Efficient computation of Harmonic Centrality on large networks: theory and practice

Master Thesis

Eugenio Angriman

Monday 10th October, 2016

Corso di Laurea Magistrale in Ingegneria Informatica

Advisors: Prof. Geppino Pucci, Prof. Andrea Pietracaprina

Università degli Studi di Padova, Scuola di Ingegneria

Anno accademico 2015-2016

Abstract

Many today's real-world applications make use of graphs to represent activities and relationships between entities in a network. An important concept in this context is the so-called *Centrality*, that is a method to identify the most influential vertexes of a graph. Centrality is represented through indexes such as the *Closeness Centrality* or the *Harmonic Centrality*. These indexes can be computed for each node in the network and they are both inversely proportional to the distance between the considered vertex and all the others. A couple of popular examples of Centrality indexes application are the recognition of the most influential people inside a social network or the identification of the most cited web pages.

However, the rapid growth of the amount of available data forces us to deal with extremely large networks. Consequently, the Centrality indexes computation for these kind of networks is often unfeasible since it is needed to be solved the All-Pairs-Shortest-Path problem, which requires a time that is at least quadratic in the size of the network. Nevertheless most of the applications often necessitate to find just a small set of vertexes having the highest centrality index values or, at least, a reliable centrality indexes estimation.

In the last few years a lot of progress has been made for the efficient computation of the *Closeness Centrality* index. D. Eppstein and J. Wang designed an approximation algorithm that efficiently estimates the value of the Closeness Centrality of each vertex of a given network while K. Okamoto, Wei C. and X proposed a fast algorithm that calculates the exact top-*k* Closeness Centralities. On the other hand, the *Harmonic Centrality* is a more recent metric and efficient algorithms for it have not been developed yet.

In this work we propose a Harmonic Centrality redesigned version of the efficient algorithms we cited above. We first provide the necessary theoretical background to prove the time and error bounds and then we present a Python implementation which makes use of graph-tool as main support library.

Contents

Dedicated to Andrea Marin, my first IT teacher. I can still remember the lesson when I wrote *"Hello World!"* for my first time. His genuine devotion was truly inspirational to me to find my own path.

Acknowledgements

My gratitude goes to my advisors Professor Geppino Pucci and to Professor Andrea Pietracaprina for proposing me such a challenging and interesting problem. Working with them on this project allowed me not only to put a great effort into a topic I am captivated by, but also to learn a lot from it.
I sincerely thank Dr. Matteo Ceccarello for his precious help and suggestions. For the assistance he provided to me I could rely on an excellent library and improve my code.
I am profoundly grateful to Michele Borassi and Elisabetta Bergamini for the detailed clarifications about their algorithms they provided to me.
To my mother, father, brother and sister who shared their great support, thank you.

Chapter 1

Introduction

Many of today's real-world applications make use of graphs in order to represent and analyze relationships between interconnected entities inside a network. An important concept of network analysis is the so-called *Centrality*. Centrality is a method to measure the influence of a node on the other nodes inside a network. In this context the influence of a node is intended as how close to all the other nodes the given node is and the distance between two nodes is represented as the length of the shortest path between them.
The problem of identifying the most central nodes in a network is extremely important among a wide range of research areas such as biology, sociology and, of course, computer science.
Centrality is represented by indexes such as *Closeness Centrality* and *Harmonic Centrality*. These indexes can be computed for each vertex in the graph that represents the network we would like to analyze. Closeness centrality was conceived by Bavelas in the early fifties when he developed the intuition that the vertexes which are more central should have lower distances to the other vertexes [2]. For each vertex in a graph, Closeness Centrality was defined as the inverse of the sum of the distances between the given vertex to all the other vertexes. However, for this definition to make sense the graph needs to be strongly connected because, without such condition, some distances would be unlimited resulting in score equal to zero for the affected vertexes. Because of this drawback, it is more troublesome to work both with directed graphs and with graphs with infinite distances but, probably, it was not Bavela's intention to use this metric with such graphs. Nevertheless it is still possible to apply Closeness Centrality to not strongly connected graphs just by not including unreachable vertexes into the sum of the distances.
An attempt to overhaul the definition of Closeness for not strongly connected graphs was made in the seventies by Nan Lin [15]. His intuition was to calculate the inverse average distance of a vertex v by weighting the Closeness of v using the square of the number of reachable vertexes from

v. By definition, he imposed isolated vertexes to have centrality equals to 1. Even though Lin's index seems to provide a reasonable solution to the problems related with the Bavela's definition of Closeness, it was ignored in the following literature.
Later in 2000 the idea underneath the concept of Harmonic Centrality was introduced by Marchiori and Latora who were facing the problem of providing an effective notion of "average shortest path" for the vertexes of a generic network. In their work [16] they proposed to replace the average distance, that was used for the Closeness centrality, with the *harmonic mean of all distances*. If we assume that $1/\infty = 0$ this kind of metric has the advantageous property to handle cleanly infinite distances we typically encounter in unconnected graphs. In fact the average of finite distances can be misleading, especially in large networks where a large number of pairs of nodes are not reachable. In this cases the average distance may be relatively low just because the graph is almost completely disconnected [3].
The formal definition of the *Harmonic Centrality* was introduced by Yannick Rochat in a talk at ASNA 2009 (Application of Social Network Analysis). He took inspiration from the work of Newman and Butts who gave a brief definition of this centrality metric as the sum of the inverted distances a few years before [20], [5]. The definition given by Yannick also includes a normalization term equals to the inverse of the number of vertexes minus one in order to obtain centrality index between zero and one that is cleaner and more preferable [25].
A couple of years later Raj Kumar Pan and Jari Saramäki adopted a very similar approach in their article on *temporal networks* [23] in order to deal with *Temporal Closeness Centrality*. Concisely, they used the same definition of Harmonic Centrality given by Yannick to compute the Temporal Closeness Centrality. The only difference is that Pan and Saramäki considered an "average temporal distance" τ_{ij} between two vertexes $i, j \in V$ instead of the shortest distance d_{ij}. As they wrote in their paper this metric allowed them "to better account for disconnected pairs of nodes".

In the last decades the amount of available data rose exponentially and the Centrality computation has became as important as computationally unfeasible. In the 2000s researchers proposed some new and faster approaches to compute the Closeness Centrality on large networks or a reasonable approximation of it.
To begin with, in 2004 David Eppstein and Joseph Wang designed a randomized approximation algorithm for the computation of Closeness Centrality in weighted graphs. Their method can estimate the centrality of all vertexes with high probability within a $(1+\varepsilon)$ factor in $\mathcal{O}(logn/\varepsilon^2(n\log n + m))$ time. This is possible by selecting a subset S of $\Theta(logn/\varepsilon^2)$ random vertexes, then they estimate the centrality of each node using only the vertexes in S as the target nodes instead the whole set V [7].

However, the majority of the applications require to calculate the top-k most central vertexes of a graph. For this purpose Kazuya Okamoto, Wei Chen and Xiang-Yang Li presented in 2008 a new algorithm that ranks the exact top-k vertexes with the highest Closeness Centrality in $\mathcal{O}((k + n^{2/3}\log^{1/3} n)(n \log n + m))$. Their strategy makes use of the Eppstein et al. algorithm in order to obtain an estimation of the centrality of each vertex. Then, they create a candidate set H with the top-$k + \hat{k}$ most central vertexes of the estimated vertexes. Finally, they compute the exact Closeness Centrality for each element inside H [21].
Another algorithm for the fast computation of the exact top-k Closeness Centralities was published in 2015 by Michele Borassi, Pierluigi Crescenzi and Andrea Marino. They designed a BFSCut function that is called for each vertex v in the graph. Briefly, this function starts the Breadth First Search using v as source and it stops as soon as an upper bound of the Closeness Centrality of v is less than the k-th Closeness Centrality [4].

So far efficient algorithms for the approximation or the exact computation of the Harmonic Centrality have not been designed yet, probably for the reason that it is a more recent metric than the Closeness Centrality. Actually the algorithm presented by Borassi et al. has already been generalized by the same authors in order to compute also the Harmonic Centrality. The purpose of our work is to provide high performance algorithms for both the computation and approximation of the Harmonic Centrality on large networks. Therefore, we re-designed the approaches described by Eppstein et al. and Okamoto et al. in order to obtain for the Harmonic Centrality the same results they achieved with the Closeness Centrality. Then, we will compare these two approaches with both the basic algorithm and the with the algorithm designed by Borassi et al. Furthermore, since neither Eppstein et al. nor Okamoto et al. specified the exact number of random samples to choose for their algorithms (they provided provided an asymptotic term only), we will change the multiplicative constants in front of the asymptotic terms in order to verify whether is it possible to reduce the running time of the algorithms without compromising their precision.

The main results we achieved through this work are the following. First of all we created a strong theoretical background that supports the efficient computation of the Harmonic Centrality index. More precisely we proved the following two statements:

- It is possible to approximate all the Harmonic centralities of a graph within a ε error bound for each vertex in $\mathcal{O}\left(\frac{\log n}{\varepsilon^2}(n \log n + m)\right)$ time through the Eppstein et al. algorithm we redesigned
- It is possible to compute the exact top-k Harmonic centralities of a graph with high probability in $\mathcal{O}\left(\left(k + n^{\frac{2}{3}}\log^{\frac{1}{3}} n\right)(n \log n + m)\right)$ time

through our new implementation of the Okamoto et al. algorithm

Furthermore, we observed that not only both the algorithms we implemented are considerably faster than the standard approach that solves the APSP problem which is quite obvious, but our new version the Eppstein et al. approximation algorithm is, in some cases, even more competitive than the Borassi et al. implementation, especially for bigger networks and higher k values.
Another interesting aspect of our experimental results concerns the precision achieved by our approximation algorithm. In short we noticed that the actual errors were considerably lower than the corresponding upper bound ε and if we lower the number of random samples the error grows linearly. This means that it is possible to save a considerable amount of time by reducing the number of random samples without compromising the algorithm's precision.
We applied these conclusions also on our revised version of the Okamoto et al. algorithm and we noticed that it could still compute the exact top-k Harmonic centralities but with a considerable reduction of its running time.

Our algorithms have been entirely implemented in Python 3.5.2 [1] and we used the library graph-tool 2.18 [24] for network support since it can be easily integrated into Python scripts but all its algorithms are written in C++ for better performances.

This thesis is organized as follows. In Chapter 2 we formally introduce the required background including notations, definitions and the terminology that will be used in the following chapters.
Chapter 3 is dedicated to a thorough description of the Eppstein et al., Okamoto et al. and Borassi et al. algorithms for the computation of the Closeness Centrality.
Chapter 4 is concerned with the description of how we adapted these algorithm for the computation of the Harmonic Centrality including a complete theoretical support.
Chapter 5 presents and comments the experimental setup and the results in terms of time and precision we obtained by executing our algorithms on several large social and authorship networks.
Finally, Chapter 6 summarizes the conclusions of this work and illustrates some indications for potential future developments.

Chapter 2

Preliminaries

In the previous chapter we mentioned the importance of the centrality concept in the large network analysis context and we illustrated two main centrality indexes which are largely used in a wide range of today's applications. We also briefly summarized three efficient strategies for both the approximation and the fast computation of the Closeness Centrality index.
Before describing in detail these techniques, let us give some fundamental definitions.

2.1 Centrality definitions

Let $G = (V, E)$ be a strongly connected graph with $n = |V|$ and $m = |E|$. The *Closeness Centrality* index is defined as follows [22]:

Definition 2.1 *Given a strongly connected graph $G = (V, E)$, the Closeness Centrality of a vertex $v \in V$ is defined as:*

$$c(v) = \frac{|V| - 1}{\sum_{w \in V} d(v, w)} \tag{2.1}$$

where $d(v, w)$ denotes the geodesic (i.e. shortest) distance from vertex v to w.

Another way to express $c(v)$ is the following [4]:

$$c(v) = \frac{|V| - 1}{f(v)}, \quad f(v) = \sum_{w \in V} d(v, w) \tag{2.2}$$

where $f(v)$ is also known as the *farness* of v.

However, if G is not strongly connected the definition becomes more complicated because $d(v, w)$ cannot be defined for unreachable vertexes. Even if we impose $d(v, w) = \infty$ for each pair of unreachable vertexes, then $c(v) = 0$ for each v that cannot reach all the vertexes in the graph, which is not very useful. The most common generalization that can be found in the literature is the following:

Definition 2.2 *Given a graph $G = (V, E)$, the closeness centrality of a vertex $v \in V$ is defined as:*

$$c(v) = \frac{(|R(v)| - 1)^2}{(|V| - 1) \sum_{w \in R(v)} d(v, w)} \tag{2.3}$$

where R(v) is the set of vertexes that are reachable from v.

On the other hand the *Harmonic Centrality* index is defined as follows:

Definition 2.3 *Given a graph $G = (V, E)$, the Harmonic Centrality of a vertex $v \in V$ is defined as:*

$$h(v) = \frac{1}{|V| - 1} \sum_{w \in V, w \neq v} \frac{1}{d(v, w)} \tag{2.4}$$

where d(v,w) represents the geodesic distance between v and w.

In the literature the normalization term is often omitted, consequently the Harmonic Centrality is also defined as:

$$h(v) = \sum_{w \in V, w \neq v} \frac{1}{d(v, w)} \tag{2.5}$$

Hereafter we will refer to the harmonic centrality according to the Definition 2.3 because it always takes values between 0 and 1 which can be compared more easily.

2.2 Description and complexity of the problem

Nearly all today's applications that exploit the concept of centrality are interested in identify the $k \geq 1$ most central nodes in a network i.e. the top-k centralities. Formally, the problem is defined as follows:

Definition 2.4 (Top-k Centrality Problem) *Given a graph $G = (V, E)$, a top-k centrality problem is to find:*

$$\text{argmax}_{\tilde{V} \subseteq V, |\tilde{V}| \geq k} \left(\min_{v \in \tilde{V}} c(v), |\tilde{V}| \right) \tag{2.6}$$

where c(v) is an arbitrary centrality index.

Note that the set $\tilde{V}$ might be greater than k for the reason that different vertexes may have the same centrality value, so they should be included.

The easiest and most naive strategy to solve this problem is to compute the centrality of each vertex in the graph, sort them and return the k most central vertexes. This is equivalent to solve the All-Pairs Shortest-Path problem (APSP) that is known to be unfeasible for large networks. Several algorithms can solve the APSP problem in $\mathcal{O}\left(nm + n^2 \log n\right)$ time [9, 12] and others in $\mathcal{O}\left(n^3\right)$ time [8] or even up to $\mathcal{O}\left(n^2 \log n\right)$ for random graphs [6, 10, 17, 19]. However, all of them are too slow, specialized, or excessively complicated and, for this reason, faster algorithms for the computation or approximation of the centrality indexes are needed.

Chapter 3

Efficient algorithms for the computation of Closeness Centrality

In the previous chapters we illustrated the importance of the top-k centrality problem and the difficulties of solving it efficiently because it is almost equivalent to the All-Pairs Shortest-Path problem. Since it is unfeasible to solve the APSP problem for large networks, researchers designed fast and reliable algorithms for both the approximation and the exact computation of the top-k Closeness Centralities of a network. In this chapter we illustrate in detail three of these algorithms and the theory underneath them.

3.1 Fast Top-k Closeness Centrality computation

We now present the algorithm designed by M. Borassi, P. Crescenzi and A. Marino for the efficient computation of the exact top-k Closeness centralities of a graph. The core of their intuition is represented by the BFSCut function they call for each vertex in the graph. This function calculates an upper bound $\tilde{c}(v)$ of the Closeness Centrality of the current vertex $v \in V$ and it stops as soon as $\tilde{c}(v)$ is less than the current k-th Closeness centrality c_k. Otherwise, it completes the BFS from node v and stores its Closeness Centrality value $c(v)$. It is important to point out that, in a worst case scenario, the complexity of this algorithm is the same as the the naive approach of solving SSSP for each all vertexes in the graph that is $\mathcal{O}(n^2 \log n + nm)$. The authors nonetheless noticed that the BFSCut function is far more efficient than solving APSP for the vast majority of the real-world cases.
Their algorithm's running time is also boosted by a degree-descending sort of all the graph's vertexes in order to run the BFSCut function for the highest degree vertexes first. This is thought to minimize the probability of performing a full BFS for non-top-k most central vertexes.
Before digging into the detailed description of this algorithm let us intro-

duce and demonstrate the correctness of the elements we will use such as the Closeness upper bound and other functions.

3.1.1 Upper bound of the Closeness Centrality

The BFSCut function takes as input two main parameters: the current node $v \in V$ and the current k-th Closeness centrality c_k. Then, it updates the value of $\tilde{c}(v)$ whenever the exploration of the d-th level of the BFS starting from v is finished ($d \geq 1$). $\tilde{c}(v)$ is then obtained from a lower bound on the *farness* of v:

Lemma 3.1 (Farness lower bound)

$$f(v) \geq \tilde{f}_d(v, r(v)) := f_d(v) - \tilde{\gamma}_{d+1}(v) + (d+2)(r(v) - n_d(v))$$

where:

- *$r(v)$ is the number of reachable vertexes from v*
- *$f_d(v)$ is the farness of node v up to distance d, that is:*

$$f_d(v) = \sum_{w \in N_d(v)} d(v,w)$$

where $N_d(v)$ is the set of vertexes at distance at most d from v i.e.:

$$N_d(v) = \{w \in V : d(v,w) \leq d\}$$

- *γ_{d+1} is the number of vertexes at distance exactly $d+1$ from v.*
- *$\tilde{\gamma}_{d+1}$ is an upper bound on γ_{d+1} and it is defined as follows:*

$$\tilde{\gamma}_{d+1} = \sum_{w \in \Gamma_d(v)} \text{outdeg}(u) \geq \gamma_{d+1}(v) = |\Gamma_{d+1}(v)|$$

where $\Gamma_{d+1}(v)$ represents the set of vertexes at distance exactly $d+1$ from v.

- *$n_{d+1}(v)$ is the number of vertexes at distance at most $d+1$ from v i.e.:*

$$n_{d+1}(v) = |N_{d+1}(v)| = |\{w \in V : d(v,w) \leq d+1\}|$$

Proof Clearly, for each $d \geq 1$ it holds that:

$$f(v) \geq f_d(v) + (d+1)\gamma_{d+1}(v) + (d+2)(r(v) - n_{d+1}(v))$$

Since $n_{d+1}(v) = \gamma_{d+1} + n_d(v)$ it follows that:

$$\begin{aligned} f(v) &\geq f_d(v) + (d+1)\gamma_{d+1}(v) + (d+2)(r(v) - \gamma_{d+1} - n_d(v)) \\ &= f_d(v) - \gamma_{d+1}(v) + (d+2)(r(v) - n_d(v)) \end{aligned}$$

Finally, since $\tilde{\gamma}_{d+1} = \sum_{u \in \Gamma_d(v)} \operatorname{outdeg}(v) \geq \gamma_{d+1}(v)$ we have:

$$f(v) \geq f_d(v) - \tilde{\gamma}_{d+1} + (d+2)(r(v) - n_d(v))$$

□

At this point the upper bound of the Closeness Centrality of v can be expressed as:

$$\tilde{c}(v) = \frac{(r(v)-1)^2}{(n-1)\tilde{f}_d(v)} \geq c(v) \tag{3.1}$$

and, apart from $r(v)$, all quantities are available as soon as all vertexes in $N_d(v)$ are visited by a BFS.

3.1.2 Computation of r(v)

The computation of $r(v)$ depends on the properties of the input graph $G = (V, E)$. More precisely, if G is *strongly connected* then $r(v) = n$ while, if G is *undirected* but not necessarily connected, $r(v)$ can be calculated in linear time. More effort is required if G is directed and not strongly connected.

Directed and not Strongly Connected Graphs

For this particular situation we assume to know for $r(v)$ an upper bound $\alpha(v) > 1$ and a lower bound $\omega(v)$. We will use $\alpha(v)$ and $\omega(v)$ to calculate a lower bound on $1/c(v)$:

Lemma 3.2

$$\frac{1}{c(v)} \geq \lambda_d(v) := (n-1)\min\left(\frac{\tilde{f}_d(v, \alpha(v))}{(\alpha(v)-1)^2}, \frac{\tilde{f}_d(v, \omega(v))}{(\omega(v)-1)^2}\right)$$

Proof From Lemma 3.1 it follows that:

$$f(v) \geq f_d(v) - \tilde{\gamma}_{d+1}(v) + (d+2)(r(v) - n_d(v))$$

If we denote $a = d+2$:

$$\begin{aligned} f(v) &\geq f_d(v) - \tilde{\gamma}_{d+1}(v) + a(r(v) - n_d(v)) \\ &= a(r(v)-1) - a(n_d(v)-1) - \tilde{\gamma}_{d+1}(v) + f_d(v) \end{aligned}$$

Finally, if we denote $b = \tilde{\gamma}_{d+1}(v) + a(n_d(v) - 1) - f_d(v)$:

$$f(v) \geq a(r(v) - 1) - b$$

where $a > 0$ (because $d > 0$), $b > 0$ (because $\tilde{\gamma}_{d+1}(v) \geq 0$) and $n_d(v) \geq 1$ (because $v \in N_d(v)$). Therefore:

$$f_d(v) = \sum_{w \in N_d(v)} d(v,w) \leq d(n_d(v) - 1) < a(n_d(v) - 1)$$

the first inequality holds because if $w = v$ then $d(v,w) = 0$, otherwise $w \in N_d(v) \Rightarrow 1 \leq d(v,w) \leq d$. The second inequality is trivial.
Considering the generalized definition of Closeness Centrality given by Equation 2.2 it follows that:

$$\frac{1}{c(v)} = (n-1)\frac{f(v)}{((r(v)-1)^2} \geq (n-1)\frac{a(r(v)-1)-b}{(r(v)-1)^2}$$

Let us denote $x = r(v) - 1$ and consider the function $g(x) = \frac{ax-b}{x^2}$ in order to study its minima. Its first order derivative $g'(x) = \frac{-ax+2b}{x^3}$ is positive for $0 < x < \frac{2b}{a}$ and negative for $x > \frac{2b}{a}$ if we consider only the positive values of x (which is reasonable if we assume $r(v) > 1$). This means that $\frac{2b}{a}$ is a local maximum and there are no local minima for $x > 0$. Consequently, for each closed interval $[x_1, x_2]$ where x_1 ad x_2 are positive, the minimum of $g(x)$ for $x > 0$ is reached in x_1 or x_2. Since $0 < \alpha(v) - 1 \leq r(v) - 1 \leq \omega(v) - 1$:

$$g(r(v) - 1) \geq \min(g(\alpha(v) - 1), g(\omega(v) - 1)) \qquad \square$$

Computing $\alpha(v)$ and $\omega(v)$ The computation of $\alpha(v)$ and $\omega(v)$ can be done during the *pre-processing* phase of the algorithm. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be the weighted acyclic graph made by the strongly connected components (SSCs) corresponding to the graph $G = (V, E)$. It is defined as follows:

- $\mathcal{V}$ is the set of SSCs of G.
- for any C,D $\in \mathcal{V}$, (C,D) $\in \mathcal{E}$ if and only if $\exists\ v \in C,\ u \in D$ s.t. $(v,u) \in$ E.
- for each SSC C $\in \mathcal{V}$ the weight $w(C) = |C|$, that is the number of vertexes in the SCC C.

