Rademacher Averages: Theory and Practice

Matteo Riondato – Labs, Two Sigma Investments LP

Dagstuhl – April 6, 2017

Outline

I. Data mining, sampling, and issues thereof

What happens in data mining?

GIVEN:

- Dataset $\mathcal{D}$
- Family $\mathcal{F}$ of functions $f : \mathcal{D} \to [a, b] \subseteq \mathbb{R}$

GOAL: Compute $\mathrm{m}_{\mathcal{D}}(f) = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} f(x)$, for each $f \in \mathcal{F}$

EXAMPLES:

- $\mathcal{D}$ is a database, $\mathcal{F}$ is a family of SQL queries;
- $\mathcal{D}$ is a graph $G = (V, E)$, $\mathcal{F}$ contains an f_v for each $v \in V$

ISSUE: *Exact* computation of *every* $\mathrm{m}_{\mathcal{D}}(f)$, $f \in \mathcal{F}$, is *too expensive on large datasets*

What should we aim for?

INTUITION: in many applications, *high-quality approximations* are sufficient

Definition (ε-approximation)

Given $\varepsilon \in (0,1)$, a ε-approximation to $\mathcal{F}$ on $\mathcal{D}$ is a collection

$$\tilde{B} = \{\tilde{m}(f), f \in \mathcal{F}\}$$

such that

$$|\tilde{m}(f) - \mathrm{m}_{\mathcal{D}}(f)| \leq \varepsilon, \text{ for each } f \in \mathcal{F}$$

A definition with relative/multiplicative error is also possible

How can we get a ε-approximation?

SAMPLING!

HOW TO PROCEED:

1. Define a probability distribution π on $\mathcal{D}$
2. Create a sample $\mathcal{S}$ by sampling *enough* points independently from $\mathcal{D}$ according to π
3. Compute $\tilde{m}(f)$ on $\mathcal{S}$, usually as $\mathrm{m}_{\mathcal{S}}(f) = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} f(x)$

WE NEED:

1. A suitable π
2. An efficient sampling scheme
3. A *sample size* $|\mathcal{S}|$, sufficient for $\tilde{B}$ to be an ε-approximation, *with prob.* $\geq 1 - \delta$
   ... or a *stopping condition* to understand whether $\mathcal{S}$ is *large enough*

Isn't it obvious how to do it?

We want to compute a quantity ℓ such that

$$\Pr\left(\exists f \in \mathcal{F} \text{ s.t. } |\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)| > \varepsilon\right) < \delta$$

The probability is taken over all samples of size ℓ.

IDEA: Use *tail bounds* for a single f, and then the *union bound* over $\mathcal{F}$

(SPOILER: It won't be that easy)

Let's make an example

Let each $f = f_A \in \mathcal{F}$ be the indicator function for some property A
($f(x) = 1$ if $x \in \mathcal{D}$ satisfies A, 0 othw.)

Then $\mathrm{m}_{\mathcal{D}}(f)$ is a proportion and $|\mathcal{S}|\mathrm{m}_{\mathcal{S}}(f)$ has a Binomial distribution $B(|\mathcal{S}|, \mathrm{m}_{\mathcal{D}}(f))$

Apply the Chernoff bound and then the union bound over $\mathcal{F}$:

$$\begin{aligned}\Pr\left(\exists f \in \mathcal{F} \text{ s.t. } |\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)| > \varepsilon\right) &\leq \sum_{f \in \mathcal{F}} \Pr\left(|\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)| > \varepsilon\right) \\ &\leq |\mathcal{F}| 2 \exp\left(-|\mathcal{S}|\varepsilon^2/3\right)\end{aligned}$$

For the r.h.s. to be at most δ, it must be

$$|\mathcal{S}| \geq \frac{3}{\varepsilon^2}\left(\ln|\mathcal{F}| + \ln\frac{1}{\delta}\right)$$

What's disappointing with this sample size?

$$|\mathcal{S}| \geq \frac{3}{\varepsilon^2}\left(\ln|\mathcal{F}| + \ln\frac{1}{\delta}\right)$$

What's disappointing with this sample size?

$$|\mathcal{S}| \geq \frac{3}{\varepsilon^2}\left(\ln|\mathcal{F}| + \ln\frac{1}{\delta}\right)$$

1. $\mathcal{F}$ may be *infinite*. E.g., in the classification setting:

   $x = (w, y)$, $y \in \{0, 1\}$, and $f_\theta(x)$ is the *loss* of a classifier c_θ on x, $\theta \in \Theta \subseteq R^\ell$

   ($\mathcal{S}$ is the training set, $|\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)|$ is the *generalization error*)

What's disappointing with this sample size?

$$|\mathcal{S}| \geq \frac{3}{\varepsilon^2}\left(\ln|\mathcal{F}| + \ln\frac{1}{\delta}\right)$$

1. $\mathcal{F}$ may be *infinite*. E.g., in the classification setting:

   $x = (w, y)$, $y \in \{0, 1\}$, and $f_\theta(x)$ is the *loss* of a classifier c_θ on x, $\theta \in \Theta \subseteq R^\ell$

   ($\mathcal{S}$ is the training set, $|\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)|$ is the *generalization error*)

2. The sample size does *not depend on any characteristic of* $\mathcal{D}$

   E.g., on properties of the graph $G = \mathcal{D}$

Why should we want such dependency on $\mathcal{D}$?

To *let the data speak*!

For a fixed $\mathcal{F}$, computing an ε-approximation to $\mathcal{F}$ may be more difficult on $\mathcal{D}_1$ than on $\mathcal{D}_2$;

$\mathcal{D}$ can give information on the *sample complexity* of computing an ε-approximation

"$\ln |\mathcal{F}|$" is a rough *measure of complexity* of the task, as it ignores $\mathcal{D}$

Are there better measures of sample complexity?

Yes! *Statistical learning theory* is "all" about them:

VC-dimension, pseudodimension, covering numbers, *Rademacher averages*, ...

