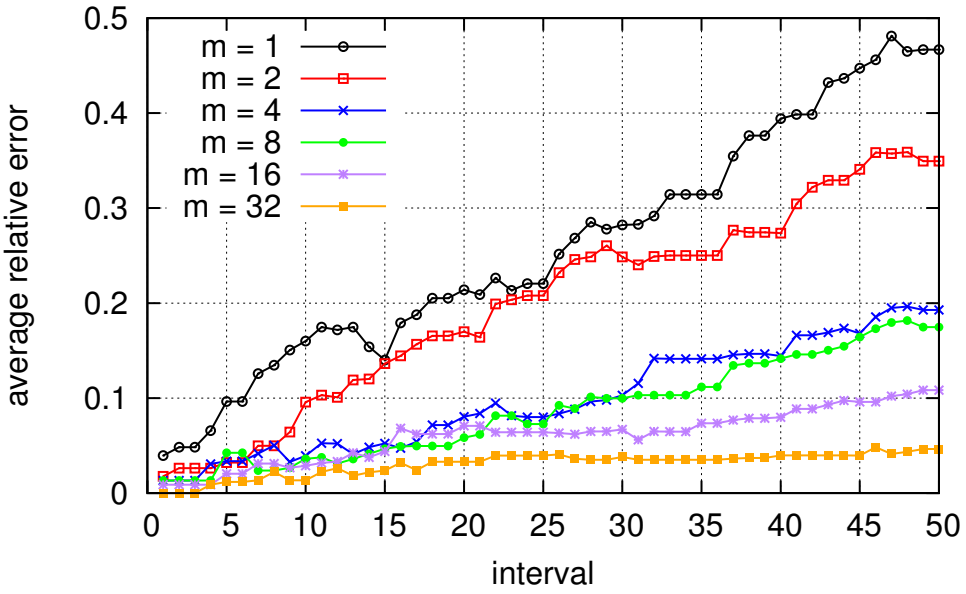


(a)

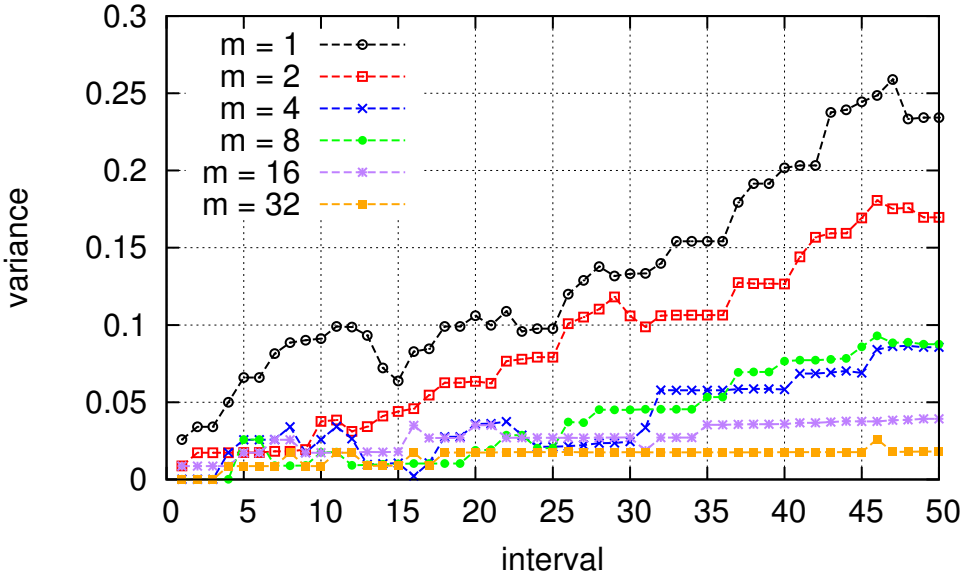


(b)

Figure 14: Average relative error (a) and variance (b) of the results returned by the DUC Algorithm over all nodes of the SocialDIS network every 5672 time steps. The relative error at each node is computed with respect to the real number of distinct nodes encountered up to the time step of evaluation. m represents the number of bitmaps and hash functions used in the implementation of the FM sketch at each node

