KADABRA is an ADaptive Algorithm for Betweenness via Random Approximation*

Michele Borassi[1] and Emanuele Natale[2]

1 **IMT Insitute for Advanced Studies, 55100 Lucca, Italy**
michele.borassi@imtlucca.it
2 **Sapienza University of Rome, 00185 Roma, Italy**
natale@di.uniroma1.it

Abstract

We present KADABRA, a new algorithm to approximate betweenness centrality in directed and undirected graphs, which significantly outperforms all previous approaches on real-world complex networks. The efficiency of the new algorithm relies on two new theoretical contributions, of independent interest.

The first contribution focuses on sampling shortest paths, a subroutine used by most algorithms that approximate betweenness centrality. We show that, on realistic random graph models, we can perform this task in time $|E|^{\frac{1}{2}+o(1)}$ with high probability, obtaining a significant speedup with respect to the $\Theta(|E|)$ worst-case performance. We experimentally show that this new technique achieves similar speedups on real-world complex networks, as well.

The second contribution is a new rigorous application of the adaptive sampling technique. This approach decreases the total number of shortest paths that need to be sampled to compute all betweenness centralities with a given absolute error, and it also handles more general problems, such as computing the k most central nodes. Furthermore, our analysis is general, and it might be extended to other settings, as well.

1998 ACM Subject Classification G.2.2 [Discrete Mathematics] Graph Theory, Graph algorithms

Keywords and phrases Betweenness centrality, shortest path algorithm, graph mining, sampling, network analysis

Digital Object Identifier 10.4230/LIPIcs.ESA.2016.20

1 Introduction

In this work we focus on estimating the *betweenness centrality*, which is one of the most famous measures of *centrality* for nodes and edges of real-world complex networks [19, 30]. The rigorous definition of betweenness centrality has its roots in sociology, dating back to the Seventies, when Freeman formalized the informal concept discussed in the previous decades in different scientific communities [6, 40, 39, 17, 13], although the definition already appeared in [4]. Since then, this notion has been very successful in network science [44, 31, 22, 30].

A probabilistic way to define the betweenness centrality[1] $\mathrm{bc}(v)$ of a node v in a graph $G = (V, E)$ is the following. We choose two nodes s and t, and we go from s to t through a shortest path π; if the choices of s, t and π are made uniformly at random, the betweenness centrality of a node v is the probability that we pass through v.

* This work was done while the authors were visiting the Simons Institute for the Theory of Computing.
[1] As explained in Section 2, to simplify notation we consider the *normalized* betweenness centrality.

In a seminal paper [14], Brandes showed that it is possible to exactly compute the betweenness centrality of all the nodes in a graph in time $\mathcal{O}(mn)$, where n is the number of nodes and m is the number of edges. A corresponding lower bound was proved in [10]: if we are able to compute the betweenness centrality of a single node in time $\mathcal{O}(mn^{1-\epsilon})$ for some $\epsilon > 0$, then the Strong Exponential Time Hypothesis [23] is false.

This result further motivates the rich line of research on computing approximations of betweenness centrality, with the goal of trading precision with efficiency. The main idea is to define a probability distribution over the set of all paths, by choosing two uniformly random nodes s, t, and then a uniformly distributed st-path $\boldsymbol{\pi}$, so that $\Pr(v \in \boldsymbol{\pi}) = \mathrm{bc}(v)$. As a consequence, we can approximate $\mathrm{bc}(v)$ by sampling paths $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\tau$ according to this distribution, and estimating $\tilde{\boldsymbol{b}}(v) := \frac{1}{\tau} \sum_{i=1}^{\tau} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v) = 1$ if $v \in \boldsymbol{\pi}_i$ (and $v \neq s, t$), 0 otherwise.

The tricky part of this approach is to provide probabilistic guarantees on the quality of this approximation: the goal is to obtain a $1 - \delta$ confidence interval $\boldsymbol{I}(v) = [\tilde{\boldsymbol{b}}(v) - \lambda_L, \tilde{\boldsymbol{b}}(v) + \lambda_U]$ for $\mathrm{bc}(v)$, which means that $\Pr(\forall v \in V, \mathrm{bc}(v) \in \boldsymbol{I}(v)) \geq 1 - \delta$. Thus, the research for approximating betweenness centrality has been focusing on obtaining, as fast as possible, the smallest possible $\boldsymbol{I}$.

Our Contribution

In this work, we propose a new and faster algorithm to approximate betweenness centrality in directed and undirected graphs, named KADABRA. In the standard task of approximating betweenness centralities with absolute error at most λ, we show that, on average, the new algorithm is more than 100 times faster than the previous ones, on graphs with approximately 10 000 nodes. Moreover, differently from previous approaches, our algorithm can perform more general tasks, since it does not need all confidence intervals to be equal. As an example, we consider the computation of the k most central nodes: all previous approaches compute all centralities with an error λ, and use this approximation to obtain the ranking. Conversely, our approach allows us to use small confidence interval only when they are needed, and allows bigger confidence intervals for nodes whose centrality values are "well separated". This way, we can compute for the first time an approximation of the k most central nodes in networks with millions of nodes and hundreds of millions of edges, like the Wikipedia citation network and the IMDB actor collaboration network.

Our results rely on two main theoretical contributions, which are interesting in their own right, since their generality naturally extends to other applications.

Balanced bidirectional breadth-first search. By leveraging on recent advanced results, we prove that, on many realistic random models of real-world complex networks, it is possible to sample a random path between two nodes s and t in time $m^{\frac{1}{2}+o(1)}$ if the degree distribution has finite second moment, or $m^{\frac{4-\beta}{2}+o(1)}$ if the degree distribution is power law with exponent $2 < \beta < 3$. The models considered are the Configuration Model [9], and all Rank-1 Inhomogeneous Random Graph models [42, Chapter 3], such as the Chung-Lu model [29], the Norros-Reittu model [32], and the Generalized Random Graph [42, Chapter 3]. Our proof techniques have the merit of adopting a unified approach that simultaneously works in all models considered. These models well represent metric properties of real-world networks [11]: indeed, our results are confirmed by practical experiments.

The algorithm used is simply a balanced bidirectional BFS (bb-BFS): we perform a BFS from each of the two endpoints s and t, in such a way that the two BFSs are likely to explore about the same number of edges, and we stop as soon as the two BFSs "touch

each other". Rather surprisingly, this technique was never implemented to approximate betweenness centrality, and it is rarely used in the experimental algorithm community. Our theoretical analysis provides a clear explanation of the reason why this technique improves over the standard BFS: this means that many state-of-the-art algorithm for real-world complex networks can be improved by the bb-BFS.

Adaptive sampling made rigorous. To speed up the estimation of the betweenness centrality, previous work make use of the technique of adaptive sampling, which consists in testing during the execution of the algorithm whether some condition on the sample obtained so far has been met, and terminating the execution of the algorithm as soon as this happens. However, this technique introduces a subtle stochastic dependence between the time in which the algorithm terminates and the correctness of the given output, which previous papers claiming a formal analysis of the technique did not realize (see Section 3 for details). With an argument based on martingale theory, we provide a general analysis of such useful technique. Through this result, we do not only improve previous estimators, but we also make it possible to define more general stopping conditions, that can be decided "on the fly": this way, with little modifications, we can adapt our algorithm to perform more general tasks than previous ones.

To better illustrate the power of our techniques, we focus on the unweighted, static graphs, and to the centrality of nodes. However, our algorithm can be easily adapted to compute the centrality of edges, to handle weighted graphs and, since its core part consists merely in sampling paths, we conjecture that it may be coupled with the existing techniques in [8] to handle dynamic graphs.