So, if vertexes v and u are in the same SSC C, then:

$$r(v) = r(u) = \sum_{D \in R(C)} w(D) \tag{3.2}$$

where $R(C)$ represents the set of SSCs that are reachable from C in G. This means that we only need to compute a lower bound $\alpha(C)$ and an upper bound $\omega(C)$ once for every SSC C in G. To do so we first compute a topological sort $\{C_1, \cdots, C_l\}$ of $\mathcal{V}$ (where if $C_i, C_j \in \mathcal{E}$, then $i < j$) such that:

- C_l is a *sink* node i.e. outdeg$(C_l) = 0$.
- All sink nodes are placed consecutively at the end of the SSCs list.

then we use a dynamic programming approach in reverse topological order starting from C_l:

$$\begin{aligned} \alpha(C) &= w(C) + \max_{(C,D) \in \mathcal{E}} \alpha(D) \\ \omega(C) &= w(C) + \sum_{(C,D) \in \mathcal{E}} \omega(D) \end{aligned} \tag{3.3}$$

Note that processing the SSCs in reverse topological order (from C_l down to C_1) ensures us that the values on the right hand side of the equation above are available when computing the values $\alpha(C)$ and $\omega(C)$.
For example, at the first iteration we must compute $\alpha(C_l)$ and $\omega(C_l)$. We know that outdeg$(C_l) = 0$ so $\alpha(C) = \omega(C) = w(C_l) = |C_l|$. This applies to every other sink node in the list and provides the needed information to compute $\alpha(C_i)$ and $\omega(C_i)$ for the remaining non-sink nodes.

3.1.3 The algorithm

Here we illustrate the algorithm's pseudo-code including the BFSCut function for each combination of directed, undirected, strongly and not strongly connected plus the dynamic programming algorithm for the computation of the SSCs in the graph.

Algorithm 1 TopK_Clos($G = (V, E), k$)

```
1: Preprocessing(G)
2: $x_k \leftarrow 0$
3: for each node $v \in V$ do
4:     $c(v) \leftarrow 0$
5: end for
6: for $v \in V$ in decreasing order of degree do
7:     $c(v) \leftarrow$ BFSCut$(v, x_k)$
8:     if $c(v) \neq 0$ then
9:         $x_k \leftarrow Kth(c)$
10:    end if
11: end for
12: return $TopK(c)$
```

Where:

- x_k is the k-th greatest closeness computed until now
- $Kth(c)$ is a function that returns the k-th biggest element of c
- $TopK(c)$ is a function that returns the k biggest elements of c

The *Preprocessing* phase of the algorithm takes linear time and can be used to compute $\alpha(v)$ and $\omega(v)$.

Algorithm 2 BFSCut(v, x_k) function in the case of strongly connected graphs

```
1: Create queue Q
2: Q.enqueue($v$)
3: Mark $v$ as visited
4: $d \leftarrow 0$; $f \leftarrow 0$; $\tilde{\gamma} \leftarrow 0$; $nd \leftarrow 0$;
5: while !$Q$.isEmpty do
6:     $u \leftarrow Q$.dequeue()
7:     if $d(v,u) > d$ then //all nodes at level $d-1$ have been visited, $\tilde{c}$
   must be updated
8:         $\tilde{f} \leftarrow f - \tilde{\gamma} + (d+2)(n - nd)$
9:         $\tilde{c} \leftarrow \frac{n-1}{\tilde{f}}$
10:        if $\tilde{c} \leq x_k$ then
11:            return 0
12:        end if
13:    end if
14:    $f \leftarrow f + d(u,v)$
15:    $\tilde{\gamma} \leftarrow \tilde{\gamma}+$ outdeg($u$)
16:    $nd \leftarrow nd + 1$
17:    for w in adjacency list of u do
18:        if $w$.visited == $false$ then
19:            $Q$.enqueue($w$)
20:            $w$.visited $\leftarrow$ $true$
21:        end if
22:    end for
23: end while
24: return $\frac{n-1}{f}$
```

Algorithm 3 BFSCut(v, x_k) function in the case of undirected graphs (not necessarily connected)

```
1: Create queue Q
2: Q.enqueue($v$)
3: Mark $v$ as visited
4: $d \leftarrow 0$; $f \leftarrow 0$; $\tilde{\gamma} \leftarrow 0$; $nd \leftarrow 0$;
5: **while** !$Q$.isEmpty **do**
6:     $u \leftarrow Q$.dequeue()
7:     **if** $d(v, u) > d$ **then**
8:         $r \leftarrow$ BFSCount($u$) //Returns the number of reachable nodes from $u$
9:         $\tilde{f} \leftarrow f - \tilde{\gamma} + (d+2)(r - nd)$
10:        $\tilde{c} \leftarrow \frac{n-1}{\tilde{f}}$
11:        **if** $\tilde{c} \leq x_k$ **then**
12:            **return** 0
13:        **end if**
14:    **end if**
15:    $f \leftarrow f + d(u, v)$
16:    $\tilde{\gamma} \leftarrow \tilde{\gamma} +$ outdeg($u$)
17:    $nd \leftarrow nd + 1$
18:    **for** w in adjacency list of u **do**
19:        **if** $w$.visited == $false$ **then**
20:            $Q$.enqueue($w$)
21:            $w$.visited $\leftarrow true$
22:        **end if**
23:    **end for**
24: **end while**
25: **return** $\frac{n-1}{f}$
```

Algorithm 4 BFSCut(v, x_k) function in the case of directed and not strongly connected graphs

```
Create queue Q
Q.enqueue($v$)
Mark $v$ as visited
$d \leftarrow 0; f \leftarrow 0; \tilde{\gamma} \leftarrow 0; nd \leftarrow 0;$
while !$Q$.isEmpty do
    $u \leftarrow Q$.dequeue()
    if $d(v, u) > d$ then
        $(\alpha, \omega) \leftarrow$ GetBounds$(u)$ $//\alpha$ and $\omega$ have been computed for all SSCs in the Preprocessing
        $f_\alpha \leftarrow f_d(u) - \tilde{\gamma} + (d+2)(\alpha - nd)$
        $f_\omega \leftarrow f_d(u) - \tilde{\gamma} + (d+2)(\omega - nd)$
        $\lambda_d \leftarrow (n-1) \min\left(\frac{f_\alpha}{(\alpha(v)-1)^2}, \frac{f_\omega}{(\omega(v)-1)^2}\right)$
        if $\lambda_d > 0$ then
            $\tilde{c} \leftarrow \frac{1}{\lambda_d}$
            if $\tilde{c} \leq x_k$ then
                return 0
            end if
        end if
    end if
    $f \leftarrow f + d(u, v)$
    $\tilde{\gamma} \leftarrow \tilde{\gamma}+$ outdeg$(u)$
    $nd \leftarrow nd + 1$
    for w in adjacency list of u do
        if $w$.visited == $false$ then
            $Q$.enqueue($w$)
            $w$.visited $\leftarrow true$
        end if
    end for
end while
return $\frac{n-1}{f}$
```

Algorithm 5 Dynamic programming algorithm to compute for each SSC in G the lower bound $\alpha(C)$ and the upper bound $\omega(C)$ to the number of reachable vertices $r(v)$

```
1: $(\mathcal{V}, \mathcal{E}) \leftarrow$ graph of SSCs
2: $\mathcal{V}' \leftarrow$ topological sort of $\mathcal{V}$
3: $l \leftarrow |\mathcal{V}|$
4: $A \leftarrow \vec{0}\ //|A| = l$
5: $\Omega \leftarrow \vec{0}\ //|\Omega| = l$
6: **for** $i = l - 1$ down to 0 **do**
7:     **if** $\mathcal{V}'[i]$.outdeg $== 0$ **then**
8:         $A[i] = \mathcal{V}[i]$.weight
9:         $\Omega[i] = \mathcal{V}[i]$.weight
10:    **else**
11:        $O \leftarrow \{w \in \mathcal{V}$ s.t. $(\mathcal{V}[i], w) \in \mathcal{E}\}$
12:        $A[i] \leftarrow \mathcal{V}[i]$.weight $+ \max_{j \in O} A[j]$
13:        $\Omega[i] \leftarrow \mathcal{V}[i]$.weight $+ \sum_{j \in O} \Omega[j]$
14:    **end if**
15: **end for**
16: **for** $i = l - 1$ down to 0 **do**
17:    **for** $v \in \mathcal{V}[i]$ **do**
18:        $\alpha(v) = A[i]$
19:        $\omega(v) = \Omega[i]$
20:    **end for**
21: **end for**
22: **return** $(A, \Omega)$
```

3.2 Fast Closeness Centrality Approximation

In this section we describe the Closeness Centrality approximation algorithm conceived by D. Eppstein and J. Wang for undirected and weighted graphs. They were inspired by a particular feature called *small world phenomenon* that has been empirically observed to be typical of many social networks [18, 28]. This kind of networks are characterized by $\mathcal{O}(\log n)$ diameters instead of $\mathcal{O}(n)$. The strategy we are going to illustrate provides a near-linear time $(1+\varepsilon)$-approximation to the Closeness Centrality of all the nodes of a network of this type.
Shortly, the main intuition is the following: instead of solving the APSP problem they compute the Single-Source Shortest-Paths (SSSP) from each node contained in a subset S of random samples to all the other vertexes ($S \subset V$). This technique allows them to estimate the centrality of each $v \in V$ to within an additive error of $\varepsilon\Delta$ in $\mathcal{O}\left(\log n/\varepsilon^2 \left(n \log n + m\right)\right)$ time with high probability, where $\varepsilon > 0$ is the upper error bound for a single vertex centrality and Δ is the diameter of the graph. The approximated vertex centrality is calculate using the average distance to the sampled vertexes.

3.2.1 The algorithm

As we can see from the pseudo-code of Algorithm 6 (RAND), this approximation algorithm takes as inputs a graph G and the number of samples k. Then, it performs two main actions: it selects uniformly at random k samples from V and it solves the SSSP problem with each of them as source. Finally it computes an inverse Closeness Centrality estimator $1/\hat{c}(v)$ of the inverse Closeness Centrality $1/c(v)$ for each $v \in V$.

Algorithm 6 RAND($G = (V, E)$, k) D. Eppstein - J. Wang Closeness Centrality approximation algorithm.

```
1: for $i = 1$ to $k$ do
2:     $v_i \leftarrow$ pick a vertex uniformly at random from $V$
3:     Solve SSSP problem with $v_i$ as source
4: end for
5: for each $v$ in $V$ do
6:     $c(v) \leftarrow \hat{c}(v)$
7: end for
```

Let us point out that k is not arbitrary but it has been defined by the authors as $\Theta\left(\log n/\varepsilon^2\right)$. The estimated value of the closeness centrality used for each vertex $v \in V$ (line 6 of the RAND algorithm) is defined as follows:

$$\hat{c}(v) = \frac{1}{\sum_{i=1}^{k} \frac{nd(v_i,v)}{k(n-1)}} \tag{3.4}$$

$\hat{c}(v)$ estimates of $1/c(v)$ as the inverse of the average distance to the sampled vertexes, which is normalized by the $\frac{n}{k(n-1)}$ term.
In conclusion, if we adopt the $\mathcal{O}(n \log n + m)$ algorithm designed by Fredman and Tarjan for solving the SSSP problem [9], the total running time of this approximation algorithm is $\mathcal{O}(km)$ for unweighted graphs and $\mathcal{O}(k(n \log n + m))$ for weighted graphs. Thus, given that $k = \Theta\left(\log n/\varepsilon^2\right)$, we obtain an $\mathcal{O}\left(\left(m \log n/\varepsilon^2\right)\right)$ algorithm for unweighted graphs and an $\mathcal{O}\left(\log n/\varepsilon^2 \left(n \log n + m\right)\right)$ algorithm for weighted graphs.

3.2.2 Theoretical analysis

So far we described how the algorithm operates, which conditions the input graph must satisfy and how many samples we should choose. Now we must demonstrate that the algorithm RAND computes the inverse Closeness Centrality estimator $\hat{c}(v)$ for each $v \in V$ to within an upper error bound of $\xi = \varepsilon\Delta$ with high probability. For this purpose we will refer to the errors on the estimated Closeness centralities as independent, bounded and identically distributed random variables in order to exploit the Hoeffding lemma on probability bounds for sums of independent random variables, that is:

Lemma 3.3 (Hoeffding [11]) *If $x_1, x_2, \ldots, x_k$ are independent random variables, $a_i \leq x_i \leq b_i$ and $\mu = \mathrm{E}\left[\sum_{i=1}^{k} \frac{x_i}{k}\right]$ is the expected mean, than for $\xi > 0$:*

$$\Pr\left\{\left|\frac{\sum_{i=i}^{k} x_i}{k} - \mu\right| \geq \xi\right\} \leq 2e^{-\frac{2k^2\xi^2}{\sum_{i=1}^{k}(b_i-a_i)^2}} \tag{3.5}$$

In other words, we will denote x_i as $\frac{1}{\hat{c}(v_i)} - \frac{1}{c(v_i)}$, $1 \leq i \leq k$.

For the reason that Hoeffding's lemma requires the empirical mean of the $x_1, x_2, \ldots, x_n$ random variables i.e. $\sum_{i=1}^{k} \frac{x_i}{k}$ to be equal to μ, we need to prove that $\mathrm{E}\left[\frac{1}{\hat{c}(v)}\right] = \frac{1}{c(v)}$.

Theorem 3.4 *Given that:*

$$c(v) = \frac{n-1}{\sum_{i=i}^{n} d(v_i, v)} \quad \text{and} \quad \hat{c}(v) = \frac{1}{\sum_{i=1}^{k} \frac{nd(v_i,v)}{k(n-1)}}$$

then, $\mathrm{E}\left[\frac{1}{\hat{c}(v)}\right] = \frac{1}{c(v)}$.

Proof It is trivial that:

$$\begin{aligned}
\mathrm{E}\left[\frac{1}{\hat{c}(v)}\right] = \mathrm{E}\left[\sum_{i=i}^{k}\frac{nd(v_i,v)}{k(n-1)}\right] = \\
= \frac{n}{k(n-1)}\mathrm{E}\left[\sum_{i=1}^{k} d(v_i,v)\right]
\end{aligned}$$

Since we can interpret the geodesic distance between $d(v_i, v)$ as a random variable and, given $X_1, X_2, \ldots, X_k$ random variables, it is known that $\mathrm{E}\left(\sum_{i=1}^{k} X_i\right) = \sum_{i=1}^{k} \mathrm{E}(X_i)$. It follows that:

$$\frac{n}{k(n-1)}\mathrm{E}\left[\sum_{i=1}^{k} d(v_i,v)\right] = \frac{n}{k(n-1)}\sum_{i=1}^{k}\mathrm{E}\left[d(v_i,v)\right]$$

The expected value of the geodesic distance between v_i and v can be expressed as: $\mathrm{E}\left[d(v_i,v)\right] = \frac{1}{n}\sum_{j=1}^{n} d(v_j,v)$. So we have that:

$$\begin{aligned}
\frac{n}{k(n-1)}\sum_{i=1}^{k}\mathrm{E}\left[d(v_i,v)\right] &= \frac{n}{k(n-1)}\sum_{i=1}^{k}\frac{1}{n}\sum_{j=1}^{n} d(v_j,v) \\
&= \frac{1}{k(n-1)}\sum_{j=1}^{n}\sum_{i=1}^{k} d(v_j,v) \\
&= \frac{1}{k(n-1)}\sum_{j=1}^{n} kd(v_j,v) \\
&= \frac{1}{n-1}\sum_{j=1}^{n} d(v_j,v) \\
&= \frac{1}{c(v)}
\end{aligned}$$

□

So far we have proven that we are operating under the hypothesis required by the Hoeffding's bound. Now we can use it to demonstrate the following theorem:

Theorem 3.5 *Given an undirected, connected and weighted graph $G = (V, E)$, with high probability the algorithm RAND computes the inverse of the Closeness Centrality estimator $\hat{c}(v)$ for each vertex $v \in V$ to within an upper error bound $\xi = \varepsilon\Delta$ using $\Theta\left(\frac{\log n}{\varepsilon^2}\right)$ samples, where $\varepsilon > 0$ and Δ is the diameter of G.*

Proof As we suggested before we can exploit the Hoeffding's bound to calculate an upper bound on the probability that the error of $\hat{c}(v)$ is greater than $\xi = \varepsilon\Delta$. This can be done by imposing:

$$x_i = \frac{nd(v_i, v)}{n-1}, \quad \mu = \frac{1}{c(v)}, \quad a_i = 0, \quad b_i = \frac{n\Delta}{n-1}$$

Thus, given that $\mathrm{E}[1/\hat{c}(v)] = 1/c(v)$ we can re-write Equation 3.5 as follows:

$$\begin{aligned}
\Pr\left\{\left|\frac{1}{\hat{c}(v)} - \frac{1}{c(v)}\right| \geq \xi\right\} &= \Pr\left\{\left|\sum_{i=1}^{k} \frac{nd(v_i,v)}{k(n-1)} - \frac{1}{c(v)}\right| \geq \xi\right\} \\
&= \Pr\left\{\left|\sum_{i=1}^{k} \frac{x_i}{k} - \mu\right| \geq \xi\right\} \\
&\leq 2e^{-\frac{2k^2\xi^2}{\sum_{i=1}^{k}(b_i - a_i)^2}} \\
&\leq 2e^{-\frac{2k^2\xi^2}{k\left(\frac{n\Delta}{n-1}\right)^2}} \\
&= 2e^{-\Omega\left(\frac{k\xi^2}{\Delta^2}\right)}
\end{aligned}$$

In order to meet the required bounds we impose $\xi = \varepsilon\Delta$ and $k = \alpha\frac{\log n}{\varepsilon^2}$, where $\alpha \geq 1$. It follows that:

$$\begin{aligned}
2e^{-\Omega\left(\frac{k\xi^2}{\Delta^2}\right)} &= e^{-\Omega\left(\alpha\frac{\varepsilon^2\Delta^2\log n}{\varepsilon^2\Delta^2}\right)} \\
&= e^{-\Omega(\alpha\log n)} \\
&= e^{-\Omega(\log n^\alpha)} \\
&\leq 1/n^\alpha
\end{aligned}$$

This implies that the probability of the error to be greater than ξ at any vertex in the graph G is upper-bounded by $1/n^\alpha$. This gives a $1/n^{\alpha-1} \leq 1/n$ probability of having errors greater than $\varepsilon\Delta$ in the whole graph.

□

3.3 Exact top-k Closeness centralities fast computation

The last algorithm we are going to present in this chapter was introduced by K. Okamoto, W. Chen and X. Li for the exact computation of the top-*k* greatest Closeness centralities for undirected, connected and weighted

graphs. The main strategy is based on the combination of the approximation algorithm we described in the previous section with the exact algorithm. More precisely, this algorithm executes the RAND algorithm with $l = \Theta(k + n^{2/3}\log^{1/3} n)$ samples first in order to find a candidate set E of top-k' vertexes, where $k' > k$. To guarantee that all final top-k vertexes fall inside the E set the authors suggest to carefully choose k' using the bound given in the proof of Theorem 3.5. Once the E set has been found the exact algorithm is used to compute the average distances for each $v \in E$ and, finally, the actual top-k greatest Closeness centralities can be extracted.
Briefly, under certain conditions, the algorithm we illustrate in this section ranks all the top-k vertexes with greatest Closeness Centrality in $\mathcal{O}((k + n^{2/3}\log^{1/3} n)(n \log n + m))$ time with high probability.

3.3.1 The algorithm

Algorithm 7 (TOPRANK) takes as input an undirected, connected and weighted graph $G = (V, E)$, the number of top ranking vertexes it should extract (k) and the number of samples (l) to be used by the approximation algorithm RAND. First of all, TOPRANK uses the RAND algorithm with a set S, $|S| = l$ of random sampled vertexes to estimate the average distance $\hat{a}_v$ for each $v \in V$. Next, it names all the vertexes in the graph to $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_n$ such that $\hat{a}_{v_1} \leq \hat{a}_{v_2} \leq \cdots \leq \hat{a}_{v_n}$, where $\hat{a}_{v_i} = 1/\hat{c}_{v_i}$, and creates the E set ($|E| = k'$) of top-k' vertexes with greatest estimated Closeness Centrality. As final step, TOPRANK computes the exact average shortest-path distances of all vertexes in E and returns the top-k Closeness centralities.

Algorithm 7 TOPRANK(G = (V,E), *k*, *l*) K. Okamoto, W. Chen, X. Li exact top-*k* Closeness centralities algorithm.

```
1: Use algorithm RAND with a set $S$ of $l$ sampled vertexes to obtain the estimated average distance $\hat{a}_v \ \forall v \in V$. Rename all vertexes to $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_n$ such that $\hat{a}_{\hat{v}_1} \leq \hat{a}_{\hat{v}_2} \leq \cdots \leq \hat{a}_{\hat{v}_n}$
2: Find $\hat{v}_k$
3: $\hat{\Delta} \leftarrow 2\min_{u \in S}\max_{v \in V} d(u,v)$ // $d(u,v)$ has been computed for all $u \in S, v \in V$ at step 1 and $\hat{\Delta}$ is determined in $\mathcal{O}(ln)$ time
4: Compute candidate set E as the set of vertexes whose estimated average distances are less than or equal $\hat{a}_{\hat{v}_k} + 2f(l)\hat{\Delta}$
5: Calculate exact average shortest-path distances of all vertexes in $E$
6: Sort the exact average distances and return the top-$k$ highest closeness centralities
```

Note that the candidate set E is computed at line 4 of Algorithm 7 as "the set of vertexes whose estimated average distances are less than or equal to $\hat{a}_{\hat{v}_i} + 2f(l)\hat{\Delta}$". The $f(l)$ function is defined as follows:

$$f(l) = \alpha\sqrt{\frac{\log n}{l}} \tag{3.6}$$

where $\alpha > 1$. The authors made this choice in order to define a $1/2n^2$ upper bound to the probability of the estimation error at any vertex in the graph of being at least $f(l)\Delta$. This is based on setting $\varepsilon = f(l)$ in the proof of Theorem 3.5, the details are illustrated in the following theoretical analysis section (3.3.2).

