

ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages

Matteo Riondato
Two Sigma Investments, LP
New York, NY, USA
matteo@twosigma.com

Eli Upfal
Dept. of Computer Science – Brown University
Providence, RI, USA
eli@cs.brown.edu

ABPAΞΑΣ (ABRAXAS): Gnostic word of mystic meaning

ABSTRACT

We present **ABRA**, a suite of algorithms to compute and maintain probabilistically-guaranteed, high-quality, approximations of the betweenness centrality of all nodes (or edges) on both static and fully dynamic graphs. Our algorithms use progressive random sampling and their analysis rely on Rademacher averages and pseudodimension, fundamental concepts from statistical learning theory. To our knowledge, this is the first application of these concepts to the field of graph analysis. Our experimental results show that **ABRA** is much faster than exact methods, and vastly outperforms, in both runtime and number of samples, state-of-the-art algorithms with the same quality guarantees.

Keywords

betweenness; centrality; pseudodimension; Rademacher averages; sampling;

1. INTRODUCTION

Centrality measures are fundamental concepts in graph analysis: they assign to each node or edge in the network a score that quantifies some notion of importance of the node/edge in the network [21]. Betweenness Centrality (BC) is a very popular centrality measure that, informally, defines the importance of a node or edge z in the network as proportional to the fraction of shortest paths in the network that go through z [2, 13] (see Sect. 3 for formal definitions).

Brandes [9] presented an algorithm (denoted **BA**) to compute the exact BC values for all nodes or edges in a graph $G = (V, E)$ in time $O(|V||E|)$ if the graph is unweighted, or time $O(|V||E| + |V|^2 \log |V|)$ if the graph has positive weights. The cost of **BA** is excessive on modern networks with millions of nodes and tens of millions of edges. Moreover, having the exact BC values may often not be needed, given the exploratory nature of the task, and a high-quality

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD '16, August 13–17, 2016, San Francisco, CA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4232-2/16/08...\$15.00

DOI: <http://dx.doi.org/10.1145/2939672.2939770>

approximation of the values is usually sufficient, provided it comes with stringent guarantees.

Today's networks are not only large, but also *dynamic*: edges are added and removed continuously. Keeping the BC values up-to-date after edge insertions and removals is a challenging task, and proposed algorithms [15, 17–19] have worst-case time and space complexities not better than from-scratch-recomputation using **BA**. Maintaining a high-quality approximation up-to-date is more feasible and more *sensible*: there is little added value in keeping track of exact BC values that change continuously.

Contributions. We focus on developing algorithms for approximating the BC of all nodes and edges in static and dynamic graphs. Our contributions are the following.

- We present **ABRA** (for “Approximating Betweenness with Rademacher Averages”), the first family of algorithms based on *progressive sampling* for approximating the BC of all nodes in static and dynamic graphs, where node and edge insertions and deletions are allowed. The BC approximations computed by **ABRA** are *probabilistically guaranteed* to be within an user-specified additive error from their exact values. We also present variants with relative error (i.e., within a multiplicative factor ε of the true value) for the top- k nodes with highest BC, and variants that use refined estimators to give better approximations with a slightly larger sample size.
- Our analysis relies on Rademacher averages [28] and pseudodimension [23], fundamental concepts from the field of statistical learning theory [30]. Exploiting known and novel results using these concepts, **ABRA** computes the approximations without having to keep track of any global property of the graph, in contrast with existing algorithms [4, 6, 24]. **ABRA** performs only “real work” towards the computation of the approximations, without having to obtain such global properties or update them after modifications of the graph. To the best of our knowledge, ours is the first application of Rademacher averages and pseudodimension to graph analysis problems, and the first to use *progressive* random sampling for BC computation. Using pseudodimension, we derive new analytical results on the sample complexity of the BC computation task, generalizing previous contributions [24], and formulating a conjecture on the connection between pseudodimension and the distribution of shortest path lengths.
- The results of our experimental evaluation on real networks show that **ABRA** outperforms, in both speed and

number of samples, the state-of-the-art methods offering the same guarantees [24].

Due to space constraints, some details and additional results have been deferred to the extended online version [25].

2. RELATED WORK

The definition of Betweenness Centrality comes from the sociology literature [2, 13], but the study of efficient algorithms to compute it started only when graphs of substantial size became available to the analysts, following the emergence of the Web. The BA algorithm by Brandes [9] is currently the asymptotically fastest algorithm for computing the exact BC values for all nodes in the network. A number of works also explored heuristics to improve BA [12, 27], but retained the same worst-case time complexity.

The use of random sampling to approximate the BC values in static graphs was proposed independently by Bader et al. [3] and Brandes and Pich [10], and successive works explored the tradeoff space of sampling-based algorithms [4–6, 24]. We focus here on related works that offer approximation guarantees similar to ours. For an in-depth discussion of previous contributions approximating BC on static graphs, we refer the reader to [24, Sect. 2].

Riondato and Kornaropoulos [24] present algorithms that employ the Vapnik-Chervonenkis (VC) dimension [30] to compute what is currently the tightest upper bound on the sample size sufficient to obtain guaranteed approximations of the BC of all nodes in a static graph. Their algorithms offer the same guarantees as ours but, to compute the sample size, they need to compute an upper bound on a characteristic quantity of the graph (the vertex diameter, namely the maximum number of nodes on any shortest path). Thanks to our use of Rademacher averages in a progressive random sampling setting, ABRA does not need to compute any characteristic quantity of the graph, and instead uses an efficient-to-evaluate stopping condition to determine when the approximated BC values are close to the exact ones. This allows ABRA to use smaller samples and be much faster than the algorithms by Riondato and Kornaropoulos [24].

A number of works [15, 17–19] focused on computing the exact BC for all nodes in a dynamic graph, taking into consideration different update models. None of these algorithm is provably asymptotically faster than a complete computation from scratch using Brandes’ algorithm [9] and they all require significant amount of space (more details about these works can be found in [4, Sect. 2]). In contrast, Bergamini and Meyerhenke [4, 5] built on the work by Riondato and Kornaropoulos [24] to derive an algorithm for maintaining high-quality approximations of the BC of all nodes when the graph is dynamic and both additions and deletions of edges are allowed. Due to the use of the algorithm by Riondato and Kornaropoulos [24] as a building block, the algorithm must keep track of the vertex diameter after an update to the graph. Our algorithm for dynamic graphs, instead, does not need this piece of information, and therefore can spend more time in computing the approximations, rather than in keeping track of global properties of the graph. Moreover, our algorithm can handle directed graphs, which is not the case for the algorithms by Bergamini and Meyerhenke [4, 5].

Hayashi et al. [16] recently proposed a data structure called *Hypergraph Sketch* to maintain the shortest path DAGs between pairs of nodes following updates to the graph. Their algorithm uses random sampling and this novel data struc-

ture allows them to maintain a high-quality, probabilistically guaranteed approximation of the BC of all nodes in a dynamic graph. Their guarantees come from an application of the simple uniform deviation bounds (i.e., the union bound) to determine the sample size, as previously done by Bader et al. [3] and Brandes and Pich [10]. As a result, the resulting sample size is excessively large, as it depends on the *number of nodes in the graph*. Our improved analysis using the Rademacher averages allows us to develop an algorithm that uses the Hypergraph Sketch with a much smaller number of samples, and is therefore faster.

Progressive random sampling with Rademacher Averages has been used by Elomaa and Kääriäinen [11] and Riondato and Upfal [26] in completely different settings.

3. PRELIMINARIES

We now introduce the formal definitions and basic results that we use throughout the paper.

3.1 Graphs and Betweenness Centrality

Let $G = (V, E)$ be a graph. G may be directed or undirected and may have non-negative weights on the edges. For any ordered pair (u, v) of different nodes $u \neq v$, let \mathcal{S}_{uv} be the set of *Shortest Paths* (SPs) from u to v , and let $\sigma_{uv} = |\mathcal{S}_{uv}|$. Given a path p between two nodes $u, v \in V$, a node $w \in V$ is *internal to p* iff $w \neq u$, $w \neq v$, and p goes through w . We denote as $\sigma_{uv}(w)$ the number of SPs from u to v that w is internal to.

DEFINITION 1 (BETWEENNESS CENTRALITY (BC) [2, 13]). *Given a graph $G = (V, E)$, the Betweenness Centrality (BC) of a node $w \in V$ is defined as*

$$b(w) = \frac{1}{|V|(|V| - 1)} \sum_{\substack{(u,v) \in V \times V \\ u \neq v}} \frac{\sigma_{uv}(w)}{\sigma_{uv}} \quad (\in [0, 1]) .$$

Many variants of BC have been proposed in the literature, including one for edges [21]. Our results can be extended to these variants, following the reduction by Riondato and Kornaropoulos [24, Sect. 6], but we do not discuss them here due to space constraints.