They allow to replace "$\ln |\mathcal{F}|$" with $g(\mathcal{F}, \mathcal{D})$ or even $g(\mathcal{F}, \mathcal{S})$: *let the data speak*!

CHALLENGES:

1) Developed for *supervised learning*;
2) Long-standing reputation of being *only of theoretical interest*;
3) Not exactly straightforward to interpret, compute, bound

Are there better measures of sample complexity?

Yes! *Statistical learning theory* is "all" about them:

VC-dimension, pseudodimension, covering numbers, *Rademacher averages*, ...

They allow to replace "$\ln|\mathcal{F}|$" with $g(\mathcal{F},\mathcal{D})$ or even $g(\mathcal{F},\mathcal{S})$: *let the data speak*!

Challenges:

1) Developed for *supervised learning*;
2) Long-standing reputation of being *only of theoretical interest*;
3) Not exactly straightforward to interpret, compute, bound:

$$\sup_{f\in\mathcal{F}}|\mathrm{m}_{\mathcal{S}}(f)-\mathrm{m}_{\mathcal{D}}(f)|\leq 2\mathbb{E}_{\boldsymbol{\lambda}}\left[\sup_{f\in\mathcal{F}}\frac{1}{\ell}\sum_{i=1}^{\ell}\lambda_i f(x_i)\right]+\sqrt{\frac{\ln\frac{3}{\delta}}{2\ell}}$$

Appeal:

Elegant, insightful theory: "Nothing is more practical than a good theory."

II. Rademacher Averages

What are Rademacher averages?

A binary r.v. λ has a *Rademacher* distribution if $\Pr(\lambda = 1) = \Pr(\lambda = -1) = 1/2$

Let $\ell = |\mathcal{S}|$ and let $\lambda_1, \ldots, \lambda_\ell$ be ℓ independent Rademacher r.v.'s.

Definition (Rademacher Average)

The *Rademacher average* of $\mathcal{F}$ is

$$\mathcal{R}_\ell(\mathcal{F}) = \mathbb{E}_{\mathcal{S},\lambda}\left[\max_{f\in\mathcal{F}} \frac{1}{\ell}\sum_{i=1}^{\ell} \lambda_i f(x_i)\right]$$

Rademacher averages enjoy enormous success in statistical learning theory

How do we get to such weird definition?

We have

$$\Pr\left(\exists f \in \mathcal{F} \text{ s.t. } |\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)| > \varepsilon\right) = \Pr\left(\max_{f\in\mathcal{F}} |\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)| > \varepsilon\right)$$
$$= \Pr\left(\max_{f\in\mathcal{F}} [\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)] > \varepsilon\right) + \Pr\left(\max_{f\in\mathcal{F}} [\mathrm{m}_{\mathcal{D}}(f) - \mathrm{m}_{\mathcal{S}}(f)] > \varepsilon\right)$$

Let's look at

$$\mathbb{E}_{\mathcal{S}}\left[\max_{f\in\mathcal{F}} [\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{D}}(f)]\right] = \mathbb{E}_{\mathcal{S}}\left[\max_{f\in\mathcal{F}} [\mathrm{m}_{\mathcal{S}}(f) - \mathbb{E}_{\mathcal{S}}[\mathrm{m}_{\mathcal{S}}(f)]]\right]$$

Theorem

$$\mathbb{E}_{\mathcal{S}}\left[\max_{f\in\mathcal{F}} [m_{\mathcal{S}}(f) - \mathbb{E}_{\mathcal{S}}[m_{\mathcal{S}}(f)]]\right] \leq 2\mathcal{R}_{\ell}(\mathcal{F})$$

Proof idea

Let $\mathcal{S}' = \{x'_1, \ldots, x'_\ell\}$ be a second sample, independent from $\mathcal{S}$

$$\mathbb{E}_{\mathcal{S}}\left[\max_{f\in\mathcal{F}}\left[\mathrm{m}_{\mathcal{S}}(f) - \mathbb{E}_{\mathcal{S}}\left[\mathrm{m}_{\mathcal{S}}(f)\right]\right]\right] = \mathbb{E}_{\mathcal{S}}\left[\max_{f\in\mathcal{F}}\left[\mathrm{m}_{\mathcal{S}}(f) - \mathbb{E}_{\mathcal{S}'}\left[\mathrm{m}_{\mathcal{S}'}(f)\right]\right]\right]$$

(Jensen's Inequality)
$$\leq \mathbb{E}_{\mathcal{S},\mathcal{S}'}\left[\max_{f\in\mathcal{F}}\left[\mathrm{m}_{\mathcal{S}}(f) - \mathrm{m}_{\mathcal{S}'}(f)\right]\right]$$
$$\leq \mathbb{E}_{\mathcal{S},\mathcal{S}'}\left[\max_{f\in\mathcal{F}}\left[\frac{1}{\ell}\sum_{i=1}^{\ell} f(x_i) - \frac{1}{\ell}\sum_{i=1}^{\ell} f(x'_i)\right]\right]$$
$$\leq \mathbb{E}_{\mathcal{S},\mathcal{S}',\lambda}\left[\max_{f\in\mathcal{F}}\left[\frac{1}{\ell}\sum_{i=1}^{\ell} \lambda_i\left(f(x_i) - f(x'_i)\right)\right]\right]$$
$$\leq \mathbb{E}_{\mathcal{S},\lambda}\left[\max_{f\in\mathcal{F}}\left[\frac{1}{\ell}\sum_{i=1}^{\ell} \lambda_i f(x_i)\right]\right] + \mathbb{E}_{\mathcal{S}',\lambda}\left[\max_{f\in\mathcal{F}}\left[\frac{1}{\ell}\sum_{i=1}^{\ell} \lambda_i f(x'_i)\right]\right]$$
$$\leq 2\mathcal{R}_\ell(\mathcal{F})$$

Where do we go from here?