3.3.2 Theoretical analysis

In this section we formally demonstrate that, under the assumptions we made on the sampling techniques and on the input graph, the TOPRANK algorithm computes exactly the top-k Closeness centralities with high probability in $\mathcal{O}((k + n^{2/3}\log^{1/3} n)(n\log n + m))$ time.
First and foremost we must prove that if we choose $f(l)$ as in Equation 3.6 than with low probability the approximation error at any vertex in the graph will be greater than $f(l)\Delta$:

Theorem 3.6 *If the $f(l)$ function is chosen as in Equation 3.6 then the error of the estimation of $c(v)$ for each $v \in V$ is greater than $f(l)\Delta$ with less than $1/2n^2$ probability.*

Proof The proof is based on setting $\varepsilon = f(l)$ in the Hoeffding's bound used in Theorem 3.5:

$$\begin{aligned}
\Pr\left\{\left|\frac{1}{\hat{c}(v)} - \frac{1}{c(v)}\right| \geq \xi\right\} &\leq 2e^{-\frac{2l^2\xi^2}{k\left(\frac{n\Delta}{n-1}\right)^2}} \\
&= \frac{2}{e^{2l\xi^2\left(\frac{n-1}{n\Delta}\right)^2}} \quad \text{(As before we set } \xi = \varepsilon\Delta) \\
&= \frac{2}{e^{\frac{\log n}{\log n}2l\varepsilon^2\left(\frac{n-1}{n}\right)^2}} \\
&= \frac{2}{n^{2l\frac{\varepsilon^2}{\log n}\left(\frac{n-1}{n}\right)^2}} \quad \left(\text{Now we set } \varepsilon = f(l) = \alpha\sqrt{\frac{\log n}{l}}\right) \\
&= \frac{1}{n^{2l\beta\frac{\log n}{l\log n}}} \; (\text{Where } \beta = \alpha^2) \\
&= \frac{1}{n^{2\beta}} \\
&\leq \frac{1}{n^2}
\end{aligned}$$

Note that in the fifth line of the proof we included both the numerator (2) and the multiplicative constant $\left(\frac{n-1}{n}\right)^2$ inside the constant $\beta \geq 1$.

$\square$

Now we have the enough elements to prove the correctness of the TOPRANK algorithm:

Theorem 3.7 *Given an undirected, connected and weighted graph $G = (V, E)$, if the distribution of the average distances is uniform with range $c\Delta$, where c is a constant and Δ is the diameter of G, the TOPRANK algorithm ranks the top-k vertexes with the greatest Closeness Centrality in $\mathcal{O}((k + n^{2/3}\log^{1/3} n)(n \log n + m))$ with high probability when we choose $l = \Theta(n^{2/3}\log^{1/3} n)$.*

We will proceed by demonstrating two main lemmas: Lemma 3.8 supports the correctness of the algorithm's output while Lemma 3.9 defines the time required by the algorithm. The strategy adopted by the authors is to prove the correctness of the TOPRANK regardless its time performances first. Then they proved that, if a particular condition is met, the same result can also be achieved in the required time limits. The results achieved by these two lemmas can be summarized in Theorem 3.7.

Lemma 3.8 *Algorithm TOPRANK ranks all the top-k vertexes with the highest Closeness Centrality correctly with high probability with any parameter l configuration.*

Proof Given that the TOPRANK algorithm computes the exact average distances for each $v \in E$ we must show that, with high probability, the set E (line 4 of Algorithm 7) contains all the top-k vertexes with the lowest exact average distance.
Let $T = \{v_1, v_2, \ldots, v_k\}$ be the set of the exact top-k Closeness centralities and $\hat{T} = \{\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_k\}$ be the set of the estimated top-k Closeness centralities returned by the RAND algorithm. Since the errors in the estimation of the average distances $\hat{a}_v$ are independent and in Theorem 3.6 we proved for any vertex v that the estimated average distance $\hat{a}_v$ is greater than $f(l)\Delta$ with probability less than $1/2n^2$ i.e.:

$$\Pr\left(\neg\left\{a_v - f(l)\Delta \leq \hat{a}_v \leq a_v + f(l)\Delta\right\}\right) \leq \frac{1}{2n^2}$$

it follows that, for each $v \in V$:

$$\begin{aligned} &\Pr\left(\neg\left\{\bigwedge_{v\in T} \hat{a}_v \leq a_v + f(l)\Delta \leq a_{v_k} + f(l)\Delta\right\}\right) \leq \\ &\sum_{i=1}^{k} \Pr\left(\neg\left\{a_{v_i} - f(l)\Delta \leq \hat{a}_{v_i} \leq a_{v_i} + f(l)\Delta\right\}\right) \leq \frac{k}{2n^2} \end{aligned} \tag{3.7}$$

This means that, with probability at least $1 - k/2n^2$, there exist at least k vertexes in T whose estimated average distance $\hat{a}_v$ is less than or equal to $a_{v_k} + f(l)\Delta$. Similarly:

$$\begin{aligned}
&\Pr\left(\neg\left\{\bigwedge_{\hat{v}\in\hat{T}} a_{\hat{v}} \le \hat{a}_{\hat{v}} + f(l)\Delta \le \hat{a}_{\hat{v}_k} + f(l)\Delta\right\}\right) \le \\
&\sum_{i=1}^{k} \Pr\left(\neg\left\{a_{\hat{v}_i} - f(l)\Delta \le \hat{a}_{\hat{v}_i} \le a_{\hat{v}_i} + f(l)\Delta\right\}\right) \le \frac{k}{2n^2}
\end{aligned} \tag{3.8}$$

which means that there exist at least k vertexes $\hat{v} \in \hat{T}$ whose real average distance $a_{\hat{v}}$ is less than or equal to $\hat{a}_{\hat{v}_k} + f(l)\Delta$ with probability greater than $1 - k/2n^2$. Thus:

$$\Pr\left(\neg\left\{a_{v_k} \le \hat{a}_{\hat{v}_k} + f(l)\Delta\right\}\right) \le \frac{k}{2n^2} \tag{3.9}$$

At this point from the 3.7 inequality we know that for each $v \in T$, $\hat{a}_v \le a_{v_k} + f(l)\Delta$. By combining this result with the 3.9 inequality it follows that:

$$\Pr\left(\neg\left\{\bigwedge_{\hat{v}\in\hat{T}} \hat{a}_v \le a_{v_k} + f(l)\Delta \le \hat{a}_{\hat{v}_k} + 2f(l)\Delta\right\}\right) \le \frac{k}{n^2} \tag{3.10}$$

Since at line 4 the TOPRANK algorithm includes in set E each vertex such that $\hat{a}_{\hat{v}} \le \hat{a}_{\hat{v}_k} + 2f(l)\hat{\Delta}$ as the final part of this proof we have to prove that $\Delta \le \hat{\Delta}$.

For any $w \in V$ we have that:

$$\begin{aligned}
\Delta = \max_{v,v'\in V} d(v,v') &\le \max_{v,v'\in V}\left(d(w,v) + d(w,v')\right) \\
&= \max_{v,v'\in V} d(w,v) + \max_{v,v'\in V} d(w,v') \\
&= 2\max_{v\in V} d(w,v)
\end{aligned}$$

and thus:

$$\Delta \le 2\min_{w\in S}\left(\max_{v\in V} d(w,v)\right) = \hat{\Delta}$$

Therefore for each $v \in T$, $\hat{a}_v \le \hat{a}_{\hat{v}_k} + 2f(l)\hat{\Delta}$ with probability at least $1 - k/n^2 \ge 1 - 1/n$ (since $k \le n$). Hence, the TOPRANK algorithm includes in E all the top-k vertexes with lowest average distance and it computes the exact top-k Closeness centralities with high probability.

□

Lemma 3.9 *If the distribution of the estimated average distances is uniform with range $c\Delta$, where c is a constant number and Δ is the diameter of the input graph G, then TOPRANK takes $\mathcal{O}((k + n^{2/3} \log^{1/3} n)(n \log n + m))$ time when we choose $l = \Theta(n^{2/3} \log^{1/3} n)$.*

Proof At line 1 the TOPRANK algorithm executes the RAND algorithm using l samples which takes $\mathcal{O}(l(n \log n + m))$ time as we saw in the previous section. Since the distribution of the estimated average distances is uniform with range $c\Delta$, there are $2nf(l)\hat{\Delta}/c\Delta$ vertexes between $\hat{a}_{\hat{v}_k}$ and $\hat{a}_{\hat{v}_k} + 2f(l)\hat{\Delta}$. Since $2nf(l)\hat{\Delta}/c\Delta \in \mathcal{O}(nf(l))$, the number of vertexes in E is $k + \mathcal{O}(nf(l))$ and therefore TOPRANK takes $\mathcal{O}((k + \mathcal{O}(nf(l)))(n \log n + m))$ time at line 5. In order to lower the total running time at lines 1 and 5 as much as possible, we should select l that minimizes $l + nf(l)$ that is:

$$\frac{\partial}{\partial l}(l + nf(l)) = 0$$
$$\frac{\partial}{\partial l}\left(l + n\alpha\sqrt{\frac{\log n}{l}}\right) = 0$$
$$1 - \frac{n\alpha}{2}\frac{\sqrt{\log n}}{l^{3/2}} = 0$$

this leads to:

$$l = \left(n \cdot \frac{\alpha}{2}\right)^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n = \Theta\left(n^{\frac{2}{3}} \cdot \log^{\frac{1}{3}} n\right)$$

In conclusion, if we choose $l = \Theta(n^{2/3} \log^{1/3} n)$ the TOPRANK algorithm takes $\mathcal{O}(n^{2/3} \log^{1/3} n(\log n + m))$ time at line 1 and it takes $\mathcal{O}((k + n^{2/3} \log^{1/3} n) \cdot (n \log n + m))$ time at line 5. Consequently, since all the other operations are asymptotically cheaper, TOPRANK takes $\mathcal{O}((k + n^{2/3} \log^{1/3} n)(n \log n + m))$ total running time.

□

3.4 Conclusions

In this chapter we illustrated three fast approaches for the approximation and the exact computation of the Closeness Centrality of a graph with high probability, we demonstrated their correctness and we calculated their asymptotic running time. In Chapter 4 we will provide a detailed description of how these algorithms can be applied to the Harmonic Centrality.

Chapter 4

Efficient Algorithms for the Harmonic Centrality

In the previous chapter we exposed in detail three efficient algorithms for both the computation and the approximation of the Closeness Centrality. In this chapter we introduce three new algorithms for the computation of the Harmonic Centrality inspired by the BFSCut function, RAND and TOPRANK.

4.1 Borassi et al. strategy applied to the Harmonic Centrality

In this section we describe how the BFSCut function could be applied for the computation of the exact top-k Harmonic centralities. The main challenge is represented by finding and proving an upper bound for $h(v)$.

4.1.1 An upper bound for h(v)

As Borassi et al. did in their work we would like to define an upper bound $\tilde{h}(v)$ to the Harmonic Centrality of node v in order to stop the BFS from that node if $\tilde{h}(v)$ is less than the kth greatest Harmonic Centrality computed until now. Of course, $\hat{h}(v)$ has to be updated whenever all the vertexes of the dth level of the BFS tree have been visited, $d \geq 1$.

Lemma 4.1

$$h(v) \leq \tilde{h}_d(v, r(v)) := h_d(v) + \frac{\tilde{\gamma}_{d+1}}{(d+1)(d+2)} + \frac{r(v) - n_d(v)}{d+2} \tag{4.1}$$

where $h_d(v)$ is the Harmonic Centrality of node v up to distance d.

Proof Of course:

$$h(v) \leq h_d(v) + \frac{\gamma_{d+1}(v)}{d+1} + \frac{r(v) - n_{d+1}(v)}{d+2}$$

Since $n_{d+1}(v) = \gamma_{d+1}(v) + n_d(v)$,

$$h(v) \leq h_d(v) + \frac{\gamma_{d+1}(v)}{d+1} + \frac{r(v) - \gamma_{d+1}(v) - n_d(v)}{d+2}$$

Finally, since $\tilde{\gamma}_{d+1} = \sum_{u \in \Gamma_d(v)} \text{outdeg}(v) \geq \gamma_{d+1}(v)$

$$h(v) \leq h_d(v) + \frac{\tilde{\gamma}_{d+1}(v)}{(d+1)(d+2)} + \frac{r(v) - n_d(v)}{d+2}$$

□

We can exploit this property to efficiently compute the top-k Harmonic centralities using Algorithm 1 and a slightly revised version of the BFSCut function reported in Algorithm 2 for strongly connected graphs which were described in Section 3.1.3.

Finally, for directed and not (strongly) connected graphs we know that the Harmonic Centrality $h(v)$ depends only on the reachable vertexes from v since the others give no contribution. In this case $r(v)$ is hard to compute but, with the purpose to find an upper bound to $h(v)$, we can try to find an upper bound to $r(v)$ since $h(v)$ is directly proportional to $r(v)$.
Borassi et. al. already provided an upper bound $\omega(v)$ to $r(v)$ so we could re-use part of Algorithm 5 of Section 3.1.3 to compute $\omega(v)$ for each vertex. The resulting algorithm is reported in Algorithm 10.

Algorithm 8 Revised BFSCut(v, h_k) function for the computation of $h(v)$ in the case of strongly connected graphs

```
1: Create queue Q
2: Q.enqueue($v$)
3: Mark $v$ as visited
4: $d \leftarrow 0$; $h \leftarrow 0$; $\tilde{\gamma} \leftarrow 0$; $nd \leftarrow 0$;
5: while !$Q$.isEmpty do
6:     $u \leftarrow Q$.dequeue()
7:     if $d(v,u) > d$ then //all nodes at level $d-1$ have been visited, $\tilde{h}$ must be updated
8:         $\tilde{h} \leftarrow h + \frac{\tilde{\gamma}}{(d+1)(d+2)} + \frac{n-nd}{d+2}$
9:         if $\frac{\tilde{h}}{n-1} \leq x_k$ then
10:            return 0
11:        end if
12:        $d \leftarrow d+1$
13:    end if
14:    if $u \neq v$ then
15:        $h \leftarrow h + \frac{1}{d(u,v)}$
16:    end if
17:    $\tilde{\gamma} \leftarrow \tilde{\gamma}+$ outdeg($u$)
18:    $nd \leftarrow nd + 1$
19:    for $w$ in adjacency list of $u$ do
20:        if $w$.visited == $false$ then
21:            $Q$.enqueue($w$)
22:            $w$.visited $\leftarrow true$
23:        end if
24:    end for
25: end while
26: return $\frac{h}{n-1}$
```

Algorithm 9 Revised BFSCut(v, x_k) function in the case of undirected graphs (not necessarily connected)

```
1: Create queue Q
2: Q.enqueue($v$)
3: Mark $v$ as visited
4: $d \leftarrow 0$; $h \leftarrow 0$; $\tilde{\gamma} \leftarrow 0$; $nd \leftarrow 0$;
5: while !$Q$.isEmpty do
6:     $u \leftarrow Q$.dequeue()
7:     if $d(v,u) > d$ then
8:         $r \leftarrow$ BFSCount$(u)$ //Returns the number of reachable nodes from $u$
9:         $\tilde{h} \leftarrow h + \frac{\tilde{\gamma}}{(d+1)(d+2)} + \frac{r-nd}{d+2}$
10:        if $\frac{\tilde{h}}{n-1} \leq x_k$ then
11:            return 0
12:        end if
13:        $d \leftarrow d + 1$
14:    end if
15:    if $u \neq v$ then
16:        $h \leftarrow h + \frac{1}{d(u,v)}$
17:    end if
18:    $\tilde{\gamma} \leftarrow \tilde{\gamma}+$ outdeg$(u)$
19:    $nd \leftarrow nd + 1$
20:    for $w$ in adjacency list of $u$ do
21:        if $w$.visited == $false$ then
22:            $Q$.enqueue($w$)
23:            $w$.visited $\leftarrow$ $true$
24:        end if
25:    end for
26: end while
27: return $\frac{h}{n-1}$
```

Algorithm 10 BFSCut(v, x_k) function in the case of directed and not strongly connected graphs

```
1: Create queue Q
2: Q.enqueue($v$)
3: Mark $v$ as visited
4: $d \leftarrow 0$; $h \leftarrow 0$; $\tilde{\gamma} \leftarrow 0$; $nd \leftarrow 0$;
5: while !$Q$.isEmpty do
6:     $u \leftarrow Q$.dequeue()
7:     if $d(v,u) > d$ then
8:         $\omega \leftarrow$ GetBound($u$) $//\omega$ has been computed for all SSCs in the Preprocessing
9:         $\tilde{h} \leftarrow h + \frac{\tilde{\gamma}}{(d+1)(d+2)} + \frac{\omega - nd}{d+2}$
10:        if $h \leq x_k$ then
11:            return 0
12:        end if
13:    end if
14:    if $u \neq v$ then
15:        $h \leftarrow h + \frac{1}{d(u,v)}$
16:    end if
17:    $\tilde{\gamma} \leftarrow \tilde{\gamma} +$ outdeg($u$)
18:    $nd \leftarrow nd + 1$
19:    for w in adjacency list of u do
20:        if $w$.visited == $false$ then
21:            $Q$.enqueue($w$)
22:            $w$.visited $\leftarrow true$
23:        end if
24:    end for
25: end while
26: return $\frac{h}{n-1}$
```

4.2 Fast Harmonic Centrality Approximation

The D. Eppstein and J. Wang strategy we presented in Section 3.2 can be adapted to the Harmonic Centrality with little effort. The steps of the algorithm are almost the same as RAND but it is necessary to provide the Harmonic Centrality's correct estimation of vertex v i.e. $\hat{h}(v)$. In conclusion we obtained an algorithm that computes with high probability a ε-approximation of the Harmonic Centrality ($\varepsilon > 0$) of an undirected, connected and weighted graph that takes $\mathcal{O}(\log n/\varepsilon^2(n \log n + m)$ time.

4.2.1 The algorithm

Similarly to the RAND algorithm, RAND_H takes as inputs a graph G and the number of samples k. Then it extracts k random samples from V and solves the SSSP problem for each extracted sample as source. Finally it computes an Harmonic Centrality estimator $\hat{h}(v)$ for each $v \in V$.

Algorithm 11 RAND_H(G = (V,E), k) Our redesigned version of the RAND algorithm for the computation of the Harmonic Centrality.

```
1: for $i = 1$ to $k$ do
2:     $v_i \leftarrow$ pick a vertex uniformly at random from $V$
3:     Solve SSSP problem with $v_i$ as source
4: end for
5: for each $v$ in $V$ do
6:     $h(v) \leftarrow \hat{h}(v)$
7: end for
```

As in the previous chapter we want the number of random samples to be $k = \Theta(\log n/\varepsilon^2)$. Thus, we define the Harmonic Centrality estimator as follows:

$$\tilde{h}(v) = \frac{n}{k(n-1)} \sum_{i=1}^{k} \frac{1}{d(v_i, v)} \tag{4.2}$$

Similarly to Equation 3.4 we are expressing the Harmonic Centrality estimator as the normalized average of the inverse distances to the sampled vertices, with n/k as normalization term.

4.2.2 Theoretical analysis

Our purpose is now to demonstrate that the RAND_H algorithm we sketched in the previous section provides a ε-approximation of the Harmonic Centrality of each $v \in V$ to within $\mathcal{O}(\log n/\varepsilon^2(n \log n + m)$ time with high probability. Since we will make use of the Hoeffding's bound (see Lemma 3.3) with the approximation error $|\hat{h}(v) - h(v)|$ as random variable, we start by demonstrating that the expected value of $\hat{h}(v)$ is equal to $h(v)$ as requested by Lemma 3.3.

Theorem 4.2 *Given that:*

$$h(v) = \frac{1}{|V|-1} \sum_{w \in V, w \neq v} \frac{1}{d(v,w)} \quad \text{and} \quad \tilde{h}(v) = \frac{n}{k(n-1)} \sum_{i=1}^{k} \frac{1}{d(v_i,v)}$$

then, $\mathrm{E}\left[\hat{h}(v)\right] = h(v)$.

Proof

$$\begin{aligned}\mathrm{E}\left[\hat{h}(v)\right] &= \mathrm{E}\left[\frac{n}{k(n-1)} \sum_{i=1}^{k} \frac{1}{d(v_i,v)}\right] \\ &= \frac{n}{k(n-1)} \mathrm{E}\left[\sum_{i=1}^{k} \frac{1}{d(v_i,v)}\right]\end{aligned}$$

Again we can interpret $1/d(v_i,v)$ as a random variable and so $\mathrm{E}\left[\sum_{i=1}^{k} \frac{1}{d(v_i,v)}\right] = \sum_{i=1}^{k} \mathrm{E}\left[\frac{1}{d(v_i,v)}\right]$ and it follows that:

$$= \frac{n}{k(n-1)} \sum_{i=1}^{k} \mathrm{E}\left[\frac{1}{d(v_i,v)}\right]$$

Since $\mathrm{E}\left[\frac{1}{d(v_i,v)}\right] = \frac{1}{n} \sum_{j=1}^{n} \frac{1}{d(v_j,v)}$ (we impose $\frac{1}{d(v_j,v)} = 0$ if $v_j = v$) we have that:

$$\begin{aligned}&= \frac{n}{k(n-1)} \sum_{i=1}^{k} \frac{1}{n} \sum_{j=1}^{n} \frac{1}{d(v_j,v)} \\ &= \frac{1}{k(n-1)} \sum_{j=1}^{n} \sum_{i=1}^{k} \frac{1}{d(v_j,v)} \\ &= \frac{1}{k(n-1)} \sum_{j=1}^{n} \frac{k}{d(v_j,v)} \\ &= \frac{1}{n-1} \sum_{j=1}^{n} \frac{1}{d(v_j,v)} \\ &= h(v)\end{aligned}$$

□

Theorem 4.2 allows us to use the Hoeffding's bound to prove the high probability bound for the RAND_H algorithm.

Theorem 4.3 *Given an undirected, connected and weighted graph $G = (V, E)$, algorithm RAND_H computes the estimator $\hat{h}(v)$ of the Harmonic Centrality $h(v)$ to within a $\varepsilon > 0$ error for all vertexes $v \in V$ using $\Theta(\log n/\varepsilon^2)$ samples with high probability.*

Proof We apply the Hoeffding's bound with the following assumptions:

$$x_i = \frac{n}{n-1}\frac{1}{d(v_i, v)}, \quad \mu = h(v), \quad a_i = 0 \quad \text{and} b_i = \frac{n}{n-1}.$$

It follows that:

$$\begin{aligned}
\Pr\left\{\left|\hat{h}(v) - h(v)\right| \geq \varepsilon\right\} &= \Pr\left\{\left|\sum_{i=1}^{k} \frac{n}{k(n-1)}\frac{1}{d(v_i, v)} - h(v)\right| \geq \varepsilon\right\} \\
&= \Pr\left\{\left|\sum_{i=1}^{k} \frac{x_i}{k} - \mu\right| \geq \varepsilon\right\} \\
&\leq 2e^{-\frac{2k^2\varepsilon^2}{\sum_{i=1}^{k}(b_i - a_i)^2}} \\
&\leq 2e^{-\frac{2k^2\varepsilon^2}{k(\frac{n}{n-1})^2}} \\
&= 2e^{-\Omega(k\varepsilon^2)}
\end{aligned}$$

Using $k = \alpha\frac{\log n}{\varepsilon^2}$ samples with $\alpha \geq 1$ leads to:

$$\begin{aligned}
2e^{-\Omega(k\varepsilon^2)} &= e^{-\Omega\left(\alpha\frac{\log n}{\varepsilon^2}\varepsilon^2\right)} \\
&= \frac{1}{e^{\Omega(\log n^{\alpha})}} \\
&\leq \frac{1}{n^{\alpha}}
\end{aligned}$$

This means that at any vertex $v \in V$ the estimation error $|\hat{h}(v) - h(v)|$ is greater than ε with probability less than $1/n^{\alpha}$ and $1/n^{1-\alpha}$ in the whole graph. In other words all the vertexes in the graph are affected by an error which is less than ε with probability at least $1 - 1/n$.