In this work we focus on computing an (ε, δ) -approximation of the collection $B = \{b(w), w \in V\}$.

DEFINITION 2 ((ε, δ) -APPROXIMATION). *Given $\varepsilon, \delta \in (0, 1)$, an (ε, δ) -approximation to B is a collection $\tilde{B} = \{\tilde{b}(w), w \in V\}$ such that*

$$\Pr(\forall w \in v, |\tilde{b}(w) - b(w)| \leq \varepsilon) \geq 1 - \delta .$$

In Sect. 4.2 we discuss a relative (i.e., multiplicative) error variant for the top- k nodes with highest BC.

3.2 Rademacher Averages

Rademacher Averages are fundamental concepts to study the rate of convergence of a set of sample averages to their expectations. They are at the core of statistical learning theory [30] but their usefulness extends way beyond the learning framework [26]. We present here only the definitions and results that we use in our work and we refer the readers to, e.g., the book by Shalev-Shwartz and Ben-David [28] for in-depth presentation and discussion.

While the Rademacher complexity can be defined on an arbitrary measure space, we restrict our discussion here to

a sample space that consists of a finite domain \mathcal{D} and a uniform distribution over that domain. Let \mathcal{F} be a family of functions from \mathcal{D} to $[0, 1]$, and let $\mathcal{S} = \{c_1, \dots, c_\ell\}$ be a sample of ℓ elements from \mathcal{D} , sampled uniformly and independently at random. For each $f \in \mathcal{F}$, the *true average* and the *sample average* of f on a sample \mathcal{S} are, respectively,

$$m_{\mathcal{D}}(f) = \frac{1}{|\mathcal{D}|} \sum_{c \in \mathcal{D}} f(c) \quad \text{and} \quad m_{\mathcal{S}}(f) = \frac{1}{\ell} \sum_{i=1}^{\ell} f(c_i). \quad (1)$$

Given \mathcal{S} , we are interested in bounding the *maximum deviation* of $m_{\mathcal{S}}(f)$ from $m_{\mathcal{D}}(f)$ among all $f \in \mathcal{F}$, i.e.,

$$\sup_{f \in \mathcal{F}} |m_{\mathcal{S}}(f) - m_{\mathcal{D}}(f)|. \quad (2)$$

For $1 \leq i \leq \ell$, let σ_i be a Rademacher r.v., i.e., a r.v. that takes value 1 with probability 1/2 and -1 with probability 1/2. The r.v.'s σ_i are independent. Consider the quantity

$$R(\mathcal{F}, \mathcal{S}) = \mathbb{E}_{\sigma} \left[\sup_{f \in \mathcal{F}} \frac{1}{\ell} \sum_{i=1}^{\ell} \sigma_i f(c_i) \right], \quad (3)$$

where the expectation is taken w.r.t. the Rademacher r.v.'s, i.e., conditionally on \mathcal{S} . The quantity $R(\mathcal{F}, \mathcal{S})$ is known as the *(conditional) Rademacher average* of \mathcal{F} on \mathcal{S} . The following is a key result in statistical learning theory, connecting $R(\mathcal{F}, \mathcal{S})$ to the maximum deviation (2).

THEOREM 1 (THM. 26.5 [28]). *Let $\eta \in (0, 1)$ and let \mathcal{S} be a collection of ℓ elements of \mathcal{D} sampled independently and uniformly at random. Then, with probability at least $1 - \eta$,*

$$\sup_{f \in \mathcal{F}} |m_{\mathcal{S}}(f) - m_{\mathcal{D}}(f)| \leq 2R(\mathcal{F}, \mathcal{S}) + \sqrt{\frac{2 \ln(2/\eta)}{\ell}}. \quad (4)$$

Thm. 1 is how the result is classically presented, but better although more complex bounds than (4) are available [22].

THEOREM 2 (THM. 3.11 [22]). *Let $\eta \in (0, 1)$ and let \mathcal{S} be a collection of ℓ elements of \mathcal{D} sampled independently and uniformly at random. Let*

$$\alpha = \frac{\ln \frac{2}{\eta}}{\ln \frac{2}{\eta} + \sqrt{(2\ell R(\mathcal{F}, \mathcal{S}) + \ln \frac{2}{\eta}) \ln \frac{2}{\eta}}}, \quad (5)$$

then, with probability at least $1 - \eta$,

$$\sup_{f \in \mathcal{F}} |m_{\mathcal{S}}(f) - m_{\mathcal{D}}(f)| \leq \frac{R(\mathcal{F}, \mathcal{S})}{1 - \alpha} + \frac{\ln \frac{2}{\eta}}{2\ell\alpha(1 - \alpha)} + \sqrt{\frac{\ln \frac{2}{\eta}}{2\ell}}. \quad (6)$$

Computing, or even estimating, the expectation in (3) w.r.t. the Rademacher r.v.'s is not straightforward and can be computationally expensive, requiring a time-consuming Monte Carlo simulation [7]. For this reason, *upper bounds to the Rademacher average* are usually employed in (4) and (6) in place of $R(\mathcal{F}, \mathcal{S})$. A powerful and efficient-to-compute bound is presented in Thm. 3. Given \mathcal{S} , consider, for each $f \in \mathcal{F}$, the vector $\mathbf{v}_{f, \mathcal{S}} = (f(c_1), \dots, f(c_\ell))$, and let $\mathcal{V}_{\mathcal{S}} = \{\mathbf{v}_f, f \in \mathcal{F}\}$ be the set of such vectors ($|\mathcal{V}_{\mathcal{S}}| \leq |\mathcal{F}|$).

THEOREM 3 ([26]). *Let $w : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ be the function*

$$w(s) = \frac{1}{s} \ln \sum_{\mathbf{v} \in \mathcal{V}_{\mathcal{S}}} \exp(s^2 \|\mathbf{v}\|^2 / (2\ell^2)), \quad (7)$$

where $\|\cdot\|$ denotes the Euclidean norm. Then

$$R(\mathcal{F}, \mathcal{S}) \leq \min_{s \in \mathbb{R}^+} w(s). \quad (8)$$

The function w is convex, continuous in \mathbb{R}^+ , and has first and second derivatives w.r.t. s everywhere in its domain, so it is possible to minimize it efficiently using standard convex optimization methods [8]. In future work, we plan to explore how to obtain a tighter bound than the one presented in Thm. 3 using recent results by Anguita et al. [1].

4. STATIC GRAPH BC APPROXIMATION

We now present and analyze **ABRA-s**, our *progressive sampling algorithm* for computing an (ϵ, δ) -approximation to the collection of exact BC values in a *static* graph. Many of the details and properties of **ABRA-s** are shared with the other **ABRA** algorithms we present.

Progressive Sampling. Progressive sampling algorithms are intrinsically *iterative*. At a high level, they work as follows. At iteration i , the algorithm extracts an approximation of the values of interest (in our case, of the BC of all nodes) from a collection \mathcal{S}_i of $S_i = |\mathcal{S}_i|$ random samples from a suitable domain \mathcal{D} (in our case, the samples are pairs of different nodes). Then, the algorithm checks a specific *stopping condition* that uses information obtained from the sample \mathcal{S}_i and from the computed approximation. If the stopping condition is satisfied, then the approximation has, with the required probability, the desired quality (in our case, it is an (ϵ, δ) -approximation). The approximation is then returned in output and the algorithm terminates. If the stopping condition is not satisfied, **ABRA-s** builds a collection \mathcal{S}_{i+1} by adding random samples to \mathcal{S}_i until it has size S_{i+1} . Then it computes a new approximation from the so-created collection \mathcal{S}_{i+1} , and checks the stopping condition again and so on.

There are two main challenges for the algorithm designer: deriving a “good” stopping condition and determining good choices for the initial sample size S_1 and the subsequent sample sizes S_{i+1} , $i \geq 1$.

Ideally, a good stopping condition is such that:

1. when satisfied, guarantees that the computed approximation has the desired quality properties (in our case, it is an (ϵ, δ) -approximation); and
 2. can be evaluated efficiently; and
 3. is tight, in the sense that is satisfied at small sample sizes.
- The stopping condition for **ABRA-s** is based on Thm. 2 and Thm. 3, and has all the above desirable properties.

The second challenge is determining the *sample schedule* $(S_i)_{i>0}$. Any monotonically increasing sequence of positive numbers can act as sample schedule, but the goal in designing a good sample schedule is to minimize the number of iterations that are needed before the stopping condition is satisfied, while minimizing the sample size S_i at the iteration i at which this happens. The sample schedule may be fixed in advance, but an *adaptive approach* that ties the sample schedule to the stopping condition can give better results, as the sample size S_{i+1} for iteration $i + 1$ can be computed using information obtained in (or up-to) iteration i . **ABRA** uses such an adaptive approach (see Sect. 4.1.1.)