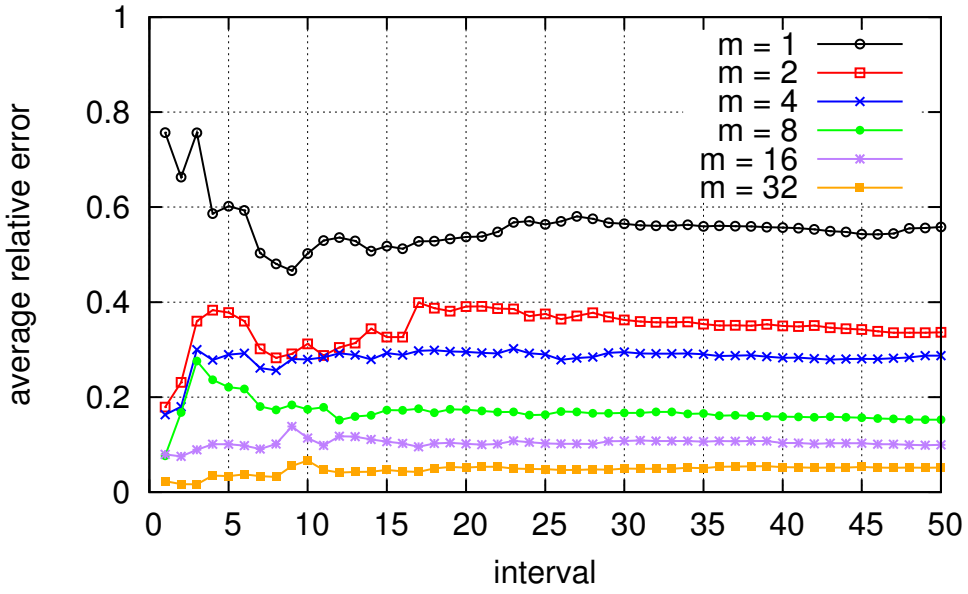


(a)

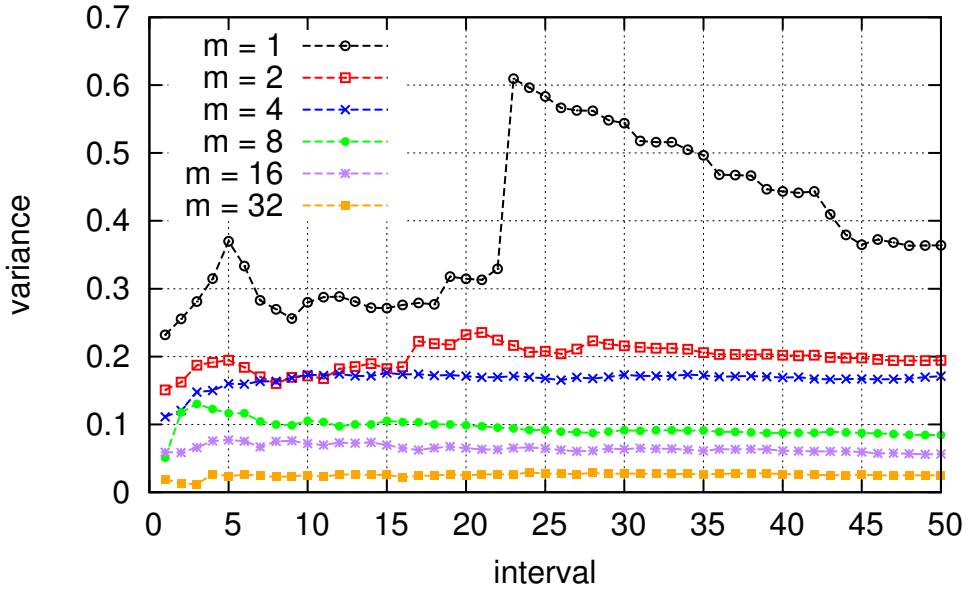


(b)

Figure 15: Average relative error (a) and variance (b) of the results returned by the DUC Algorithm over all nodes of the MACRO network every 172 time steps. The relative error at each node is computed with respect to the real number of distinct nodes encountered up to the time step of evaluation. m represents the number of bitmaps and hash functions used in the implementation of the FM sketch at each node



(a)



(b)

Figure 16: Average relative error (a) and variance (b) of the results returned by the DUC Algorithm over all nodes of the MemeTracker network every 1727 time steps. The relative error at each node is computed with respect to the real number of distinct nodes encountered up to the time step of evaluation. m represents the number of bitmaps and hash functions used in the implementation of the FM sketch at each node

actually zero incoming connections. The algorithm's estimation for such nodes will be trivially 0, so taking into account these nodes is not so interesting for evaluating the performance of the algorithm, since counting them when averaging over all estimations will lead to lower error values that do not reflect the average error on the values that the algorithm actually estimates. Thus, we analyze the average error and variance over only a subset of all nodes, excluding from the analysis all nodes whose real distinct count value is 0 (such nodes account for about half of the total number of nodes).

We are now ready to comment Figure 16, that shows the results of plotting error and variance values for the subset of nodes of the MemeTracker network defined above. Once more, it is experimentally shown how increasing the parameter m of the FM sketches leads to more accurate results and lower variance over the error values at every node. It can be noticed how, as in the case of the SocialDIS network (Figure 14), the error curves converge around some average error values after an initial adjusting period of time. This means that the life time of this network (about 24 hours) is enough for the network's nodes to reach a certain level of "stability" in their estimations, something that was not possible in the case of the MACRO network (Figure 15).

4.4.3 ENS Algorithm

The ENS Algorithm computes, at each node, an estimation of the network size. As we said while presenting the algorithm, depending on how the network evolves and without any previous assumption on this evolution, it can be impossible for a node to have an idea of the size of the whole network. This is due to the fact that a network can in general be partitioned in a number of disjoint sub-network, whose components never have any interactions with nodes in other sub-networks; in this case, a good estimation can only be computed for the size of the network's portion that a node and its neighbours "see".