Related Work

Computing Betweenness Centrality. With the recent event of big data, the major shortcoming of betweenness centrality has been the lack of efficient methods to compute it [14]. In the worst case, the best exact algorithm to compute the centrality of all the nodes is due to Brandes [14], and its time complexity is $\mathcal{O}(mn)$: the basic idea of the algorithm is to define the dependency $\delta_s(v) = \sum_{t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$, which can be computed in time $\mathcal{O}(m)$, for each $v \in V$ (we denote by $\sigma_{st}(v)$ the number of shortest paths from s to t passing through v, and by σ_{st} the number of st-shortest paths). In [10], it is also shown that Brandes algorithm is almost optimal on sparse graphs: an algorithm that computes the betweenness centrality of a single vertex in time $\mathcal{O}(mn^{1-\epsilon})$ falsifies widely believed complexity assumptions, such as the Strong Exponential Time Hypothesis [23], the Orthogonal Vector conjecture [2], or the Hitting Set conjecture [45]. Corresponding results in the dense, weighted case are available in [1]: computing the betweenness centrality exactly is as hard as computing the All Pairs Shortest Path, and computing an approximation with a given relative error is as hard as computing the diameter. For both these problems, there is no algorithm with running-time $\mathcal{O}(n^{3-\epsilon})$, for any $\epsilon > 0$. This shows that, for dense graphs, having an additive approximation rather than a multiplicative one is essential for a provably fast algorithm to exist. These negative results further motivates the already rich line of research on approaches that overcome this barrier. A first possibility is to use heuristics, that do not provide analytical guarantees on their performance [38, 21, 43]. Another line of research has defined variants of betweenness centrality, that might be easier to compute [15, 33, 18]. Finally, a third line of research has investigated approximation algorithms, which trade accuracy for speed [24, 16, 22, 26]. Our work follows the latter approach. The first approximation algorithm proposed in the literature [24] adapts Eppstein and Wang's approach for computing closeness centrality [20],

using Hoeffding's inequality and the union bound technique. This way, it is possible to obtain an estimate of the betweenness centrality of every node that is correct up to an additive error λ with probability δ, by sampling $\mathcal{O}(\frac{D^2}{\lambda^2}\log\frac{n}{\delta})$ nodes, where D is the diameter of the graph. In [22], it is shown that this can lead to an overestimation. Riondato and Kornaropoulos improve this sampling-based approach by sampling single shortest paths instead of the whole dependency of a node [36], introducing the use of the VC-dimension. As a result, the number of samples is decreased to $\frac{c}{\lambda^2}(\lfloor\log_2(\mathrm{VD}-2)\rfloor+1+\log(\frac{1}{\delta}))$, where VD is the vertex diameter, that is, the minimum number of nodes in a shortest path in G (it can be different from $D+1$ if the graph is weighted). This use of the VC-dimension is further developed and generalized in [37]. Finally, many of these results were adapted to handle dynamic networks [8, 37].

Approximating the top-k betweenness centrality set. Let us order the nodes $v_1, ..., v_n$ such that $\mathrm{bc}(v_1) \geq ... \geq \mathrm{bc}(v_n)$ and define $TOP(k) = \{(v_i, \mathrm{bc}(v_i)) : i \leq k\}$. In [36] and [37], the authors provide an algorithm that, for any given δ, ϵ, with probability $1-\delta$ outputs a set $\widetilde{TOP}(k) = \{(v_i, \tilde{\boldsymbol{b}}(v_i))\}$ such that: i) If $v \in TOP(k)$ then $v \in \widetilde{TOP}(k)$ and $|\,\mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v)| \leq \epsilon\,\mathrm{bc}(v)$; ii) If $v \in \widetilde{TOP}(k)$ but $v \notin TOP(k)$ then $\tilde{\boldsymbol{b}}(v) \leq (\mathbf{b}_k - \epsilon)(1+\epsilon)$ where $\mathbf{b}_k$ is the k-th largest betweenness given by a preliminary phase of the algorithm.

Adaptive sampling. In [5, 37], the number of samples required is substantially reduced using the adaptive sampling technique introduced by Lipton and Naughton in [28, 27]. Let us clarify that, by adaptive sampling, we mean that the termination of the sampling process depends on the sample observed so far (in other cases, the same expression refers to the fact that the distribution of the new samples is a function of the previous ones [3], while the sample size is fixed in advance). Except for [34], previous approaches tacitly assume that there is little dependency between the stopping time and the correctness of the output: indeed, they prove that, for each *fixed* τ, the probability that the estimate is wrong at time τ is below δ. However, the stopping time $\boldsymbol{\tau}$ is a random variable, and in principle there might be dependency between the event $\boldsymbol{\tau} = \tau$ and the event that the estimate is correct at time τ. As for [34], they consider a specific stopping condition and their proof technique does not seem to extend to other settings. For a more thorough discussion of this issue, we defer the reader to Section 3.

Bidirectional BFS. The possibility of speeding up a breadth-first search for the shortest-path problem by performing, at the same time, a BFS from the final end-point, has been considered since the Seventies [35]. Unfortunately, because of the lack of theoretical results dealing with its efficiency, the bidirectional BFS has apparently not been considered a fundamental heuristic improvement [25]. However, in [36] (and in some public talks by M. Riondato), the bidirectional BFS was proposed as a possible way to improve the performance of betweenness centrality approximation algorithms.

Structure of the Paper

In Section 2, we describe our algorithm, and in Section 3 we discuss the main difficulty of the adaptive sampling, and the reasons why our techniques are not affected. In Section 4, we define the balanced bidirectional BFS, and we sketch the proof of its efficiency on random graphs. In Section 5, we show that our algorithm can be adapted to compute the k most central nodes. In Section 6 we experimentally show the effectiveness of our new algorithm. Finally, all our proofs are in the appendix.

2 Algorithm Overview

To simplify notation, we always consider the *normalized* betweenness centrality of a node v, which is defined by:

$$\mathrm{bc}(v) = \frac{1}{n(n-1)} \sum_{s \neq v \neq t} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

where σ_{st} is the number of shortest paths between s and t, and $\sigma_{st}(v)$ is the number of shortest paths between s and t that pass through v. Furthermore, to simplify the exposition, we use bold symbols to denote random variables, and light symbols to denote deterministic quantities. On the same line of previous works, our algorithm samples random paths $\boldsymbol{\pi}_1, \ldots, \boldsymbol{\pi}_\tau$, where $\boldsymbol{\pi}_i$ is chosen by selecting uniformly at random two nodes s, t, and then selecting uniformly at random one of the shortest paths from s to t. Then, it estimates $\mathrm{bc}(v)$ with $\tilde{\boldsymbol{b}}(v) := \frac{1}{\tau} \sum_{i=1}^{\tau} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v) = 1$ if $v \in \boldsymbol{\pi}_i$, 0 otherwise. By definition of $\boldsymbol{\pi}_i$, $\mathbb{E}\left[\tilde{\boldsymbol{b}}(v)\right] = \mathrm{bc}(v)$.

The tricky part is to bound the distance between $\tilde{\boldsymbol{b}}(v)$ and its expected value. With a straightforward application of Hoeffding's inequality (see Appendix B of the full version [12]), it is possible to prove that $\Pr\left(\left|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)\right| \geq \lambda\right) \leq 2e^{-2\tau\lambda^2}$. A direct application of this inequality considers a union bound on all possible nodes v, obtaining $\Pr(\exists v \in V, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \geq \lambda) \leq 2ne^{-2\tau\lambda^2}$. This means that the algorithm can safely stop as soon as $2ne^{-2\tau\lambda^2} \leq \delta$, that is, after $\tau = \frac{1}{2\lambda^2} \log(\frac{2n}{\delta})$ steps.

In order to improve this idea, we can start from the Chernoff bound (see Appendix B of the full version [12]), instead of Hoeffding inequality, obtaining that $\Pr\left(\left|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)\right| \geq \lambda\right) \leq 2\exp(-\frac{\tau\lambda^2}{2(\mathrm{bc}(v)+\lambda/3)})$.

If we assume the error λ to be small, this inequality is stronger than the previous one for all values of $\mathrm{bc}(v) < \frac{1}{4}$ (a condition which holds for almost all nodes, in almost all graphs considered). However, in order to apply this inequality, we have to deal with the fact that we do not know $\mathrm{bc}(v)$ in advance, and hence we do not know when to stop. Intuitively, to solve this problem, we make a "change of variable", and we rewrite the previous inequality as

$$\Pr\left(\mathrm{bc}(v) \leq \tilde{\boldsymbol{b}}(v) - f\right) \leq \delta_L^{(v)} \quad \text{and} \quad \Pr\left(\mathrm{bc}(v) \geq \tilde{\boldsymbol{b}}(v) + g\right) \leq \delta_U^{(v)}, \tag{1}$$

for some functions $f = f(\tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \tau), g = g(\tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \tau)$. Our algorithm fixes at the beginning the values $\delta_L^{(v)}, \delta_U^{(v)}$ for each node v, and, at each step, it tests if $f(\tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \tau)$ and $g(\tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \tau)$ are small enough. If this condition is satisfied, the algorithm stops. Note that this approach lets us define very general stopping conditions, that might depend on the centralities computed until now, on the single nodes, and so on.

▶ Remark. Instead of fixing the values $\delta_L^{(v)}, \delta_U^{(v)}$ at the beginning, one might want to decide them during the algorithm, depending on the outcome. However, this is not formally correct, because of dependency issues (for example, (1) does not even make sense, if $\delta_L^{(v)}, \delta_U^{(v)}$ are random). Finding a way to overcome this issue is left as a challenging open problem (more details are provided in Section 3).