4.1 Algorithm Description and Analysis

ABRA-s takes as input a graph $G = (V, E)$, which may be directed or undirected, and may have non-negative weights

on the edges, and two parameters $\varepsilon, \delta \in (0, 1)$. It outputs a collection $\tilde{B} = \{\tilde{\mathbf{b}}(w), w \in V\}$ that is an (ε, δ) -approximation of the betweenness centralities $B = \{\mathbf{b}(w), w \in V\}$. The algorithm samples from $\mathcal{D} = V \times V, u \neq v$.

For each node $w \in V$, let $f_w : \mathcal{D} \rightarrow [0, 1]$ be the function

$$f_w(u, v) = \frac{\sigma_{uv}(w)}{\sigma_{uv}}, \quad (9)$$

i.e., $f_w(u, v)$ is the fraction of shortest paths (SPs) from u to v that go through w . Let $\mathcal{F} = \{f_w, w \in V\}$ be the set of these functions. Given this definition, we have that

$$\begin{aligned} m_{\mathcal{D}}(f_w) &= \frac{1}{|\mathcal{D}|} \sum_{(u,v) \in \mathcal{D}} f_w(u, v) \\ &= \frac{1}{|V|(|V|-1)} \sum_{\substack{(u,v) \in V \times V \\ u \neq v}} \frac{\sigma_{uv}(w)}{\sigma_{uv}} \\ &= \mathbf{b}(w). \end{aligned}$$

Let now $\mathcal{S} = \{(u_i, v_i), 1 \leq i \leq \ell\}$ be a collection of ℓ pairs (u, v) from \mathcal{D} . For the sake of clarity, we define

$$\tilde{\mathbf{b}}(w) = m_{\mathcal{S}}(f_w) = \frac{1}{\ell} \sum_{i=1}^{\ell} f_w((u_i, v_i)).$$

For each $w \in V$ consider the vector

$$\mathbf{v}_w = (f_w(u_1, v_1), \dots, f_w(u_{\ell}, v_{\ell})).$$

It is easy to see that $\tilde{\mathbf{b}}(w) = \|\mathbf{v}_w\|_1 / \ell$. Let now $\mathcal{V}_{\mathcal{S}}$ be the set of these vectors:

$$\mathcal{V}_{\mathcal{S}} = \{\mathbf{v}_w, w \in V\} \quad (|\mathcal{V}_{\mathcal{S}}| \leq |V|).$$

If we have complete knowledge of this set of vectors, then we can compute the quantity

$$\omega^* = \min_{s \in \mathbb{R}^+} \frac{1}{s} \ln \sum_{\mathbf{v} \in \mathcal{V}_{\mathcal{S}}} \exp(s^2 \|\mathbf{v}\|^2 / (2\ell^2)),$$

then use ω^* in (5) in place of $R(\mathcal{F}, \mathcal{S})$ to obtain α , and combine (6), (7), and (8) to obtain

$$\Delta_{\mathcal{S}} = \frac{\omega^*}{1 - \alpha} + \frac{\ln \frac{2}{\eta}}{2\ell\alpha(1 - \alpha)} + \sqrt{\frac{\ln \frac{2}{\eta}}{2\ell}},$$

and finally check whether $\Delta_{\mathcal{S}} \leq \varepsilon$. This is **ABRA-s**'s stopping condition. When it holds, we can just return the collection $\tilde{B} = \{\tilde{\mathbf{b}}(w) = \|\mathbf{v}_w\|_1 / \ell, w \in V\}$ since, from the definition of $\Delta_{\mathcal{S}}$ and Thms. 2 and 3, we have that \tilde{B} is an (ε, δ) -approximation to the exact betweenness values.

ABRA-s works as follows. Suppose for now that we fix a priori a monotonically increasing sequence $(S_i)_{i>0}$ of sample sizes (we show in Sect. 4.1.1 how to compute the sample schedule adaptively on the fly). The algorithm builds a collection \mathcal{S} by sampling pairs (u, v) independently and uniformly at random from \mathcal{D} until \mathcal{S} has size S_1 . After each pair of nodes has been sampled, **ABRA-s** performs an $s - t$ SP computation from u to v and then backtracks from v to u along the SPs just computed, to keep track of the set $\mathcal{V}_{\mathcal{S}}$ of vectors (details given below). For clarity of presentation, let \mathcal{S}_1 denote \mathcal{S} when it has size exactly S_1 , and analogously for \mathcal{S}_i and $S_i, i > 1$. Once \mathcal{S}_i has been built, **ABRA-s** computes $\Delta_{\mathcal{S}_i}$ using $\eta = \delta/2^i$ and checks whether $\Delta_{\mathcal{S}_i}$ is at most

ε . If so, then it returns \tilde{B} . Otherwise, **ABRA-s** iterates and continues adding samples from \mathcal{D} to \mathcal{S} until it has size S_2 , and so on until $\Delta_{\mathcal{S}_i} \leq \varepsilon$ holds. The pseudocode for **ABRA-s** is presented in Alg. 1, including the steps to update $\mathcal{V}_{\mathcal{S}}$ and to adaptively choose the sample schedule (Sect. 4.1.1). We now prove the correctness of the algorithm.

Algorithm 1: ABRA-s: absolute error approximation of BC on static graphs

input : Graph $G = (V, E)$, accuracy parameter $\varepsilon \in (0, 1)$, confidence parameter $\delta \in (0, 1)$

output: Set \tilde{B} of BC approximations for all nodes in V

- 1 $\mathcal{D} \leftarrow \{(u, v) \in V \times V, u \neq v\}$
- 2 $S_0 \leftarrow 0, S_1 \leftarrow \frac{(1+8\varepsilon+\sqrt{1+16\varepsilon}) \ln(2/\delta)}{4\varepsilon^2}$
- 3 $\mathbf{0} = (0)$
- 4 $\mathcal{V} = \{\mathbf{0}\}$
- 5 **foreach** $w \in V$ **do** $M[w] = \mathbf{0}$
- 6 $\mathbf{c}_0 \leftarrow |V|$
- 7 $i \leftarrow 1, j \leftarrow 1$
- 8 **while** *True* **do**
- 9 **for** $\ell \leftarrow 1$ **to** $S_i - S_{i-1}$ **do**
- 10 $(u, v) \leftarrow \text{uniform_random_sample}(\mathcal{D})$
- 11 $\text{compute_SPs}(u, v)$ //Truncated SP computation
- 12 **if** *reached* v **then**
- 13 **foreach** $z \in P_u[v]$ **do** $\sigma_{zv} \leftarrow 1$
- 14 **foreach** *node* w *on a SP from* u *to* v , *in reverse order by* $d(u, w)$ **do**
- 15 $\sigma_{uw}(w) \leftarrow \sigma_{uw}\sigma_{wv}$
- 16 $\mathbf{v} \leftarrow M[w]$
- 17 $\mathbf{v}' \leftarrow \mathbf{v} \cup \{(j, \sigma_{uv}(w))\}$
- 18 **if** $\mathbf{v}' \notin \mathcal{V}$ **then**
- 19 $\mathbf{c}_{\mathbf{v}'} \leftarrow 1$
- 20 $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{v}'\}$
- 21 **else** $\mathbf{c}_{\mathbf{v}'} \leftarrow \mathbf{c}_{\mathbf{v}'} + 1$
- 22 $M[w] \leftarrow \mathbf{v}'$
- 23 **if** $\mathbf{c}_{\mathbf{v}'} > 1$ **then** $\mathbf{c}_{\mathbf{v}} \leftarrow \mathbf{c}_{\mathbf{v}} - 1$
- 24 **else** $\mathcal{V} \leftarrow \mathcal{V} \setminus \{\mathbf{v}\}$
- 25 **foreach** $z \in P_u[w]$ **do** $\sigma_{zv} \leftarrow \sigma_{zv} + \sigma_{wv}$
- 26 **end**
- 27 **end**
- 28 $j \leftarrow j + 1$
- 29 **end**
- 30 $\omega_i^* \leftarrow \min_{s \in \mathbb{R}^+} \frac{1}{s} \ln \sum_{\mathbf{v} \in \mathcal{V}_{\mathcal{S}_i}} \exp(s^2 \|\mathbf{v}\|^2 / (2S_i^2))$
- 31 $\alpha_i \leftarrow \frac{(i+1) \ln \frac{2}{\delta}}{(i+1) \ln \frac{2}{\delta} + \sqrt{(2S_i\omega_i^* + (i+1) \ln \frac{2}{\delta})(i+1) \ln \frac{2}{\delta}}}$
- 32 $\Delta_{\mathcal{S}_i} \leftarrow \frac{\omega_i^*}{1 - \alpha_i} + \frac{(i+1) \ln \frac{2}{\delta}}{2S_i\alpha_i(1 - \alpha_i)} + \sqrt{(i+1) \frac{\ln \frac{2}{\delta}}{2S_i}}$
- 33 **if** $\Delta_{\mathcal{S}_i} \leq \varepsilon$ **then break**
- 34 **else**
- 35 $S_{i+1} \leftarrow \text{nextSampleSize}()$
- 36 $i \leftarrow i + 1$
- 37 **end**
- 38 **end**
- 39 **return** $\tilde{B} \leftarrow \{\tilde{\mathbf{b}}(w) \leftarrow \|M[w]\|_1 / S_i, w \in V\}$