So, more precisely, the ENS Algorithm computes at each node the size of the sub-network to which the node itself belongs. But the concept of "sub-network" of a dynamic network can be tricky to be exactly defined in a global way. Consider for instance a simple dynamic network with two disjoint subgroups G_1 and G_2 of users. Until the groups are disjoint, there are no "points of contact" between users from different groups, and each user can simply estimate the size of her own group. Suppose then that two users u_1 and u_2 belonging to G_1 and G_2 respectively interact at some point in time; they exchange their FM sketches so the new group they belong to is actually the whole network, and their estimation now refers to the supergroup obtained by merging G_1 and G_2. However, all the other nodes still only have a partial view of the network, and continue to do so until they interact with either u_1 or u_2. That is, there is in general no consensus between users on how many disjoint components there are and their sizes. When evaluating the performance of the algorithm, then, we actually measure the goodness of the estimation at a node with respect to the *local* view of the network structure at the node.

As in the case of the distinct interactions counting algorithm, we are interested in evaluating the errors that the algorithm introduces with respect to the real values

at each node. Thus, once again we use as evaluation metric the average relative error and associated variance computed in 50 regularly distributed points in time during the evolution of the three networks used for the simulations. At each time step chosen for the evaluation, the relative error between the algorithm's estimation up to that time step and the actual sub-network size is computed for each node, and the average error and variance over all these values is considered. Finally, the average error and variance of all 50 interval are plotted.

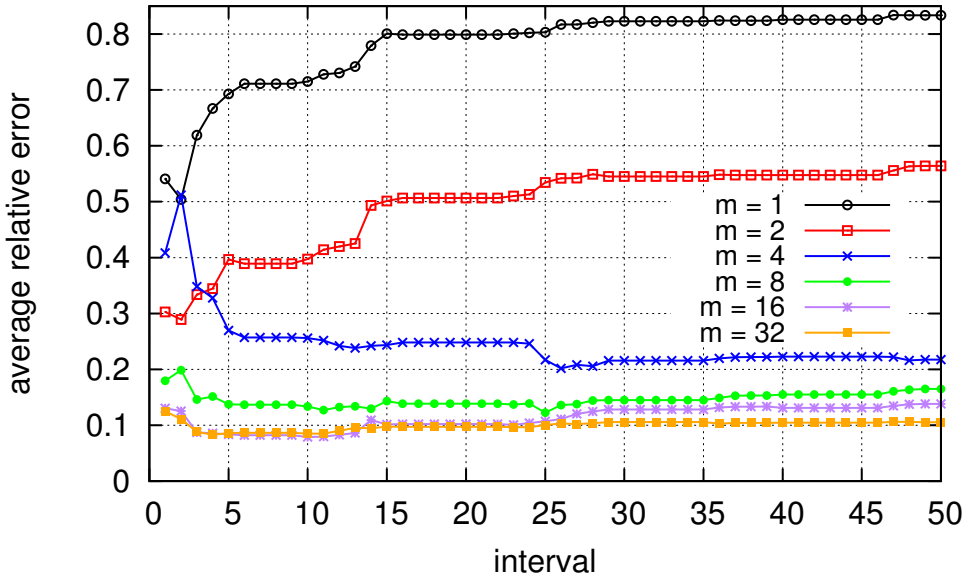
Moreover, as in the DUC Algorithm, a FM sketch is used at each node. We recall that a number m of hash functions and bitmaps can be used in the implementation of a FM sketch, with a trade-off between the space and computational power needed at each node – and network overhead in the connections, in this case – and the accuracy of the estimations. In this case the value of m is particularly important, since it affects not only the space to store the sketch and the complexity of the update operation, but also the amount of information that is exchanged between two nodes at each interaction. Indeed, all the m bitmaps must be exchanged in order to merge the two sketches, so the value of m must be suitably tuned.

We run multiple instances of the experiments with varying values of m, ranging from 1 to 32.