In order to implement this idea, we still need to solve an issue: (1) holds for each *fixed* time τ, but the stopping time of our algorithm is a random variable $\boldsymbol{\tau}$, and there might be dependency between the value of $\boldsymbol{\tau}$ and the probability in (1). To this purpose, we use Mcdiarmid's inequality (see Appendix B of the full version [12]), that holds even if $\boldsymbol{\tau}$ is a random variable. However, to use this inequality, we need to assume that $\boldsymbol{\tau} < \omega$ for some deterministic ω: in our algorithm, we choose $\omega = \frac{c}{\lambda^2}\left(\lfloor \log_2(\mathrm{VD} - 2) \rfloor + 1 + \log\left(\frac{2}{\delta}\right)\right)$, because,

by the results in [36], after ω samples, the maximum error is at most λ, with probability $1 - \frac{\delta}{2}$. Furthermore, also f and g should be modified, since they now depend on the value of ω. The pseudocode of the algorithm obtained is available in Algorithm 1 (as was done in previous approaches, we can easily parallelize the while loop in Line 5).

Algorithm 1: our algorithm for approximating betweenness centrality.

Input : a graph $G = (V, E)$
Output: for each $v \in V$, an approximation $\tilde{\boldsymbol{b}}(v)$ of $\mathrm{bc}(v)$ such that $\Pr\left(\forall v, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \leq \lambda\right) \geq 1 - \delta$

1 $\omega \leftarrow \frac{c}{\lambda^2}\left(\lfloor \log_2(\mathrm{VD} - 2) \rfloor + 1 + \log\left(\frac{2}{\delta}\right)\right)$;
2 $(\delta_L^{(v)}, \delta_U^{(v)}) \leftarrow$ `computeDelta`();
3 $\tau \leftarrow 0$;
4 **foreach** $v \in V$ **do** $\tilde{\boldsymbol{b}}(v) \leftarrow 0$;
5 **while** $\tau < \omega$ *and not* `haveToStop` $(\tilde{\boldsymbol{b}}, \delta_L, \delta_U, \omega, \tau)$ **do**
6 $\quad \boldsymbol{\pi} =$ `samplePath`();
7 $\quad$ **foreach** $v \in \boldsymbol{\pi}$ **do** $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v) + 1$;
8 $\quad \tau \leftarrow \tau + 1$;
9 **end**
10 **foreach** $v \in V$ **do** $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v)/\tau$;
11 **return** $\tilde{\boldsymbol{b}}$

Algorithm 2: the function `computeDelta`.

Input : a graph $G = (V, E)$, and two values $\lambda_L^{(v)}, \lambda_U^{(v)}$ for each $v \in V$
Output: for each $v \in V$, two values $\delta_L^{(v)}, \delta_U^{(v)}$

1 $\alpha \leftarrow \frac{\omega}{100}$;
2 $\epsilon \leftarrow 0.0001$;
3 **foreach** $i \in [1, \alpha]$ **do**
4 $\quad \boldsymbol{\pi} =$ `samplePath`();
5 $\quad$ **foreach** $v \in \boldsymbol{\pi}$ **do** $\tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v) + 1$;
6 **end**
7 **foreach** $v \in V$ **do**
8 $\quad \tilde{\boldsymbol{b}}(v) \leftarrow \tilde{\boldsymbol{b}}(v)/\alpha$;
9 $\quad c_L(v) \leftarrow \frac{2\tilde{\boldsymbol{b}}(v)\omega}{(\lambda_L^{(v)})^2}$;
10 $\quad c_U(v) \leftarrow \frac{2\tilde{\boldsymbol{b}}(v)\omega}{(\lambda_U^{(v)})^2}$;
11 **end**
12 Binary search to find C such that $\sum_{v \in V} \exp\left(-\frac{C}{c_L(v)}\right) + \exp\left(-\frac{C}{c_U(v)}\right) = \frac{\delta}{2} - \epsilon\delta$;
13 **foreach** $v \in V$ **do**
14 $\quad \delta_L^{(v)} \leftarrow \exp\left(-\frac{C}{c_L(v)}\right) + \frac{\epsilon\delta}{2n}$;
15 $\quad \delta_U^{(v)} \leftarrow \exp\left(-\frac{C}{c_U(v)}\right) + \frac{\epsilon\delta}{2n}$;
16 **end**
17 **return** $\boldsymbol{b}$;

The correctness of the algorithm follows from the following theorem, which is the base of our adaptive sampling, and which we prove in Section 2.1 (where we also define the functions f and g).

▶ **Theorem 1.** *Let $\tilde{\boldsymbol{b}}(v)$ be the output of Algorithm 1, and let $\boldsymbol{\tau}$ be the number of samples at the end of the algorithm. Then, with probability $1 - \delta$, the following conditions hold:*

- *if $\boldsymbol{\tau} = \omega$, $|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| < \lambda$ for all v;*
- *if $\boldsymbol{\tau} < \omega$, $-f(\boldsymbol{\tau}, \tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \omega) \leq \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \leq g(\boldsymbol{\tau}, \tilde{\boldsymbol{b}}(v), \delta_U^{(v)}, \omega)$ for all v.*

Algorithm 3: The function `haveToStop` to compute the top-k nodes.

Input : for each node v, the values of $\tilde{\boldsymbol{b}}(v), \delta_L^{(v)}, \delta_U^{(v)}$, and the values of ω and τ
Output : True if the algorithm should stop, False otherwise

Sort nodes in decreasing order of $\tilde{\boldsymbol{b}}(v)$, obtaining $v_1, \dots, v_n$;
for $i \in [1, \dots, k]$ **do**
  if $f(\tilde{\boldsymbol{b}}(v_i), \delta_L^{(v_i)}, \omega, \tau) > \lambda$ *or* $g(\tilde{\boldsymbol{b}}(v_i), \delta_U^{(v_i)}, \omega, \tau) > \lambda$ **then**
    if $\tilde{\boldsymbol{b}}(v_{i-1}) - f(\tilde{\boldsymbol{b}}(v_{i-1}), \delta_L^{(v_{i-1})}, \omega, \tau) < \tilde{\boldsymbol{b}}(v_i) + g(\tilde{\boldsymbol{b}}(v_i), \delta_U^{(v_i)}, \omega, \tau)$ *or* $\tilde{\boldsymbol{b}}(v_i) - f(\tilde{\boldsymbol{b}}(v_i), \delta_L^{(v_i)}, \omega, \tau) < \tilde{\boldsymbol{b}}(v_{i+1}) + g(\tilde{\boldsymbol{b}}(v_{i+1}), \delta_U^{(v_{i+1})}, \omega, \tau)$ **then**
      return *False*;
    end
  end
end
for $i \in [k+1, \dots, n]$ **do**
  if $f(\tilde{\boldsymbol{b}}(v_i), \delta_L^{(v_i)}, \omega, \tau) > \lambda$ *or* $g(\tilde{\boldsymbol{b}}(v_i), \delta_U^{(v_i)}, \omega, \tau) > \lambda$ **then**
    if $\tilde{\boldsymbol{b}}(v_k) - f(\tilde{\boldsymbol{b}}(v_k), \delta_L^{(v_k)}, \omega, \tau) < \tilde{\boldsymbol{b}}(v_i) + g(\tilde{\boldsymbol{b}}(v_i), \delta_U^{(v_i)}, \omega, \tau)$ **then**
      return *False*;
    end
  end
end
return *True*;

▶ Remark. This theorem says that, at the beginning of the algorithm, we know that, with probability $1-\delta$, one of the two conditions will hold when the algorithm stops, independently of the final value of $\boldsymbol{\tau}$. This is essential to avoid the stochastic dependence that we discuss in Section 3.

In order to apply this theorem, we choose λ such that our goal is reached if all centralities are known with error at most λ. Then, we choose the function `haveToStop` in a way that our goal is reached if the stopping condition is satisfied. This way, our algorithm is correct, both if $\boldsymbol{\tau} = \omega$ and if $\boldsymbol{\tau} < \omega$. For example, if we want to compute all centralities with bounded absolute error, we simply choose λ as the bound we want to achieve, and we plug the stopping condition $f, g \leq \lambda$ in the function `haveToStop`. Instead, if we want to compute an approximation of the k most central nodes, we need a different definition of f and g, which is provided in Section 5.

To complete the description of this algorithm, we need to specify the following functions.

`computeDelta`: The algorithm works for any choice of the $\delta_L^{(v)}, \delta_U^{(v)}$s, but a good choice yields better running times. We adopt the heuristic given in Algorithm 2, which we discuss in Appendix D of the full version.

`samplePath`: In order to sample a path between two random nodes s and t, we use a balanced bidirectional BFS (see Appendix E of the full version [12] for a detailed description).