THEOREM 4 (CORRECTNESS). *The collection \tilde{B} returned by **ABRA-s** is a (ε, δ) -approximation.*

PROOF. The claim follows from the definitions of $\mathcal{S}, \mathcal{V}_{\mathcal{S}}, \mathcal{F}, f_w$ for $w \in V, \tilde{\mathbf{b}}(w), \Delta_{\mathcal{S}_i}$, Thm. 3, and from the fact that, at each iteration i , Thm. 2 holds with probability $\delta/2^i$. \square

Computing and maintaining the set $\mathcal{V}_{\mathcal{S}}$. We now discuss in details how **ABRA-s** efficiently maintain the set $\mathcal{V}_{\mathcal{S}}$ of

vectors, which is used to compute the value Δ_S and the values $\tilde{\mathbf{b}}(w) = \|\mathbf{v}_w\|_1/|\mathcal{S}|$ in \tilde{B} . In addition to \mathcal{V}_S , **ABRA-s** also maintains a map M from V to \mathcal{V}_S (i.e., $M[w]$ is a vector $\mathbf{v}_w \in \mathcal{V}_S$), and a counter \mathbf{c}_v for each $\mathbf{v} \in \mathcal{V}_S$, denoting how many nodes $w \in V$ have $M[w] = \mathbf{v}$.

At the beginning of the execution of the algorithm, we have $\mathcal{S} = \emptyset$ and also $\mathcal{V}_S = \emptyset$. Nevertheless, **ABRA-s** initializes \mathcal{V}_S to contain one special empty vector $\mathbf{0}$, with no components, and M so that $M[w] = \mathbf{0}$ for all $w \in V$, and $\mathbf{c}_0 = |V|$ (lines 3 and following in Alg. 1).

After having sampled a pair (u, v) from \mathcal{D} , **ABRA-s** updates \mathcal{V}_S , M and the counters as follows. First, it performs (line 11) a $s-t$ SP computation from u to v using any SP algorithm (e.g., BFS or Dijkstra) modified, as discussed by Brandes [9, Lemma 3], to keep track, for each node w encountered during the computation, of the SP distance $\mathbf{d}(u, w)$ from u to w , of the number σ_{uw} of SPs from u to w , and of the set $\mathbf{P}_u(w)$ of (immediate) predecessors of w along the SPs from u .¹ Once v has been reached (and only if it has been reached), the algorithm starts backtracking from v towards u along the SPs it just computed (line 14). During this backtracking, the algorithm visits the nodes along the SPs in inverse order of SP distance from u , ties broken arbitrarily. For each visited node w different from u and v , **ABRA-s** computes the value $f_w(u, v) = \sigma_{uv}(w)$ of SPs from u to v that go through w , which is obtained as

$$\sigma_{uv}(w) = \sigma_{uw} \times \sum_{z: w \in \mathbf{P}_u(z)} \sigma_{zv}$$

where the value σ_{uw} is obtained during the $s-t$ SP computation, and the values σ_{zw} are computed recursively during the backtracking (line 25) [9]. After computing $\sigma_{uv}(w)$, the algorithm takes the vector $\mathbf{v} \in \mathcal{V}_S$ such that $M[w] = \mathbf{v}$ and creates a new vector \mathbf{v}' by appending $\sigma_{uv}(w)$ to the end of \mathbf{v} .² Then it adds \mathbf{v}' to the set \mathcal{V}_S , updates $M[w]$ to \mathbf{v}' , and increments the counter \mathbf{c}_v by one (lines 16 to 22). Finally, the algorithm decrements the counter \mathbf{c}_v by one, and if \mathbf{c}_v becomes equal to zero, **ABRA-s** removes \mathbf{v} from \mathcal{V}_S (line 24). At this point, the algorithm moves to analyzing another node w' with distance from u less or equal to the distance of w from u . It is easy to see that when the backtracking reaches u , the set \mathcal{V}_S , the map M , and the counters, have been correctly updated.

We remark that to compute Δ_{S_i} and \tilde{B} and to keep the map M up to date, **ABRA-s** does not actually need to store the vectors in \mathcal{V}_S (even in sparse form), but it is sufficient to maintain their ℓ_1 - and Euclidean norms, which require much less space.

4.1.1 Computing the sample schedule

We now discuss how to compute the initial sample size S_1 at the beginning of **ABRA-s** (line 2 of Alg. 1) and the sample size S_{i+1} at the end of iteration i of the main loop (line 35). We remark that any sample schedule $(S_i)_{i>0}$ can be used, and our method is an heuristic that nevertheless exploits

¹Storing the set of immediate predecessors is not necessary. By not storing it, we can reduce the space complexity from $O(|E|)$ to $O(|V|)$, at the expense of some additional computation at runtime.

²**ABRA-s** uses a sparse representation for the vectors $\mathbf{v} \in \mathcal{V}_S$, storing only the non-zero components of each \mathbf{v} as pairs (j, g) , where j is the component index and g is the value of that component.

all available information at the end of each iteration to the most possible extent, with the goal of increasing the chances that the stopping condition is satisfied at the next iteration.

As initial sample size S_1 we choose

$$S_1 \geq \frac{(1 + 8\varepsilon + \sqrt{1 + 16\varepsilon}) \ln(4/\delta)}{4\varepsilon^2} . \quad (10)$$

To understand the intuition behind this choice, recall (6), and consider that, at the beginning of the algorithm, we obviously have no information about $\mathbf{R}(\mathcal{F}, \mathcal{S}_1)$, except that it is *non-negative*. Consequently we also cannot compute α as in (5) using $\eta = \delta/2$, but we can easily see that $\alpha \in [0, 1/2]$. From the fact that $\mathbf{R}(\mathcal{F}, \mathcal{S}) \geq 0$, we have that, for the r.h.s. of (6) to be at most ε (i.e., for the stopping condition to be satisfied after the first iteration of the algorithm), it is necessary that

$$\frac{\ln \frac{4}{\delta}}{2S_1\alpha(1-\alpha)} + \sqrt{\frac{\ln \frac{4}{\delta}}{2S_1}} \leq \varepsilon .$$

Then, using the fact that the above expression decreases as α increases, we use $\alpha = 1/2$, i.e., its maximum attainable value, to obtain the following inequality, where S_1 acts as the unknown:

$$\frac{2 \ln(4/\delta)}{S_1} + \sqrt{\frac{\ln(4/\delta)}{2S_1}} \leq \varepsilon .$$

Solving for S_1 under the domain constraints $S_1 \geq 1$, $\delta \in (0, 1)$, and $\varepsilon \in (0, 1)$ gives the unique solution in (10).

Computing the next sample size S_{i+1} at the end of iteration i (in the pseudocode in Alg. 1, this is done by calling `nextSampleSize()` on line 35) is slightly more involved. The intuition is to assume that ω_i^* , which is an upper bound on $\mathbf{R}(\mathcal{F}, \mathcal{S}_i)$, is also an upper bound on $\mathbf{R}(\mathcal{F}, \mathcal{S}_{i+1})$, whatever \mathcal{S}_{i+1} will be, and whatever size it may have. At this point, we can ask what is the minimum size $S_{i+1} = |\mathcal{S}_{i+1}|$ for which $\Delta_{S_{i+1}}$ would be at most ε , under the assumption that $\mathbf{R}(\mathcal{F}, \mathcal{S}_{i+1}) \leq \omega_i^*$. More formally, we want to solve the inequality

$$\begin{aligned} & \left(1 + \frac{(i+2) \ln \frac{2}{\delta}}{\sqrt{(2S_{i+1}\omega_i^* + (i+2) \ln \frac{2}{\delta})(i+2) \ln \frac{2}{\delta}}} \right) \\ & \times \left(\omega_i^* + \frac{(i+2) \ln \frac{2}{\delta} + \sqrt{(2S_{i+1}\omega_i^* + (i+2) \ln \frac{2}{\delta})(i+2) \ln \frac{2}{\delta}}}{2S_{i+1}} \right) \\ & + \sqrt{\frac{(i+2) \ln \frac{2}{\delta}}{2S_{i+1}}} \leq \varepsilon \end{aligned} \quad (11)$$

where S_{i+1} acts as the unknown. The l.h.s. of this inequality is obtained by plugging (5) into (6) and using ω_i^* in place of $\mathbf{R}(\mathcal{F}, \mathcal{S})$, S_{i+1} in place of ℓ , $\delta/2^{i+1}$ in place of η , and slightly reorganize the terms for readability. Finding the solution to the above inequality requires computing the roots of the cubic equation (in x)

$$\begin{aligned} & -8 \left((i+2) \ln \frac{2}{\delta} \right)^3 + \left((i+2) \ln \frac{2}{\delta} \right)^2 (-16\omega_i^* + (1+4\varepsilon)^2)x \\ & - 4 \left((i+2) \ln \frac{2}{\delta} \right) (\omega_i^* - \varepsilon)^2 (1+4\varepsilon)x^2 + 4(b-f)^4 x^3 = 0 . \end{aligned} \quad (12)$$

One can verify that the roots of this equation are all reals. The roots are presented in Table 1. The solution to inequality (11) is that S_{i+1} should be larger than one of these roots,

but which of the roots it should be larger than depends on the values of ω_i^* , δ , and ε . In practice, we compute each of the roots and then choose the smallest positive one such that, when S_{i+1} equals to this root, then (11) is satisfied.