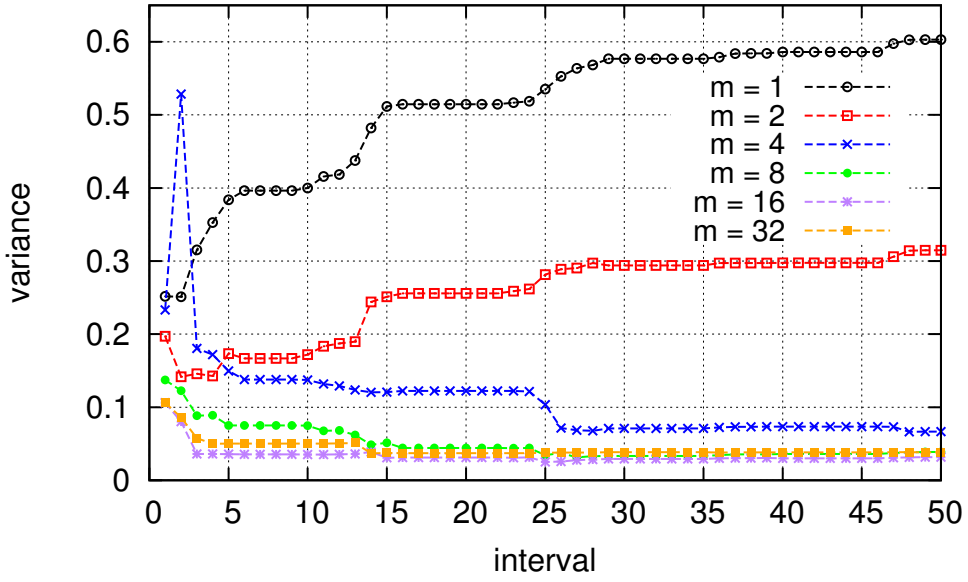
Figure 17 and 18 show the results of plotting the average relative error values and associated variance for the SocialDIS and MACRO networks, respectively. An analysis similar to the one for the results of the DUC Algorithm can be done. Once again, it is experimentally proved the very bad performance when using a trivial implementation of the FM sketch with just one bitmap and one hash function, and how increasing the parameter m of the FM sketch implementations we get both lower average error values, and thus more accurate estimations. It can again be noted how even choosing a relatively small value of m (for instance, 4) it is possible to have a reasonable level of accuracy for each estimation, given low average relative errors over all nodes and low variance associated to this set of values.

As regards the MemeTracker network, again we do not consider in the evaluation isolated nodes – whose sub-network's size is trivially one – so to not unfairly boost the performance of the algorithm. Moreover, the directed nature of this network calls for additional considerations. Here the problem of having nodes with different partial views of the network is even more pronounced, due to the unidirectional nature of the transmission links. While in undirected networks two nodes which interact among them both share their knowledge with each other, in directed networks only one of the two nodes (the "receiver" of the communication) will benefit from the interaction and update its (sub)network estimation. This inability of directed networks of properly sharing information leads to even more pronounced differences in the information about the structure of the network that each node holds.

Figure 19 shows the average relative error and variance over 50 intervals of the MemeTracker network. Although the results show a trend analogous to the ones of the other two networks and the average error values are more or less the same, it can be noticed how in this case the variance values are definitely higher, meaning that the deviations of the nodes' estimations are larger than before; this is due to the fact that having less information circulating among nodes leads in some cases to very wrong estimations.

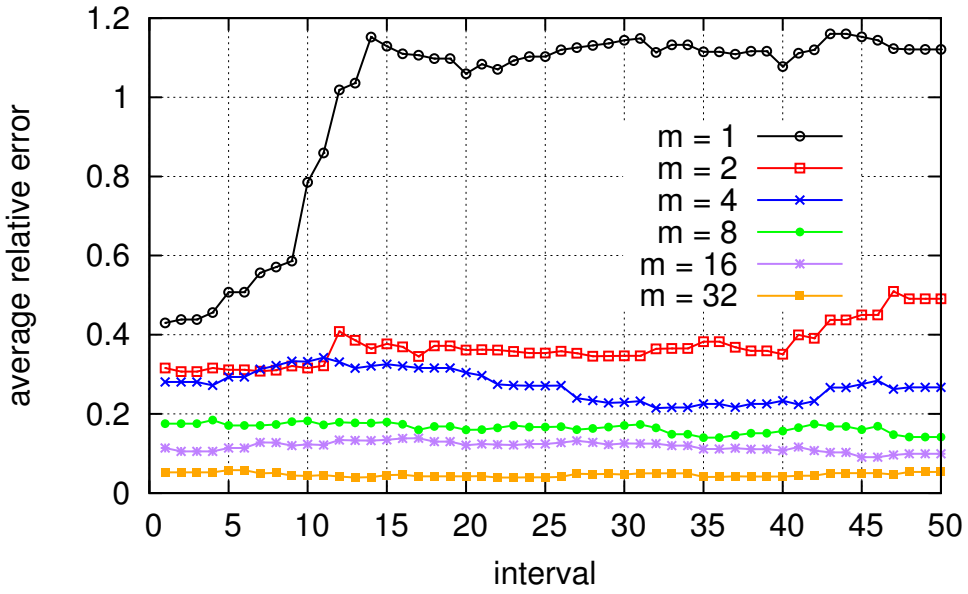


(a)

