ABRA: Approximating Betweenness Centrality through Sampling with Rademacher Averages

Matteo Riondato (Two Sigma Investments LP, matteo@twosigma.com) Eli Upfal (Department of Computer Science, Brown University, eli@cs.brown.edu)
22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD’16)

What problem are we studying?

Motivation: Identifying the most important vertices in a graph allows us to study properties such as network robustness, information diffusion, and influence propagation, that give important insights on the structure and the dynamics of the graph.

Example: Find most influential CEOs, most informed journalists, most central locations in a city, most websites, etc.

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

To find the most important, i.e., central, vertices in \( G \) we need to measure a measure of importance, i.e., a centrality measure that assigns a score to each vertex, based on some concept of importance (using degree, distance, ...) 

Betweenness centrality uses information on Shortest Paths (SP) distance to assign the score.

For each pair of different nodes \((u, w)\) in \( V \) via \( \sum_{v \in V} \) let:

\[ f(u, w) = \frac{1}{|V|^2 - |V|} \sum_{v \in V} \text{d}(u, v) \text{d}(v, w) \]

Definition: Betweenness centrality of \( c \) in \( V \):

\[ b(c) = \frac{1}{|V|^2 - |V|} \sum_{u \neq c \neq w} \text{d}(u, c) \text{d}(c, w) \]

Task: Compute the betweenness centrality score \( b(c) \) for all nodes \( c \) in \( V \)

How can we compute all the \( b(c) \)'s exactly?

Fast Algorithms:

- All-Pairs SP - Aggregation (sum for each node)
  - Takes \( O(n^2) \) time due to the aggregation

Brandes’ Algorithm (BA, [Brandes, 2001]):
  - For each node \( u \) in \( V \):
    1) Run Single Source SP from \( u 
    2) Backtrack along SP DAG,

  - Approximately incrementing BC of nodes along the path.

  Takes \( O(nm) \) (unweighted G) or \( O((n^2+m^2 log n) \text{ (weighted G)})

- Exact algorithms do not scale with \( |V| \)

How can we speed up BC computation?

Tradeoff: Computational accuracy for speed

Only perform a few SPs computations between pairs of nodes \((u, w)\) sampled at random.

Computed BC values are approximate but fast to compute.

Approximate BC: OK; centrality computation is an exploratory task

Key question: How many pairs to sample to get an approximation of the desired quality?

Our contribution:

ABRA: a fast approximation algorithm for BC that uses Progressive Random Sampling and Rademacher Averages to obtain probabilistic guarantees on the quality of the output.

How do we define an approximation of the BC of \( v \)?

Let \( \sigma = \{(v_1, v_2), \ldots, (v_{|V|})\} \) be a bag (\( |V| = m \) and, e, \( e \in (0, 1) \))

\[ \text{Definition: (e, \epsilon)-approximation to } B \]

\[ \text{As } \Sigma \in B \{ (b(v_1), \ldots, (b(v_{|V|})) \text{ s.t.} \]

\[ \text{Pr}_{(e, \epsilon) \Sigma} \leq (b(v_1) - b(v_2)) > e \] \[ \text{ABRA returns an (e, \epsilon)-approximation to } B \]

It uses Progressive Random Sampling to decide how much to sample.

What is Progressive Random Sampling (PRS)?

How much shall we sample in order to obtain an \((e, \epsilon)-\)approximation?

State of the art (RK [Richter and Kornarpoulos, 2013]):
  - sample single SPs at each step: lots of wasted work
  - use fixed sample sizes for word-case graphs

This work (ABRA):
  - samples pairs of nodes, and uses all SPs among them
  - PRS: let’s start sampling: the data tells us when to stop

Progressive Random Sampling (PRS) adaptive sampling scheme

Outline of a PRS algorithm for BC: centrality

1. Initialize \( \Sigma = \emptyset \) for all \( v \) in \( V \)
2. Repeat until the stopping condition is satisfied:
   a. Sample a pair \((u, w)\) from \( V \) uniformly at random
   b. Increment by \( \text{d}(u, v) \text{d}(v, w) \)
   c. Pr[\( (u, v) \in \Sigma \) ] = \( (b(v_1) - b(v_2)) > e \] ounce of progress

ABRA implements the above scheme for PRS algorithms for BC centrality.