The assumption $R(\mathcal{F}, S_{i+1}) \leq \omega_i^*$, which is not guaranteed to be true, is what makes our procedure for selecting the next sample size an *heuristics*. Nevertheless, using information available at the current iteration to compute the sample size for the next iteration is more sensible than having a fixed sample schedule, as it tunes the growth of the sample size to the quality of the current sample. Moreover, it removes from the user the burden of choosing a sample schedule, effectively eliminating one parameter of the algorithm.

4.2 Relative-error Top-k Approximation

In practical applications it is usually necessary (and sufficient) to identify the nodes with highest BC, as they act, in some sense, as the “primary information gateways” of the network. In this section we present a variant **ABRA-k** of **ABRA-s** to compute a high-quality approximation of the set $\text{TOP}(k, G)$ of the top- k nodes with highest BC in a graph G . The approximation $\tilde{\mathbf{b}}(w)$ returned by **ABRA-k** for a node w is within a *multiplicative* factor ε from its exact value $\mathbf{b}(w)$, rather than an additive factor ε as in **ABRA-s**. This higher accuracy has a cost in terms of the number of samples needed to compute the approximations.

Formally, assume to order the nodes in the graph in decreasing order by BC, ties broken arbitrarily, and let b_k be the BC of the k -th node in this ordering. Then the set $\text{TOP}(k, G)$ is defined as the set of nodes with BC at least b_k , and can contain more than k nodes:

$$\text{TOP}(k, G) = \{(w, \mathbf{b}(w)) : w \in V \text{ and } \mathbf{b}(w) \geq b_k\} .$$

The algorithm **ABRA-k** follows the same approach as the algorithm for the same task by Riondato and Kornaropoulos [24, Sect. 5.2] and works in two phases. Let δ_1 and δ_2 be such that $(1 - \delta_1)(1 - \delta_2) \geq (1 - \delta)$. In the first phase, we run **ABRA-s** with parameters ε and δ_1 . Let ℓ' be the k -th highest value $\tilde{\mathbf{b}}(w)$ returned by **ABRA-s**, ties broken arbitrarily, and let $\tilde{b}' = \ell' - \varepsilon$.

In the second phase, we use a variant **ABRA-r** of **ABRA-s** with a modified stopping condition based on relative-error versions of Thms. 1 and 3 (Thms. 11 and 12 from Appendix D of the extended online version [25]), which take ε , δ_2 , and $\lambda = \tilde{b}'$ as parameters. The parameter λ plays a role in the stopping condition. Indeed, **ABRA-r** is the same as **ABRA-s**, with the only crucial difference in the definition of the quantity Δ_{S_i} , which is now:

$$\Delta_{S_i} = 2 \min_{s \in \mathbb{R}^+} \frac{1}{s} \ln \sum_{\mathbf{v} \in \mathcal{V}} \exp \left(\frac{s^2 \|\mathbf{v}\|^2}{\lambda 2 S_i^2} \right) + \frac{3}{\lambda} \sqrt{i \frac{\ln(4/\delta)}{2 S_i}} . \quad (13)$$

THEOREM 5. *Let $\tilde{B} = \{\tilde{\mathbf{b}}(w), w \in V\}$ be the output of **ABRA-r**. Then \tilde{B} is such that*

$$\Pr \left(\exists w \in V : \frac{|\tilde{\mathbf{b}}(w) - \mathbf{b}(w)|}{\max\{\lambda, \mathbf{b}(w)\}} > \varepsilon \right) < \delta .$$

The proof follows the same steps as the proof for Thm. 4, using the above definition of Δ_{S_i} and applying Thms. 11 and 12 from Appendix D of the extended online version [25] instead of Thms. 2 and 3.

Let ℓ'' be the k^{th} highest value $\tilde{\mathbf{b}}(w)$ returned by **ABRA-r** (ties broken arbitrarily) and let $\tilde{b}'' = \ell'' / (1 + \varepsilon)$. **ABRA-k** then returns the set

$$\widetilde{\text{TOP}}(k, G) = \{(w, \tilde{\mathbf{b}}(w)) : w \in V \text{ and } \tilde{\mathbf{b}}(w) \geq \tilde{b}''\} .$$

We have the following result showing the properties of the collection $\widetilde{\text{TOP}}(k, G)$.

THEOREM 6. *With probability at least $1 - \delta$, the set $\widetilde{\text{TOP}}(k, G)$ is such that:*

1. *for any pair $(v, \mathbf{b}(v)) \in \text{TOP}(k, G)$, there is one pair $(v, \tilde{\mathbf{b}}(v)) \in \widetilde{\text{TOP}}(k, G)$ (i.e., we return a superset of the top- k nodes with highest betweenness) and this pair is such that $|\tilde{\mathbf{b}}(v) - \mathbf{b}(v)| \leq \varepsilon \mathbf{b}(v)$;*
2. *for any pair $(w, \tilde{\mathbf{b}}(w)) \in \widetilde{\text{TOP}}(k, G)$ such that $(w, \mathbf{b}(w)) \notin \text{TOP}(k, G)$ (i.e., any false positive) we have that $\tilde{\mathbf{b}}(w) \leq (1 + \varepsilon) \mathbf{b}_k$ (i.e., the false positives, if any, are among the nodes returned by **ABRA-k** with lower BC estimation).*

The proof and the pseudocode for **ABRA-k** can be found in Appendix A of the extended online version [25].

4.3 Special Cases

In this section we consider some special restricted settings that make computing an high-quality approximation of the BC of all nodes easier. One example of such restricted settings is when the graph is *undirected* and every pair of distinct nodes is either connected with a *single* SP or there is no path between the two nodes (because they belong to different connected components). Examples of these settings are many road networks, where the unique SP condition is often enforced [14]. Riondato and Kornaropoulos [24, Lemma 2] showed that, in this case, the number of samples needed to compute a high-quality approximation of the BC of all nodes is *independent* of any property of the graph, and only depends on the quality controlling parameters ε and δ . The algorithm by Riondato and Kornaropoulos [24] works differently from **ABRA-s**, as it samples one SP at a time and only updates the BC estimation of nodes along this path, rather than sampling a pair of nodes and updating the estimation of all nodes on any SPs between the sampled nodes. Nevertheless, as shown in the following theorem, we can actually even generalize the result by Riondato and Kornaropoulos [24], as shown in Thm. 7. The statement and the proof of this theorem use pseudodimension [23], an extension of the Vapnik-Chervonenkis (VC) dimension to real-valued functions. Details about pseudodimension and the proof of Thm. 7 can be found in Appendix B of the extended online version [25]. Corollary 1 shows how to modify **ABRA-s** to take Thm. 7 into account.

THEOREM 7. *Let $G = (V, E)$ be a graph such that it is possible to partition the set $\mathcal{D} = \{(u, v) \in V \times V, u \neq v\}$ in two classes: a class $A = \{(u^*, v^*)\}$ containing a single pair of different nodes (u^*, v^*) such that $\sigma_{u^*v^*} \leq 2$ (i.e., connected by either at most two SPs or not connected), and a class $B = \mathcal{D} \setminus A$ of pairs (u, v) of nodes with $\sigma_{uv} \leq 1$ (i.e., either connected by a single SP or not connected). Then the pseudodimension of the family of functions*

$$\{f_w : \mathcal{D} \rightarrow [0, 1], w \in V\},$$

where f_w is defined as in (9), is at most 3.