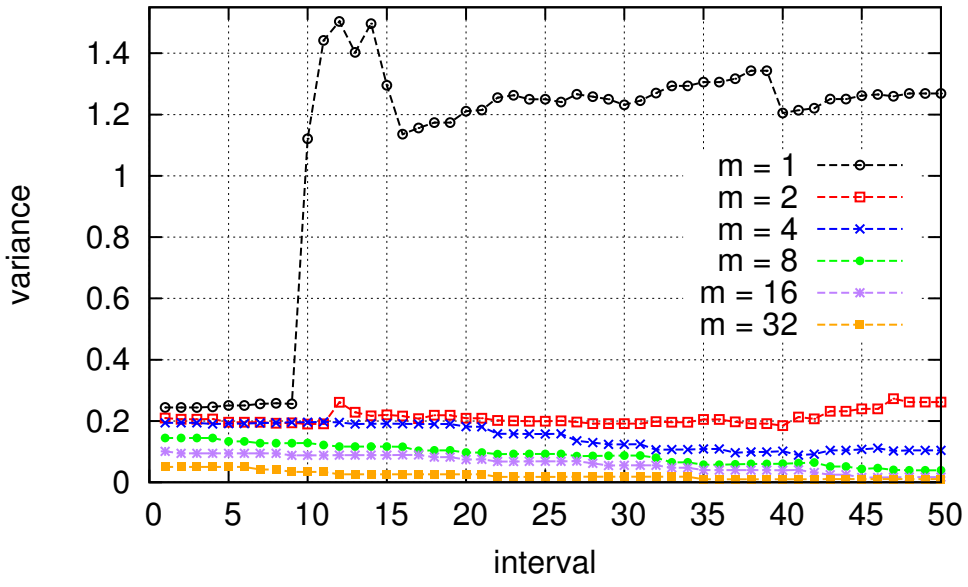


(b)

Figure 17: Average relative error (a) and variance (b) of the results returned by the ENS Algorithm over all nodes of the SocialDIS network every 5672 time steps. The relative error at each node is computed with respect to the real size up to the time step of evaluation of the sub-network to which the node belongs. m represents the number of bitmaps and hash functions used in the implementation of the FM sketch at each node

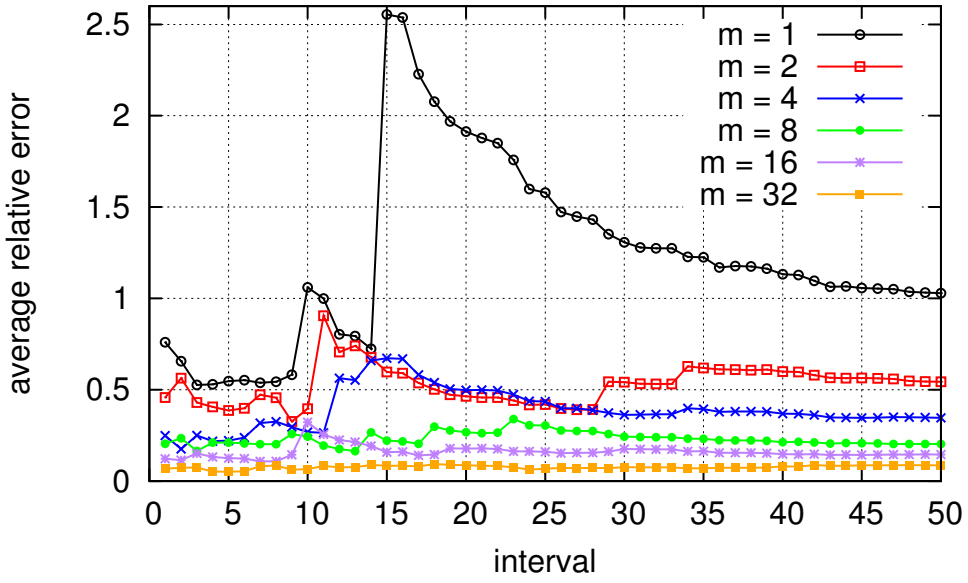


(a)

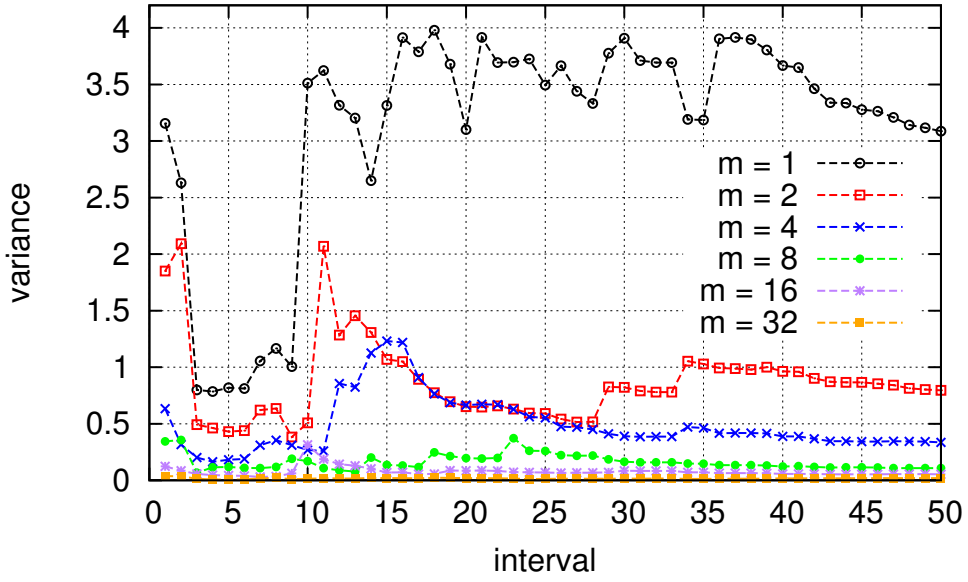


(b)