2.1 Proof of Theorem 1

In our algorithm, we sample $\boldsymbol{\tau}$ shortest paths $\boldsymbol{\pi}_i$, where $\boldsymbol{\tau}$ is a random variable such that $\boldsymbol{\tau} = \tau$ can be decided by looking at the first τ paths sampled (see Algorithm 1). Furthermore, thanks to Eq. (3) in [36], we assume that $\boldsymbol{\tau} \leq \omega$ for some fixed $\omega \in \mathbb{R}^+$ such that, after ω steps, $\Pr(\forall v, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| \leq \lambda) \geq 1 - \frac{\delta}{2}$. When the algorithm stops, our estimate of the betweenness is $\tilde{\boldsymbol{b}}(v) := \frac{1}{\boldsymbol{\tau}} \sum_{i=1}^{\boldsymbol{\tau}} \boldsymbol{X}_i(v)$, where $\boldsymbol{X}_i(v)$ is 1 if v belongs to $\boldsymbol{\pi}_i$, 0 otherwise.

To estimate the error, we use the following theorem.

▶ **Theorem 2.** *For each node v and for every fixed real numbers δ_L, δ_U, it holds*

$$\Pr\left(\mathrm{bc}(v) \le \tilde{\boldsymbol{b}}(v) - f\left(\tilde{\boldsymbol{b}}(v), \delta_L, \omega, \boldsymbol{\tau}\right)\right) \le \delta_L \quad \textit{and}$$
$$\Pr\left(\mathrm{bc}(v) \ge \tilde{\boldsymbol{b}}(v) + g\left(\tilde{\boldsymbol{b}}(v), \delta_U, \omega, \boldsymbol{\tau}\right)\right) \le \delta_U,$$

where

$$f\left(\tilde{\boldsymbol{b}}(v), \delta_L, \omega, \boldsymbol{\tau}\right) = \frac{1}{\boldsymbol{\tau}} \log \frac{1}{\delta_L} \left(\frac{1}{3} - \frac{\omega}{\boldsymbol{\tau}} + \sqrt{\left(\frac{1}{3} - \frac{\omega}{\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}(v)\omega}{\log \frac{1}{\delta_L}}} \right) \quad \textit{and} \tag{2}$$

$$g\left(\tilde{\boldsymbol{b}}(v), \delta_U, \omega, \boldsymbol{\tau}\right) = \frac{1}{\boldsymbol{\tau}} \log \frac{1}{\delta_U} \left(\frac{1}{3} + \frac{\omega}{\boldsymbol{\tau}} + \sqrt{\left(\frac{1}{3} + \frac{\omega}{\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}(v)\omega}{\log \frac{1}{\delta_U}}} \right). \tag{3}$$

Before proving Theorem 2, let us see how this theorem implies Theorem 1. To simplify notation, we often omit the arguments of the function f and g.

Proof of Theorem 1. Let $\boldsymbol{E}_1$ be the event $(\boldsymbol{\tau} = \omega \wedge \exists v \in V, |\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)| > \lambda)$, and let $\boldsymbol{E}_2$ be the event $(\boldsymbol{\tau} < \omega \wedge (\exists v \in V, -f \ge \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \vee \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \ge g))$. Let us also denote $\tilde{\boldsymbol{b}}_\tau(v) = \frac{1}{\tau}\sum_{i=1}^{\tau} \boldsymbol{X}_i(v)$ (note that $\tilde{\boldsymbol{b}}_{\boldsymbol{\tau}}(v) = \tilde{\boldsymbol{b}}(v)$).

By our choice of ω and Eq. (3) in [36],

$$\Pr(\boldsymbol{E}_1) \le \Pr(\exists v \in V, |\tilde{\boldsymbol{b}}_\omega(v) - \mathrm{bc}(v)| > \lambda) \le \frac{\delta}{2}$$

where $\tilde{\boldsymbol{b}}_\omega(v)$ is the approximate betweenness of v after ω samples. Furthermore, by Theorem 2,

$$\begin{aligned}\Pr(\boldsymbol{E}_2) &\le \sum_{v \in V} \Pr(\boldsymbol{\tau} < \omega \wedge -f \ge \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v)) + \Pr(\boldsymbol{\tau} < \omega \wedge \mathrm{bc}(v) - \tilde{\boldsymbol{b}}(v) \le g) \\ &\le \sum_{v \in V} \delta_L^{(v)} + \delta_U^{(v)} \le \frac{\delta}{2}.\end{aligned}$$

By a union bound, $\Pr(\boldsymbol{E}_1 \vee \boldsymbol{E}_2) \le \Pr(\boldsymbol{E}_1) + \Pr(\boldsymbol{E}_1) \le \delta$, concluding the proof of Theorem 1. ◀

Thus, it remains to prove Theorem 2.

Proof of Theorem 2. Since this theorem deals with a single node v, let us simply write $\mathrm{bc} = \mathrm{bc}(v), \tilde{\boldsymbol{b}} = \tilde{\boldsymbol{b}}(v), \boldsymbol{X}_i = \boldsymbol{X}_i(v)$. Let us consider $\boldsymbol{Y}^\tau = \sum_{i=1}^{\tau} (\boldsymbol{X}_i - \mathrm{bc})$ (we recall that $\boldsymbol{X}_i = 1$ if v is in the i-th path sampled, $\boldsymbol{X}_i = 0$ otherwise). Clearly, $\boldsymbol{Y}^\tau$ is a martingale, and $\boldsymbol{\tau}$ is a stopping time for $\boldsymbol{Y}^\tau$: this means that also $\boldsymbol{Z}^\tau = \boldsymbol{Y}^{\min(\boldsymbol{\tau}, \tau)}$ is a martingale.

Let us apply Mcdiarmid's inequality (see e.g. Theorem 8 in Appendix B of the full version [12]) to the martingales $\boldsymbol{Z}$ and $-\boldsymbol{Z}$: for each fixed $\lambda_L, \lambda_U > 0$ we have

$$\Pr\left(\boldsymbol{Z}^\omega \ge \lambda_L\right) = \Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \ge \lambda_L\right) \le \exp\left(-\frac{\lambda_L^2}{2\left(\omega\,\mathrm{bc} + \lambda_L/3\right)}\right) = \delta_L \quad \text{and} \tag{4}$$

$$\Pr\left(-\boldsymbol{Z}^\omega \ge \lambda_U\right) = \Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \le -\lambda_U\right) \le \exp\left(-\frac{\lambda_U^2}{2\left(\omega\,\mathrm{bc} + \lambda_U/3\right)}\right) = \delta_U. \tag{5}$$

We now show how to prove (2) from (4). The way to derive (3) from (5) is analogous.

If we express λ_L as a function of δ_L we get

$$\lambda_L^2 = 2\log\frac{1}{\delta_L}\left(\omega\,\mathrm{bc} + \frac{\lambda_L}{3}\right) \iff \lambda_L^2 - \frac{2}{3}\lambda_L \log\frac{1}{\delta_L} - 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L} = 0,$$

which implies that

$$\lambda_L = \frac{1}{3}\log\frac{1}{\delta_L} \pm \sqrt{\frac{1}{9}\left(\log\frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L}}.$$

Since (4) holds for any positive value λ_L, it also holds for the value corresponding to the positive solution of this equation, that is,

$$\lambda_L = \frac{1}{3}\log\frac{1}{\delta_L} + \sqrt{\frac{1}{9}\left(\log\frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L}}.$$

Plugging this value into (4), we obtain

$$\Pr\left(\boldsymbol{\tau}\tilde{\boldsymbol{b}} - \boldsymbol{\tau}\,\mathrm{bc} \geq \frac{1}{3}\log\frac{1}{\delta_L} + \sqrt{\frac{1}{9}\left(\log\frac{1}{\delta_L}\right)^2 + 2\omega\,\mathrm{bc}\log\frac{1}{\delta_L}}\right) \leq \delta_L. \tag{6}$$

By assuming $\tilde{\boldsymbol{b}} - \mathrm{bc} \geq \frac{1}{3\boldsymbol{\tau}}\log(\frac{1}{\delta_L})$, the event in (6) can be rewritten as

$$(\boldsymbol{\tau}\,\mathrm{bc})^2 - 2\,\mathrm{bc}\left(\boldsymbol{\tau}^2\tilde{\boldsymbol{b}} + \omega\log\frac{1}{\delta_L} - \frac{1}{3}\boldsymbol{\tau}\log\frac{1}{\delta_L}\right) - \frac{2}{3}\log\frac{1}{\delta_L}\boldsymbol{\tau}\tilde{\boldsymbol{b}} + \left(\boldsymbol{\tau}\tilde{\boldsymbol{b}}\right)^2 \geq 0.$$