□

Finally, since the time-expensive operations executed by algorithm RAND_H are equivalent to the operations of algorithm RAND (solving l times the SSSP problem), we can conclude that RAND_H achieves a total running time of $\mathcal{O}(\log n/\varepsilon^2(n\log n+m))$ and returns the correct output with high probability.

4.3 Fast top-k Harmonic centralities exact computation

The last algorithm we worked on is the by K. Okamoto, W. Chen and X. Li exact approach we exposed in Section 3.3. Our purpose was to modify the TOPRANK algorithm in order to efficiently compute the exact top-k Harmonic centralities with high probability. As the authors did, we exploited the approximation algorithm RAND_H we introduced in the previous section in order to create a candidate set H, $|H| = k' > k$. Then, we adopted the exact approach to compute the exact Harmonic centralities for all $v \in H$. In the end we obtained a $\mathcal{O}((k+n^{2/3}\log^{1/3} n)(n\log n+m))$ algorithm that calculates the exact top-K Harmonic centralities of an undirected, connected and weighted graph with high probability.

4.3.1 The algorithm

Similarly to TOPRANK, the TOPRANK_H algorithm takes as input an undirected, connected and weighted graph $G = (V, E)$, the number of top ranking vertexes k and how many samples the RAND_H algorithm should use to calculate the Harmonic Centrality estimators $\tilde{h}(v)$. Then, as we can see from Algorithm 12 pseudo-code, all vertexes in V are named according to their approximated Harmonic Centrality value i.e. $\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_2$ such that: $\hat{h}_{\hat{v}_1} \le \hat{h}_{\hat{v}_2} \le \cdots \le \hat{h}_{\hat{v}_n}$. Next, the candidate set H is created as the set of vertexes whose estimated Harmonic centrality is greater than or equal to $\hat{h}_{\hat{v}_k} - 2f(l)$. More formally:

$$H = \left\{v_i \in V : \hat{h}_{\hat{v}_i} \ge \hat{h}_{\hat{v}_k} - 2f(l)\right\} \tag{4.3}$$

where $f(l)$ is defined as in Equation 3.6

We now need to demonstrate the correctness of the TOPRANK_H algorithm and its time requirements.

Algorithm 12 TOPRANK_H($G = (V, E)$, k, l) Our redesigned version of the TOPRANK algorithm for the computation of the top-k exact Harmonic centralities.

```
1: Use algorithm RAND_H with a set $S$ of $l$ sampled vertices to obtain the estimated harmonic centrality $\hat{h}_v$ $\forall v \in V$. Rename all vertices to $\hat{v}_1, \hat{v}_2, \dots, \hat{v}_n$ such that $\hat{h}_{\hat{v}_1} \leq \hat{h}_{\hat{v}_2} \leq \dots \leq \hat{h}_{\hat{v}_n}$
2: Find $\hat{h}_k$
3: Compute candidate set $H$ as the set of vertices whose estimated harmonic centralities are greater than or equal to $\hat{h}_{\hat{v}_k} - 2f(l)$
4: Calculate exact harmonic centralities for all vertices in $H$
5: Sort the exact harmonic centralities and return the top-$k$
```

4.3.2 Theoretical analysis

As in the previous chapter we start by proving that the RAND_H algorithm will give us an ε-approximation of the Harmonic Centrality of each vertex $v \in V$ with high probability. Then we will use this result to demonstrate the time performances of TOPRANK_H and the correctness of its output.

Theorem 4.4 *If the $f(l)$ function is chosen as in Equation 3.6 then, if $\varepsilon > 0$:*

$$\forall v \in V, \quad |\hat{h}(v) - h(v)| < \varepsilon$$

with high probability.

Proof The strategy is to use the Hoeffding's bound setting $\varepsilon = f(l)$ i.e.:

$$\begin{aligned}
\Pr\left\{\left|\hat{h}(v) - h(v)\right| \geq \varepsilon\right\} &\leq 2e^{-2l^2\varepsilon^2/l(\frac{n}{n-1})^2} \\
&= \frac{2}{e^{2l\varepsilon^2\left(\frac{n-1}{n}\right)^2}} \\
&= \frac{2}{e^{2\frac{\log n}{\log n}l\varepsilon^2\left(\frac{n-1}{n}\right)^2}} \\
&= \frac{2}{n^{2\frac{l\varepsilon^2}{\log n}\left(\frac{n-1}{n}\right)^2}} \qquad \left(\text{We now set } \varepsilon = \alpha'\sqrt{\frac{\log n}{l}}\right) \\
&= \frac{1}{n^{2\beta\frac{l\log n}{l\log n}}} \qquad (\text{Where } \beta = \alpha'^2 \geq 1) \\
&= \frac{1}{n^{2\beta}} \\
&\leq \frac{1}{n^2}
\end{aligned}$$

Note that in the fifth line we included both the numerator 2 and the factor $(\frac{n-1}{n})^2$ inside the constant $\beta \geq 1$.

□

At this point we are ready to demonstrate the correctness of the TOPRANK_H algorithm using the result we achieved with Theorem 4.4. Basically we have to prove the following theorem:

Theorem 4.5 *Given an undirected, connected and weighted graph $G = (V, E)$, if the distribution the of the estimated Harmonic centralities among the vertexes in V is uniform in range cU, where $c > 0$ and U = $[0, 1]$, with high probability the* TOPRANK_H *algorithm ranks the top-k vertexes with the greatest Harmonic Centrality in $\mathcal{O}((k + n^{2/3} \log^{1/3} n)(n \log n + m))$ if we choose $l = \Theta(n^{2/3} \log^{1/3} n)$ samples.*

We can prove this theorem by separating it into two lemmas: Lemma 4.6 and 4.7.

Lemma 4.6 *The* TOPRANK_H *algorithm ranks all the top-k vertexes with the greatest Harmonic Centrality correctly with high probability and with any parameter l configuration.*

Proof We need to prove that with high probability the candidate set H created at the 3rd line of the TOPRANK_H algorithm contains all the top-k vertexes with the greatest Harmonic Centrality.
Let $T = \{v_1, v_2, \ldots, v_k\}$ be the set of the exact top-k Harmonic centralities and $\hat{T} = \{\hat{v}_1, \hat{v}_2, \ldots, \hat{v}_k\}$ be the set of the top-k estimated Harmonic centralities returned by the RAND_H algorithm. Since in Lemma 4.4 we demonstrated that $|\hat{h}(v) - h(v)| \geq \varepsilon$ for each $v \in V$ with probability less than $1/2n^2$:

$$\Pr\left(\neg\left\{h_v - f(l) \leq \hat{h}_v \leq h_v + f(l)\right\}\right) \leq \frac{1}{2n^2}$$

it follows that, for each $v \in V$:

$$\Pr\left(\neg\left\{\bigwedge_{v \in T} \hat{h}_v \geq h_v - f(l) \geq h_{v_k} - f(l)\right\}\right) \leq$$
$$\sum_{i=1}^{k} \Pr\left(\neg\left\{h_{v_i} - f(l) \leq \hat{h}_{v_i} \leq h_{v_i} + f(l)\right\}\right) \leq \frac{k}{2n^2}$$

This inequality means that, with probability at least $1 - k/2n^2$, there are at least k vertexes in H whose estimated Harmonic Centrality is greater than $h_{v_k} - f(l)$. Furthermore, we have that:

$$\Pr\left(\neg\left\{\bigwedge_{\hat{v}\in\hat{T}} h_{\hat{v}} \geq \hat{h}_{\hat{v}} - f(l) \geq \hat{h}_{\hat{v}_k} - f(l)\right\}\right) \leq$$
$$\sum_{i=1}^{k} \Pr\left(\neg\left\{h_{\hat{v}_i} - f(l) \leq \hat{h}_{\hat{v}_i} \leq h_{\hat{v}_i} + f(l)\right\}\right) \leq \frac{k}{2n^2}$$

which shows that, with probability at least $1 - k/2n^2$, there are at least k vertexes in $\hat{H}$ whose exact Harmonic Centrality is greater than $\hat{h}_{\hat{v}_k} - f(l)$. Moreover, this implies that $h_{v_k} \geq \hat{h}_{\hat{v}_k} - f(l)$ with high probability. If we combine this result with the first inequality we obtain that:

$$\Pr\left(\neg\left\{\bigwedge_{v\in T} h_{\hat{v}} \geq h_{v_k} - f(l) \geq \hat{h}_{\hat{v}_k} - 2f(l)\right\}\right) \leq \frac{k}{n^2}$$

Therefore, for each $v \in H$, $\hat{h}_v \leq \hat{a}_{\hat{v}_k} - 2f(l)$ with probability at least $1 - 1/n$ (since $k \leq n$). In conclusion, we proved that the TOPRANK_H algorithm includes all the top-k vertexes with greatest Harmonic Centrality in the candidate set H with high probability.

$\square$

Now we can evaluate the complexity of the TOPRANK_H algorithm which will have a smaller additive constant than TOPRANK since it does not need to approximate the graph diameter $\hat{\Delta}$.

Lemma 4.7 *If the distribution of the estimated Harmonic centralities among the vertexes in V is uniform in range* $c\mathrm{U}$, *where* $c > 0$ *and* $\mathrm{U} = [0, 1]$, *with high probability* TOPRANK_H *ranks the top-k vertexes with the greatest Harmonic Centrality in* $\mathcal{O}((k + n^{2/3}\log^{1/3} n)(n\log n + m))$ *if we choose* $l = \Theta(n^{2/3}\log^{1/3} n)$ *samples.*

Proof We know from Chapter 3 that solving the SSSP problem takes $\mathcal{O}(n\log n + m)$ time. Therefore TOPRANK_H takes $\mathcal{O}(l(n\log n + m))$ time at its first step.
Since the distribution of the estimated Harmonic centralities is uniform in range cU then there are $2nf(l)/c$ vertexes between $\hat{h}_{\hat{v}_k} - 2f(l)$ and, obviously, $2nf(l)/c \in \mathcal{O}(nf(l))$. Thus, the number of vertexes in H is $k + \mathcal{O}(nf(l))$ and this implies that TOPRANK_H takes $\mathcal{O}((k + \mathcal{O}(nf(l)))(n\log n + m))$ time at line 4.
Finally, as the authors did in Lemma 3.9, we choose l in order to minimize the total running time at line 4 that is $l = \Theta(n^{2/3}\log^{1/3} n)$. Therefore, under the assumptions we made, the TOPRANK_H algorithm takes $\mathcal{O}((k + n^{2/3}\log^{1/3} n)(n\log n + m))$ time. $\square$

4.4 Conclusions

In this chapter we presented a redesigned version of the algorithms we exposed in Chapter 3. Our aim was to obtain new strategies to approximate and calculate efficiently the Harmonic Centrality of the vertexes of a graph. Up there, we achieved our objective from a theoretical point of view. In the next chapter we will report and comment the experimental results achieved by the implementation of our algorithms. We will examine their performances in terms of time and precision and provide a comparison between them and both the naive algorithm and Borassis et al. .

Chapter 5

Experimental Results

This chapter is dedicated to the experiments we performed on several benchmark networks. We measured the performances in terms of time and precision of our Python implementation of the algorithms we exposed in the previous chapter.

Our purpose was firstly to verify the correctness of our theoretical results into a practical scenario. Therefore we compared the running time of our randomized algorithms with the time requested by both solving APSP and the Borassi's et al. algorithm. Then we analyzed the errors made by the approximated algorithm RAND_H in terms of average absolute error, average relative error and error variance. For what concerns the TOPRANK_H algorithm we checked whether the top-k greatest Harmonic centralities it found were correct or not.

Since the RAND_H algorithm achieved excellent results in terms of precision (the error was far below the high probability bound ε) we performed several additional experiments in order to see whether it was possible to boost the algorithm's time performances by halving the number of random samples without violating the error bound ε. We noticed that, with such configuration, the RAND_H precision was not compromised. Thus we proceeded by applying the same modification also in the TOPRANK_H algorithm and, finally, halving again the number of random samples of the RAND_H algorithm.
We observed that, despite the lower number of samples, the precision of the TOPRANK_H algorithm was slightly affected but it could be adjusted with a little more time cost. On the other hand a considerable amount of running time was saved. Furthermore the running time of the RAND_H algorithm was reduced by almost a quarter of the time required in the first experiments (some post-processing operations are always needed) while the precision was reduced by about half of its original values (in other words the error approximately doubled).

5.1 Introduction

Before proceeding with the analysis of the experimental results let us introduce the metrics we will use to measure the time and precision performances of our algorithms. In this section we also define the constants we modified in order to modify the sampling techniques.

5.1.1 Performance metrics

Since we are going to compare the time performance between two algorithms, we introduce a time gain metric:

$$\text{gain} = \frac{t_n - t_a}{t_n} \tag{5.1}$$

where t_a denotes the time needed by the tested algorithm and t_n represents the time required by the algorithm we are comparing it with. In the following sections we will often express this metric as a percentage (i.e. gain·100).

For what concerns the precision we introduce several metrics. We denote with *error* the overall absolute error made by the RAND_H algorithm:

$$error = \sum_{v \in V} \left| h(v) - \hat{h}(v) \right| \tag{5.2}$$

where, as in Chapter 4, $\hat{h}(v)$ represents the approximated value of the Harmonic Centrality of vertex v. In order to compare the errors between two networks we use the average error:

$$avg\_error = \frac{1}{n} \sum_{v \in V} \left| h(v) - \hat{h}(v) \right| = \frac{error}{n} \tag{5.3}$$

Furthermore we would like to measure the gap between the actual error and its corresponding upper bound ε. For this reason we introduce the *d_bound* metric:

$$d\_bound(v) = \frac{\left| \hat{h}(v) - h(v) \right|}{\varepsilon} \tag{5.4}$$

Since we are going to compare the precision on different networks, it is more convenient for us to consider the average value of d_bound that is:

$$d\_bound = \frac{1}{n\varepsilon} \sum_{v \in V} \left| h(v) - \hat{h}(v) \right| = \frac{error}{n\varepsilon} \tag{5.5}$$

such that $d\_bound \in [0,1]$. $d\_bound = 1$ means that the RAND_H did not do any better than the upper-bound, in other words for each vertex in the graph the Harmonic Centrality estimation is affected by an error of ε. Conversely, $d\_bound = 0$ represents that RAND_H computed all the exact the Harmonic centrality.
Another aspect we took into account in our experimental analysis is the variance of the errors. By interpreting the error at each node as a random variables we have that:

$$\begin{aligned} \mathrm{var}(error) &= \mathrm{E}\left[(error - avg\_error)^2\right] \\ &= \sum_{v \in V} \frac{\left(\left|\hat{h}(v) - h(v)\right| - avg\_error\right)^2}{n-1} \end{aligned} \tag{5.6}$$

We also include the maximum error since it could show us whether there exist some isolated but considerably high errors. These kind of errors cannot be noticed if we average them with thousands of other much smaller errors:

$$\text{max error} = \max_{v \in V} \left|\hat{h}(v) - h(v)\right| \tag{5.7}$$

The last metric we used is the relative error since, in some cases, the absolute error can be misleading. For example, if there was a node $v \in V$ such that $h(v) = 10^{-4}$ and RAND_H calculated $\hat{h}(v) = 2 \cdot 10^{-4}$ the absolute error would be 10^{-4} which, as we will see in the following sections, is considerably small, but the relative error would be 2. As before we compute the average relative error of a graph in order compare different networks:

$$\delta = \sum_{v \in V} \frac{\left|\hat{h}(v) - h(v)\right|}{h(v)} \tag{5.8}$$

We will also focus on the error among the top-k Harmonic centralities, which are often much more interesting than the remaining nodes. For this purpose we examined the *avg_error* and the relative error metrics with different values of k.

5.1.2 Constants

It is important to point out that our implementation needs to deal with real numbers and not with asymptotic values. The RAND_H algorithm chooses $k = \Theta(\log n/\varepsilon^2)$ random samples, so we can denote k as:

$$k = \left\lceil C \cdot \frac{\log n}{\varepsilon^2} \right\rceil \tag{5.9}$$

where $C > 0$.

On the other hand, the TOPRANK_H algorithm strongly depends on the number of samples (i.e. l) used for the execution of the RAND_H algorithm and on the constant (i.e. α) it uses to determine the candidate set H. More precisely, l is defined as:

$$l = \Theta\left(n^{\frac{2}{3}} \log^{\frac{1}{3}} n\right) \tag{5.10}$$

Unfortunately, the authors did not specify the exact value for l so hereafter we will refer to l as:

$$l = \left\lceil \beta \cdot n^{\frac{2}{3}} \log^{\frac{1}{3}} n \right\rceil \tag{5.11}$$

where β is a positive constant. The α constant is instead crucial for the computation of the $f(l)$ function we defined in Equation 3.6. $f(l)$ determines $\hat{k}$ that is the number of additional vertexes added to the candidate set H. In this case the authors specified that α must be > 1 but did not provided its exact value. So, from this point forward, we will refer to $f(l)$ as follows:

$$f(l) = \left\lceil \alpha \cdot \sqrt{\frac{\log n}{\varepsilon^2}} \right\rceil \tag{5.12}$$

where $\alpha > 1$.

In our sets of experiments we used C and β to control the number of random samples extracted by RAND_H and α to compensate the potential lack of precision in TOPRANK_H.

5.2 Experimental setup

Our experiments are based on a 18 benchmark graphs dataset we reported in Table 5.1. We downloaded these networks from SNAP [14], Network Repository [26] and Konect [13]. 9 of them (see Table 5.1) represent authorship networks which are unweighted bibartite graphs consisting of links between authors and their works. The remaining 9 graphs represent part of several well-known social networks.

Each graph is given through a single text file that defines an edge list. A single line of such file counts two to three spaced numbers where the first

Authorship networks

Network name	Nodes	Edges
Wikinews (en)	159,990	901,416
Wiktionary (de)	145,301	1,229,501
DBpedia producers	138,841	207,268
Github	120,867	440,237
Wikiquote (en)	93,445	549,210
arXiv cond-mat	89,356	144,340
Wikibooks (fr)	27,754	201,727
Wikinews (fr)	25,042	193,618
Writers	22,015	58,595

Social networks

Network name	Nodes	Edges
Gowalla	196,591	950,327
Epinions trust	131,828	841,372
Epinions	63,947	606,512
Brightkite	56,739	212,945
Gplus	23,628	39,242
Anybeat	12,645	67,053
Wiki-elec	7,118	107,071
Advogato	6,539	51127
Facebook	4,039	88,234

Table 5.1: Benchmark networks dataset

two numbers are the couple of nodes representing the edge. The third number is optional and indicates the weight of the edge. Figure 5.1 provides an explanatory example. Note that in some cases the order of the nodes is used to specify the direction of the edge but, since we are working with undirected graphs, we did not take into account this information.

```
1  2  1.5
1  3  2.3
```

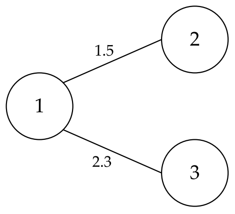


Figure 5.1: On the left: a couple of line of edge list file. On the right: the corresponding graph.

We implemented our algorithms using Python together with the graph-tool library that implements several useful graph I/O functions and algorithms. In Appendix A we included the most relevant parts of our code. Note that when we compute the time gain of our algorithm on solving the APSP problem we measure t_n as the time required by the instruction:

```
closeness(G, harmonic=True, norm=True)
```

that is the graph-tool algorithm to compute all the normalized Harmonic centralities of the G graph by iterating SSSP using each node of G as source.

We also point out that each experimental result we reported hereafter is an average on four run of the algorithms.

5.3 RAND_H: first set of experiments

In this section we report and analyze the performances achieved by the modified version of the RAND_H algorithm in terms of time and precision. This was the first set of experiment we made and it was thought to confirm the theoretical results into a practical environment. For this purpose we imposed $C = 1$.

5.3.1 Time performances

In Table 5.2 we report the time performances obtained by the RAND_H algorithm. We performed this experiment twice in order to compare the results between two different values of the upper bound: $\varepsilon_1 = 0.05$ and $\varepsilon_2 = 0.3$. We did not choose values for ε below 0.05 because thy would have required to extract a number of samples equal to the number of vertexes in the network. Nevertheless lower ε values can be used for bigger networks than the ones we examined.

It is easy to see from the *Samples* columns that the number of random samples increases as we consider networks with a higher number of nodes and that it is proportionate to $1/\varepsilon^2$. This just show us that our implementation of the random sampling technique is correct.
Moreover, if we look at the gain, we see that it increases as we augment the size of the network. This means that, for a fixed value of ε, RAND_H is asymptotically faster than solving APSP. From the social network table it is also evident that, for the smallest sized networks and for $\varepsilon = 0.05$ or lower, RAND_H obtains very little time gain. A clearer view of the gain trend is provided by Figure 5.2, where, instead of $\varepsilon_2 = 0.3$, we chosen $\varepsilon_2 = 0.1$ for a better comparison between the two curves.

Authorship networks

Network name	$\varepsilon = 0.3$		$\varepsilon = 0.05$	
	Samples	Gain	Samples	Gain
Wikinews (en)	134	99.90%	4794	96.90%
Wiktionary (de)	133	99.89%	4756	96.95%
DBpedia producers	133	99.86%	4737	96.14%
Github	131	99.87%	4682	96.12%
Wikiquote (en)	128	99.80%	4579	95.18%
arXiv cond-mat	128	99.77%	4561	93.54%
Wikibooks (fr)	115	99.39%	4093	84.03%
Wikinews (fr)	114	99.36%	4052	81.85%
Writers	112	99.05%	4001	73.34%

Social networks

Network name	$\varepsilon = 0.3$		$\varepsilon = 0.05$	
	Samples	Gain	Samples	Gain
Gowalla	136	99.89%	4877	97.08%
Epinions trust	132	99.87%	4717	96.45%
Epinions	124	99.76%	4427	93.73%
Brightkite	123	99.65%	4379	91.30%
Gplus	113	98.91%	4029	75.19%
Anybeat	106	98.16%	3779	55.18%
Wiki-elec	100	97.55%	3549	40.34%
Advogato	99	95.42%	3515	7.63%
Facebook	93	95.18%	3323	5,53%

Table 5.2: Time performance improvement of the RAND_H algorithm on our dataset. Samples: number of samples extracted by the algorithm. Gain: time gain percentage as defined in Equation 5.1. ε: error upper bound that was used for the experiment.