Theorem (McDiarmid's inequality)

Let $X_1, \ldots, X_\ell$ be independent random variables and let $h(x_1, \ldots, x_\ell)$ be a function s.t. a change in variable x_i can change the value of the function by no more than c_i:

$$\sup_{x_1,\ldots,x_\ell,x_i'} \left|h(x_1, \ldots, x_i, \ldots, x_\ell) - h(x_1, \ldots, x_i', \ldots, x_\ell)\right| \leq c_i \ .$$

Then, for any $\epsilon > 0$

$$\Pr\left(h(X_1, \ldots, X_\ell) - \mathbb{E}[h(X_1, \ldots, X_\ell)] > \epsilon\right) \leq \exp\left(-2\epsilon^2 \Big/ \sum_{i=1}^{\ell} c_i^2\right) \ .$$

How do we use McDiarmid's inequality?

Recall the *bounded differences condition* from McDiarmid's inequality:

$$\sup_{x_1,\ldots,x_\ell,x_i'} \left|h(x_1,\ldots,x_i,\ldots,x_\ell) - h(x_1,\ldots,x_i',\ldots,x_\ell)\right| \leq c_i \ .$$

The function $h(x_1,\ldots,x_\ell) = h(\mathcal{S}) = \max_{f\in\mathcal{F}} [\mathsf{m}_{\mathcal{S}}(f) - \mathsf{m}_{\mathcal{D}}(f)]$
satisfies the condition with $c_i = \frac{|b-a|}{\ell}$ (in the rest of the talk we assume $|b-a| = 1$)

The same holds for the function $h(x_1,\ldots,x_\ell) = h(\mathcal{S}) = \mathbb{E}_\lambda \left[\max_{f\in\mathcal{F}} \frac{1}{\ell}\sum_{i=1}^{\ell} \lambda_i f(x_i)\right]$

This function is the *empirical Rademacher average* of $\mathcal{F}$ on $\mathcal{S}$.
We denote it with $\mathcal{R}_{\mathcal{S}}(\mathcal{F})$. Its expectation is $\mathsf{R}_\ell(\mathcal{F})$.

Let's put everything together

Let's start from

$$\mathbb{E}\left[\max_{f\in\mathcal{F}}\left[\mathsf{m}_{\mathcal{S}}(f)-\mathsf{m}_{\mathcal{D}}(f)\right]\right]\leq 2\mathcal{R}_{\ell}(\mathcal{F})$$

Now apply McDiarmid to $\max_{f\in\mathcal{F}}\left[\mathsf{m}_{\mathcal{S}}(f)-\mathsf{m}_{\mathcal{D}}(f)\right]$. With probability $\geq 1-\delta/3$:

$$\max_{f\in\mathcal{F}}\left[\mathsf{m}_{\mathcal{S}}(f)-\mathsf{m}_{\mathcal{D}}(f)\right]\leq 2\mathcal{R}_{\ell}(\mathcal{F})+\sqrt{\frac{\ln\frac{3}{\delta}}{\ell}}$$

Now apply McDiarmid to $\mathcal{R}_{\mathcal{S}}(\mathcal{F})$. With probability at least $\geq 1-2\delta/3$:

$$\max_{f\in\mathcal{F}}\left[\mathsf{m}_{\mathcal{S}}(f)-\mathsf{m}_{\mathcal{D}}(f)\right]\leq 2\mathcal{R}_{\mathcal{S}}(\mathcal{F})+3\sqrt{\frac{\ln\frac{3}{\delta}}{\ell}}$$

Applying McDiarmid to $\max_{f\in\mathcal{F}}\left[\mathsf{m}_{\mathcal{D}}(f)-\mathsf{m}_{\mathcal{S}}(f)\right]$. With probability at least $\geq 1-\delta$:

$$\max_{f\in\mathcal{F}}\left|\mathsf{m}_{\mathcal{S}}(f)-\mathsf{m}_{\mathcal{D}}(f)\right|\leq 2\mathcal{R}_{\mathcal{S}}(\mathcal{F})+3\sqrt{\frac{\ln\frac{3}{\delta}}{\ell}}$$

Formally...

Theorem

Let $\mathcal{S}$ be a fixed sample of size ℓ. With probability at least $1-\delta$,

$$\max_{f\in\mathcal{F}} |m_{\mathcal{S}}(f) - m_{\mathcal{D}}(f)| \leq 2\mathcal{R}_{\mathcal{S}}(\mathcal{F}) + 3\sqrt{\frac{\ln\frac{3}{\delta}}{\ell}}$$

The quantity on the r.h.s. depends only on the sample $\mathcal{S}$.

We can compute the "quality" of a sample from the sample itself!

... but how do we compute $\mathcal{R}_{\mathcal{S}}(\mathcal{F})$?

How do we compute $\mathcal{R}_{\mathcal{S}}(\mathcal{F})$?

Given $\mathcal{S}$, define, for each $f \in \mathcal{F}$,

$$\mathbf{v}_f = (f(x_1), \ldots, f(x_\ell))$$

Consider the *set*

$$\mathcal{V}_{\mathcal{S}} = \{\mathbf{v}_f, f \in \mathcal{F}\}$$

If the co-domain of the functions $f \in \mathcal{F}$ is finite, then $\mathcal{V}_{\mathcal{S}}$ is finite

Then $|\mathcal{V}_{\mathcal{S}}| \leq |\mathcal{F}|$, and usually $|\mathcal{V}_{\mathcal{S}}| \ll |\mathcal{F}|$.

Theorem (Massart's Finite Class Lemma)

$$\mathcal{R}_{\mathcal{S}}(\mathcal{F}) \leq \max_{\mathbf{v} \in \mathcal{V}_{\mathcal{S}}} \|\mathbf{v}\|_2 \frac{\sqrt{2 \ln |\mathcal{V}_{\mathcal{S}}|}}{\ell}$$

If we can keep track of $\mathcal{V}_{\mathcal{S}}$, or bound the max. $\|\cdot\|_2$, we have a bound to $\mathcal{R}_{\mathcal{S}}(\mathcal{F})$.

Why does it make sense?

$$\mathcal{R}_{\mathcal{S}}(\mathcal{F}) = \mathbb{E}_{\lambda}\left[\max_{f\in\mathcal{F}} \frac{1}{\ell}\sum_{i=1}^{\ell} \lambda_i f(x_i)\right]$$

Assume $\mathcal{F}$ contains classifiers from $\mathbb{R}$ to $\{-1, 1\}$.