By solving the previous quadratic equation w.r.t. bc we get

$$\mathrm{bc} \leq \tilde{\boldsymbol{b}} + \log\frac{1}{\delta_L}\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}} - \sqrt{\left(\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}} + \frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}}\right)^2 - \left(\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}}\right)^2 + \frac{2}{3\boldsymbol{\tau}}\frac{\tilde{\boldsymbol{b}}}{\log\frac{1}{\delta_L}}}\right),$$

where we only considered the solution which upper bounds bc, since we assumed $\tilde{\boldsymbol{b}} - \mathrm{bc} \geq \frac{1}{3\tau}\log(\frac{1}{\delta_L})$. After simplifying the terms under the square root in the previous expression, we get

$$\mathrm{bc} \leq \tilde{\boldsymbol{b}} + \log\frac{1}{\delta_L}\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}} - \sqrt{\left(\frac{\omega}{\boldsymbol{\tau}^2} - \frac{1}{3\boldsymbol{\tau}}\right)^2 + \frac{2\tilde{\boldsymbol{b}}\omega}{\boldsymbol{\tau}^2\log\frac{1}{\delta_L}}}\right),$$

which means that

$$\Pr\left(\mathrm{bc} \leq \tilde{\boldsymbol{b}} - f\left(\tilde{\boldsymbol{b}}, \delta_L, \omega, \boldsymbol{\tau}\right)\right) \leq \delta_L,$$

concluding the proof. ◀

3 Adaptive Sampling

In this section, we highlight the main technical difficulty in the formalization of adaptive sampling, which previous works claiming analogous results did not address. Furthermore, we sketch the way we overcome this difficulty: our argument is quite general, and it could be easily adapted to formalize these claims.

As already said, the problem is the stochastic dependence between the time $\boldsymbol{\tau}$ in which the algorithm terminates and the event $\boldsymbol{A}_\tau$ = "at time τ, the estimate is within the required distance from the true value", since both $\boldsymbol{\tau}$ and $\boldsymbol{A}_\tau$ are functions of the same random sample. Since it is typically possible to prove that $\Pr(\neg\boldsymbol{A}_\tau) \leq \delta$ for every fixed τ, one may be tempted

to argue that also $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}}) \leq \delta$, by applying these inequalities at time $\boldsymbol{\tau}$. However, this is not correct: indeed, if we have no assumptions on $\boldsymbol{\tau}$, $\boldsymbol{\tau}$ could even be defined as the smallest τ such that $\boldsymbol{A}_{\tau}$ does not hold!

More formally, if we want to link $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}})$ to $\Pr(\neg \boldsymbol{A}_{\tau})$, we have to use the law of total probability, that says that:

$$\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}}) = \sum_{\tau=1}^{\infty} \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} = \tau) \Pr(\boldsymbol{\tau} = \tau) \tag{7}$$

$$= \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} < \tau) \Pr(\boldsymbol{\tau} < \tau) + \Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} \geq \tau) \Pr(\boldsymbol{\tau} \geq \tau). \tag{8}$$

Then, if we want to bound $\Pr(\neg \boldsymbol{A}_{\boldsymbol{\tau}})$, we need to assume that

$$\Pr(\neg A_{\boldsymbol{\tau}} \mid \boldsymbol{\tau} = \tau) \leq \Pr(\neg A_{\tau}) \quad \text{or that} \quad \Pr(\neg A_{\tau} \mid \boldsymbol{\tau} \geq \tau) \leq \Pr(\neg A_{\tau}), \tag{9}$$

which would allow to bound (7) or (8) from above. The equations in (9) are implicitly assumed to be true in previous works adopting adaptive sampling techniques. Unfortunately, because of the stochastic dependence, it is quite difficult to prove such inequalities, even if some approaches managed to overcome these difficulties [34].

For this reason, our proofs avoid dealing with such relations: in the proof of Theorem 1, we fix a deterministic time ω, we impose that $\boldsymbol{\tau} \leq \omega$, and we apply the inequalities with $\tau = \omega$. Then, using martingale theory, we convert results that hold at time ω to results that hold at the stopping time $\boldsymbol{\tau}$ (see Section 2.1).

4 Balanced Bidirectional BFS

A major improvement of our algorithm, with respect to previous counterparts, is that we sample shortest paths through a balanced bidirectional BFS, instead of a standard BFS. In this section, we describe this technique, and we bound its running time on realistic models of random graphs, with high probability. The idea behind this technique is very simple: if we need to sample a uniformly random shortest path from s to t, instead of performing a full BFS from s until we reach t, we perform at the same time a BFS from s and a BFS from t, until the two BFSs touch each other (if the graph is directed, we perform a "forward" BFS from s and a "backward" BFS from t).

More formally, assume that we have visited up to level l_s from s and to level l_t from t, let $\Gamma^{l_s}(s)$ be the set of nodes at distance l_s from s, and similarly let $\Gamma^{l_t}(t)$ be the set of nodes at distance l_t from t. If $\sum_{v \in \Gamma^{l_s}(s)} \deg(v) \leq \sum_{v \in \Gamma^{l_t}(t)} \deg(v)$, we process all nodes in $\Gamma^{l_s}(s)$, otherwise we process all nodes in $\Gamma^{l_t}(t)$ (since the time needed to process level l_s is proportional to $\sum_{v \in \Gamma^{l_s}(s)} \deg(v)$, this choice minimizes the time needed to visit the next level). Assume that we are processing the node $v \in \Gamma^{l_s}(s)$ (the other case is analogous). For each neighbor w of v we do the following:

- if w was never visited, we add w to $\Gamma^{l_s+1}(s)$;
- if w was already visited in the BFS from s, we do not do anything;
- if w was visited in the BFS from t, we add the edge (v, w) to the set Π of candidate edges in the shortest path.

After we have processed a level, we stop if $\Gamma^{l_s}(s)$ or $\Gamma^{l_t}(t)$ is empty (in this case, s and t are not connected), or if Π is not empty. In the latter case, we select an edge from Π, so that the probability of choosing the edge (v, w) is proportional to $\sigma_{sv}\sigma_{wt}$ (we recall that σ_{xy} is the number of shortest paths from x to y, and it can be computed during the BFS as in [16]).

Then, the path is selected by considering the concatenation of a random path from s to v, the edge (v, w), and a random path from w to t. These random paths can be easily chosen by backtracking, as shown in [36] (since the number of paths might be exponential in the input size, in order to avoid pathological cases, we assume that we can perform arithmetic operations in $\mathcal{O}(1)$ time).

4.1 Analysis on Random Graph

In order to show the effectiveness of the balanced bidirectional BFS, we bound its running time in several models of random graphs: the Configuration Model (CM, [9]), and Rank-1 Inhomogeneous Random Graph models (IRG, [42, Chapter 3]), such as the Chung-Lu model [29], the Norros-Reittu model [32], and the Generalized Random Graph [42, Chapter 3]. In these models, we fix the number n of nodes, and we give a weight ρ_u to each node. In the CM, we create edges by giving ρ_u half-edges to each node u, and pairing these half-edges uniformly at random; in IRG we connect each pair of nodes (u, v) independently with probability close to $\rho_u\rho_v/\sum_{w\in V}\rho_w$. With some technical assumptions discussed in Appendix E of the full version [12], we prove the following theorem.

▶ **Theorem 3.** *Let G be a graph generated through the aforementioned models. Then, for each fixed $\epsilon > 0$, and for each pair of nodes s, t, w.h.p., the time needed to compute an st-shortest path through a bidirectional BFS is $\mathcal{O}(n^{\frac{1}{2}+\epsilon})$ if the degree distribution λ has finite second moment, $\mathcal{O}(n^{\frac{4-\beta}{2}}+\epsilon)$ if λ is a power law distribution with $2 < \beta < 3$.*

Sketch of proof. The idea of the proof is that the time needed by a bidirectional BFS is proportional to the number of visited edges, which is close to the sum of the degrees of the visited nodes, which are very close to their weights. Hence, we have to analyze the weights of the visited edges: for this reason, if V' is a subset of V, we define the volume of V' as $\rho_{V'} = \sum_{v\in V'}\rho_v$.

Our visit proceeds by "levels" in the BFS trees from s and t: if we never process a level with total weight at least $n^{\frac{1}{2}+\epsilon}$, since the diameter is $\mathcal{O}(\log n)$, the volume of the set of processed vertices is $\mathcal{O}(n^{\frac{1}{2}+\epsilon}\log n)$, and the number of visited edges cannot be much bigger (for example, this happens if s and t are not connected). Otherwise, assume that, at some point, we process a level l_s in the BFS from s with total weight $n^{\frac{1}{2}+\epsilon}$: then, the corresponding level l_t in the BFS from t has also weight $n^{\frac{1}{2}+\epsilon}$ (otherwise, we would have expanded from t, because weights and degrees are strongly correlated). We use the "birthday paradox": levels l_s+1 in the BFS from s, and level l_t+1 in the BFS from t are random sets of nodes with size close to $n^{\frac{1}{2}+\epsilon}$, and hence there is a node that is common to both, w.h.p.. This means that the time needed by the bidirectional BFS is proportional to the volume of all levels in the BFS tree from s, until l_s, plus the volume of all levels in the BFS tree from t, until l_t (note that we do not expand levels l_s+1 and l_t+1). All levels except the last have volume at most $n^{\frac{1}{2}+\epsilon}$, and there are $\mathcal{O}(\log n)$ such levels because the diameter is $\mathcal{O}(\log n)$: it only remains to estimate the volume of the last level.