These observations were quite predictable from the theory underneath this algorithm. Recall that while APSP complexity in the worst case is $\mathcal{O}(n^2 \log n + nm)$ for weighted graphs and $\mathcal{O}(nm)$ for unweighted graphs, RAND_H requires $\mathcal{O}(\log n/\varepsilon^2(n \log n + m))$ time in the first case and $\mathcal{O}(m \log n/\varepsilon^2)$ in the latter case. Consequently, the time gain deterioration we see from $\varepsilon = 0.3$ to $\varepsilon = 0.05$ is a straightforward consequence of lowering ε. Finally, the fact that RAND_H is asymptotically faster than solving APSP is confirmed by the increasing values in the *gain* column as we consider larger networks.

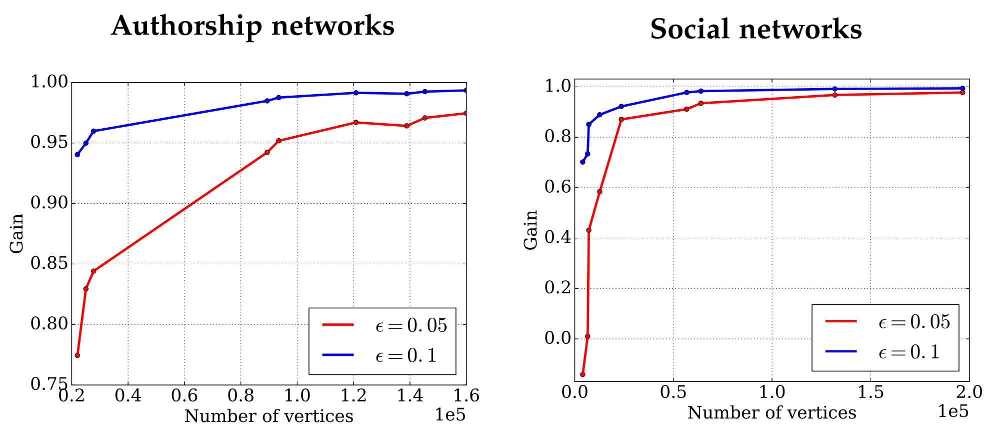


Figure 5.2: Time gain of the RAND_H algorithm.

5.3.2 Precision

We now focus our analysis of the RAND_H algorithm on two main networks (*Wikiquote (en)* and *Wiktionary (de)*) so it is easier for us to study the precision through the metrics we introduced in Section 5.1.1. Data is reported in Table 5.3.

To begin with, it is trivial to observe that, as we loosen the precision bound ε, RAND_H needs to extract less samples. In particular, if we lift ε by 10 times its value, the number of samples raises to 100 times its previous value. This is totally in agreement with the theory and can be observed by comparing the $\varepsilon = 0.05$ and the $\varepsilon = 0.5$ rows of Table 5.3.
The increase of the time gain for higher values of ε is a direct and obvious consequence of the reduction of the number of samples. In Figure 5.3a we reported the time gain trend with $\varepsilon = 0.05$ and $\varepsilon = 0.1$ so the curves are more similar and easy to compare. From these graphs it is evident that the time gain heavily depends on the networks size in terms of number of nodes and on the choice of ε. We can also notice that the time gain is negligible for graphs with less than about 1000 vertexes while it is considerably high for graph with more than 100.000 vertexes.

A surprising and positive aspect of these results concerns the errors made by the algorithm. If we look at the average error column we see not only that its values are always below ε as we expected from the theory, but, for each ε we chosen, that it also keeps a 10^{-3} magnitude which is a good result. In accordance with the theory, we can see from both the table and Figure 5.3b that the average error grows linearly as we augment ε. Furthermore, it is important to remark that the average error values have been calculated by considering their absolute value. This means that in the worst case scenario of our experiment (i.e. $\varepsilon = 0.5$) on average RAND_H made errors which are

less than 2% the requested bound ε. The ε/avg.error ratio can be observed more clearly from Figure 5.3g.
Next, the d_bound metric we introduced in Section 5.1.1 to measure the effectiveness of the estimated $\hat{h}(v)$ according to the chosen ε, always takes values below 0.016. Since d_bound $\in [0, 1]$ and $\hat{h}(v)$ is as much good as d_bound(v) approaches to 0, this result confirms that RAND_H achieved substantially good precision performances for each ε we chosen. From Figure 5.3d we can better see that d_bound, despite some swinging, does not grow linearly as the average error does and that the algorithm was generally more precise for the *Wiktionary (de)* networks which has about 5000 less nodes but over 300.000 more edges than *Wikiquote (en)*.

As we already mentioned in Section 5.1.1, the average error and the d_bound metrics can give us just a partial view of the overall precision of the algorithm we are analyzing. For this reason we also took into account parameters such as the absolute error variance, the relative error and the maximum error.
To begin with, if we examine the *variance* column of Table 5.3 we can conclude that, even though the reported values grow proportionally faster than the upper bound ε, all of them are located between $\sim 10^{-6}$ and $\sim 10^{-5}$, which are considerably small values compared to their corresponding average error. A clearer view of the distribution of the error is provided by Figures 5.4 and 5.5 where we also took into account the sign of the error. It is easy to verify that the histograms corresponding to the lowest values of ε, which have the lowest variance values, are also the narrowest, meaning that the error values are closer to their average than in the other cases. From Figure 5.3e we can also observe that the error variance sharply increases for both the considered networks when $\varepsilon \geq 0.35$ which correspond to the last four flattest histograms of Figures 5.4 and 5.5.
Then, the average relative error column shows us that the average error values are coherent with the actual error distribution since, in all the experiment we performed, it has always been below 0.02. Similarly to the average error, from both the values reported in the table and Figure 5.3f we can see that the relative error is heavily influenced by ε. Finally, in accordance with the theory, we can see that, as we augment the error upper bound, the relative error grows linearly.
From the maximum error column we can see that in the worst case it is about 0.045 with $\varepsilon = 0.45$ but, when $\varepsilon \leq 0.1$, it is always below 0.01. Since typical top-k ($k \leq 1000$) Harmonic Centrality values are $h(v) \simeq 0.5$, these kind of error are not significant. We can also observe from Figure 5.3c that, as the other metrics, the maximum error is directly proportional to ε.

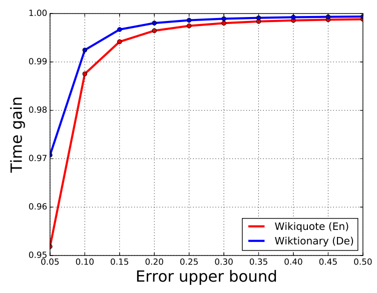
Wikiquote (en)

ε	**Samples**	**Gain**	**Avg err**	**d_bound**	**Var**	**Avg. δ**	**Max err**
0.05	4579	95.18%	0.33e-3	6.50e-3	0.20e-6	1.54e-3	2.70e-3
0.10	1146	98.75%	1.03e-3	10.29e-3	1.56e-6	4.54e-3	6.10e-3
0.15	510	99.42%	1.76e-3	11.72e-3	5.19e-6	6.36e-3	10.51e-3
0.20	287	99.65%	1.42e-3	7.10e-3	3.40e-6	6.43e-3	11.08e-3
0.25	184	99.75%	2.04e-3	8.17e-3	6.18e-6	6.61e-3	11.04e-3
0.30	128	99.80%	2.38e-3	7.92e-3	8.00e-6	7.79e-3	15.69e-3
0.35	94	99.84%	1.46e-3	4.18e-3	2.92e-6	4.86e-3	8.71e-3
0.40	73	99.86%	3.73e-3	9.33e-3	19.45e-6	11.90e-3	28.27e-3
0.45	58	99.87%	2.11e-3	4.69e-3	9.60e-6	6.87e-3	21.82e-3
0.50	47	99.88%	4.86e-3	9.72e-3	35.57e-6	15.18e-3	27.17e-3

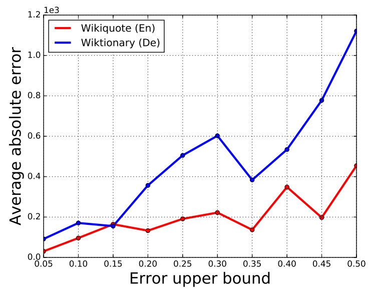
Wiktionary (de)

ε	**Samples**	**Gain**	**Avg err**	**d_bound**	**Var**	**Avg. δ**	**Max err**
0.05	4756	97.08%	0.63e-3	12.60e-3	0.66e-6	1.68e-3	6.94e-3
0.10	1190	99.25%	1.17e-3	11.74e-3	1.91e-6	3.07e-3	6.11e-3
0.15	529	99.67%	1.07e-3	7.12e-3	1.83e-6	2.97e-3	8.39e-3
0.20	298	99.80%	2.46e-3	12.28e-3	9.01e-6	6.50e-3	15.91e-3
0.25	191	99.86%	3.48e-3	13.91e-3	19.52e-6	9.01e-3	27.28e-3
0.30	133	99.89%	4.15e-3	13.82e-3	28.71e-6	11.00e-3	37.25e-3
0.35	98	99.91%	2.64e-3	7.55e-3	10.96e-6	7.22e-3	20.40e-3
0.40	75	99.92%	3.68e-3	9.19e-3	20.65e-6	10.06e-3	25.70e-3
0.45	60	99.93%	5.35e-3	11.90e-3	44.46e-6	14.13e-3	44.64e-3
0.50	49	99.94%	7.72e-3	15.43e-3	81.77e-6	19.93e-3	35.41e-3

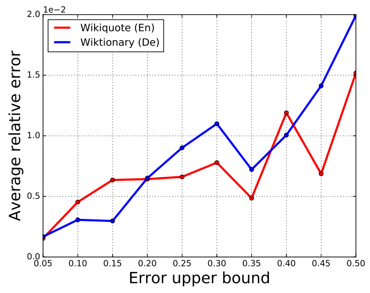
Table 5.3: ε: precision bound used for the current experiment. Samples: number of random samples extracted by RAND_H. Gain: time gain percentage. Avg. err: average error on the Harmonic Centrality of each vertex as defined in Equation 5.3. Var: variance of the error as defined in Equation 5.6. Avg. δ: average of the relative error as defined in Equation 5.8. Max err: maximum error made by RAND_H as defined in Equation 5.7.



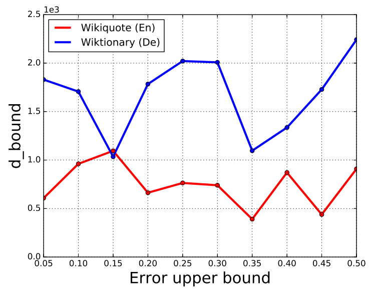
(a)



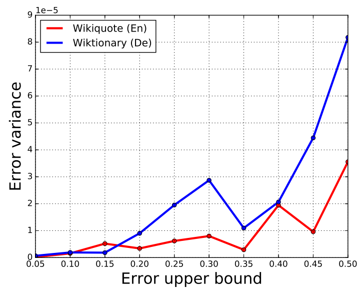
(b)



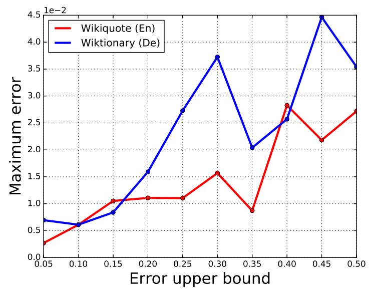
(c)



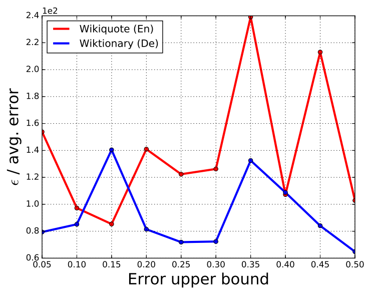
(d)



(e)



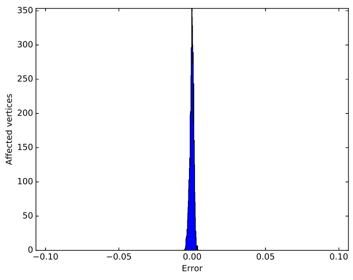
(f)



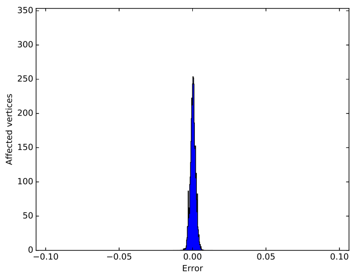
(g)

Figure 5.3: Precision metrics for Eppstein algorithm with different choices of the upper bound ε, represented data is referred to Table 5.3.

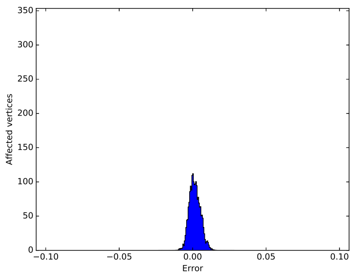
Wiktionary (de)



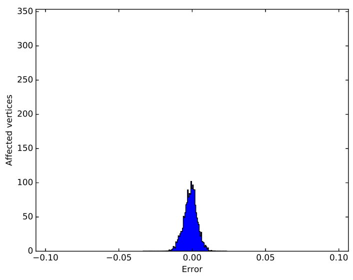
(a) $\varepsilon = 0.05$



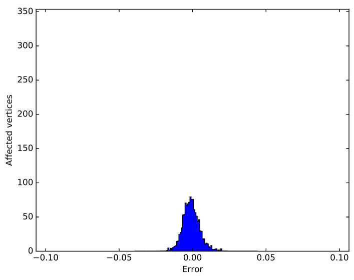
(b) $\varepsilon = 0.1$



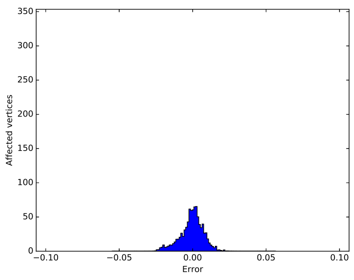
(c) $\varepsilon = 0.15$



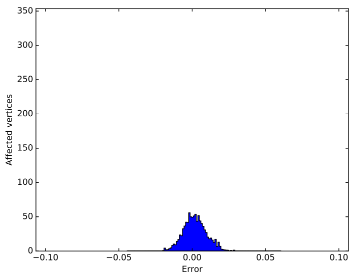
(d) $\varepsilon = 0.2$



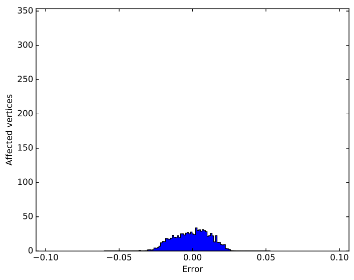
(e) $\varepsilon = 0.25$



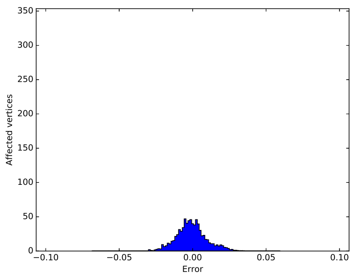
(f) $\varepsilon = 0.3$



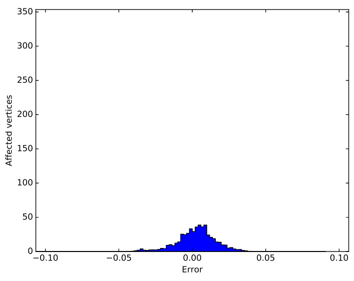
(g) $\varepsilon = 0.35$



(h) $\varepsilon = 0.4$



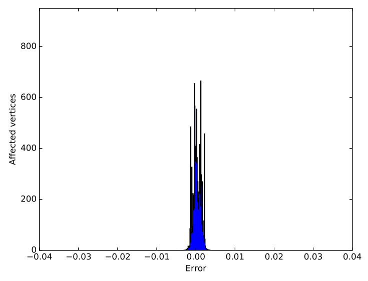
(i) $\varepsilon = 0.45$



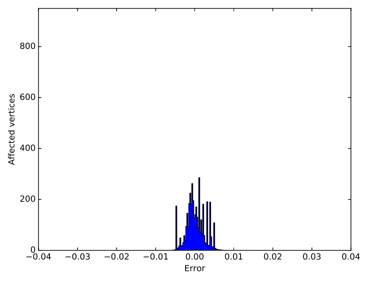
(j) $\varepsilon = 0.5$

Figure 5.4: Error distribution of Eppstein algorithm for different values of ε.

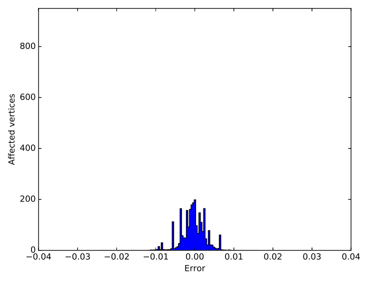
Wikiquote (en)



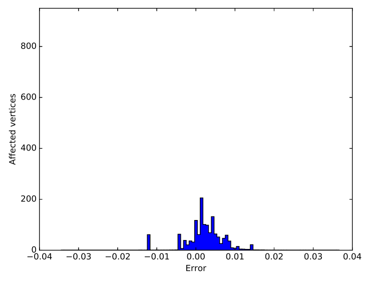
(a) $\varepsilon = 0.05$



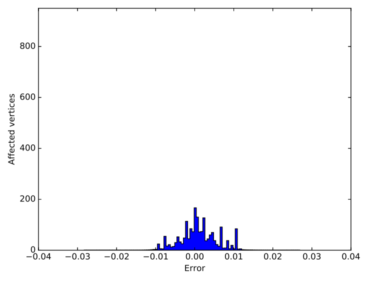
(b) $\varepsilon = 0.1$



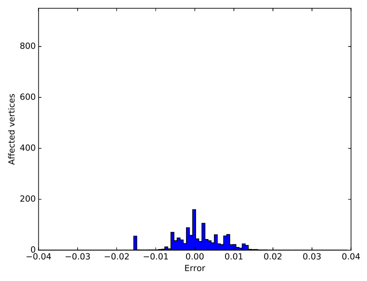
(c) $\varepsilon = 0.15$



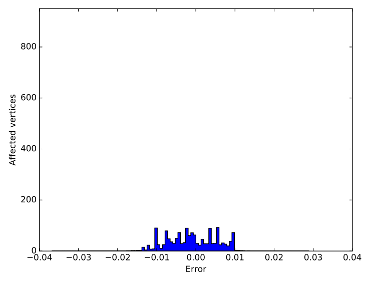
(d) $\varepsilon = 0.2$



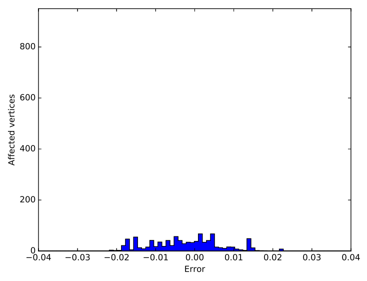
(e) $\varepsilon = 0.25$



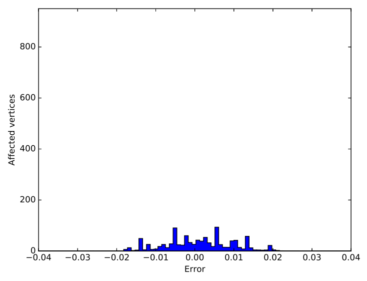
(f) $\varepsilon = 0.3$



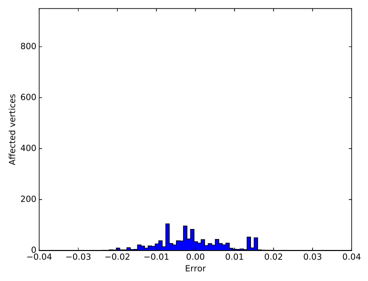
(g) $\varepsilon = 0.35$



(h) $\varepsilon = 0.4$



(i) $\varepsilon = 0.45$



(j) $\varepsilon = 0.5$

Figure 5.5: Error distribution of Eppstein algorithm for different values of ε.

5.3.3 Top-k analysis

We now focus our analysis on the precision achieved by the RAND_H algorithm among the top-k vertexes with greatest Harmonic Centrality value. To begin with we will report the average error and the average relative error among the top-k centralities. Then, we will check whether the top-K approximated Harmonic centralities correspond to the exact top-k centralities or not. If not we will count how many more vertexes should be added in order to obtain all the exact top-k most central vertexes.

Top-k errors

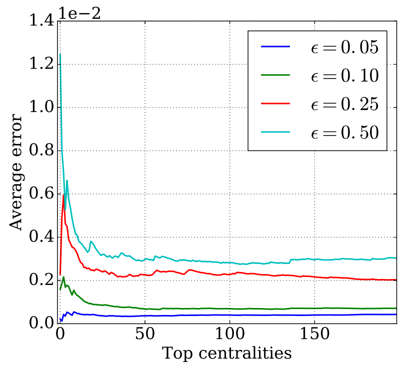
As we can see from Figures 5.6a and 5.6c the average error seems to be higher among the first top 20 centralities and then it follows a horizontal asymptote. This means that, unfortunately, the majority of the errors are grouped among the top-20 centralities. However, these results could be meaningless without taking into consideration the actual values of the top Harmonic centralities and the relative error. It is clear from Figure 5.7 that the top-20 centralities are also substantially higher than the others and this could explain the greater magnitude of the absolute error among the top-20 centralities. Conversely, Figures 5.6b and 5.6d show us that even the relative error is considerably higher among the top-20 most central vertexes than in the others.

In order to have a complete view of the precision of RAND_H we also payed attention to the absolute error and the relative error trend among the less central vertexes. In Figure 5.8 we reported the same kind of data displayed in Figure 5.6 but extended to $k = n$. A peak of both the absolute and the relative error is still evident among the highest centralities and, for the remaining centralities, the trend is approximately constant. We can also notice an absolute minimum immediately after the initial error peak in all the four graphs meaning that the RAND_H maximum precision is between about the top-50 and the top-400 centralities.

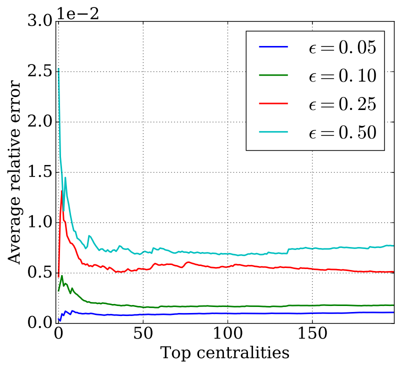
Exact top-k comparison

We now examine the precision achieved by the RAND_H algorithm among the top-k centralities from a different point of view. Instead of measuring the approximation error we are going to study whether RAND_H correctly ranks the top-k Harmonic centralities or not. If not, we will observe how many more approximated vertexes should be added in order to obtain the actual top-k centralities.