Figure 18: Average relative error (a) and variance (b) of the results returned by the ENS Algorithm over all nodes of the MACRO network every 172 time steps. The relative error at each node is computed with respect to the real size up to the time step of evaluation of the sub-network to which the node belongs. m represents the number of bitmaps and hash functions used in the implementation of the FM sketch at each node



(a)



(b)

Figure 19: Average relative error (a) and variance (b) of the results returned by the ENS Algorithm over all nodes of the MemeTracker network every 1727 time steps. The relative error at each node is computed with respect to the real size up to the time step of evaluation of the sub-network to which the node belongs. m represents the number of bitmaps and hash functions used in the implementation of the FM sketch at each node

5 Conclusions

In this article we study the problem of computing importance measures of nodes in an evolving network in a fully decentralized way, without any central point of computation or knowledge at each node of the whole network structure and assumptions on its future evolution. In particular, we focus on the decentralized computation of a PageRank-like measure and of two kinds of degree centrality measures.

CWP and CWE are lightweight algorithms which compute an approximation of the distribution given by the PageRank algorithm at each node by exchanging "tokens". The expected results of the two algorithms are equal when applied on undirected networks, while in the case of directed ones CWP outperforms CWE due to the "sink nodes anomaly". We analyze the idea between the two algorithms from a theoretical point of view as a problem of random walks on dynamic graphs, showing how their expected results are equal to the real PageRanks for static networks and in some important dynamic cases. We also support this theoretical intuition through an experimental analysis of the algorithms, evaluating them in terms of precision at k and showing how they give reasonable levels of precision for increasing values of k.

We also propose algorithms for computing two degree centrality measures on evolving networks: absolute degree, which counts the number of distinct other users encountered during the evolution of the network, and volume degree, which counts the total number of interactions. The absolute degree is computed at each node by the DUC Algorithm, which makes use of sketches by Flajolet and Martin (or FM sketches) to avoid duplicate updates and to solve the distinct counting problem. The experimental analysis shows how the algorithm returns good results in terms of average relative error introduced by the estimations at each node and associated variance computed on snapshots of the network at fixed time intervals. We show however a trade-off between the hardware resources required for the FM sketches implementations and the accuracy of the estimations. The volume degree is computed by the TIC Algorithm, which simply counts the number of interactions at each node and can be seen as a "bonus feature" to be implemented without significant additional costs along with one or more of the other algorithms.

Finally, we introduce an algorithm (namely the ENS Algorithm) for computing the size of the network component to which each node belongs, again making use of FM sketches. We describe how nodes in the same sub-component can have different estimations of its size, depending on how the network actually evolves. Again the performance of the algorithm in shown in terms of average relative error and associated variance over fixed time intervals.

In the design of the algorithms, much attention has been paid to their implementability on real-world devices. To make the algorithms as generally implementable as possible, we try to reduce as much as possible the computations required by each node. In particular, in the algorithms analysis we highlight the hardware resources required for the computations at each node and the network overhead introduced in the link transmissions, showing how their simplicity makes it possible to implement them also on low-power devices so to allow them to be deployed in a greater number of real-world applications.

Future work There are number of possible future developments of our work.

One possibility is to focus on introducing procedures for the decentralized computation of other centrality measures, like for instance betweenness centrality or more complex link analysis algorithms such as HITS and SALSA, briefly mentioned in Section 2.2. No centrality measure is inherently better than another one, but very much like the static case different metrics can be of interest case by case, depending on the specific network application.

A second line of work involves studying evolving networks with spacial and/or temporal dependencies. Dropping the assumption of independence between different time steps of the network makes the problem a great deal harder to analyze from a theoretical point of view.

Finally, another interesting future development regards privacy and security issues. A node can learn a good amount of information about the other nodes it encounters, by analyzing for example the frequency with wich it meets them. It would be preferable to avoid such possibility, for instance by encripting the information exchanges between nodes. Another issue is to analyze the case of selfish or byzantine behaviour of nodes. In our analysis we assume that the nodes can "trust" each other, meaning that nodes do not lie about their local importance values. In general, this could be not true in some applications, and counter-measures should be employed to avoid having such possibility in evolving network applications.

References

[1] Utku Günay Acer, Petros Drineas, and Alhussein A. Abouzeid. "Random walks in time-graphs". In: *Proceedings of the Second International Workshop on Mobile Opportunistic Networking*. MobiOpp '10. Pisa, Italy: ACM, 2010, pp. 93–100. ISBN: 978-1-60558-925-1. DOI: 10.1145/1755743.1755761. URL: http://doi.acm.org/10.1145/1755743.1755761.