What are the challenges? What are our solutions?

The challenges:

1. Developing a stopping condition that:
   - can be checked fast
   - guarantees that the output is a \((e, \epsilon)-\)approximation
   - can be satisfied at small sample sizes
2. Developing a method to choose the next sample size

Our contribution: ABRA, the first algorithm that:

1. Uses PRS to obtain an \((e, \epsilon)-\)approximation of BC
2. Uses a stopping condition that only requires the solution of an unconstrained convex minimization problem
3. Does not need to compute any global property of the graph
4. Computes the optimal next sample size on the fly
5. Uses Rademacher Averages in a graph mining setting

Previous contributions using PRS gave no guarantees and used predefined sample sizes

What do we really need?

Let \( \Sigma \) be the collection of pairs sampled by ABRA up to iteration \( \ell \)

We need a way to compute the sum of pairs sampled by ABRA up to iteration \( \ell \)

\[ \text{Pr}_{\varepsilon_2} \left( \sum_{(u, v) \in \Sigma} \text{d}(u, v) \text{d}(v, w) > e \right) \]

Notation: \( \theta \) is an upper bound to the pseudo-dimension (VC-dimension for real-valued functions)

Conjecture: Let \( e \) be the maximum positive integer for which there exists a set \( \Sigma = \{(u_1, v_1), \ldots, (u_{|V|})\} \) of \( \ell \) distinct pairs of distinct vertices such that

\[ \sum_{(u, v) \in \Sigma} \varepsilon_{2, \ell} \leq (\ell / 2) \]

then the pseudo-dimension is at most \( e \)

Figure: The conjecture is true for \( \ell \leq 4 \) this graph satisfies the conjecture for \( e = 4 \) and has pseudo-dimension 4

Experimental evaluation

We implemented ABRA in C++ as an extension to NetworkKit.

We tested ABRA on datasets from the SNAP repository.

Run on AMD Phenom II X4 955 processor and 16GB of RAM, running FreeBSD 11

Results

1. In all runs, the maximum error was smaller than \( e \)
2. Average errors 100 times smaller than \( \theta \)
3. 99% percentile error 10 times smaller than \( e \)

ABRA requires up to 4x fewer samples than state-of-the-art ARK

ABRA is up to 7x faster than ARK, 40x faster than BA

97.7% of the running time is spent doing useful work (sampling)

2.1% spent checking the stopping condition

The automatic sample schedule outperforms fixed geometric ones

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

<table>
<thead>
<tr>
<th>Sample Size</th>
<th>Runtime (sec.)</th>
<th>Speedup</th>
<th>Breakdown</th>
<th>Reduction</th>
<th>Absolute Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>12</td>
<td>1</td>
<td>10/9</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>20</td>
<td>6</td>
<td>2</td>
<td>20/8</td>
<td>2</td>
<td>0.001</td>
</tr>
<tr>
<td>40</td>
<td>3</td>
<td>4</td>
<td>40//</td>
<td>4</td>
<td>0.0001</td>
</tr>
<tr>
<td>80</td>
<td>1.5</td>
<td>8</td>
<td>80/0</td>
<td>8</td>
<td>0.00001</td>
</tr>
</tbody>
</table>

![Figure: Actual error evaluation. The vertical axis has a logarithmic scale](image.png)

13. Future work

- Extend ABRA to other centrality measures
- Improve bounds to Rademacher averages
- Develop different sampling schemes to use all compute SPs
- Use Murtagh’s theory rather than Union Bound over iterations

14. Acknowledgments

This work was supported, in part, by NSF grant IIS-1247581 and NIH grant R01-CA161775

15. Extended version