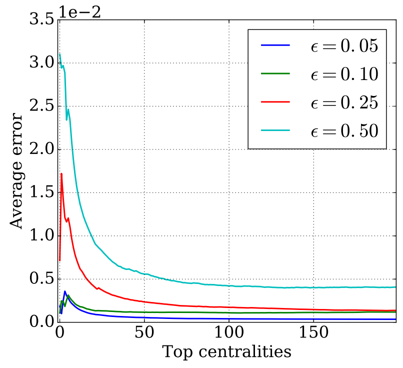
In order to provide a well-comparable performance metric, let us define a top-k precision ratio:



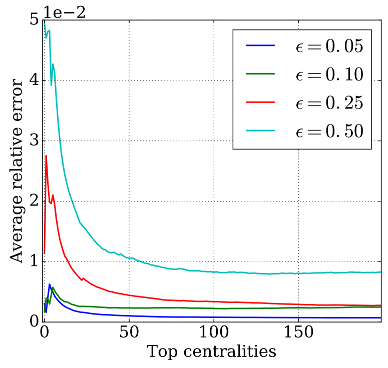
(a) Wikinews (en)



(b) Wikinews (en)



(c) Wiktionary (de)



(d) Wiktionary (de)

Figure 5.6: Average error and average relative error among the top-k centralities of two networks ($1 \leq k \leq 250$).

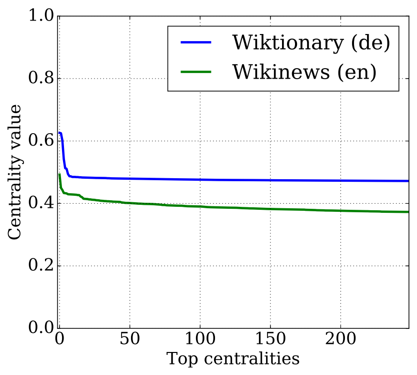
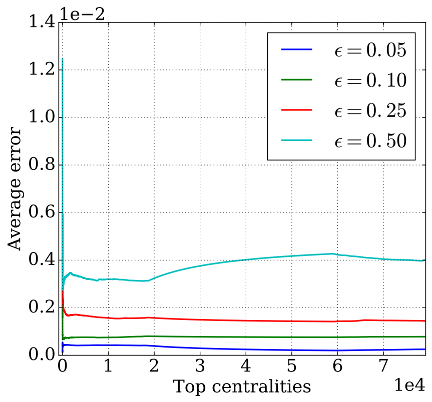
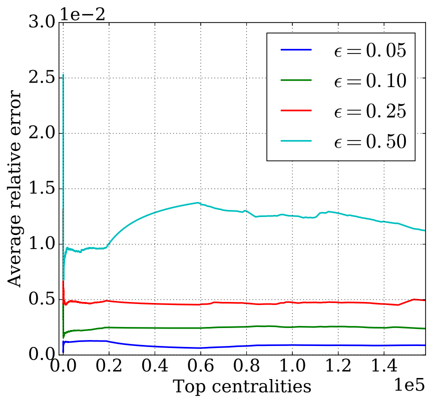


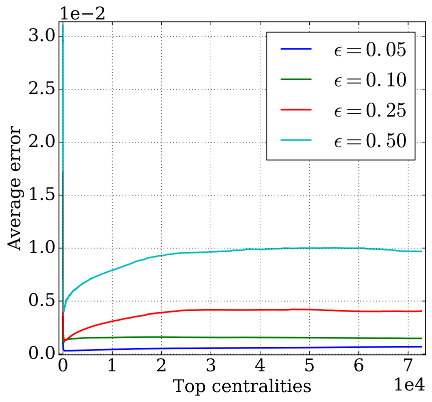
Figure 5.7: Top-k Harmonic Centrality values, $1 \leq k \leq 250$.



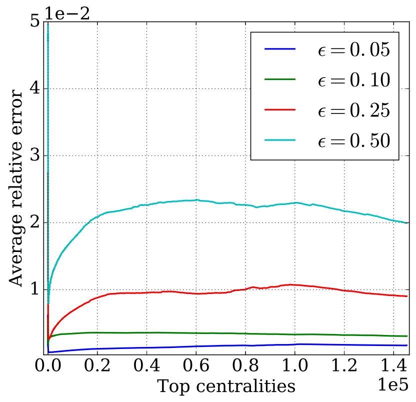
(a) Wikinews (en)



(b) Wikinews (en)



(c) Wiktionary (de)



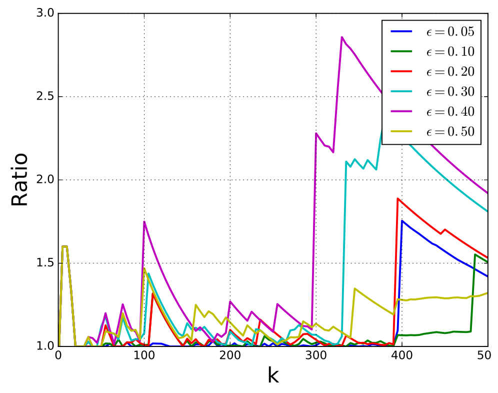
(d) Wiktionary (de)

Figure 5.8: Average error and average relative error among the top-k centralities of two networks ($1 \leq k \leq n$).

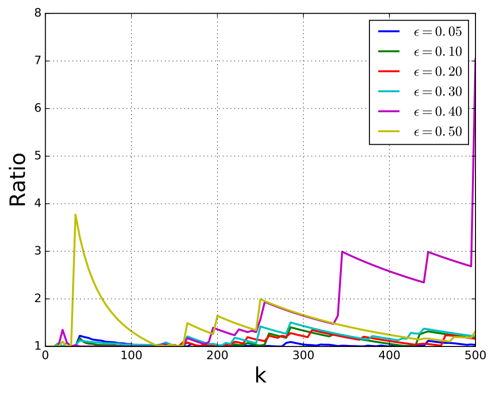
$$\text{ratio} = \frac{k + \hat{k}}{k} \tag{5.13}$$

where $\hat{k}$ is the number of vertexes which have been added to the approximated top-k set in order to obtain all the actual top-k most central nodes. Another metric that we use is the number of exact top-k vertexes the RAND_H missed in the estimation, which we denoted with Δ.

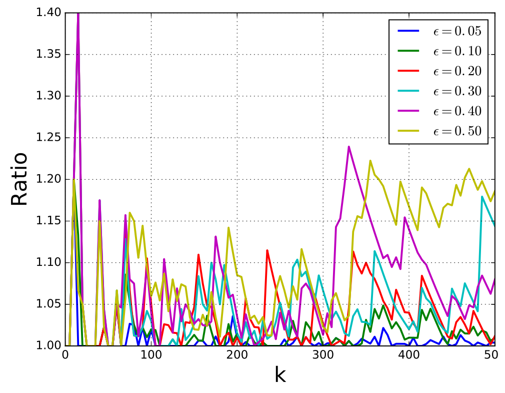
If we look at Tables 5.4 and 5.5 we can make the following observations. First of all the first Harmonic centrality is always correctly computed in both cases and for each value of ε. This is a considerably positive result since it shows us that, even with a loose precision bound, the most important centrality can be easily identified. Moreover, as we consider higher k values,



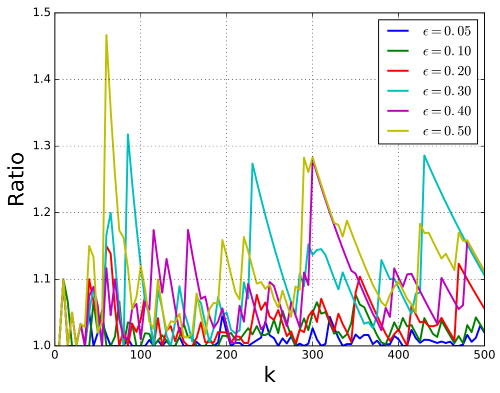
(a) Wikinews (en)



(b) Wiktionary (de)



(c) Gowalla



(d) Brightkite

Figure 5.9: Top-k precision ratio among four networks.

the precision drops but apparently it does not strongly depend on ε as we could expect. More precisely we can see that $\hat{k}$ values for a fixed k cannot be interpreted as a monotonically increasing function (with ε as variable) but, conversely, we can easily identify some peaks and nadirs. A clearer view is provided by Figure 5.9 since it takes into account much more k values. Unfortunately it is complicated to observe recognize a precise trend because the graphs are very noisy. We suppose that this is due to high similarity between consecutive centrality values, especially for greater values of k. Still we can identify pretty clearly that the worst performance curves (yellow and purple) are associated with the greatest ε values (0.4 and 0.5).

In conclusion we verified that, when k is small ($k \sim 10$), RAND_H precisely ranks the top-k Harmonic centralities and, by examining Figure 5.9 graphs, we also noticed the correlation between the ranking precision and ε.

Wiktionary (de)

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

$k = 1$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

$k = 5$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	2	5	1.50
0.50	0	0	1.00

$k = 10$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	1	1	1.05
0.20	0	0	1.00
0.25	1	3	1.15
0.30	0	0	1.00
0.35	1	1	1.05
0.40	2	7	1.35
0.45	0	0	1.00
0.50	2	2	1.10

$k = 20$

ε	Δ	$\hat{k}$	**Ratio**
0.05	9	9	1.18
0.10	3	3	1.06
0.15	2	2	1.04
0.20	0	0	1.00
0.25	3	3	1.06
0.30	5	5	1.10
0.35	2	2	1.04
0.40	0	0	1.00
0.45	4	4	1.08
0.50	2	82	2.64

$k = 50$

ε	Δ	$\hat{k}$	**Ratio**
0.05	5	5	1.05
0.10	1	1	1.01
0.15	1	1	1.01
0.20	0	0	1.00
0.25	3	3	1.03
0.30	4	4	1.04
0.35	2	2	1.02
0.40	0	0	1.00
0.45	4	4	1.04
0.50	2	32	1.32

$k = 100$

Table 5.4: Precision of the RAND_H algorithm among the top-k vertexes. ε: upper bound value, Δ: number of missed correct centralities, $\hat{k}$: number of vertexes added to obtain all the exact top-k set, ratio: as defined in Equation 5.13.

Wikinews (en)

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

$k = 1$

ε	Δ	$\hat{k}$	**Ratio**
0.05	1	3	1.60
0.10	1	3	1.60
0.15	2	13	3.60
0.20	1	3	1.60
0.25	1	3	1.60
0.30	1	3	1.60
0.35	1	3	1.60
0.40	1	3	1.60
0.45	1	3	1.60
0.50	1	3	1.60

$k = 5$

ε	Δ	$\hat{k}$	**Ratio**
0.05	4	6	1.60
0.10	4	6	1.60
0.15	4	10	2.00
0.20	4	6	1.60
0.25	4	6	1.60
0.30	4	6	1.60
0.35	4	6	1.60
0.40	4	6	1.60
0.45	4	6	1.60
0.50	4	6	1.60

$k = 10$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

$k = 20$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	1	1	1.02
0.20	0	0	1.00
0.25	0	0	1.00
0.30	3	6	1.12
0.35	0	0	1.00
0.40	2	5	1.10
0.45	1	1	1.02
0.50	0	0	1.00

$k = 50$

ε	Δ	$\hat{k}$	**Ratio**
0.05	1	1	1.01
0.10	1	1	1.01
0.15	2	3	1.03
0.20	1	1	1.01
0.25	4	44	1.44
0.30	3	8	1.08
0.35	3	4	1.04
0.40	3	75	1.75
0.45	5	61	1.61
0.50	3	47	1.47

$k = 100$

Table 5.5: See Table 5.4 for further details.

5.3.4 Comparison with Borassi et al.

In this section we compare the RAND_H algorithm with the Borassi et al.'s strategy in terms of running time. It is important to point out that, for a given network, their algorithm computes the exact top-k centralities with high probability while RAND_H approximates all the Harmonic centralities. Moreover, it can also deal with directed and not strongly connected graphs while RAND_H takes as input undirected and connected graphs only.

Unfortunately a Python implementation of the Borassi et al.'s algorithm adapted to the Harmonic Centrality does not exist yet so we are going to use their function that computes the Closeness Centrality:

centrality.TopCloseness(G, k)

that returns the top-k Closeness centralities of G. This function has been included into the NetworKit framework which includes a Python interface but the main algorithms have been written in C++ [27].

In Table 5.6 we reported RAND_H time gain on Borassi et al. on two authorship networks and two social networks. It is clear that, if we choose a low upper bound value ($\varepsilon \sim 0.05$) the gain is always negative, meaning that RAND_H required more time than Borassi et al. On the other hand, if we take into consideration higher k values we see that RAND_H recovers more and more in terms of running time since it does not depend on k while the BFSCut function time performances are inversely proportional to k. This aspect is emphasized in larger networks cases such as *Gowalla* or *Wiktionary (de)*.

Furthermore we should point out that the Borassi et al.'s algorithm supports multithreading while our RAND_H implementation does not. A multi-threaded implementation of RAND_H would doubtlessly achieve far better results. Nevertheless it is impressive that, in several cases, our sequential implementation is still quicker than a multi-threaded implementation of the Borassi et al.'s strategy.

5.4 RAND_H: second set of experiments

Up until now the choice of the number of random samples have been implemented as

$$\text{samples} = \left\lceil \frac{\log n}{\varepsilon^2} \right\rceil$$

In other words we imposed $C = 1$. However, this could not be the optimal choice into a practical scenario. In fact a lower C value would certainly diminish the number of samples and so the algorithm's running time but we cannot predict the actual impact on the precision since the theory imposes us to choose $C \geq 1$. In the following experiments we are going to analyze how the RAND_H's time and precision changes if we linearly lower the number of samples.

5.4.1 C = 0.5: time and precision performances

Our first choice of C is 0.5 since we want to understand the consequences on the RAND_H's precision of selecting exactly half of the samples we were selecting before.

Network	ε	**k**	**Gain**
Gowalla	0.05	1	-92155%
		50	-406.61%
		500	-296.17%
	0.15	1	-10717%
		50	40.60%
		500	53.55%
	0.3	1	-3652%
		50	79.39%
		500	83.89%
	0.5	1	-1901%
		50	89.01%
		500	91.40%

Network	ε	**k**	**Gain**
Wikinews (en)	0.05	1	-97886%
		50	-28507%
		500	-703.81%
	0.15	1	-11636%
		50	-3326%
		500	3.72%
	0.3	1	-3840%
		50	-1050%
		500	67.67%
	0.5	1	-2195%
		50	-570.08%
		500	81.17%

Network	ε	**k**	**Gain**
Anybeat	0.05	1	-86902%
		50	-14554%
		500	-2976%
	0.15	1	-10833%
		50	-1741%
		500	-286.56%
	0.3	1	-3742.73%
		50	-547.26%
		500	-35.86%
	0.5	1	-2255%
		50	-296.77%
		500	16.72%

Network	ε	**k**	**Gain**
Wiktionary (de)	0.05	1	-359.61%
		50	-297.47%
		500	-226.79%
	0.15	1	48.21%
		50	55.21%
		500	63.17%
	0.3	1	83.21%
		50	85.48%
		500	88.06%
	0.5	1	90.30%
		50	91.61%
		500	93.10%

Table 5.6: Time performance comparison between RAND_H and Borassi et al.'s algorithms. ε: error upper bound used by RAND_H. k: number centralities extracted by the Borassi et al.'s algorithm. Gain: time gain of RAND_H on Borassi et al.

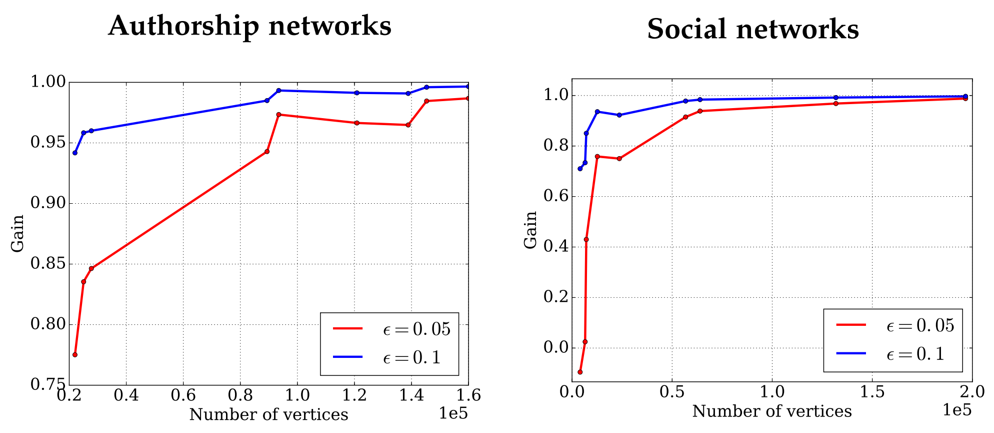


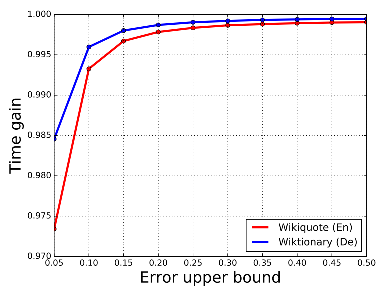
Figure 5.10: Time gain of the RAND_H algorithm, $C = 0.5$

By looking at Table 5.7 we can observe that, since we halved the number of samples, the time gain is greater than in Table 5.3. The number of shortest paths RAND_H has to compute are exactly half than before but some post-processing operations for output formatting are still necessary. Thus the overall running time is slightly more than half the time required in the previous experiment. Figures 5.10 and 5.11a also show us that the time gain is still directly proportional to the network size as we expected.

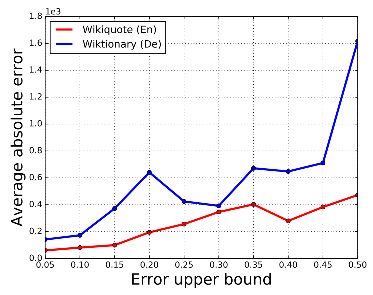
We can also see that, despite the lower number of samples, the accuracy decreased linearly. Above all, the average absolute and relative errors still have the same scale than in Table 5.3. This result is significant since the errors made by RAND_H are still much lower than the corresponding upper bound ε. By comparing the graphs represented in Figures 5.3 and 5.11 we can verify that the trend of the precision metrics did not substantially change. The main differences are represented by the improvement of the time gain and the slight deterioration of the precision metrics which still remain remarkably positive if compared to their corresponding ε value.

5.4.2 C = 0.5: top-k analysis

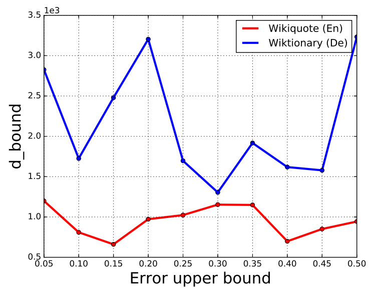
Another interesting aspect we considered in our analysis is the precision that the RAND_H algorithm achieved among the top-k most central vertexes. More precisely, we performed again the comparison with the exact top-k Harmonic centralities we did in the previous experiment, this time with half of random samples. As we can see from Tables 5.8 and 5.9 the top-k ranking precision is still remarkably good if compared with the previous experiment, especially for the lower k values. Surprisingly the *Wiktionary (de)* network obtained even better results than before since the number of outliers in most of the cases is less than in Table 5.4. On the other hand RAND_H had a slight worse ranking precision on the *Wikinews (en)* network since it ranked



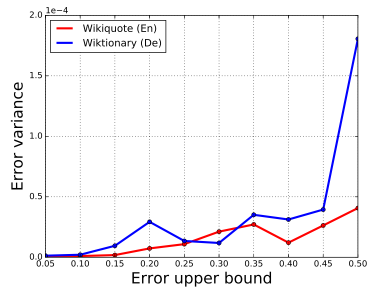
(a)



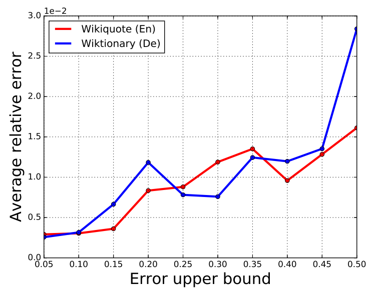
(b)



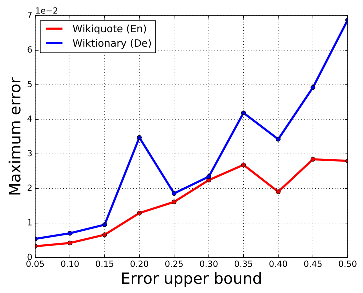
(c) d_bound over ε.



(d)



(e)



(f)

Figure 5.11: Precision metrics for the RAND_H algorithm with different choices of the upper bound ε. $C = 0.5$. Represented data is referred to Table 5.7.

Wikiquote (en)

ε	**Samples**	**Gain**	**Avg. err**	**d_bound**	**Var**	**Avg. δ**	**Max err**
0.05	2290	97.34%	0.64e-3	12.83e-3	0.61e-6	2.91e-3	3.28e-3
0.10	573	99.33%	0.87e-3	8.66e-3	1.02e-6	3.04e-3	4.23e-3
0.15	255	99.67%	1.06e-3	7.07e-3	1.83e-6	3.61e-3	6.61e-3
0.20	144	99.78%	2.08e-3	10.41e-3	7.34e-6	8.34e-3	12.89e-3
0.25	93	99.83%	2.74e-3	10.95e-3	10.87e-6	8.80e-3	16.12e-3
0.30	65	99.87%	3.70e-3	12.34e-3	21.20e-6	11.88e-3	22.43e-3
0.35	48	99.88%	4.30e-3	12.30e-3	27.14e-6	13.51e-3	26.83e-3
0.40	37	99.89%	2.99e-3	7.48e-3	12.10e-6	9.58e-3	19.05e-3
0.45	29	99.90%	4.10e-3	9.10e-3	26.30e-6	12.84e-3	28.45e-3
0.50	24	99.90%	5.05e-3	10.10e-3	40.69e-6	16.12e-3	27.99e-3

Wiktionary (de)

ε	**Samples**	**Gain**	**Avg err**	**d_bound**	**Var**	**Avg. δ**	**Max err**
0.05	2378	98.46%	0.97e-3	19.45e-3	1.33e-6	2.56e-3	5.42e-3
0.10	595	99.60%	1.19e-3	11.87e-3	2.10e-6	3.16e-3	7.05e-3
0.15	265	99.80%	2.56e-3	17.06e-3	9.50e-6	6.65e-3	9.52e-3
0.20	150	99.87%	4.41e-3	22.05e-3	29.29e-6	11.84e-3	34.75e-3
0.25	96	99.90%	2.92e-3	11.67e-3	13.49e-6	7.81e-3	18.57e-3
0.30	67	99.92%	2.69e-3	8.98e-3	11.83e-6	7.59e-3	23.43e-3
0.35	50	99.93%	4.62e-3	13.19e-3	35.12e-6	12.44e-3	41.88e-3
0.40	38	99.94%	4.46e-3	11.14e-3	31.24e-6	11.97e-3	34.25e-3
0.45	30	99.94%	4.89e-3	10.86e-3	39.44e-6	13.53e-3	49.23e-3
0.50	25	99.95%	11.13e-3	22.25e-3	180.60e-6	28.40e-3	68.75e-3

Table 5.7: Same experiment as in Table 5.3, $C = 0.5$.

the actual first Harmonic Centrality as the 20th using $\varepsilon = 0.05$ and we overall registered more outliers than in the previous case.
A more extended view is provided by Figure 5.12 where we can still notice the peaks and nadirs trend as in the $C = 1$ case.