Assume that $\lambda_1, \ldots, \lambda_\ell$ are the labels of training set $\mathcal{S}$. Then

$$\lambda_i f(x_i, \lambda_i) = 1$$

when f correctly classifies x_i. -1 otherwise.

$\mathcal{R}_{\mathcal{S}}(\mathcal{F})$ is high when, for any labeling $\lambda_1, \ldots, \lambda_\ell$, there is a function $f \in \mathcal{F}$ that correctly classify many points of the training set.

If this is the case, then $\mathcal{F}$ can essentially fit ℓ random noise points, so it is a rich class

Thus, learning the "correct" classifier requires more training points

Recap and comments

Rade.Avg. allow to compute a bound to the maximum deviation from the sample

For ε-approximation, keep sampling until the bound is less than ε (caveats)

Keeping track of $\mathcal{V}_{\mathcal{S}}$ is not always straightforward, but it's the key task

The bounds presented here have much tighter variants

There are relative/multiplicative error variants

III. Betweenness centrality estimation with Rademacher Averages

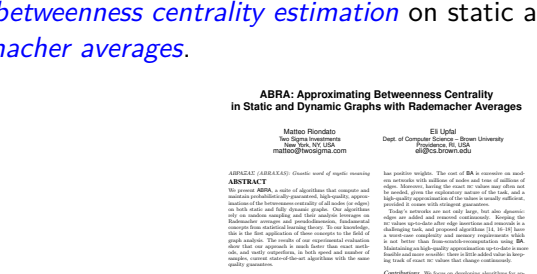
What am I going to talk about?

ABRA: A *sampling*-based algorithm for *betweenness centrality estimation* on static and dynamic *graphs*. Its analysis uses *Rademacher averages*.

Joint work with *Eli Upfal* (Brown);

ACM KDD'16;

Journal under submission,
http://bit.ly/abra-betweenness.



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

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

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

ABRAZAT, (ABRAXAS): Gnostic word of cryptic meaning

ABSTRACT

We present **ABRA**, a suite of algorithms that compute and maintain probabilistically-guaranteed, high-quality, approximations of the betweenness centrality of all nodes (or edges) of a graph (or network). The algorithms in the suite leverage only on random sampling and their analysis leverages 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 analytics. The results of our experimental evaluation show that our approach is much faster than exact methods, and vastly outperforms, in both speed and number of samples, current state-of-the-art algorithms with the same quality guarantees.

Categories and Subject Descriptors

G.2.2 [**Discrete Mathematics**]: Graph Theory—*Graph algorithms*; H.2.8 [**Database Management**]: Database Applications—*Data mining*

Keywords

centrality; pseudodimension; sampling;

1. INTRODUCTION

Centrality measures are fundamental concepts in graph analysis. The centrality (importance) of a node or edge is a weak a score that quantifies some notion of importance of the node/edge in the network [20]. 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, 12].

Brandes [9] presented an algorithm (denoted **BA**) that computes the exact BC values for all nodes or edges in a graph $G = (V, E)$, in time $\mathcal{O}(|V||E|)$ if the graph is unweighted, and 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 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 updated in such networks is an extremely costly challenging task, and proposed algorithms [14, 16–18] have a worst-case complexity and memory requirements which is not better than from-scratch-recomputation using **BA**. Maintaining an 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 vertices and edges in static and dynamic graphs. Our contributions are the following:

- We introduce **ABRA**, a family of algorithms (denoted with "*Rademacher Averages*"), the first family of algorithms based on *progressive sampling* for approximating the BC of all vertices in static and dynamic graphs, where vertex and edge insertions and deletions are allowed. The approximations computed by **ABRA** are probabilistically guaranteed to be within an user-specified additive error from their exact values. We also present variants that focus on (estimating relative) error for the top-k vertices 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 [27] and pseudodimension [22], fundamental concepts from the field of statistical learning theory [29]. Exploiting known and novel results using these concepts, **ABRA** computes the approximations without having to keep track of any shortest path, thus greatly improving over existing algorithms [4, 6, 23]. **ABRA** performs only "real work" towards the computation of the approximations, without having to compute each 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 new analytical results on the number of samples needed for approximations, generalizing previous contributions [23], and formulating a conjecture on the connection between pseudodimension and the distribution of shortest path lengths.

DOI: http://dx.doi.org/XX.XXXX/XXXXXXX.XXXXXXX

What are the important nodes in a graph?

Let $G = (V, E)$ be a *graph* with $|V| = n$ nodes and $|E| = m$ edges.

QUESTION: Can we find the *most important nodes* in G?
I.e. (almost), can we *rank the nodes by importance*?

PREREQUISITE: *Quantify* the importance of a node through a *numerical score*.

Definition (Centrality measure)

A function $f : V \to \mathbb{R}^+$ expressing the importance of a node.
The *higher* is $f(x)$, the *more important* is $x \in V$.

MOTIVATION: Find *relevant* webpages on the web, *influential* participants in a social network, *key* concepts in a E-R graph, ...

EXAMPLES: degree, PageRank, closeness, *betweenness*, ...
Each centrality measure quantifies importance in a very specific way.

What is betweenness centrality?

INTUITION: Assume that

1) every node wants to communicate with every node; and

2) communication progresses along *Shortest Paths (SPs)*.

Then, *the higher the no. of SPs that a node v belongs to, the more important v is.*

Definition (Betweenness Centrality (BC))

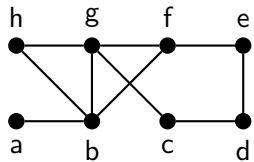
For each node $x \in V$, the *betweeness* $\mathrm{b}(x)$ of x is:

$$\mathrm{b}(x) = \frac{1}{n(n-1)} \sum_{u \neq x \neq v \in V} \frac{\sigma_{uv}(x)}{\sigma_{uv}} \qquad \in [0,1]$$

- σ_{uv}: number of SPs from u to v, $u, v \in V$;
- $\sigma_{uv}(x)$: number of SPs from u to v that go through x.