By definition of the models, the probability that a node v with weight ρ_v belongs to the last level is about $\frac{\rho_v\rho_{\mathbf{\Gamma}^{l_s-1}(s)}}{M} \leq \rho_v n^{-\frac{1}{2}+\epsilon}$: hence, the expected volume of $\mathbf{\Gamma}^{l_s}(s)$ is at most $\sum_{v\in V}\rho_v \Pr(v\in\mathbf{\Gamma}^{l_s-1}(s)) \leq \sum_{v\in V}\rho_v^2 n^{-\frac{1}{2}+\epsilon}$. Through standard concentration inequalities, we prove that this random variable is concentrated: hence, we only need to compute this expected value. If the degree distribution has finite second moment, then $\sum_{v\in V}\rho_v^2 = \mathcal{O}(n)$, concluding the proof. If the degree distribution is power law with $2 < \beta < 3$, then we have to consider separately nodes v such that $\rho_v < n^{\frac{1}{2}}$ and such that $\rho_v > n^{\frac{1}{2}}$. In the first case, $\sum_{\rho_v<n^{\frac{1}{2}}}\rho_v^2 \approx \sum_{d=0}^{n^{\frac{1}{2}}} nd^2\lambda(d) \approx \sum_{d=0}^{n^{\frac{1}{2}}} nd^{2-\beta} \approx n^{1+\frac{3-\beta}{2}}$. In the second case, we prove that

the volume of the set of nodes with weight bigger than $n^{\frac{1}{2}}$ is at most $n^{\frac{4-\beta}{2}}$. Hence, the total volume of $\mathbf{\Gamma}^{l_s}(s)$ is at most $n^{-\frac{1}{2}+\epsilon}n^{1+\frac{3-\beta}{2}} + n^{\frac{4-\beta}{2}} \approx n^{\frac{4-\beta}{2}}$. ◀

5 Computing the k Most Central Nodes

Differently from previous works, our algorithm is more flexible, making it possible to compute the betweenness centrality of different nodes with different precision. This feature can be exploited if we only want to rank the nodes: for instance, if v is much more central than all the other nodes, we do not need a very precise estimation on the centrality of v to say that it is the top node. Following this idea, in this section we adapt our approach to the approximation of the ranking of the k most central nodes: as far as we know, this is the first approach which computes the ranking without computing a λ-approximation of all betweenness centralities, allowing significant speedups. Clearly, we cannot expect our ranking to be always correct, otherwise the algorithm does not terminate if two of the k most central nodes have the same centrality. For this reason, the user fixes a parameter λ, and, for each node v, the algorithm does one of the following:

- it provides the exact position of v in the ranking;
- it guarantees that v is not in the top-k;
- it provides a value $\tilde{\boldsymbol{b}}(v)$ such that $|\operatorname{bc}(v) - \tilde{\boldsymbol{b}}(v)| \leq \lambda$.

In other words, similarly to what is done in [36], the algorithm provides a set of $k' \geq k$ nodes containing the top-k nodes, and for each pair of nodes v, w in this subset, either we can rank correctly v and w, or v and w are almost even, that is, $|\operatorname{bc}(v) - \operatorname{bc}(w)| \leq 2\lambda$. In order to obtain this result, we plug into Algorithm 1 the aforementioned conditions in the function `haveToStop` (see Algorithm 3 in the appendix).

Then, we have to adapt the function `computeDelta` to optimize the $\delta_L^{(v)}$s and the $\delta_U^{(v)}$s to the new stopping condition: in other words, we have to choose the values of $\lambda_L^{(v)}$ and $\lambda_U^{(v)}$ that should be plugged into the function `computeDelta` (we recall that the heuristic `computeDelta` chooses the $\delta_L^{(v)}$s so that we can guarantee as fast as possible that $\tilde{\boldsymbol{b}}(v) - \lambda_L^{(v)} \leq \operatorname{bc}(v) \leq \tilde{\boldsymbol{b}}(v) + \lambda_U^{(v)}$). To this purpose, we estimate the betweenness of all nodes with few samples and we sort all nodes according to these approximate values $\tilde{b}(v)$, obtaining $v_1, \ldots, v_n$. The basic idea is that, for the first k nodes, we set $\lambda_U^{(v_i)} = \frac{\tilde{b}(v_{i-1}) - \tilde{b}(v_i)}{2}$, and $\lambda_L^{(v_i)} = \frac{\tilde{b}(v_i) - \tilde{b}(v_{i+1})}{2}$ (the goal is to find confidence intervals that separate the betweenness of v_i from the betweenness of v_{i+1} and v_{i-1}). For nodes that are not in the top-k, we choose $\lambda_L^{(v)} = 1$ and $\lambda_U^{(v)} = \tilde{b}(v_k) - \lambda_L^{(v_k)} - \tilde{b}(v_i)$ (the goal is to prove that v_i is not in the top-k). Finally, if $\tilde{b}(v_i) - \tilde{b}(v_{i+1})$ is small, we simply set $\lambda_L^{(v_i)} = \lambda_U^{(v_i)} = \lambda_L^{(v_{i+1})} = \lambda_U^{(v_{i+1})} = \lambda$, because we do not know if $\operatorname{bc}(v_{i+1}) > \operatorname{bc}(v_i)$, or viceversa.

6 Experimental Results

In this section, we test the four variations of our algorithm on several real-world networks, in order to evaluate their performances. The platform for our tests is a server with 1515 GB RAM and 48 Intel(R) Xeon(R) CPU E7-8857 v2 cores at 3.00GHz, running Debian GNU Linux 8. The algorithms are implemented in C++, and they are compiled using gcc 5.3.1. The source code of our algorithm is available at `https://sites.google.com/a/imtlucca.it/borassi/publications`.

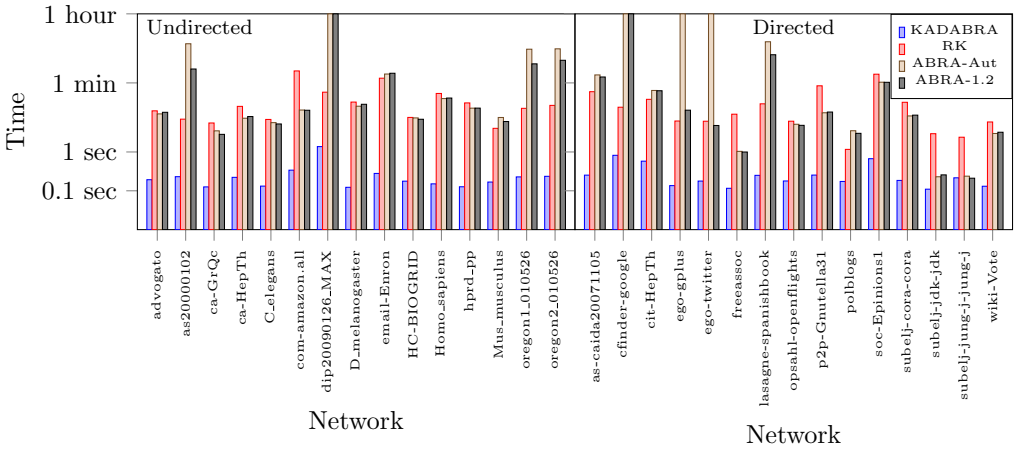

Figure 1 The time needed by the different algorithms, on all the graphs of our dataset.

Comparison with the State of the Art

The first experiment compares the performances of our algorithm KADABRA with the state of the art. The first competitor is the RK algorithm [36], available in the open-source *NetworKit* framework [41]. This algorithm uses the same estimator as our algorithm, but the stopping condition is different: it simply stops after sampling $k = \frac{c}{\epsilon^2}\left(\lfloor \log_2(\mathrm{VD} - 2) \rfloor + 1 + \log\left(\frac{1}{\delta}\right)\right)$, and it uses a heuristic to upper bound the vertex diameter. Following suggestions by the author of the *NetworKit* implementation, we set to 20 the number of samples used in the latter heuristic [7].