[2] Romas Aleliunas et al. "Random walks, universal traversal sequences, and the complexity of maze problems". In: *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*. SFCS '79. Washington, DC, USA: IEEE Computer Society, 1979, pp. 218–223. DOI: 10.1109/SFCS.1979.34. URL: http://dx.doi.org/10.1109/SFCS.1979.34.

[3] Noga Alon, Yossi Matias, and Mario Szegedy. "The space complexity of approximating the frequency moments". In: *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. STOC '96. Philadelphia, Penn-

sylvania, USA: ACM, 1996, pp. 20–29. ISBN: 0-89791-785-5. DOI: 10.1145/237814.237823. URL: http://doi.acm.org/10.1145/237814.237823.

[4] C. Avin, M. Koucký, and Z. Lotker. "How to explore a fast-changing world (cover time of a simple random walk on evolving graphs)". In: *Automata, Languages and Programming* (2008), pp. 121–132.

[5] K. Avrachenkov et al. "Monte Carlo Methods in PageRank Computation: When One Iteration is Sufficient". In: *SIAM J. Numer. Anal.* 45.2 (Feb. 2007), pp. 890–904. ISSN: 0036-1429. DOI: 10.1137/050643799. URL: http://dx.doi.org/10.1137/050643799.

[6] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. "Fast incremental and personalized PageRank". In: *Proc. VLDB Endow.* 4.3 (Dec. 2010), pp. 173–184. ISSN: 2150-8097. URL: http://dl.acm.org/citation.cfm?id=1929861.1929864.

[7] Ziv Bar-Yossef et al. "Counting Distinct Elements in a Data Stream". In: *Proceedings of the 6th International Workshop on Randomization and Approximation Techniques*. RANDOM '02. Springer-Verlag, 2002, pp. 1–10. ISBN: 3-540-44147-6. URL: http://dl.acm.org/citation.cfm?id=646978.711822.

[8] Luca Becchetti, Andrea Vitaletti, and Giuseppe Persiano. *Private communication*. 2013.

[9] Dan Braha and Yaneer Bar-Yam. "From centrality to temporary fame: Dynamic centrality in complex networks". In: *Complexity* 12.2 (2006), pp. 59–63.

[10] Sergey Brin and Lawrence Page. "The anatomy of a large-scale hypertextual Web search engine". In: *Computer Networks and ISDN Systems* 30 (1998). Proceedings of the Seventh International World Wide Web Conference. ISSN: 0169-7552. DOI: 10.1016/S0169-7552(98)00110-X. URL: http://www.sciencedirect.com/science/article/pii/S016975529800110X.

[11] Iacopo Carreras et al. “Eigenvector centrality in highly partitioned mobile networks: Principles and applications”. In: *Advances in Biologically Inspired Information Systems* (2007), pp. 125–147.

[12] Andrea Clementi et al. “Distributed Community Detection in Dynamic Graphs”. In: *arXiv preprint arXiv:1302.5607* (2013).

[13] O. Denysyuk and L. Rodrigues. *Random walk on directed dynamic graphs*. 2006.

[14] Daniel Figueiredo et al. “Characterizing continuous time random walks on time varying graphs”. In: *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*. SIGMETRICS ’12. London, England, UK: ACM, 2012, pp. 307–318. ISBN: 978-1-4503-1097-0. DOI: 10.1145/2254756.2254794. URL: http://doi.acm.org/10.1145/2254756.2254794.

[15] Philippe Flajolet and G Nigel Martin. “Probabilistic counting algorithms for data base applications”. In: *Journal of computer and system sciences* 31.2 (1985), pp. 182–209.

[16] H Habiba, C Tantipathananandh, and T Berger-Wolf. *Betweenness centrality measure in dynamic networks*. Tech. rep. Technical report, DIMACS, Tech Rep, 2007.

[17] Marios Hadjieleftheriou, John W Byers, and John Kollios. *Robust sketching and aggregation of distributed data streams*. Tech. rep. Boston University Computer Science Department, 2005.

[18] Pan Hui, Jon Crowcroft, and Eiko Yoneki. “Bubble rap: social-based forwarding in delay tolerant networks”. In: *Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*. MobiHoc ’08. Hong Kong, Hong Kong, China: ACM, 2008, pp. 241–250. ISBN: 978-1-60558-073-9. DOI: 10.1145/1374618.1374652. URL: http://doi.acm.org/10.1145/1374618.1374652.

[19] Pan Hui et al. "Pocket switched networks and human mobility in conference environments". In: *Proceedings of the 2005 ACM SIGCOMM workshop on Delay-tolerant networking*. WDTN '05. Philadelphia, Pennsylvania, USA: ACM, 2005, pp. 244–251. ISBN: 1-59593-026-4. DOI: 10.1145/1080139.1080142. URL: http://doi.acm.org/10.1145/1080139.1080142.