Roughly: $\mathrm{b}(x)$: the *weighted fraction of SPs that go through x*, among all SPs in G.

May I give an example?



Node x	a	b	c	d	e	f	g	h
b(x)	0	0.250	0.125	0.036	0.054	0.080	0.268	0

How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

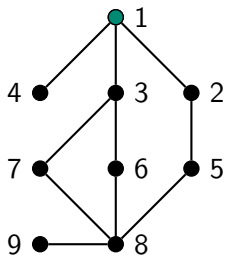
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



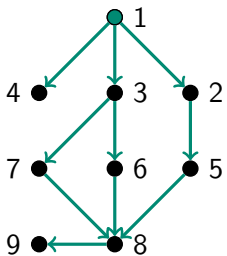
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



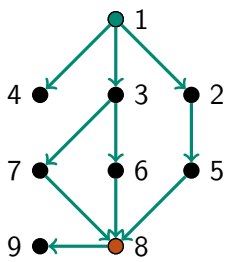
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



(update to $\mathrm{b}(v)$ not shown)

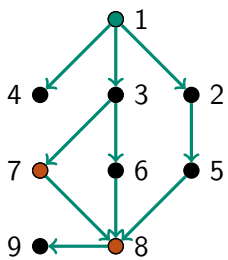
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



(update to $\mathrm{b}(v)$ not shown)

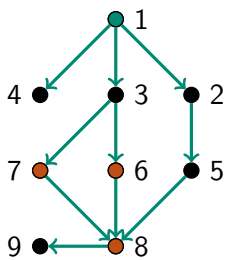
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



(update to $\mathrm{b}(v)$ not shown)

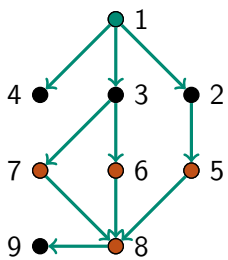
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



(update to $\mathrm{b}(v)$ not shown)

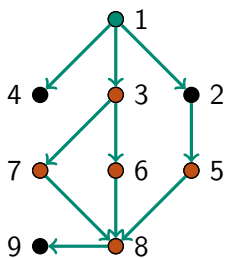
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



(update to $\mathrm{b}(v)$ not shown)

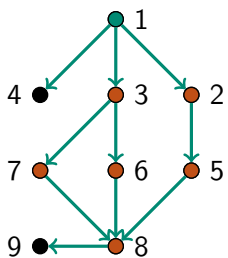
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



(update to $\mathrm{b}(v)$ not shown)

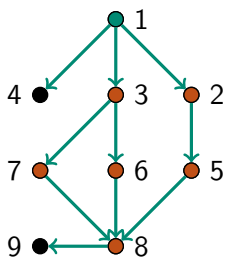
How do we compute it exactly for all vertices?

Brandes's Algorithm (BA) [Brandes 2001]

For each vertex $s \in V$:

1) Build the SP DAG from s via Dijkstra/BFS;
2) Traverse the SP DAG from the most distant node towards s, in *reverse order* of distance. During the walk, *appropriately* increment $\mathrm{b}(v)$ of each non-leaf node v traversed.

Source s: 1



(update to $\mathrm{b}(v)$ not shown)

TIME COMPLEXITY: $O(nm + n^2 \log n)$
n Dijkstra's, plus n backward walks, taking at most n each

Too much even with just 10^4 nodes.

What kind of approximation are we looking for?

Let $\tilde{\mathrm{b}}(x)$ denote a value that *approximates* $b(x)$, $x \in V$.

Definition ((ε, δ)-approximation)

Let $\varepsilon \in (0,1)$, and $\delta \in (0,1)$ be user-specified parameters;

An (ε, δ)-approximation is a set $\{\tilde{\mathrm{b}}(x), x \in V\}$ of n values such that

$$\Pr(\exists x \in V \text{ s.t. } |\mathrm{b}(x) - \tilde{\mathrm{b}}(x)| > \varepsilon) \leq \delta \ .$$

I.e., with prob. $\geq 1 - \delta$, *for all* $x \in V$, $\tilde{\mathrm{b}}(x)$ *is within* ε *of* $\mathrm{b}(x)$.

An (ε, δ)-approximation offers *uniform probabilistic guarantees* over all the nodes.

Are there algorithms to compute (ε, δ)-approximations?

Yes, they use *random sampling*, but of different "objects":

- [Brandes and Pich 2007]: sampling of *source nodes for BA* uniformly at random;
- [R. and Kornaropoulos 2015]: sampling of *SPs* non-uniformly at random;

NOTE: they *obtain approximations by performing fewer computations*, *not* by running BA on a *smaller graph*.

Key question in sampling algorithms:

How many samples does the algorithm need to obtain a (ε, δ)-approximation?

ANSWER:

- [Brandes and Pich 2007]: $O\left(\frac{1}{\varepsilon^2}\left(\ln n + \ln \frac{1}{\delta}\right)\right)$ source nodes;
- [R. and Kornaropoulos 2015]: $O\left(\frac{1}{\varepsilon^2}\left(\log_2 D + \ln \frac{1}{\delta}\right)\right)$ SPs (D: diameter of G);

What's wrong with these algorithms?

[Brandes and Pich 2007]:

- the *sample size* does not depend on the *edge structure* of G, only on n;
- *lots of work per sample* (SSSP, i.e., full exploration of the graph).

[R. and Kornaropoulos 2015]:

- the *sample size* is derived by considering the *worst-case graph* of diameter D;
- lots of *wasted work* per sample ($s - t$ SP computation, but a single SP is used);
- must compute an upper bound to the diameter before sampling can start.

Our algorithm, *ABRA*, solves these issues.

How does ABRA solve these issues?

ABRA computes an (ε, δ)-*approximation* using *progressive random sampling*.