The second competitor is the ABRA algorithm [37], available at `http://matteo.rionda.to/software/ABRA-radebetw.tbz2`. This algorithm samples pairs of nodes (s, t), and it adds the fraction of st-paths passing from v to the approximation of the betweenness of v, for each node v. The stopping condition is based on a key result in statistical learning theory, and there is a scheduler that decides when it should be tested. Following the suggestions by the authors, we use both the automatic scheduler ABRA-Aut, which uses a heuristic approach to decide when the stopping condition should be tested, and the geometric scheduler ABRA-1.2, which tests the stopping condition after $(1.2)^i k$ iterations, for each integer i.

The test is performed on a dataset made by 15 undirected and 15 directed real-world networks, taken from the datasets SNAP (`snap.stanford.edu/`), LASAGNE (`piluc.dsi.unifi.it/lasagne`), and KONECT (`http://konect.uni-koblenz.de/networks/`). As in [37], we have considered all values of $\lambda \in \{0.03, 0.025, 0.02, 0.015, 0.01, 0.005\}$, and $\delta = 0.1$. All the algorithms have to provide an approximation $\tilde{\boldsymbol{b}}(v)$ of $\mathrm{bc}(v)$ for each v such that $\Pr\left(\forall v, \left|\tilde{\boldsymbol{b}}(v) - \mathrm{bc}(v)\right| \leq \lambda\right) \geq 1 - \delta$. In Figure 1, we report the time needed by the different algorithms on every graph for $\lambda = 0.005$ (the behavior with different values of λ is very similar). More detailed results are reported in Appendix F of the full version [12].

From the figure, we see that KADABRA is much faster than all the other algorithms, on all graphs: on average, our algorithm is about 100 times faster than RK in undirected graphs, and about 70 times faster in directed graphs; it is also more than 1 000 times faster than ABRA. The latter value is due to the fact that the ABRA algorithm has large running times on few networks: in some cases, it did not even conclude its computation within one hour. The authors confirmed that this behavior might be due to some bugs in the code, which seems to affect it only on specific graphs: indeed, in most networks, the performances

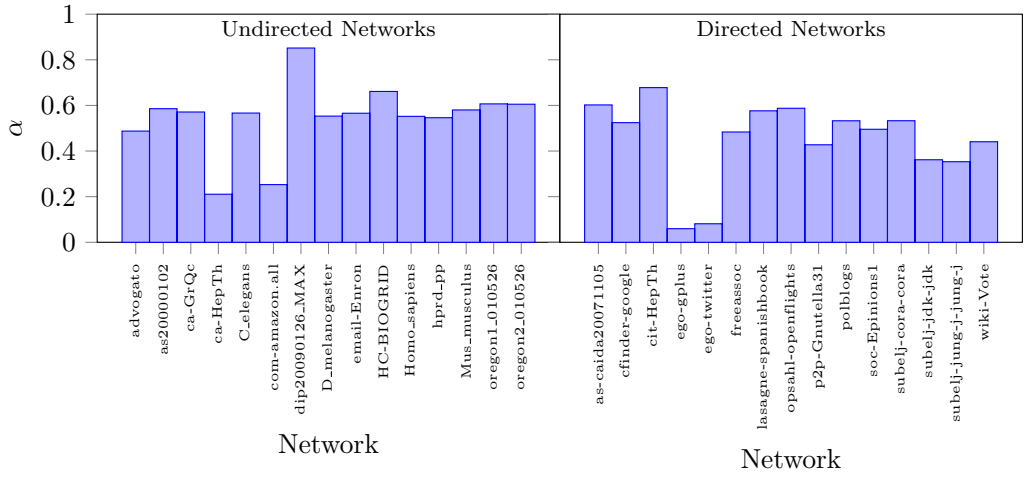

Figure 2 The exponent α such that the average number of edges visited during a bidirectional BFS is n^α.

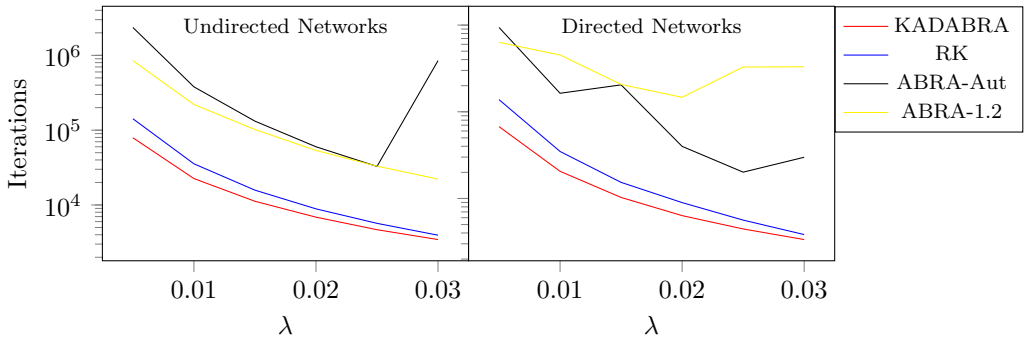

Figure 3 The average number of samples needed by the different algorithms.

of ABRA are better than those of the RK algorithm (but, still, not better than KADABRA).

In order to explain these data, we take a closer look at the improvements obtained through the bidirectional BFS, by considering the average number of edges m_{avg} that the algorithm visits in order to sample a shortest path (for all our competitors, $m_{\text{avg}} = m$, since they perform a full BFS). In Figure 2, for each graph in our dataset, we plot $\alpha = \frac{\log(m_{\text{avg}})}{\log(m)}$ (intuitively, this means that the average number of edges visited is m^α).

The figure shows that, apart from few cases, the number of edges visited is close to $n^{\frac{1}{2}}$, confirming the results in Section 4. This means that, since many of our networks have approximately 10 000 edges, the bidirectional BFS is about 100 times faster than the standard BFS. Finally, for each value of λ, we report in Figure 3 the number of samples needed by all the algorithms, averaged over all the graphs in the dataset.

From the figure, KADABRA needs to sample the smallest amount of shortest paths, and the average improvement over RK grows when λ tends to 0, from a factor 1.14 (resp., 1.14) if $\lambda = 0.03$, to a factor 1.79 (resp., 2.05) if $\lambda = 0.005$ in the case of undirected (resp., directed) networks. Again, the behavior of ABRA is highly influenced by the behavior on few networks, and as a consequence the average number of samples is higher. In any case, also in the graphs where ABRA has good performances, KADABRA still needs a smaller number of samples.

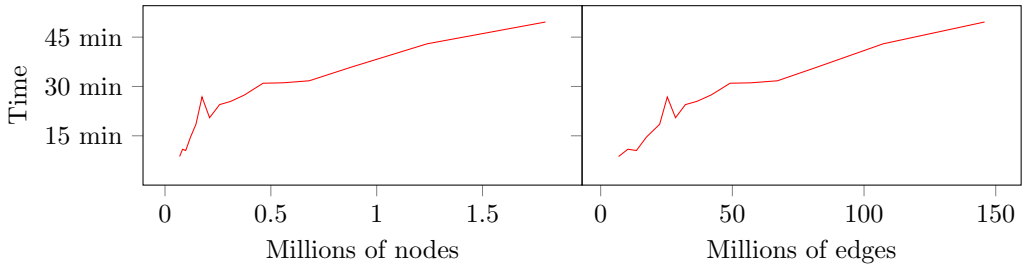

Figure 4 The total time of computation of KADABRA on increasing snapshots of the IMDB graph.

Computing Top-k Centralities

In the second experiment, we let KADABRA compute the top-k betweenness centralities of large graphs, which were unfeasible to handle with the previous algorithms.

The first set of graph is a series of temporal snapshots of the IMDB actor collaboration network, in which two actors are connected if they played together in a movie. The snapshots are taken every 5 years from 1940 to 2010, including a last snapshot in 2014, with 1 797 446 nodes and 145 760 312 edges. The graphs are extracted from the IMDB website (`http://www.imdb.com`), and they do not consider TV-series, awards-shows, documentaries, game-shows, news, realities and talk-shows, in accordance to what was done in `http://oracleofbacon.org`.

The other graph considered is the Wikipedia citation network, whose nodes are Wikipedia pages, and which contains an edge from page p_1 to page p_2 if the text of page p_1 contains a link to page p_2. The graph is extracted from DBPedia 3.7 (`http://wiki.dbpedia.org/`), and it consists of 4 229 697 nodes and 102 165 832 edges.

We have run our algorithm with $\lambda = 0.0002$ and $\delta = 0.1$: as discussed in Section 5, this means that either two nodes are ranked correctly, or their centrality is known with precision at most λ. As a consequence, if two nodes are not ranked correctly, the difference between their real betweenness is at most 2λ. The full results are available in Appendix G of the full version [12].