Let	$\begin{cases} z = 48\omega_i^* + (1 + 4\varepsilon)^2 \\ w = -1 - 12\varepsilon + 8(27(\omega_i^*)^2 + (21 - 8\varepsilon)\varepsilon^2 + 18\omega_i^*(1 + \varepsilon)) \\ y = 12\sqrt{3} - 1 + 2\omega_i^* + 2\varepsilon \sqrt{-(27(\omega_i^*)^2 - \varepsilon^2(1 + 16\varepsilon) - \omega_i^*(1 + 18\varepsilon))} \\ \theta = \arg(-w + jy)/3 \text{ where } j \text{ is the imaginary unity and } \arg(\ell) \text{ is the argument of the complex number } \ell \end{cases}$
Root 1	$\frac{1}{3}(i + 2)(\ln \frac{2}{\delta})((1 + 4\varepsilon) - \sqrt{z} \cos \theta)(\omega_i^* - \varepsilon)^{-2}$
Root 2	$\frac{1}{6}(i + 2)(\ln \frac{2}{\delta})(2(1 + 4\varepsilon) + \sqrt{z}(\cos \theta + \sqrt{3} \sin \theta))(\omega_i^* - \varepsilon)^{-2}$
Root 3	$\frac{1}{6}(i + 2)(\ln \frac{2}{\delta})(2(1 + 4\varepsilon) + \sqrt{z}(\cos \theta - \sqrt{3} \sin \theta))(\omega_i^* - \varepsilon)^{-2}$

Table 1: Roots of the cubic equation (12) for the computation of the next sample size.

COROLLARY 1. *Suppose to augment ABRA-s with the additional stopping condition instructing to return the set $\tilde{B} = \{\mathbf{b}(w), w \in V\}$ after a total of*

$$r = \frac{c}{\varepsilon^2} \left(3 + \ln \frac{1}{\delta} \right)$$

pairs of nodes have been sampled from \mathcal{D} . The set \tilde{B} is an (ε, δ) -approximation.

The bound in Thm. 7 is strict, i.e., there exists a graph for which the pseudodimension is exactly 3 [24, Lemma 4]. Moreover, as soon as we relax the requirement in Thm. 7 and allow two pairs of nodes to be connected by two SPs, there are graphs with pseudodimension 4 (Lemma 4 in Appendix B of the extended online version [25]).

For the case of *directed* networks, it is currently an open question whether a high-quality (i.e., within ε) approximation of the BC of all nodes can be computed from a sample whose size is independent of properties of the graph, but it is known that, even if possible, the constant would not be the same as for the undirected case [24, Sect. 4.1].

We conjecture that, given some information on how many pair of nodes are connected by x shortest paths, for $x \geq 0$, it should be possible to derive a strict bound on the pseudodimension associated to the graph.

4.4 Improved Estimators

Geisberger et al. [14] present an improved estimator for BC using random sampling. Their experimental results show that the quality of the approximation is significantly improved, but they do not present any theoretical analysis. Their algorithm, which follows the work of Brandes and Pich [10] differs from ours as it samples nodes and performs a Single-Source-Shortest-Paths (SSSP) computation from each of the sampled nodes. We can use an adaptation of their estimator in a variant of our algorithm, and we can prove that this variant is still probabilistically guaranteed to compute an (ε, δ) -approximation of the BC of all nodes, therefore removing the main limitation of the original work, which offered no quality guarantees. We now present this variant considering, for ease of discussion, the special case of the linear scaling estimator by Geisberger et al. [14]. This technique can be extended to the generic parameterized estimators they present.

The intuition behind the improved estimator is to increase the estimation of the BC for a node w proportionally to the ratio between the SP distance $d(u, w)$ from the first component u of the pair (u, v) to w and the SP distance $d(u, v)$ from u to v . Rather than sampling pairs of nodes, the algorithm samples triples (u, v, d) , where d is a *direction*, (either \leftarrow or \rightarrow), and updates the betweenness estimation differently depending on d , as follows. Let $\mathcal{D}' = \mathcal{D} \times \{\leftarrow, \rightarrow\}$ and

for each $w \in V$, define the function g_w from \mathcal{D}' to $[0, 1]$ as:

$$g_w(u, v, d) = \begin{cases} \frac{\sigma_{uv}(w)}{\sigma_{uv}} \frac{d(u, w)}{d(u, v)} & \text{if } d = \rightarrow \\ \frac{\sigma_{uv}(w)}{\sigma_{uv}} \left(1 - \frac{d(u, w)}{d(u, v)} \right) & \text{if } d = \leftarrow \end{cases}$$

Let \mathcal{S} be a collection of ℓ elements of \mathcal{D}' sampled uniformly and independently at random with replacement. Our estimation $\tilde{\mathbf{b}}(w)$ of the BC of a node w is

$$\tilde{\mathbf{b}}(w) = \frac{2}{\ell} \sum_{(u, v, d) \in \mathcal{S}} g_w(u, v, d) = 2m_{\mathcal{S}}(f_w) .$$

The presence of the factor 2 in the estimator calls for a single minor adjustment in the definition of Δ_{S_i} which, for this variant of ABRA-s, becomes

$$\Delta_{S_i} = \frac{\omega_i^*}{1 - \alpha_i} + \frac{(i + 1) \ln \frac{2}{\delta}}{2S_i \alpha_i (1 - \alpha_i)} + \sqrt{(i + 1) \frac{2 \ln \frac{2}{\delta}}{S_i}}$$

i.e., w.r.t. the original definition of Δ_{S_i} , there is an additional factor 4 inside the square root of the third term on the r.h.s..

The output of this variant of ABRA-s is still a high-quality approximation of the BC of all nodes, i.e., Thm. 4 still holds with this new definition of Δ_{S_i} . This is due to the fact that the results on the Rademacher averages presented in Sect. 3.2 can be extended to families of functions whose co-domain is an interval $[a, b]$, rather than just $[0, 1]$ [28].

5. DYNAMIC GRAPH BC APPROXIMATION

In this section we present an algorithm, named ABRA-d, that computes and keeps up to date an high-quality approximation of the BC of all nodes in a *fully dynamic graph*, i.e., in a graph where nodes and edges can be added or removed over time. Our algorithm builds on the recent work by Hayashi et al. [16], who introduced two fast data structures called the Hypergraph Sketch and the Two-Ball Index: the Hypergraph Sketch stores the BC estimations for all nodes, while the Two-Ball Index is used to store the SP DAGs and to understand which parts of the Hypergraph Sketch needs to be modified after an update to the graph (i.e., an edge or node insertion or deletion). Hayashi et al. [16] show how to populate and update these data structures to maintain an (ε, δ) -approximation of the BC of all nodes in a fully dynamic graph. Using the novel data structures results in orders-of-magnitude speedups w.r.t. previous contributions [4, 5]. The algorithm by Hayashi et al. [16] is based on a static random sampling approach which is identical to the one described for ABRA-s, i.e., pairs of nodes are sampled and the BC estimation of the nodes along the SPs between the two nodes are updated as necessary. Their analysis on the number of samples necessary to obtain an (ε, δ) -approximation of the

BC of all nodes uses the union bound, resulting in a number of samples that depends on the logarithm of the number of nodes in the graph, i.e., $O(\varepsilon^{-2}(\log(|V|/\delta)))$ pairs of nodes must be sampled.

ABRA-d builds and improves over the algorithm presented by Hayashi et al. [16] as follows. Instead of using a static random sampling approach with a fixed sample size, we use the progressive sampling approach and the stopping condition that we use in **ABRA-s** to understand when we sampled enough to first populate the Hypergraph Sketch and the Two-Ball Index. Then, after each update to the graph, we perform the same operations as in the algorithm by Hayashi et al. [16], with the crucial addition, after these operation have been performed, of keeping the set \mathcal{V}_S of vectors and the map M (already used in **ABRA-s**) up to date, and checking whether the stopping condition is still satisfied. If it is not, additional pairs of nodes are sampled and the Hypergraph Sketch and the Two-Ball Index are updated with the estimations resulting from these additional samples. The sampling of additional pairs continues until the stopping condition is satisfied, potentially according to a sample schedule either automatic, or specified by the user. As we show in Sect. 6, the overhead of additional checks of the stopping condition is minimal. On the other hand, the use of the progressive sampling scheme based on the Rademacher averages allows us to sample much fewer pairs of nodes than in the static sampling case based on the union bound: Riondato and Kornaropoulos [24] already showed that it is possible to sample much less than $O(\log |V|)$ nodes, and, as we show in our experiments, our sample sizes are even smaller than the ones by Riondato and Kornaropoulos [24]. The saving in the number of samples results in a huge speedup, as the running time of the algorithms are, in a first approximation, linear in the number of samples, and in a reduction in the amount of space required to store the data structures, as they now store information about fewer SP DAGs.

THEOREM 8. *The set $\tilde{B} = \{\tilde{\mathbf{b}}(w), w \in V\}$ returned by **ABRA-d** after each update has been processed is such that*

$$\Pr(\exists w \in V \text{ s.t. } |\tilde{\mathbf{b}}(w) - \mathbf{b}(w)| > \varepsilon) < \delta .$$

The proof follows from the correctness of the algorithm by Hayashi et al. [16] and of **ABRA-s** (Thm. 4).