ABRA in two lines (details later)

- ABRA immediately *starts sampling*, computing the approximation as it goes.
- At predefined intervals, ABRA checks a *stopping condition* to understand, *using the sample*, whether the current approximation has the desired quality.

The analysis of correctness uses *Rademacher averages* and *pseudodimension*.

Challenge

The stopping condition must be *fast to check* and *satisfied at small sample sizes*

Intuition from a Rademacher Average p.o.v.

$\mathcal{D}$ is the set of pairs of different nodes (u, v) in V:

$$\mathcal{D} = \{(u, v), (u, v) \in V \times V, u \neq v\}$$

Sample from $\mathcal{D}$ uniformly at random

$\mathcal{F}$ contains one function f_w for each $w \in V$. $f_w : \mathcal{D} \to [0, 1]$:

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

ABRA keeps track of the set $\mathcal{V}_\mathcal{S}$ of vectors $\mathbf{v}_{f_w}$, $w \in V$, and uses it to compute bounds to the maximum deviations.

How does ABRA work?

ABRA from 30,000 ft:

INPUT: G, ε, δ, *sample schedule* $(S_i)_{i \geq 1}$
OUTPUT: (ε, δ)-approximation

How does ABRA work?

ABRA from 30,000 ft:

INPUT: G, ε, δ, *sample schedule* $(S_i)_{i \geq 1}$
OUTPUT: (ε, δ)-approximation

$\mathcal{T} \leftarrow$ set of triples $(r_1 \in \mathbb{R}, r_2 \in \mathbb{R}, C \subseteq V))$, initially containing only $(0, 0, V)$;
$\tilde{b}(x) \leftarrow 0$, for all $x \in V$; $i \leftarrow 1$, $S_0 \leftarrow 0$

How does ABRA work?

ABRA from 30,000 ft:

INPUT: G, ε, δ, *sample schedule* $(S_i)_{i \geq 1}$
OUTPUT: (ε, δ)-approximation

$\mathcal{T} \leftarrow$ set of triples $(r_1 \in \mathbb{R}, r_2 \in \mathbb{R}, C \subseteq V))$, initially containing only $(0, 0, V)$;
$\tilde{b}(x) \leftarrow 0$, for all $x \in V$; $i \leftarrow 1$, $S_0 \leftarrow 0$
At iteration i:
For $j \leftarrow 1$ to $S_i - S_{i-1}$:

How does ABRA work?

ABRA from 30,000 ft:

INPUT: G, ε, δ, *sample schedule* $(S_i)_{i \geq 1}$
OUTPUT: (ε, δ)-approximation

$\mathcal{T} \leftarrow$ set of triples $(r_1 \in \mathbb{R}, r_2 \in \mathbb{R}, C \subseteq V))$, initially containing only $(0, 0, V)$;
$\tilde{b}(x) \leftarrow 0$, for all $x \in V$; $i \leftarrow 1$, $S_0 \leftarrow 0$
At iteration i:
For $j \leftarrow 1$ to $S_i - S_{i-1}$:
1) *Sample* a pair (u, v) of nodes uniformly at random from $V \times V$;
2) Get the *SP DAG* from u to v (Dijkstra/BFS);
3) *Increment* $\tilde{b}(x)$ by $\sigma_{uv}(x)/\sigma_{uv}$ for all $x \in V$ *internal* to the SP DAG;
4) *Update* $\mathcal{T}$; `// stay tuned`

How does ABRA work?

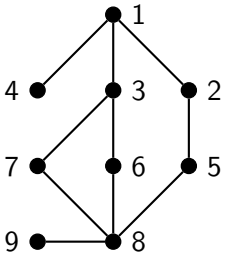
ABRA from 30,000 ft:

INPUT: G, ε, δ, *sample schedule* $(S_i)_{i \geq 1}$
OUTPUT: (ε, δ)-approximation

$\mathcal{T} \leftarrow$ set of triples $(r_1 \in \mathbb{R}, r_2 \in \mathbb{R}, C \subseteq V))$, initially containing only $(0, 0, V)$;
$\tilde{b}(x) \leftarrow 0$, for all $x \in V$; $i \leftarrow 1$, $S_0 \leftarrow 0$
At iteration i:
  For $j \leftarrow 1$ to $S_i - S_{i-1}$:
    1) *Sample* a pair (u, v) of nodes uniformly at random from $V \times V$;
    2) Get the *SP DAG* from u to v (Dijkstra/BFS);
    3) *Increment* $\tilde{b}(x)$ by $\sigma_{uv}(x)/\sigma_{uv}$ for all $x \in V$ *internal* to the SP DAG;
    4) *Update* $\mathcal{T}$; `// stay tuned`
  Check a stopping condition using $\mathcal{T}$; `// stay tuned`
    If the stopping condition is *satisfied, then output* $\{\tilde{b}(x)/S_i, x \in V\}$, else iterate;

How does ABRA really work?

1) *Sample* a pair (u, v) of nodes uniformly at random from $V \times V$;
2) Get the *SP DAG* from u to v (Dijkstra/BFS);
3) *Increment* $\tilde{\mathrm{b}}(x)$ by $\sigma_{uv}(x)/\sigma_{uv}$ for all $x \in V$ *internal* to the SP DAG;
4) *Update* $\mathcal{T}$;

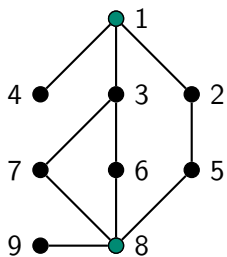


Node v	$\tilde{\mathrm{b}}(v)$
1	1
2	1/2
3	1/2
4	0
5	1
6	1/2
7	1/2
8	1/2
9	0

How does ABRA really work?