All the graphs were processed in less than one hour, apart from the Wikipedia graph, which was processed in approximately 1 hour and 38 minutes. In Figure 4, we plot the running times for the actor graphs: from the figure, it seems that the time needed by our algorithm scales slightly sublinearly with respect to the size of the graph. This result respects the results in Section 4, because the degrees in the actor collaboration network are power law distributed with exponent $\beta \approx 2.13$ (`http://konect.uni-koblenz.de/networks/actor-collaboration`). Finally, we observe that the ranking is quite precise: indeed, most of the times, there are very few nodes in the top-5 with the same ranking, and the ranking rarely contains significantly more than 10 nodes.

Acknowledgements. The authors would like to thank Matteo Riondato for several constructive comments on an earlier version of this work. We also thank Elisabetta Bergamini, Richard Lipton, and Sebastiano Vigna for helpful discussions and Holger Dell for his help with the experiments.

References

1 Amir Abboud, Fabrizio Grandoni, and Virginia Vassilevska Williams. Subcubic equivalences between graph centrality problems, apsp and diameter. In *Proceedings of the Twenty-Sixth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1681–1697. SIAM, 2015.

2 Amir Abboud, Virginia V. Williams, and Joshua Wang. Approximation and fixed parameter subquadratic algorithms for radius and diameter. In *Proceedings of the 26th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 377–391, may 2016. URL: `http://arxiv.org/abs/1506.0179`.

3 Ankit Aggarwal, Amit Deshpande, and Ravi Kannan. Adaptive Sampling for k-Means Clustering. In Irit Dinur, Klaus Jansen, Joseph Naor, and José Rolim, editors, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, number 5687 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009.

4 Jac M Anthonisse. The rush in a directed graph. *Stichting Mathematisch Centrum. Mathematische Besliskunde*, BN(9/71):1–10, 1971.

5 David A. Bader, Shiva Kintali, Kamesh Madduri, and Milena Mihail. Approximating betweenness centrality. *The 5th Workshop on Algorithms and Models for the Web-Graph*, 2007.

6 Alex Bavelas. A mathematical model for group structures. *Human organization*, 7(3):16–30, 1948.

7 Elisabetta Bergamini. private communication, 2016.

8 Elisabetta Bergamini and Henning Meyerhenke. Fully-dynamic approximation of betweenness centrality. In *ESA*, 2015.

9 Béla Bollobás. A probabilistic proof of an asymptotic formula for the number of labelled regular graphs. *European Journal of Combinatorics*, 1(4):311–316, 1980. `doi:10.1016/S0195-6698(80)80030-8`.

10 Michele Borassi, Pierluig Crescenzi, and Michel Habib. Into the square - On the complexity of some quadratic-time solvable problems. In *Proceedings of the 16th Italian Conference on Theoretical Computer Science (ICTCS)*, pages 1–17, 2015.

11 Michele Borassi, Pierluigi Crescenzi, and Luca Trevisan. An Axiomatic and an Average-Case Analysis of Algorithms and Heuristics for Metric Properties of Graphs. *arXiv:1604.01445 [cs]*, April 2016.

12 Michele Borassi and Emanuele Natale. Kadabra is an adaptive algorithm for betweenness via random approximation. *arXiv preprint arXiv:1604.08553*, 2016.

13 Stephen P. Borgatti and Martin G. Everett. A graph-theoretic perspective on centrality. *Social Networks*, 28:466–484, 2006.

14 Ulrik Brandes. A faster algorithm for betweenness centrality. *The Journal of Mathematical Sociology*, 25(2):163–177, jun 2001. `doi:10.1080/0022250X.2001.9990249`.

15 Ulrik Brandes. On variants of shortest-path betweenness centrality and their generic computation. *Social Networks*, 30:136–145, 2008.

16 Ulrik Brandes and Christian Pich. Centrality Estimation in Large Networks. *International Journal of Bifurcation and Chaos*, 17(07):2303–2318, 2007. `doi:10.1142/S0218127407018403`.

17 Bernard S Cohn and McKim Marriott. Networks and centres of integration in indian civilization. *Journal of social Research*, 1(1):1–9, 1958.

18 Shlomi Dolev, Yuval Elovici, and Rami Puzis. Routing betweenness centrality. *J. ACM*, 57, 2010.

19 David A. Easley and Jon M. Kleinberg. Networks, crowds, and markets - reasoning about a highly connected world. In *DAGLIB*, 2010.

20 David Eppstein and Joseph Wang. Fast approximation of centrality. *J. Graph Algorithms Appl.*, 8:39–45, 2001.

21 Dóra Erdős, Vatche Ishakian, Azer Bestavros, and Evimaria Terzi. A divide-and-conquer algorithm for betweenness centrality. In *Proceedings of the 2015 SIAM International Conference on Data Mining*, pages 433–441, 2015.

22 Robert Geisberger, Peter Sanders, Dominik Schultes, and Daniel Delling. Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In Catherine C. McGeoch, editor, *Experimental Algorithms: 7th International Workshop, WEA 2008*, pages 319–333. Springer Berlin Heidelberg, 2008.

23 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which Problems Have Strongly Exponential Complexity? *Journal of Computer and System Sciences*, 63(4):512–530, dec 2001. doi:10.1006/jcss.2001.1774.

24 Riko Jacob, Dirk Koschützki, Katharina Anna Lehmann, Leon Peeters, and Dagmar Tenfelde-Podehl. Algorithms for centrality indices. In *DAGSTUHL*, 2004.

25 Hermann Kaindl and Gerhard Kainz. Bidirectional heuristic search reconsidered. *J. Artif. Intell. Res. (JAIR)*, 7:283–317, 1997.

26 Yeon-sup Lim, Daniel S Menasché, Bruno Ribeiro, Don Towsley, and Prithwish Basu. Online estimating the k central nodes of a network. *Proceedings of IEEE NSW*, pages 118–122, 2011.

27 Richard J. Lipton and Jeffrey F. Naughton. Query Size Estimation by Adaptive Sampling. *Journal of Computer and System Sciences*, 51(1):18–25, August 1995. doi:10.1006/jcss.1995.1050.

28 Richard J. Lipton and Naughton, Jeffrey F. Estimating the size of generalized transitive closures. In *Proceedings of the 15th Int. Conf. on Very Large Data Bases*, 1989.

29 Linyuan Lu and Fan R. K. Chung. *Complex graphs and networks*. Number no. 107 in CBMS regional conference series in mathematics. American Mathematical Society, 2006.

30 Mark Newman. *Networks: an introduction*. OUP Oxford, 2010.

31 Mark EJ Newman. Scientific collaboration networks. ii. shortest paths, weighted networks, and centrality. *Physical review E*, 64(1):016132, 2001.

32 Ilkka Norros and Hannu Reittu. On a conditionally Poissonian graph process. *Advances in Applied Probability*, 38(1):59–75, 2006.

33 Jürgen Pfeffer and Kathleen M Carley. k-centralities: local approximations of global measures based on shortest paths. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 1043–1050. ACM, 2012.

34 Andrea Pietracaprina, Matteo Riondato, Eli Upfal, and Fabio Vandin. Mining Top-K Frequent Itemsets Through Progressive Sampling. *Data Mining and Knowledge Discovery*, 21(2):310–326, September 2010. doi:10.1007/s10618-010-0185-7.

35 Ira Pohl. *Bi-directional and heuristic search in path problems*. PhD thesis, Dept. of Computer Science, Stanford University., 1969.

36 Matteo Riondato and Evgenios M Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery*, 30(2):438–475, 2015.

37 Matteo Riondato and Eli Upfal. ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages. *arXiv preprint 1602.05866*, pages 1–27, 2016. arXiv:1602.05866.

38 Ahmet Erdem Sariyüce, Erik Saule, Kamer Kaya, and Ümit V Çatalyürek. Shattering and compressing networks for betweenness centrality. In *SIAM Data Mining Conference (SDM). SIAM*, 2013.

39 Marvin E Shaw. Group structure and the behavior of individuals in small groups. *The Journal of psychology*, 38(1):139–149, 1954.

40 Alfonso Shimbel. Structural parameters of communication networks. *The bulletin of mathematical biophysics*, 15(4):501–507, 1953.

41 Christian L. Staudt, Aleksejs Sazonovs, and Henning Meyerhenke. Networkit: an interactive tool suite for high-performance network analysis. *arXiv preprint 1403.3005*, pages 1–25, 2014.

42 Remco van der Hofstad. Random graphs and complex networks. Vol. II. Manuscript, 2014.

43 Flavio Vella, Giancarlo Carbone, and Massimo Bernaschi. Algorithms and heuristics for scalable betweenness centrality computation on multi-gpu systems. *CoRR*, abs/1602.00963, 2016.

44 Stanley Wasserman and Katherine Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.

45 Ryan Williams and Huacheng Yu. Finding orthogonal vectors in discrete structures. In *Proceedings of the 24th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 1867–1877, 2014. doi:10.1137/1.9781611973402.135.