6. EXPERIMENTAL EVALUATION

In this section we presents the results of our experimental evaluation. We measure and analyze the performances of **ABRA-s** in terms of its runtime and sample size and accuracy, and compared them with those of the exact algorithm **BA** [9] and the approximation algorithm **RK** [24], which offers the same guarantees as **ABRA-s** (computes an (ε, δ) -approximation the BC of all nodes).

Implementation and Environment. We implement **ABRA-s** and **ABRA-d** in C++, as an extension of the NetworKit library [29]. The code is available from <http://matteo.riondato/software/ABRA-radebetw.tbz2>. We performed the experiments on a machine with a AMD Phenom™ II X4 955 processor and 16GB of RAM, running FreeBSD 11.

Datasets and Parameters. We use graphs of various nature (communication, citations, P2P, and social networks)

from the SNAP repository [20]. The characteristics of the graphs are reported in the leftmost column of Table 2.

In our experiments we varied ε in the range [0.005, 0.3], and we also evaluate a number of different sampling schedules (see Sect. 6.2). In all the results we report, δ is fixed to 0.1. We experimented with different values for this parameter, and, as expected, it has a very limited impact on the nature of the results, given the logarithmic dependence of the sample size on δ . We performed five runs for each combination of parameters. The variance between the different runs was essentially insignificant, so we report, unless otherwise specified, the results for a random run.

6.1 Runtime and Speedup

Our main goal was to develop an algorithm that can compute an (ε, δ) -approximation of the BC of all nodes as fast as possible. Hence we evaluate the runtime and the speedup of **ABRA-s** w.r.t. **BA** and **RK**. The results are reported in columns 3 to 5 (from the left) of Table 2 (the values for $\varepsilon = 0.005$ are missing for Email-Enron and Cit-HepPh because in these case both **RK** and **ABRA-s** were slower than **BA**). As expected, the runtime is a perfect linear function of the sample size (column 9), which in turns grows as ε^{-2} . The speedup w.r.t. the exact algorithm **BA** is significant and naturally decreases quadratically with ε . More interestingly **ABRA-s** is always faster than **RK**, sometimes by a significant factor. At first, one may think that this is due to the reduction in the sample size (column 10), but a deeper analysis shows that this is only one component of the speedup, which is almost always greater than the reduction in sample size. The other component can be explained by the fact that **RK** must perform an expensive computation (computing the vertex diameter [24] of the graph) to determine the sample size before it can start sampling, while **ABRA-s** can immediately start sampling and rely on the stopping condition, whose computation is inexpensive, as we will discuss. The different speedups for different graphs are due to different characteristics of the graphs: when the SP DAG between two nodes has many paths, **ABRA-s** does more work per sample than **RK**, which only backtracks along a single SP of the DAG, hence the speedup is smaller.

Runtime breakdown. The main challenge in designing a stopping condition for progressive sampling algorithm is striking the right balance between the strictness of the condition (i.e., it should stop early) and the efficiency in evaluating it. We now comment on the efficiency, and will report about the strictness in Sect. 6.2 and 6.3. In columns 6 to 8 of Table 2 we report the breakdown of the runtime into the main components. It is evident that evaluating the stopping condition amounts to an insignificant fraction of the runtime, and most of the time is spent in computing the samples (selection of nodes, execution of SP algorithm, update of the BC estimations). The amount in the “Other” column corresponds to time spent in logging and checking invariants. We can then say that our stopping condition is extremely efficient to evaluate, and **ABRA-s** is almost always doing “real” work to improve the estimation.

6.2 Sample Size and Sample Schedule

We evaluate the final sample size of **ABRA-s** and the performances of the “automatic” sample schedule (Sect. 4.1.1). The results are reported in columns 9 and 10 of Table 2.

Graph	ϵ	Runtime (sec.)	Speedup w.r.t.		Runtime Breakdown (%)			Sample Size	Reduction w.r.t. RK	Absolute Error ($\times 10^5$)		
			BA	RK	Sampling	Stop Cond.	Other			max	avg	stddev
Soc-Epinions1 Directed V = 75, 879 E = 508, 837	0.005	483.06	1.36	2.90	99.983	0.014	0.002	110,705	2.64	70.84	0.35	1.14
	0.010	124.60	5.28	3.31	99.956	0.035	0.009	28,601	2.55	129.60	0.69	2.22
	0.015	57.16	11.50	4.04	99.927	0.054	0.018	13,114	2.47	198.90	0.97	3.17
	0.020	32.90	19.98	5.07	99.895	0.074	0.031	7,614	2.40	303.86	1.22	4.31
	0.025	21.88	30.05	6.27	99.862	0.092	0.046	5,034	2.32	223.63	1.41	5.24
	0.030	16.05	40.95	7.52	99.827	0.111	0.062	3,668	2.21	382.24	1.58	6.37
P2p-Gnutella31 Directed V = 62, 586 E = 147, 892	0.005	100.06	1.78	4.27	99.949	0.041	0.010	81,507	4.07	38.43	0.58	1.60
	0.010	26.05	6.85	4.13	99.861	0.103	0.036	21,315	3.90	65.76	1.15	3.13
	0.015	11.91	14.98	4.03	99.772	0.154	0.074	9,975	3.70	109.10	1.63	4.51
	0.020	7.11	25.09	3.87	99.688	0.191	0.121	5,840	3.55	130.33	2.15	6.12
	0.025	4.84	36.85	3.62	99.607	0.220	0.174	3,905	3.40	171.93	2.52	7.43
	0.030	3.41	52.38	3.66	99.495	0.262	0.243	2,810	3.28	236.36	2.86	8.70
Email-Enron Undirected V = 36, 682 E = 183, 831	0.010	202.43	1.18	1.10	99.984	0.013	0.003	66,882	1.09	145.51	0.48	2.46
	0.015	91.36	2.63	1.09	99.970	0.024	0.006	30,236	1.07	253.06	0.71	3.62
	0.020	53.50	4.48	1.05	99.955	0.035	0.010	17,676	1.03	290.30	0.93	4.83
	0.025	31.99	7.50	1.11	99.932	0.052	0.016	10,589	1.10	548.22	1.21	6.48
	0.030	24.06	9.97	1.03	99.918	0.061	0.021	7,923	1.02	477.32	1.38	7.34
Cit-HepPh Undirected V = 34, 546 E = 421, 578	0.010	215.98	2.36	2.21	99.966	0.030	0.004	32,469	2.25	129.08	1.72	3.40
	0.015	98.27	5.19	2.16	99.938	0.054	0.008	14,747	2.20	226.18	2.49	5.00
	0.020	58.38	8.74	2.05	99.914	0.073	0.013	8,760	2.08	246.14	3.17	6.39
	0.025	37.79	13.50	2.02	99.891	0.091	0.018	5,672	2.06	289.21	3.89	7.97
	0.030	27.13	18.80	1.95	99.869	0.108	0.023	4,076	1.99	359.45	4.45	9.53

Table 2: Runtime, speedup, breakdown of runtime, sample size, reduction, and absolute error

As expected, the sample size grows with ϵ^{-2} . We already commented on the fact that **ABRA-s** uses a sample size that is consistently (up to 4 \times) smaller than the one used by **RK** and how this is part of the reason why **ABRA-s** is much faster than **RK**. In Fig. 1 we show the behavior (on P2p-Gnutella31, figures for other graphs can be found in Appendix C of the extended online version [25]) of the final sample size chosen by the automatic sample schedule in comparison with *static geometric sample schedules*, i.e., schedules for which the sample size at iteration $i+1$ is c times the size of the sample size at iteration i . We can see that the *automatic sample schedule is always better than the geometric ones*, sometimes significantly depending on the value of c (e.g., more than 2 \times decrease w.r.t. using $c = 3$ for $\epsilon = 0.05$). Effectively this means that the automatic sample schedule really frees the user from having to selecting a parameter whose impact on the performances of the algorithm may be devastating (larger final sample size implies higher runtime). Moreover, thanks to the automatic sample schedule, **ABRA-s** always terminates after just two iterations, while this was not the case for the geometric sample schedules (taking even 5 iterations in some cases): this means that effectively the automatic sample schedules “jumps” directly to a sample size for which the stopping condition will be verified. We can sum up the results and say that the stopping condition of **ABRA-s** stops at small sample sizes, smaller than those used in **RK**, and the automatic sample schedule we designed is extremely efficient at choosing the right successive sample size.