In conclusion, despite the *Wikinews (en)* experiment with $\varepsilon = 0.05$, we can say that the overall RAND_H ranking precision was not compromised by the samples reduction and, for $k \sim 20$, it still precisely ranks the top-k Harmonic centralities. Furthermore, Table 5.10 also shows us that, if we impose $C = 0.5$, RAND_H becomes more competitive with respect to the algorithm designed by Borassi et al., especially for larger networks and for k bigger than 50.

Wiktionary (de)

ε	Δ	$\hat{k}$	**Ratio**	ε	Δ	$\hat{k}$	**Ratio**	ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00	0.05	0	0	1.00	0.05	0	0	1.00
0.10	0	0	1.00	0.10	0	0	1.00	0.10	0	0	1.00
0.15	0	0	1.00	0.15	0	0	1.00	0.15	0	0	1.00
0.20	0	0	1.00	0.20	0	0	1.00	0.20	0	0	1.00
0.25	0	0	1.00	0.25	0	0	1.00	0.25	1	1	1.10
0.30	0	0	1.00	0.30	0	0	1.00	0.30	0	0	1.00
0.35	0	0	1.00	0.35	0	0	1.00	0.35	0	0	1.00
0.40	0	0	1.00	0.40	0	0	1.00	0.40	0	0	1.00
0.45	0	0	1.00	0.45	0	0	1.00	0.45	1	6	1.60
0.50	0	0	1.00	0.50	0	0	1.00	0.50	1	1	1.10
	$k = 1$				$k = 5$				$k = 10$		
ε	Δ	$\hat{k}$	**Ratio**	ε	Δ	$\hat{k}$	**Ratio**	ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00	0.05	4	4	1.08	0.05	2	2	1.02
0.10	0	0	1.00	0.10	2	2	1.04	0.10	0	0	1.00
0.15	0	0	1.00	0.15	0	0	1.00	0.15	0	0	1.00
0.20	0	0	1.00	0.20	1	1	1.02	0.20	1	1	1.01
0.25	1	2	1.10	0.25	0	0	1.00	0.25	0	0	1.00
0.30	1	3	1.15	0.30	0	0	1.00	0.30	0	0	1.00
0.35	2	6	1.30	0.35	1	1	1.02	0.35	1	1	1.01
0.40	1	1	1.05	0.40	0	0	1.00	0.40	0	0	1.00
0.45	2	5	1.25	0.45	1	1	1.02	0.45	1	1	1.01
0.50	3	6	1.30	0.50	5	5	1.10	0.50	5	5	1.05
	$k = 20$				$k = 50$				$k = 100$		

Table 5.8: Precision of the RAND_H algorithm among the top-k vertexes. ε: upper bound value, Δ: number of missed correct centralities, $\hat{k}$: number of vertexes added to obtain all the exact top-k set, ratio: as defined in Equation 5.13. $C = 0.5$

Wikinews (en)

ε	Δ	$\hat{k}$	**Ratio**
0.05	1	19	20.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

$k = 1$

ε	Δ	$\hat{k}$	**Ratio**
0.05	2	15	4.00
0.10	1	3	1.60
0.15	1	3	1.60
0.20	1	3	1.60
0.25	1	3	1.60
0.30	1	3	1.60
0.35	1	3	1.60
0.40	1	3	1.60
0.45	1	3	1.60
0.50	1	3	1.60

$k = 5$

ε	Δ	$\hat{k}$	**Ratio**
0.05	5	10	2.00
0.10	4	6	1.60
0.15	4	6	1.60
0.20	4	6	1.60
0.25	4	6	1.60
0.30	4	6	1.60
0.35	4	6	1.60
0.40	4	6	1.60
0.45	4	6	1.60
0.50	4	6	1.60

$k = 10$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

$k = 20$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	2	6	1.12
0.35	1	24	1.48
0.40	2	12	1.24
0.45	2	22	1.44
0.50	5	22	1.44

$k = 50$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	1	1	1.01
0.15	2	42	1.42
0.20	1	1	1.01
0.25	4	6	1.06
0.30	7	52	1.52
0.35	4	58	1.58
0.40	4	16	1.16
0.45	6	48	1.48
0.50	7	63	1.63

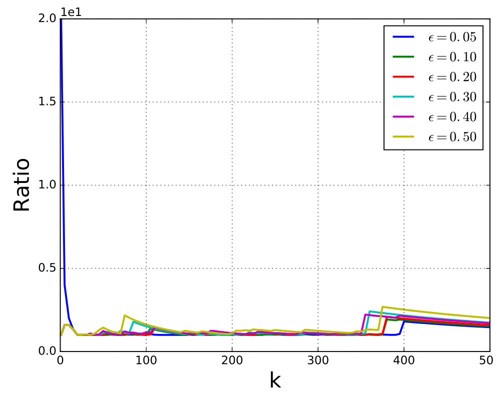
$k = 100$

Table 5.9: See Table 5.8 for further details.

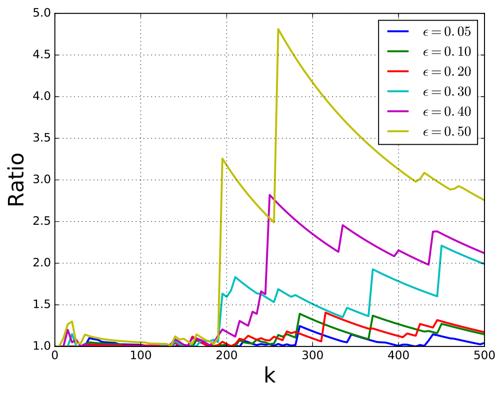
5.4.3 C = 0.25: time and precision performances

Since the results we obtained using $C = 0.5$ were quite encouraging, we expect that a further reduction of the number of samples could make the RAND_H algorithm save even more time without compromising its precision. More precisely we suppose that the error keeps growing linearly as we halve again the selected samples.

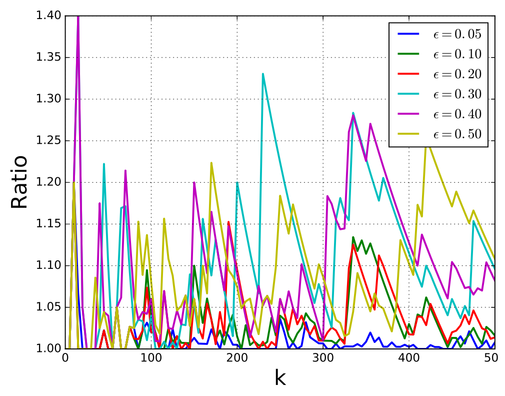
By comparing the results reported in Table 5.11 and in Figure 5.14 with the ones we obtained in the previous experiments (Tables 5.3 and 5.7) we can make the following considerations. First of all the time gain rose again thanks to the smaller number of samples (see also Figure 5.13). Then, the precision metrics changed as we expected. In particular metrics which are



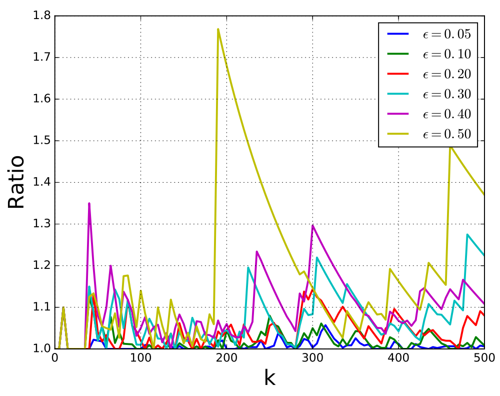
(a) Wikinews (en)



(b) Wiktionary (de)



(c) Gowalla



(d) Brightkite

Figure 5.12: RAND_H top-k precision ratio, $C = 0.5$

directly proportional to the errors (the average error, d_bound, the relative error and the maximum error) are nearly two times their original value we registered with $C = 1$. Therefore, selecting $\lceil 0.25 \cdot \log n / \varepsilon^2 \rceil$ samples does not compromise the precision of the RAND_H algorithm and, at the same time, it saves almost 75% of running time.

5.4.4 C = 0.25: top-k analysis

As we did in the previous sections we also examined the ranking precision among the top-k vertexes. Similarly to the $C = 0.5$ case, we can see in Tables 5.12 and 5.13 that RAND_H is still considerably accurate in identifying the top-k most central vertexes, especially when $k \sim 10$. We recorded the majority of the errors when $k \geq 100$ since these centralities have quite similar values and, according with Figure 5.15 RAND_H fails to rank them correctly

Network	ε	**k**	**Ratio**
		1	-49400%
	0.05	50	-171.83%
		500	-112.57%
		1	-6223%
	0.15	50	65.28%
Gowalla		500	72.85%
		1	-2274%
	0.3	50	86.96%
		500	89.80%
		1	-1481%
	0.5	50	91.31%
		500	93.21%

Network	ε	**k**	**Ratio**
	0.05	1	-50890%
	0.05	50	-14786%
	0.05	500	-318.29%
	0.15	1	-6577%
	0.15	50	-1849%
Wikinews	0.15	500	45.23%
(en)	0.3	1	-2484%
	0.3	50	-654.51%
	0.3	500	78.80%
	0.5	1	-1827%
	0.5	50	-462.75%
	0.5	500	84.19%

Network	ε	**k**	**Ratio**
	0.05	1	-50384%
	0.05	50	-8403%
	0.05	500	-1684%
	0.15	1	-6697%
	0.15	50	-1045%
Anybeat	0.15	500	-140.34%
	0.3	1	-2741%
	0.3	50	-378.56%
	0.3	500	-0.45%
	0.5	1	-1947%
	0.5	50	-244.80%
	0.5	500	27.63%

Network	ε	**k**	**Ratio**
		1	-142.54%
	0.05	50	-109.75%
		500	-72.45%
		1	68.77%
	0.15	50	73.00%
Wiktionary		500	77.80%
(de)		1	87.57%
	0.3	50	89.25%
		500	91.16%
		1	91.62%
	0.5	50	92.76%
		500	94.04%

Table 5.10: Comparison with Borassi et al. algorithm. $C = 0.5$.

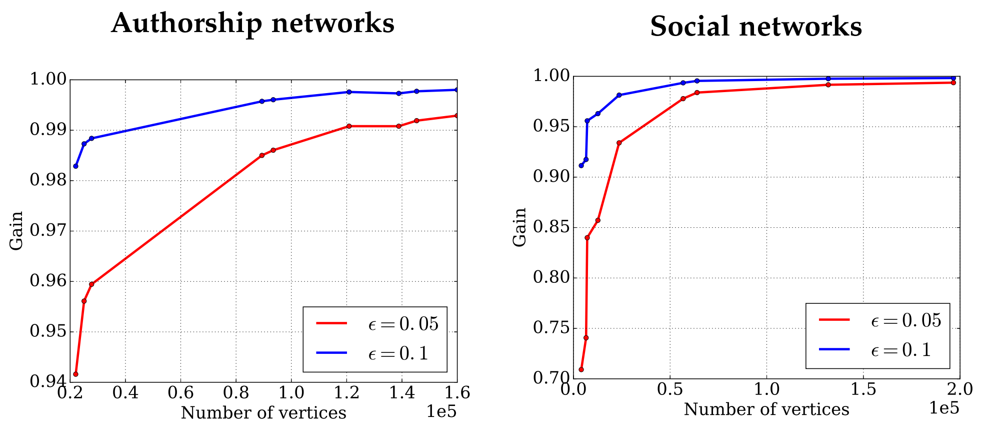
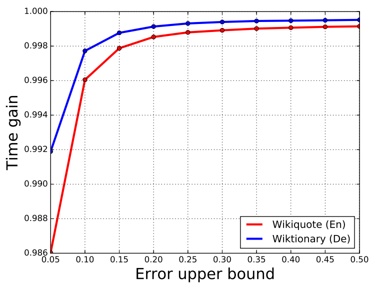
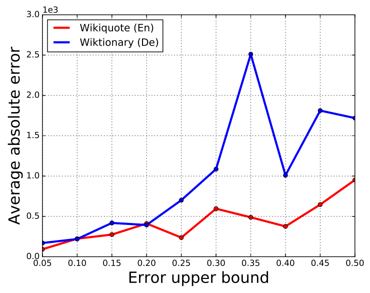


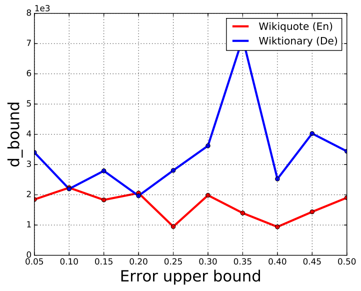
Figure 5.13: Time gain of the RAND_H algorithm, $C = 0.25$



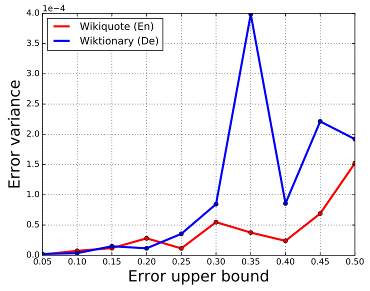
(a)



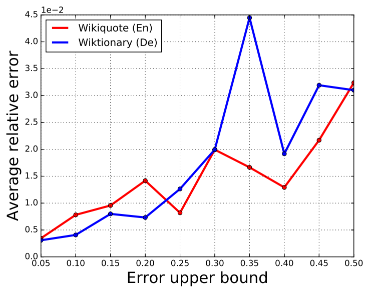
(b)



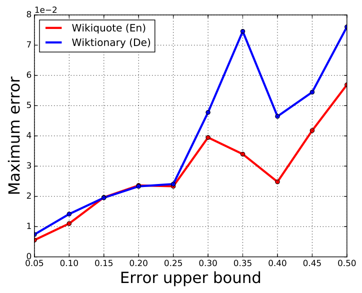
(c)



(d)



(e)



(f)

Figure 5.14: Precision metrics for the RAND_H algorithm with different choices of the upper bound ε. $C = 0.25$. Represented data is referred to Table 5.11.

Wikiquote (en)

ε	**Samples**	**Ratio**	**Avg err**	**d_bound**	**Var**	**Avg. δ**	**Max err**
0.05	1146	98.60%	0.99e-3	19.75e-3	1.26e-6	3.46e-3	5.55e-3
0.10	287	99.60%	2.39e-3	23.91e-3	7.41e-6	7.81e-3	11.01e-3
0.15	128	99.79%	2.94e-3	19.59e-3	11.79e-6	9.56e-3	19.67e-3
0.20	73	99.85%	4.40e-3	22.02e-3	28.06e-6	14.17e-3	23.62e-3
0.25	47	99.88%	2.54e-3	10.15e-3	11.41e-6	8.20e-3	23.34e-3
0.30	33	99.89%	6.37e-3	21.24e-3	54.79e-6	19.93e-3	39.48e-3
0.35	24	99.90%	5.23e-3	14.95e-3	37.55e-6	16.65e-3	33.97e-3
0.40	19	99.91%	4.02e-3	10.06e-3	23.85e-6	12.92e-3	24.80e-3
0.45	15	99.91%	6.92e-3	15.38e-3	68.99e-6	21.70e-3	41.77e-3
0.50	12	99.91%	10.19e-3	20.38e-3	152.32e-6	32.39e-3	56.90e-3

Wiktionary (de)

ε	**Samples**	**Ratio**	**Avg err**	**d_bound**	**Var**	**Avg. δ**	**Max err**
0.05	1190	99.19%	1.17e-3	23.41e-3	2.12e-6	3.09e-3	7.41e-3
0.10	298	99.77%	1.51e-3	15.10e-3	3.55e-6	4.08e-3	14.16e-3
0.15	133	99.88%	2.88e-3	19.22e-3	14.93e-6	7.99e-3	19.55e-3
0.20	75	99.91%	2.71e-3	13.54e-3	11.50e-6	7.32e-3	23.31e-3
0.25	49	99.93%	4.83e-3	19.31e-3	35.46e-6	12.64e-3	24.07e-3
0.30	34	99.94%	7.48e-3	24.92e-3	84.56e-6	19.95e-3	47.77e-3
0.35	25	99.95%	17.28e-3	49.37e-3	398.59e-6	44.47e-3	74.58e-3
0.40	20	99.95%	6.95e-3	17.37e-3	85.69e-6	19.17e-3	46.45e-3
0.45	16	99.95%	12.47e-3	27.71e-3	221.33e-6	31.92e-3	54.46e-3
0.50	13	99.95%	11.83e-3	23.66e-3	192.01e-6	31.00e-3	76.05e-3

Table 5.11: Same experiment as in Table 5.3, $C = 0.25$.

because its lower precision.

We also repeated the comparison with the Borassi et al. algorithm and, in accordance with Table 5.14, we noticed again a remarkable improvement. However, their algorithm is still much more competitive than RAND_H for $k = 1$.

Wiktionary (de)

k = 1

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

k = 5

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

k = 10

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	1	5	1.50
0.35	2	7	1.70
0.40	1	5	1.50
0.45	1	1	1.10
0.50	1	1	1.10

k = 20

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	2	2	1.10
0.15	0	0	1.00
0.20	0	0	1.00
0.25	1	2	1.10
0.30	3	6	1.30
0.35	1	2	1.10
0.40	3	9	1.45
0.45	2	7	1.35
0.50	2	2	1.10

k = 50

ε	Δ	$\hat{k}$	**Ratio**
0.05	1	1	1.02
0.10	2	2	1.04
0.15	4	4	1.08
0.20	1	81	2.62
0.25	7	7	1.14
0.30	2	2	1.04
0.35	1	1	1.02
0.40	1	84	2.68
0.45	14	15	1.30
0.50	10	101	3.02

k = 100

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	2	2	1.02
0.15	3	3	1.03
0.20	1	31	1.31
0.25	7	7	1.07
0.30	2	2	1.02
0.35	1	1	1.01
0.40	1	34	1.34
0.45	15	15	1.15
0.50	10	51	1.51

Table 5.12: Precision of the RAND_H algorithm among the top-*k* vertexes. ε: upper bound value, Δ: number of missed correct centralities, $\hat{k}$: number of vertexes added to obtain all the exact top-*k* set, ratio: as defined in Equation 5.13. $C = 0.25$

Wikinews (en)

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

$k = 1$

ε	Δ	$\hat{k}$	**Ratio**
0.05	1	1	1.20
0.10	1	3	1.60
0.15	1	3	1.60
0.20	1	3	1.60
0.25	1	3	1.60
0.30	1	3	1.60
0.35	1	3	1.60
0.40	1	3	1.60
0.45	1	3	1.60
0.50	1	3	1.60

$k = 5$

ε	Δ	$\hat{k}$	**Ratio**
0.05	4	4	1.40
0.10	5	10	2.00
0.15	4	6	1.60
0.20	4	6	1.60
0.25	4	6	1.60
0.30	4	6	1.60
0.35	4	6	1.60
0.40	4	6	1.60
0.45	4	6	1.60
0.50	4	6	1.60

$k = 10$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	0	0	1.00
0.25	0	0	1.00
0.30	0	0	1.00
0.35	0	0	1.00
0.40	0	0	1.00
0.45	0	0	1.00
0.50	0	0	1.00

$k = 20$

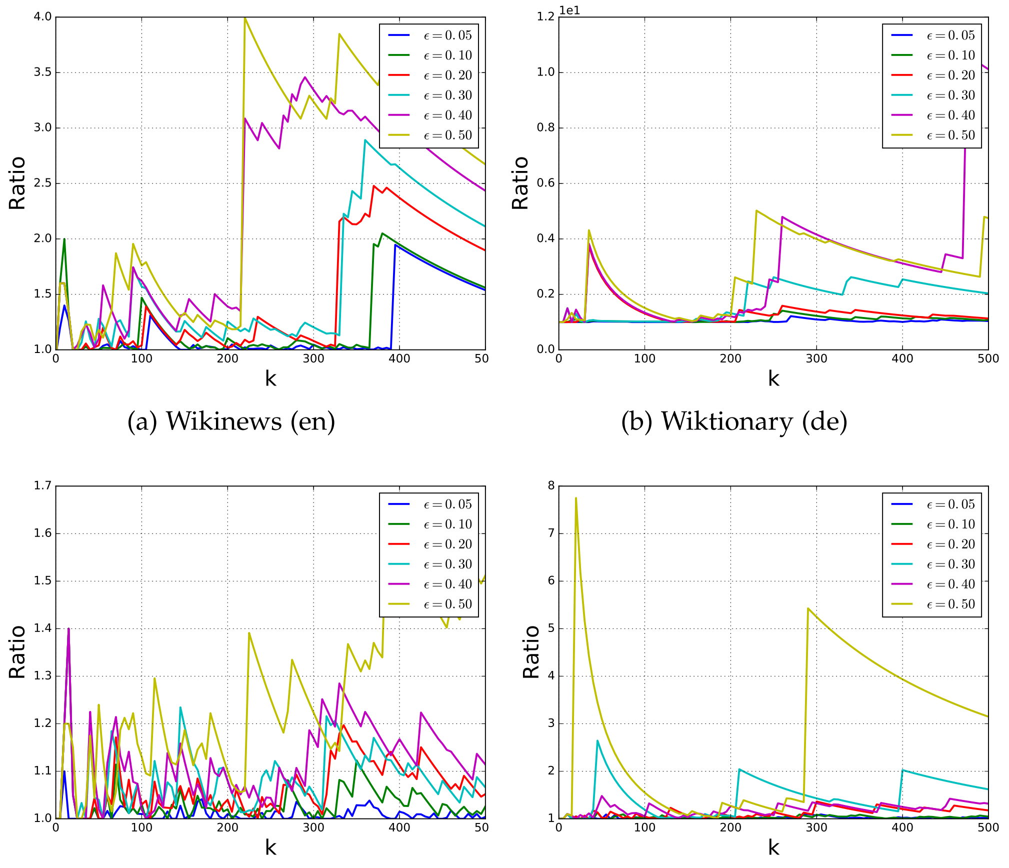
ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	0	0	1.00
0.15	0	0	1.00
0.20	1	2	1.04
0.25	1	4	1.08
0.30	3	14	1.28
0.35	2	34	1.68
0.40	3	5	1.10
0.45	4	16	1.32
0.50	3	9	1.18

$k = 50$

ε	Δ	$\hat{k}$	**Ratio**
0.05	0	0	1.00
0.10	1	47	1.47
0.15	2	43	1.43
0.20	1	4	1.04
0.25	1	5	1.05
0.30	5	57	1.57
0.35	2	64	1.64
0.40	7	62	1.62
0.45	2	52	1.52
0.50	12	76	1.76

$k = 100$

Table 5.13: See Table 5.8 for further details.