1) *Sample* a pair (u, v) of nodes uniformly at random from $V \times V$;
2) Get the *SP DAG* from u to v (Dijkstra/BFS);
3) *Increment* $\tilde{b}(x)$ by $\sigma_{uv}(x)/\sigma_{uv}$ for all $x \in V$ *internal* to the SP DAG;
4) *Update* $\mathcal{T}$;

Sampled pair: $(1, 8)$

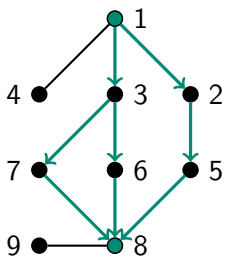


Node v	$\tilde{b}(v)$
1	1
2	1/2
3	1/2
4	0
5	1
6	1/2
7	1/2
8	1/2
9	0

How does ABRA really work?

1) *Sample* a pair (u, v) of nodes uniformly at random from $V \times V$;
2) Get the *SP DAG* from u to v (Dijkstra/BFS);
3) *Increment* $\tilde{b}(x)$ by $\sigma_{uv}(x)/\sigma_{uv}$ for all $x \in V$ *internal* to the SP DAG;
4) *Update* $\mathcal{T}$;

Sampled pair: $(1, 8)$ SPs: $\sigma_{1,8} = 3$

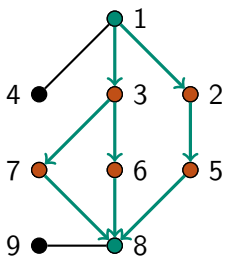


Node v	$\tilde{b}(v)$	$\sigma_{1,8}(v)/\sigma_{1,8}$
1	1	0
2	1/2	1/3
3	1/2	2/3
4	0	0
5	1	1/3
6	1/2	1/3
7	1/2	1/3
8	1/2	0
9	0	0

How does ABRA really work?

1) *Sample* a pair (u, v) of nodes uniformly at random from $V \times V$;
2) Get the *SP DAG* from u to v (Dijkstra/BFS);
3) *Increment* $\tilde{b}(x)$ by $\sigma_{uv}(x)/\sigma_{uv}$ for all $x \in V$ *internal* to the SP DAG;
4) *Update* $\mathcal{T}$;

Sampled pair: $(1, 8)$ SPs: $\sigma_{1,8} = 3$

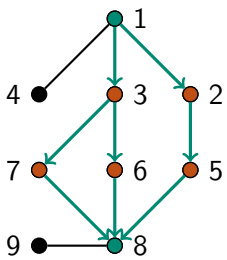


Node v	$\tilde{b}(v)$	$\sigma_{1,8}(v)/\sigma_{1,8}$
1	1	0
2	5/6	1/3
3	7/6	2/3
4	0	0
5	5/6	1/3
6	5/6	1/3
7	5/6	1/3
8	1/2	0
9	0	0

How does ABRA really work?

1) *Sample* a pair (u, v) of nodes uniformly at random from $V \times V$;
2) Get the *SP DAG* from u to v (Dijkstra/BFS);
3) *Increment* $\tilde{\mathrm{b}}(x)$ by $\sigma_{uv}(x)/\sigma_{uv}$ for all $x \in V$ *internal* to the SP DAG;
4) *Update* $\mathcal{T}$; `// stay tuned`

Sampled pair: $(1, 8)$ SPs: $\sigma_{1,8} = 3$



Node v	$\tilde{\mathrm{b}}(v)$	$\sigma_{1,8}(v)/\sigma_{1,8}$
1	1	0
2	5/6	1/3
3	7/6	2/3
4	0	0
5	5/6	1/3
6	5/6	1/3
7	5/6	1/3
8	1/2	0
9	0	0

What is $\mathcal{T}$?

Assume to observe ABRA after it has sampled k pairs of nodes: $(u_1, v_1), \ldots, (u_k, v_k)$.

For any $x \in V$, let $\mathbf{v}_x = \left(\frac{\sigma_{u_1,v_1}(x)}{\sigma_{u_1 v_1}}, \ldots, \frac{\sigma_{u_k v_k}(x)}{\sigma_{u_k v_k}} \right)$;

Let $\mathcal{V}_k = \{\mathbf{v}_x, x \in V\}$. There may be distinct nodes x and y s.t. $\mathbf{v}_x = \mathbf{v}_y$;
$\mathcal{V}_k$ induces a *partitioning* of V into classes $C_{\mathbf{v}}$ indexed by the elements of $\mathcal{V}_k$.

$\mathcal{T}$ contains *one and only* one element $(\|\mathbf{v}\|_1, \|\mathbf{v}\|_2, C_{\mathbf{v}})$ for each class $C_{\mathbf{v}}$ in the partitioning. At the start, all nodes belong to one class, and $\mathcal{T} = \{(0, 0, V)\}$

As ABRA takes more samples, *the partitioning is refined*, and $\mathcal{T}$ changes;

ABRA leverages properties of the refining process to *track the partitioning efficiently*.
$\mathcal{T}$ is updated efficiently after each sample.

What is the stopping condition?

ABRA outputs the approximations $(\tilde{\mathsf{b}}(x))_{x \in V}$ *when the stopping condition is satisfied*.

The stopping condition:

1. uses $\mathcal{T}$ to obtain $r = \max_{\mathbf{v} \in \mathcal{V}_\mathcal{S}} \|\mathbf{v}\|$ and $|\mathcal{V}_\mathcal{S}|$;
2. then uses r $|\mathcal{V}_\mathcal{S}|$ in Massart's Lemma to compute an upper bound ω to $\mathcal{R}_\mathcal{S}(\mathcal{F})$;
3. then uses ω and δ to obtain a bound ξ to $\max_{x \in V} |\tilde{\mathsf{b}}(x) - \mathsf{b}(x)|$.
4. and finally checks whether $\xi \leq \varepsilon$.

KEY THEOREM: The output is a (ε, δ)-approximation.

CAVEAT: Need union bound over all possible iterations, so at iteration i use $\delta' = \delta / 2^i$

Will ABRA ever stop sampling?

Yes, it will.

Theorem

Let θ be the size of the largest Weakly Connected Component in G.

After having sampled

$$\frac{1}{\varepsilon^2}\left(\log_2 \theta + \ln(1/\delta)\right)$$

pairs of nodes, then ABRA can stop: the output will be an (ε, δ)-approximation.