[20] Jon M. Kleinberg. "Authoritative sources in a hyperlinked environment". In: *J. ACM* 46.5 (Sept. 1999), pp. 604–632. ISSN: 0004-5411. DOI: 10.1145/324133.324140. URL: http://doi.acm.org/10.1145/324133.324140.

[21] P. Krishna et al. "A cluster-based approach for routing in dynamic networks". In: *SIGCOMM Comput. Commun. Rev.* 27.2 (Apr. 1997), pp. 49–64. ISSN: 0146-4833. DOI: 10.1145/263876.263885. URL: http://doi.acm.org/10.1145/263876.263885.

[22] Fabian Kuhn, Nancy Lynch, and Rotem Oshman. "Distributed computation in dynamic networks". In: *Proceedings of the 42nd ACM symposium on Theory of computing*. STOC '10. Cambridge, Massachusetts, USA: ACM, 2010, pp. 513–522. ISBN: 978-1-4503-0050-6. DOI: 10.1145/1806689.1806760. URL: http://doi.acm.org/10.1145/1806689.1806760.

[23] Amy N. Langville and Carl D. Meyer. "A Survey of Eigenvector Methods for Web Information Retrieval". In: *SIAM Rev.* 47.1 (Jan. 2005), pp. 135–161. ISSN: 0036-1445. DOI: 10.1137/S0036144503424786. URL: http://dx.doi.org/10.1137/S0036144503424786.

[24] R. Lempel and S. Moran. "The stochastic approach for link-structure analysis (SALSA) and the TKC effect". In: *Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking*. Amsterdam, The Netherlands: North-Holland Publishing Co., 2000, pp. 387–401. URL: http://dl.acm.org/citation.cfm?id=347319.346324.

[25] Kristina Lerman, Rumi Ghosh, and Jeon Hyung Kang. "Centrality metric for dynamic networks". In: *Proceedings of the Eighth Workshop on Mining and Learning with Graphs*. MLG '10. Washington, D.C.: ACM, 2010, pp. 70–

77. ISBN: 978-1-4503-0214-2. DOI: 10.1145/1830252.1830262. URL: http://doi.acm.org/10.1145/1830252.1830262.

[26] Wei Liu et al. "On Compressing Weighted Time-evolving Graphs". In: (2012).

[27] László Lovász. "Random walks on graphs: A survey". In: *Combinatorics, Paul erdos is eighty* 2.1 (1993), pp. 1–46.

[28] *MACRO Project*. URL: http://wiserver.dis.uniroma1.it/cms/index.php/projects/15-macro.

[29] *Markov chain (Wikipedia)*. URL: http://en.wikipedia.org/wiki/Markov_chain.

[30] *MemeTracker Project*. URL: http://www.memetracker.org.

[31] Rajeev Motwani and Prabhakar Raghavan. *Randomized algorithms*. Cambridge university press, 1995.

[32] Mark EJ Newman. "The structure and function of complex networks". In: *SIAM review* 45.2 (2003), pp. 167–256.

[33] Joshua O'Madadhain and Padhraic Smyth. "EventRank: a framework for ranking time-varying networks". In: *Proceedings of the 3rd international workshop on Link discovery*. LinkKDD '05. Chicago, Illinois: ACM, 2005, pp. 9–16. ISBN: 1-59593-215-1. DOI: 10.1145/1134271.1134273. URL: http://doi.acm.org/10.1145/1134271.1134273.

[34] Larry Page et al. "The PageRank Citation Ranking: Bringing Order to the Web". In: (1998). URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=?doi=10.1.1.38.5427.

[35] A. Platis, N. Limnios, and M. Le Du. "Hitting time in a finite non-homogeneous Markov chain with applications". In: *Applied Stochastic Models and Data Analysis* 14.3 (1998), pp. 241–253. ISSN: 1099-0747. DOI: 10.1002/(SICI)1099-0747(199809)14:3<241::AID-ASM354>3.0.CO;2-3. URL: http://dx.doi.org/10.1002/(SICI)1099-0747(199809)14:3<241::AID-ASM354>3.0.CO;2-3.

[36] Markus Sobek. "Google dance–the index update of the Google search engine". In: *Electronic report* (2003).

[37] *SocialDIS Project*. URL: `http://wiserver.dis.uniroma1.it/cms/index.php/projects/3-socialdis`.

[38] Robin J Wilson. *Introduction to graph theory*. New York, NY, USA: John Wiley & Sons, Inc., 1986. ISBN: 0-470-20616-0.

[39] B.B. Xuan, A. Ferreira, and A. Jarry. "Computing shortest, fastest, and foremost journeys in dynamic networks". In: *International Journal of Foundations of Computer Science* 14.02 (2003), pp. 267–285.

[40] Dian Zhang et al. "An RF-based system for tracking transceiver-free objects". In: *Pervasive Computing and Communications, 2007. PerCom'07. Fifth Annual IEEE International Conference on*. IEEE. 2007, pp. 135–144.