(a) Wikinews (en)

(b) Wiktionary (de)

(c) Gowalla

(d) Brightkite

Figure 5.15: RAND_H top-k precision ratio, $C = 0.25$

Network	ε	**k**	**Ratio**	**Network**	ε	**k**	**Ratio**
Gowalla	0.05	1	-25733%	Wikinews (en)	0.05	1	-27325%
		50	-41.86%			50	-7906%
		500	-10.94%			500	-124.98%
	0.15	1	-3576%		0.15	1	-4111%
		50	79.81%			50	-1129%
		500	84.21%			500	65.45%
	0.3	1	-1707%		0.3	1	-2007%
		50	90.07%			50	-515.41%
		500	92.24%			500	82.71%
	0.5	1	-1315%		0.5	1	-1615%
		50	92.23%			50	-400.74%
		500	93.92%			500	85.93%

Network	ε	**k**	**Ratio**	**Network**	ε	**k**	**Ratio**
Anybeat	0.05	1	-29766%	Wiktionary (de)	0.05	1	-27.36%
		50	-4930%			50	-10.14%
		500	-955.94%			500	9.45%
	0.15	1	-4174%		0.15	1	80.63%
		50	-620.01%			50	83.25%
		500	-51.13%			500	86.23%
	0.3	1	-2131%		0.3	1	90.52%
		50	-275.91%			50	91.80%
		500	21.10%			500	93.26%
	0.5	1	-1647%		0.5	1	92.45%
		50	-194.35%			50	93.47%
		500	38.22%			500	94.63%

Table 5.14: Comparison with Borassi et al. algorithm, $C = 0.25$.

5.5 TOPRANK_H

In this section we report and analyze the performances achieved by the TOPRANK_H algorithm in terms of time and precision through three main sets of experiments. The first set had been thought to validate the theoretical results we achieved in Chapter 4 into a practical environment so we imposed $\beta = 1$ and $\alpha = 1.01$. The other two sets were meant to verify whether it is possible to boost the time performance of the TOPRANK_H algorithm without compromising its precision since it should exactly compute the top-k Harmonic centralities.

5.5.1 First set of experiments: $\beta = 1$, $\alpha = 1.01$

Let us summarize the main features of Table 5.15. It is trivial to observe that, as we consider higher values of k, the time performances diminish, especially for smaller networks. This is due to the greater number of shortest paths needed to be computed and to the initial overhead of the algorithm. This phenomena can be observed more easily in Table 5.16 and in Figure 5.16.
Moreover, the time gain gets better as we increase the network size, especially from the number of vertexes point of view. This represents an evident consequence of the initial overhead due to the execution of the RAND_H algorithm. A straightforward view is provided by Figure 5.16.
Finally, although TOPRANK_H was designed to compute the exact top-k Harmonic centralities with high probability, it failed in one of the considered cases ($k = 100$ of the *Wikinews (en)* network). This probably means that we did not choose the optimal combination of the α and β constants. In the following experiments we will try to overhaul this aspect by changing the constants values.

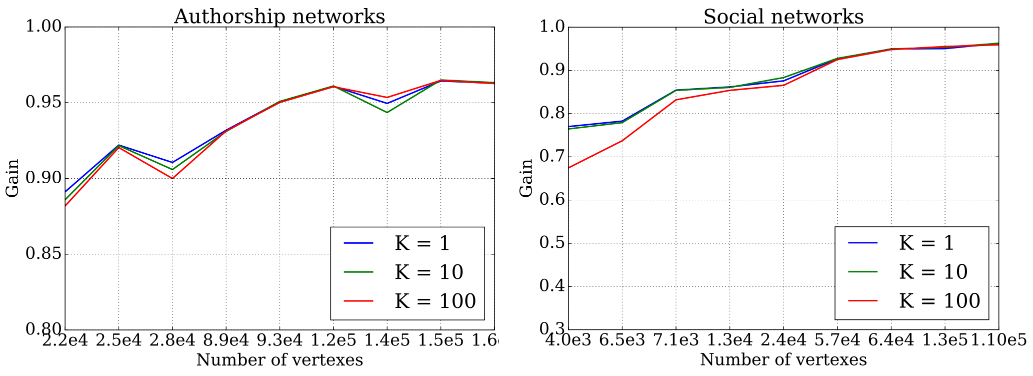


Figure 5.16: Time performances achieved by the TOPRANK_H algorithm.

Authorship networks

Name	**Samples**	$k=1$		$k=10$		$k=100$		
		$\hat{k}$	Gain	$\hat{k}$	Gain	$\hat{k}$	Gain	Prec.
Wikinews (en)	6745	0	96.285%	2	96.325%	3	96.266%	98.50%
Wiktionary (de)	6309	0	96.433%	4	96.499%	31	96.473%	100%
DBpedia prod.	6112	0	94.954%	5	94.353%	33	95.348%	100%
Github	5551	0	96.086%	1	96.101%	13	96.055%	100%
Wikiquote (en)	4642	0	95.071%	1	95.082%	18	95.020%	100%
arXiv cond-mat	4499	1	93.185%	2	93.105%	46	93.135%	100%
Wikibooks (fr)	1991	0	91.056%	3	90.588%	46	89.997%	100%
Wikinews (fr)	1853	0	92.203%	0	92.170%	57	92.039%	100%
Writers	1693	2	89.116%	1	88.593%	52	88.184%	100%

Social networks

Name	**Samples**	$k=1$		$k=10$		$k=100$		
		$\hat{k}$	Gain	$\hat{k}$	Gain	$\hat{k}$	Gain	Prec.
Gowalla	7782	0	96.284%	0	96.252%	13	95.931%	100%
Epinions trust	5896	0	96.086%	1	96.101%	13	96.055%	100%
Epinions	3564	0	94.975%	3	94.948%	18	94.824%	100%
Brightkite	3280	1	92.709%	2	92.790%	21	92.516%	100%
Gplus	1779	0	87.587%	12	88.351%	33	86.553%	100%
Anybeat	1148	1	86.178%	14	86.072%	49	85.390%	100%
Wiki-elec	767	1	85.440%	14	85.390%	43	83.197%	100%
Advogato	723	1	78.265%	18	77.922%	82	73.723%	100%
Facebook	515	1	77.006%	18	76.461%	222	67.446%	100%

Table 5.15: Samples: number of samples used by the RAND_H first estimation. k: number of top centralities the algorithm extracted. $\hat{k}$: number of samples added to the candidate set H. Gain: time gain in the basic algorithm (as defined in Equation 5.1). Prec.: precision achieved by the algorithm expressed as the correct number of ranked centralities over k (note that if we omitted this column for a specific k value it means that the precision was always 100%).

Network	k	**Time (s)**	**Gain**
Wikinews (en)	1	324.04	96.05%
	5	304.51	96.29%
	10	304.08	96.29%
	15	303.19	96.30%
	20	300.45	96.34%
	25	300.64	96.33%
	30	300.55	96.33%
	35	301.88	96.32%
	40	300.93	96.33%
	45	300.68	96.33%
	50	301.76	96.32%
	100	302.38	96.31%
	150	303.61	96.30%
	200	304.86	96.28%
	300	307.90	96.25%
	500	313.43	96.18%
	750	342.66	95.82%
	1000	346.75	95.77%

Table 5.16: k: number of top centralities the algorithms extracted (TOPRANK_H and solving APSP). Time: time in seconds required by the TOPRANK_H algorithm. Gain: time gain on solving APSP.

Comparison with Borassi et al.

As we can see from Table 5.17 it is clear that TOPRANK_H is worse than the Borassi et al.'s strategy for each value of k we choose. Very likely this is due to the lack of multithreading support of our implementation. However we can still verify that time gain is gradually recovered by TOPRANK_H as we consider bigger k values.

5.5.2 Second set of experiments: $\beta = 0.5$, $\alpha = 1.01$

Into this set of experiments we halved the number of random samples used by the RAND_H algorithm. As we saw in the RAND_H experiments this saves a remarkable amount of time and entails little impact on the algorithm's precision. We were encouraged to undertake this experiment since we supposed the execution of the RAND_H algorithm took the majority of the TOPRANK_H running time. In Table 5.15 we can see that the number of samples used by the approximation algorithm is much higher than candidate set size, $|H| = k + \hat{k}$. This means in that experiment RAND_H solved much more SSSPs than TOPRANK_H did.

If we consider the results reported in Table 5.18 we can conclude that, as we

Network	k	Gain	Network	k	Gain
Gowalla	1	-152753%	Wikinews (en)	1	-143302%
	10	-872.46%		10	-111309%
	100	-778.72%		100	-19974%

Network	k	Gain	Network	k	Gain
Anybeat	1	-28806%	Wiktionary (de)	1	-460.51%
	10	-8458%		10	-399.55%
	100	-3661%		100	-362.23%

Table 5.17: k: number of top centralities to extract. Gain: time gain of the TOPRANK_H algorithm on Borassi et al.

Name	Samples	$k=1$		$k=10$			$k=100$	
		$\hat{k}$	Gain	$\hat{k}$	Gain	Prec.	$\hat{k}$	Gain
Gowalla	3891	0	98.14%	2	98.16%	100%	21	98.09%
Wikinews (en)	3373	0	98.02%	4	98.01%	100%	13	97.98%
Wiktionary (de)	3155	0	98.17%	8	98.12%	100%	57	98.05%
Anybeat	575	1	91.86%	13	91.61%	95%	203	89.83%

Table 5.18: Samples: number of samples used by the RAND_H algorithm. $\hat{k}$: number of additive approximated Harmonic centralities added to the candidate set H. Gain: time gain achieved on solving APSP. Prec.: precision expressed as the number of centralities correctly computed over k.

expected, the time performances improved from the previous experiment (Table 5.16). Moreover, the overall precision did not substantially drop. This also represents an additional proof of the RAND_H's good precision even if it selects half of the random samples it is supposed to select.

Comparison with Borassi et al.

Despite the lower number of samples, Table 5.19 shows us that TOPRANK_H is still much less competitive than Borassi et al. even tough it achieved remarkable improvements if compared the previous case (Table 5.17).

5.5.3 Third set of experiments: $\beta = 0.5$, $\alpha = 1.1$

Since the TOPRANK_H algorithm should compute the exact top-k centralities with high probability, it should not have made any mistake. Since β does not seem to have significant impact on the algorithm's precision, as

Network	**k**	**Improvement ratio**	**Network**	**k**	**Improvement ratio**
	1	-76348%		1	-76532%
Gowalla	10	-378.28%	Wikinews (en)	10	-60112%
	100	-313.00%		100	-10773%
Network	**k**	**Improvement ratio**	**Network**	**k**	**Improvement ratio**
	1	-16920%		1	-187.61%
Anybeat	10	-5058%	Wiktionary (de)	10	-168.42%
	100	-2517%		100	-155.82%

Table 5.19: k: number of top centralities to extract. Gain: time gain of the TOPRANK_H algorithm on Borassi et al. . $\beta = 0.5$, $\alpha = 1.01$.

Name	**Samples**	$k = 1$		$k = 10$		$k = 100$		
		$\hat{k}$	Gain	$\hat{k}$	Gain	$\hat{k}$	Gain	Prec.
Gowalla	3891	0	98.16%	4	98.21%	25	98.17%	100%
Wikinews (en)	3373	0	98.13%	4	98.13%	11	98.02%	100%
Wiktionary (de)	3155	0	98.28%	7	98.26%	64	98.1%	100%
Anybeat	575	1	91.27%	16	91.14%	138	89.29%	100%

Table 5.20: Samples: number of samples used by the RAND_H algorithm. $\hat{k}$: number of additive approximated Harmonic centralities added to the candidate set H. Gain: time gain achieved on solving APSP. Prec.: precision expressed as the number of centralities correctly computed over k.

last set of experiments we modified the value of α to 1.1 in order to add more approximated centralities to the candidate set H.

By comparing Table 5.20 with 5.18 we can observe that, even though the α modification from 1.01 to 1.1 had very little effect on $\hat{k}$, it was enough to reach 100% precision for each network we analyzed for this purpose. On the other hand we noticed a negative but not severe impact on the running time due to the greater number of SSSP problems the algorithm has to solve (this can be verified by comparing the *Gain* columns of Tables 5.20 and 5.18).

Chapter 6

Conclusion and future work

We adapted to the Harmonic Centrality two existing randomized algorithms for both the approximation and exact computation of the Closeness Centrality of the nodes of a network. We provided the required theoretical support to prove the correctness of the approaches we developed and then we verified these achievement into a practical environment. We wrote a Python implementation of the algorithms we presented in Chapter 4 and tested them on an eighteen large benchmark networks dataset. As we expected from the theory, both the RAND_H and the TOPRANK_H algorithms required much less running time than solving the APSP problem. Furthermore, we noticed that the errors affecting Harmonic Centrality values estimated by RAND_H were much lower than the corresponding upper bound ε. We also showed that the relative errors were very low too and this encouraged us to pick less random samples to boost the RAND_H time performances without compromising the overall precision. We observed satisfying results also with half and a quarter of random samples since both the running time and the precision decreased linearly compared to the number of random samples. In our analysis RAND_H has been proven to be also an efficient top-k centrality ranker even if used with an high upper bound ($\varepsilon \leq 0.25$), especially for small values of k. We also calculated that, in many cases, the algorithm was less competitive than the Borassi et al. strategy from the running time point of view, most likely because our implementation does not support multi-threading. On the other hand, with large networks and higher k values, our RAND_H implementation still required much less time than Borassi et al..

Concerning the TOPRANK_H algorithm we discovered that choice of the α constant is crucial for the algorithm's precision. The algorithm did not always correctly ranked the top-k Harmonic centralities even though α was greater than 1. Besides a 100% precision could be achieved by increasing α by 0.09. Similarly to RAND_H, the TOPRANK_H running time was remarkably reduced by lowering the number of random samples and this did not compromise its ranking precision. Unfortunately, in our experiments this algorithm was never more competitive than Borassi et al. but we verified that, as we rise k, it gradually recovers the disadvantage.

6.1 Future developments

Our work could be improved in several different ways such as:

- Multithreading: a multi-threaded implementation could dramatically reduce the running time of the algorithm we designed.
- Graph tool library: unfortunately the graph tool library shortest path instruction does not support the computation of the shortest path from all the vertexes to a subset of vertexes. This could enhance the time performances of the RAND_H algorithm since, at the moment, it implements this requirement through a for-loop.
- TOPRANK_H time analysis: as we illustrated in Chapter 4 this algorithm's two main phases. First the Harmonic centralities are estimated through the RAND_H algorithm and then it computes the exact Harmonic Centrality of each vertex in the candidate set H. The precision and the time required by these phases can be controlled through the α and β constants. A deeper analysis of these two phases could be done in order to understand what is the optimal choice that would allow us to obtain a 100% precision by minimizing the algorithm's running time.
- Better implementation: probably a more efficient implementation of both the RAND_H and the TOPRANK_H algorithm could lower their running time.

Appendix A

Appendix

A.1 Implemented algorithms code

```
from graph_tool.all import *
import random
import constants
import sys
import math
import numpy as np

# Returns the optimal number of samples
# in order to get the required precision
def numberOfSamples(n, prec):
  if n <= 0 or prec <= 0 or prec > 1:
    return 1

  # Scientific round, return values
  # between 1 and n
  return min(n, max(1, math.ceil(0.5 + constants.samplesConstants *
    math.log(n) / pow(prec, 2))))

# Function to perform the Eppstein
# algorithm for the Harmonic centrality
def Rand_H(G, prec):

    # Number of nodes of graph G
    n = G.num_vertices()

  # In some cases (e.g. Toprank_H) we call
  # Rand_H(G, prec) function with a pre-
  # calculated number of samples
  if (prec >= 1):
    # Precision interpreted as number of samples
    l = prec
  else:
    # Precision interpreted as itself, now
    # the optimal number of samples is
    # calculated through the function
    l = numberOfSamples(n, prec)

    # List of unique random chosen vertices
    r_chosen = [G.vertex(v) for v in random.sample(range(0, n - 1),
    l)]

    # Performs SSSP from each node in v to each node in
    # the set 'r_chosed' (in accordance with the paper).

    mult_fact = (n / (l * (n - 1)))
    max_dist = G.num_vertices() + 1

    approx_harmonics = [shortest_distance(G, source=G.vertex(v),
    max_dist=max_dist).get_array() for v in r_chosen]
    transposed_h = np.transpose(approx_harmonics)

    return [(1. / dists[(dists < max_dist) * (dists > 0)]).sum() *
    mult_fact for dists in transposed_h]
```

Listing A.1: RAND_H algorithm Python code

```
from epps import Rand_H
import sys
import math
from operator import itemgetter
from exact_harmonic import harmonic
from bisect import bisect_left
import constants

# Function f(l)
def f(alpha, n, l):
  if l == 0 or n < 1:
    return 0
  return alpha * math.sqrt(math.log(n)) / l

# Function to calculate the most appropriate value for 'l'
def lcalc(n):
  if n <= 0:
    return None

  # Asymptotic value for 'l' as reported in
  # the paper
  l = constants.oka_samples_const * pow(n, 2 / 3) * pow(math.log(n),
   1 / 3)

  # Scientific rounding of 'l' because
  # 'l' must be an integer
  return math.ceil(l + 0.5)

def Toprank_H(G, k):
  # 1 if we totally trust in Rand_H
  # high if not
  alpha = constants.oka_const
  # Number of nodes in G
  n = G.num_vertices()

  # Samples for Rand_H
  l = lcalc(n)

  # Estimated harmonic centrality calculated with Eppstein and 'l'
    samples
  # Order must not be reversed in order to use bisect_left correctly
  epps_dict = Rand_H(G, l)
  epps_dict = {str(i): epps_dict[i] for i in range(0, n)}
  hs = sorted(epps_dict.items(), key=itemgetter(1))

  # Calculating h_k - 2f(l):
  threshold = hs[n - k - 1][1] - 2 * f(alpha, n, l)

  # Index of the threshold in the 'harmonics' list
  E_index = bisect_left([x[1] for x in hs], threshold)

  # Check if there is at least one centrality to select
  if E_index > n - 1:
    return {}

  # Calculating exact topk harmonic centralities
  # for each node in set E
  topK = sorted((harmonic(G, [x[0] for x in hs[E_index: n]])).items
    (), key=itemgetter(1), reverse=True)

  # Including also other nodes with harmonic
  # centrality equals to h_k
  to_k = k - 1
  while to_k < len(topK) - 1 and topK[to_k + 1] == topK[to_k]:
    to_k += 1

  # Top k harmonic centralities
  return [topK[0: to_k + 1], n - E_index - 1]
```

Listing A.2: TOPRANK_H algorithm Python code

```
from graph_tool.all import *

# Calculates exact harmonic centrality for the set of vertices S
def harmonic(G, S):
  return {v: closeness(G, source=G.vertex(v), harmonic=True, norm=
    True) for v in S}
```

Listing A.3: Harmonic Centrality exact algorithm Python code

Bibliography

[1] Python 3 documentation. https://docs.python.org/3/.

[2] Alex Bavelas. Communication patterns in task-oriented groups. *Journal of the acoustical society of America*, 1950.

[3] Paolo Boldi and Sebastiano Vigna. Axioms for centrality. *CoRR*, abs/1308.2140, 2013.

[4] Michele Borassi, Pierluigi Crescenzi, and Andrea Marino. Fast and simple computation of top-k closeness centralities. *CoRR*, abs/1507.01490, July 2015.

[5] Carter T Butts. Sna: tools for social network analysis. 2009.

[6] Colin Cooper, Alan Frieze, Kurt Mehlhorn, and Volker Priebe. Average-case complexity of shortest-paths problems in the vertex-potential model. In *International Workshop on Randomization and Approximation Techniques in Computer Science*, pages 15–26. Springer, 1997.

[7] David Eppstein and Joseph Wang. Fast approximation of centrality. In *Proceedings of the Twelfth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '01, pages 228–229, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.

[8] Robert W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962.

[9] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[10] Alan M Frieze and Geoffrey R Grimmett. The shortest-path problem for graphs with random arc-lengths. *Discrete Applied Mathematics*, 10(1):57–77, 1985.

[11] Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

[12] Donald B Johnson. Efficient algorithms for shortest paths in sparse networks. *Journal of the ACM (JACM)*, 24(1):1–13, 1977.

[13] Jérôme Kunegis. Konect: The koblenz network collection. In *Proceedings of the 22Nd International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350, New York, NY, USA, 2013. ACM.

[14] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[15] Nan Lin. *Foundations of Social Research*. McGraw-Hill, New York, 1976.

[16] Massimo Marchiori and Vito Latora. Harmony in the small-world. *Physica A: Statistical Mechanics and its Applications*, 285(3):539–546, 2000.

[17] Kurt Mehlhorn and Volker Priebe. On the all-pairs shortest-path algorithm of moffat and takaoka. *Random Structures & Algorithms*, 10(1-2):205–220, 1997.

[18] Stanley Milgram. The small world problem. *Psychology today*, 2(1):60–67, 1967.

[19] Alistair Moffat and Tadao Takaoka. An all pairs shortest path algorithm with expected time o(nˆ2\logn). *SIAM Journal on Computing*, 16(6):1023–1031, 1987.

[20] Mark EJ Newman. The structure and function of complex networks. *SIAM review*, 45(2):167–256, 2003.

[21] Kazuya Okamoto, Wei Chen, and Xiang-Yang Li. Ranking of closeness centrality for large-scale social networks. In *International Workshop on Frontiers in Algorithmics*, pages 186–195. Springer, 2008.

[22] Paul W Olsen, Alan G Labouseur, and Jeong-Hyon Hwang. Efficient top-k closeness centrality search. In *2014 IEEE 30th International Conference on Data Engineering*, pages 196–207. IEEE, 2014.

[23] Raj Kumar Pan and Jari Saramäki. Path lengths, correlations, and centrality in temporal networks. *Phys. Rev. E*, 84:016105, Jul 2011.

[24] Tiago P. Peixoto. The graph-tool python library. *figshare*, 2014.

[25] Yannick Rochat. Closeness centrality extended to unconnected graphs: The harmonic centrality index. In *ASNA*, number EPFL-CONF-200525, 2009.

[26] Ryan A. Rossi and Nesreen K. Ahmed. An interactive data repository with visual analytics. *SIGKDD Explor.*, 17(2):37–41, 2016.

[27] Christian Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: An interactive tool suite for high-performance network analysis. *CoRR*, abs/1403.3005, 2014.

[28] Duncan J Watts. *Small worlds: the dynamics of networks between order and randomness*. Princeton university press, 1999.