INTUITION:

$\log_2 \theta$ is an *upper bound* to the *pseudodimension* of the problem.

(pseudodimension: *VC-dimension* for real-valued functions).

Can we do better?

The bound to the pseudodimension is *disappointing*: for a connected G, $\theta = \log_2 |V|$.

We could get the same result with a simple union bound.

Can we do better?

The bound to the pseudodimension is *disappointing*: for a connected G, $\theta = \log_2 |V|$.

We could get the same result with a simple union bound.

Conjecture

Let G be a graph and let κ be the maximum positive integer for which there exists a set $L = \{(u_1, v_1), \ldots, (u_\kappa, v_\ell)\}$ of ℓ distinct pairs of distinct vertices such that

$$\sum_{i=1}^{\kappa} \sigma_{u_i v_i} \geq \binom{\ell}{\lfloor \ell/2 \rfloor}$$

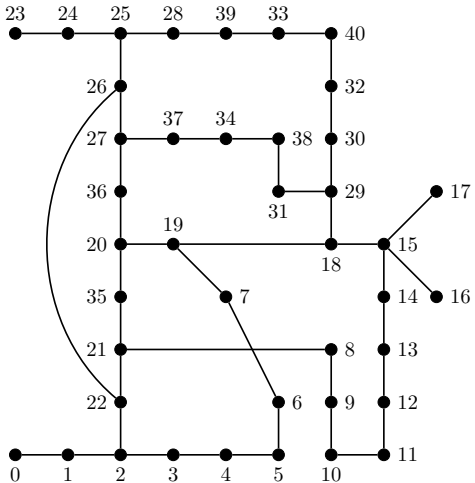
then the pseudodimension is at most κ.

E.g., if there is at most 1 SP between each pair of nodes (road networks), then pseudodimension is at most 3.

Is the conjecture tight?

The conjecture is tight at least up to $\kappa = 4$.

This graph satisfies the conjecture for $\kappa = 4$ and has pseudodimension $d = 4$:



How does ABRA perform in practice?

Great!

IMPLEMENTATION: C++, extension of *NetworKit*.

DATASETS: from *SNAP* (Soc-Epinions1, P2p-Gnutella, Email-Enron, Cit-HepPh).

ACCURACY: Maximum error was *always* $< \varepsilon$, Avg. and min. errors were $\ll \varepsilon$.

FINAL SAMPLE SIZE:

$O(25 \cdot 10^4)$ for $\varepsilon = 0.01$, $O(10^3)$ for $\varepsilon = 0.03$ (varies across graphs);
Smaller than RK ($\approx 2x$ to $4x$ fewer samples).

RUNTIME:

Faster than BA ($\approx 5x$ for $\varepsilon = 0.01$, $\approx 40x$ for $\varepsilon = 0.03$);

Faster than RK ($\approx 3x$ for $\varepsilon = 0.01$, $\approx 6x$ for $\varepsilon = 0.03$).

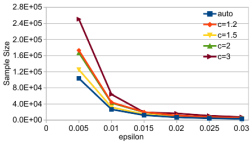
Did we do our homework?

This slide is close to *unreadable on purpose.*

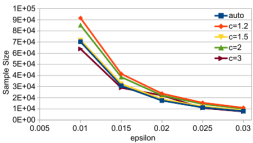
Graph	ε	Runtime (sec.)	Speedup w.r.t. BA	Speedup w.r.t. RK	Runtime Breakdown (%) Sampling	Stop Cond.	Other	Sample Size	Reduction w.r.t. RK	Absolute Error ($\times 10^5$) max	avg	stddev
Soc-Epinions1 Directed $\lvert V\rvert = 75,879$ $\lvert E\rvert = 508,837$	0.005	483.06	1.36	2.90	99.983	0.014	0.002	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 $\lvert V\rvert = 62,586$ $\lvert E\rvert = 147,892$	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 $\lvert V\rvert = 36,682$ $\lvert E\rvert = 183,831$	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 $\lvert V\rvert = 34,546$ $\lvert E\rvert = 421,578$	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

Did we really do our homework?

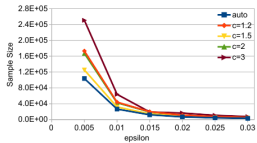
This slide is close to *unreadable on purpose.*



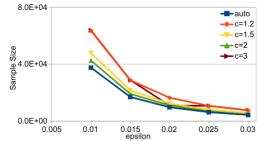
(a) P2p-Gnutella



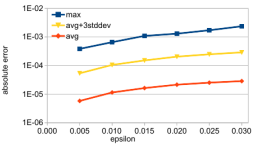
(b) Email-Enron



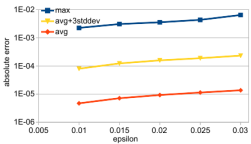
(c) Soc-Epinions1



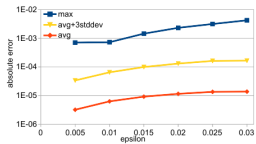
(d) Cit-HepPh



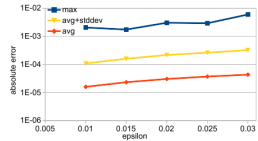
(a) P2p-Gnutella



(b) Email-Enron



(c) Soc-Epinions1



(d) Cit-HepPh

To sum up...

ABRA is an algorithm to estimate betweenness centrality of all nodes

It uses progressive random sampling of pairs of nodes, plus BFS/Dijkstra for each pair

It keeps track of $\mathcal{V}_S$ as it samples

The analysis relies on Rademacher averages

It is very fast, showing the practicality of Rademacher averages

IV. Conclusions

To sum up...

Rademacher averages: a very powerful tool for analyzing sampling algorithms

They are efficient in practice, no longer only of theoretical interest

We also used them for other key tasks in data analysis, e.g., frequent pattern mining

They can also be used to control the FWER in multiple hypotheses statistical testing

Rademacher chaos: promising extension to limited dependence sampling case

Online Rademacher avg.: extensions to non-stationary time-series analysis

Thank you!

Matteo Riondato @teorionda http://matteo.rionda.to