6.3 Accuracy

We evaluate the accuracy of **ABRA-s** by measuring the absolute error $|\widehat{b}(v) - b(v)|$. The theoretical analysis guarantees that this quantity should be at most ϵ for all nodes, with probability at least $1 - \delta$. A first important result is that in *all* the thousands of runs of **ABRA-s**, the maximum

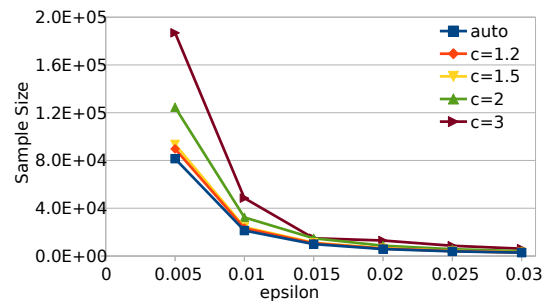


Figure 1: Final sample size for different sample schedules on P2p-Gnutella

error was *always* smaller than ϵ (not just with probability $> 1 - \delta$). We report statistics about the absolute error in the three rightmost columns of Table 2 and in Fig. 2 (figures for the other graphs are in Appendix C of the extended online version [25]). The minimum error (not reported) was always 0. The maximum error is *an order of magnitude smaller than ϵ* , and the average error is around *three orders of magnitude smaller than ϵ* , with a very small standard deviation. As expected, the error grows as ϵ^{-2} . In Fig. 2 we show the behavior of the maximum, average, and average plus three standard deviations (approximately corresponding to the 95% percentile) for Soc-Epinions1 (the vertical axis has a logarithmic scale), to appreciate how most of the errors are almost two orders of magnitude smaller than ϵ .

All these results show that **ABRA-s** is *very accurate, more than what is guaranteed by the theoretical analysis*. This can be explained by the fact that the bounds to the sampling size, the stopping condition, and the sample schedule are *conservative*, in the sense that **ABRA-s** may be sampling more than necessary to obtain an (ϵ, δ) -approximation. Tightening any of these components would result in a less

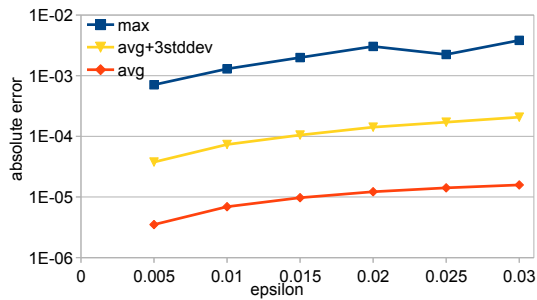


Figure 2: Absolute error evaluation – Soc-Epinions1

conservative algorithm that offers the same approximation quality guarantees, and is an interesting research direction.

6.4 Dynamic BC Approximation

We did not evaluate ABRA-d experimentally, but, given its design, it is reasonable to expect that, when compared to previous contributions offering the same quality guarantees [5, 16], it would exhibit similar or even larger speedups and reductions in the sample size than what ABRA-s had w.r.t. RK. Indeed, the algorithm by Bergamini and Meyerhenke [4] uses RK as a building block and it needs to constantly keep track of (an upper bound on) the vertex diameter of the graph, a very expensive operation. On the other hand, the analysis of the sample size by Hayashi et al. [16] uses very loose simultaneous deviation bounds (the union bound). As already shown by Riondato and Kornaropoulos [24], the resulting sample size is extremely large and they already showed how RK can use a smaller sample size. Since we built over the work by Hayashi et al. [16] and ABRA-s improves over RK, we can reasonably expect it to have better performances than the algorithm by Hayashi et al. [16]

7. CONCLUSIONS

We presented ABRA, a family of sampling-based algorithms for computing and maintaining high-quality approximations of (variants of) the BC of all nodes in static and dynamic graphs with updates (both deletions and insertions). We discussed a number of variants of our basic algorithms, including finding the top- k nodes with higher BC, using improved estimators, and special cases when there is a single SP. ABRA greatly improves, theoretically and experimentally, the current state of the art. The analysis relies on Rademacher averages and on pseudodimension. To our knowledge this is the first application of these concepts to graph mining. In the future we plan to investigate stronger bounds to the Rademacher averages, give stricter bounds to the sample complexity of BC by studying the pseudodimension of the class of functions associated to it, and extend our study to other network measures.

Acknowledgements. This work was supported in part by NSF grant IIS-1247581 and NIH grant R01-CA180776.

8. REFERENCES

- [1] D. Anguita, A. Ghio, L. Oneto, and S. Ridella. A deep connection between the Vapnik-Chervonenkis entropy and the Rademacher complexity. *IEEE Trans. Neural Netw. Learn. Sys.*, 25(12):2202–2211, 2014.
- [2] J. M. Anthonisse. The rush in a directed graph. TR BN 9/71, Stichting Mathematisch Centrum, 1971.
- [3] D. A. Bader, S. Kintali, K. Madduri, and M. Mihail. Approximating betweenness centrality. *Algorithms and Models for the Web-Graph*, 124–137, 2007.
- [4] E. Bergamini and H. Meyerhenke. Fully-dynamic approximation of betweenness centrality. *ESA’15*.
- [5] E. Bergamini and H. Meyerhenke. Approximating betweenness centrality in fully-dynamic networks. *Internet Mathematics*, to appear.
- [6] E. Bergamini, H. Meyerhenke, and C. L. Staudt. Approximating betweenness centrality in large evolving networks. *ALENEX’15*, 2015.
- [7] S. Boucheron, O. Bousquet, and G. Lugosi. Theory of classification : A survey of some recent advances. *ESAIM: Probability and Statistics*, 9:323–375, 2005.
- [8] S. Boyd and L. Vandenberghe. *Convex optimization*. Cambridge university press, 2004.
- [9] U. Brandes. A faster algorithm for betweenness centrality. *J. Math. Sociol.*, 25(2):163–177, 2001.
- [10] U. Brandes and C. Pich. Centrality estimation in large networks. *Int. J. Bifurc. Chaos*, 17(7):2303–2318, 2007.
- [11] T. Elomaa and M. Kääriäinen. Progressive Rademacher sampling. *AAAI’2001*, 2001.
- [12] D. Erdős, V. Ishakian, A. Bestavros, and E. Terzi. A divide-and-conquer algorithm for betweenness centrality. *SDM’15*, 2015.
- [13] L. C. Freeman. A set of measures of centrality based on betweenness. *Sociometry*, 40:35–41, 1977.
- [14] R. Geisberger, P. Sanders, and D. Schultes. Better approximation of betweenness centrality. *ALENEX’08*, 2008.
- [15] O. Green, R. McColl, and D. Bader. A fast algorithm for streaming betweenness centrality. *PASSAT’12*, 2012.
- [16] T. Hayashi, T. Akiba, and Y. Yoshida. Fully dynamic betweenness centrality maintenance on massive networks. *VLDB’16*, 2015.
- [17] M. Kas, M. Wachs, K. M. Carley, and L. R. Carley. Incremental algorithm for updating betweenness centrality in dynamically growing networks. *ASONAM’13*, 2013.
- [18] N. Kourtellis, G. D. F. Morales, and F. Bonchi. Scalable online betweenness centrality in evolving graphs. *IEEE Trans. Knowl. Data Eng.*, 27(9):2494–2506, 2015.
- [19] M.-J. Lee, J. Lee, J. Y. Park, R. H. Choi, and C.-W. Chung. QUBE: A quick algorithm for updating betweenness centrality. *WWW’12*, 2012.
- [20] J. Leskovec and A. Krevl. SNAP Datasets <http://snap.stanford.edu/data>, 2014.
- [21] M. E. J. Newman. *Networks – An Introduction*. Oxford University Press, 2010.
- [22] L. Oneto, A. Ghio, D. Anguita, and S. Ridella. An improved analysis of the Rademacher data-dependent bound using its self bounding property. *Neur. Netw.*, 44:0, 2013.
- [23] D. Pollard. *Convergence of stochastic processes*. Springer-Verlag, 1984.
- [24] M. Riondato and E. M. Kornaropoulos. Fast approximation of betweenness centrality through sampling. *Data Mining Knowl. Disc.*, 30(2) 438–475, 2015.
- [25] M. Riondato and E. Upfal. Approximating betweenness centrality in static and dynamic graphs with Rademacher averages (extended version) <http://riondato.to/papers/RiondatoUpfal-ABRA-ext.pdf>, 2016.
- [26] M. Riondato and E. Upfal. Mining frequent itemsets through progressive sampling with Rademacher averages. *KDD’15*, 2015.
- [27] A. E. Sariyüce, E. Saule, K. Kaya, and U. V. Çatalyürek. Shattering and compressing networks for betweenness centrality. *SDM’15*, 2015.
- [28] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.
- [29] C. Staudt, A. Sazonovs, and H. Meyerhenke. NetworKit: A tool suite for high-performance network analysis. *Network Science*, to appear.
- [30] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1999.