Estimating the Percolation Centrality of Large Networks through Pseudo-dimension Theory

Alane M. de Lima
Department of Computer Science
Federal University of Paraná
P.O. Box 19031
Curitiba – Paraná, Brazil, 81531-980
amlima@inf.ufpr.br

Murilo V. G. da Silva
Department of Computer Science
Federal University of Paraná
P.O. Box 19031
Curitiba – Paraná, Brazil, 81531-980
murilo@inf.ufpr.br

André L. Vignatti
Department of Computer Science
Federal University of Paraná
P.O. Box 19031
Curitiba – Paraná, Brazil, 81531-980
vignatti@inf.ufpr.br

ABSTRACT

In this work we investigate the problem of estimating the percolation centrality of every vertex in a graph. This centrality measure quantifies the importance of each vertex in a graph going through a contagious process. It is an open problem whether the percolation centrality can be computed in $O(n^{3-c})$ time, for any constant $c > 0$. In this paper we present a $\tilde{O}(m)$ randomized approximation algorithm for the percolation centrality for every vertex of G, generalizing techniques developed by Riondato, Upfal and Kornaropoulos. The estimation obtained by the algorithm is within ϵ of the exact value with probability $1 - \delta$, for *fixed* constants $0 < \epsilon, \delta < 1$. In fact, we show in our experimental analysis that in the case of real-world complex networks, the output produced by our algorithm is significantly closer to the exact values than its guarantee in terms of theoretical worst case analysis.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; **Shortest paths**; **Approximation algorithms analysis**; **Sample complexity and generalization bounds**.

KEYWORDS

percolation centrality; approximation algorithms; pseudo-dimension; sample complexity; graph algorithms

ACM Reference Format:
Alane M. de Lima, Murilo V. G. da Silva, and André L. Vignatti. 2020. Estimating the Percolation Centrality of Large Networks through Pseudo-dimension Theory. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403235

1 INTRODUCTION

The importance of a vertex in a graph can be quantified using centrality measures. In this paper we deal with the *percolation centrality*, a measure relevant in applications where graphs are used to model a contagious process in a network (e.g., disease transmission or misinformation spreading). Centrality measures can be defined in terms of local properties, such as the vertex degree, or global properties, such as the betweenness centrality or the percolation centrality. The betweenness centrality of a vertex v, roughly speaking, is the fraction of shortest paths containing v as an intermediate vertex. The percolation centrality generalizes the betweenness centrality by allowing weights on the shortest paths, and the weight of a shortest path depends on the disparity between the degree of contamination of the two end vertices of such path.

The study of the percolation phenomenon in a physical system was introduced by [6] in the context of the passage of a fluid in a medium. In graphs, percolation centrality was proposed by Piraveenan *et. al* (2013) [13], where the medium are the vertices of a graph G and each vertex v in G has a *percolation state* (reflecting the "degree of contamination" of v). The percolation centrality of v is a function that depends on the topological connectivity and the states of the vertices of G (the appropriate formal definitions are given in Section 2).

The best known algorithms that exactly compute the betweenness centrality for every vertex of a graph depends on computing all its shortest paths [14] and, consequently, the same applies to the computation of percolation centrality. The best known algorithm for this task for weighted graphs runs in time $O\left(n^3/2^{c\sqrt{\log n}}\right)$, for some constant c [19]. Currently it is a central open problem in graph theory whether this problem can be solved in $O(n^{3-c})$, for any $c > 0$ and the hypothesis that there is no such algorithm is used in hardness arguments in some works[1, 2]. In the particular case of sparse graphs, which are common in applications, the complexity of the exact computation for the betweenness centrality can be improved to $O(n^2)$. However, the same is not known to be the true for percolation centrality and no subcubic algorithm is known even in such restricted scenario.

The present paper uses techniques developed in the work of Riondato and Kornaropoulos (2016) [14] and Riondato and Upfal (2018) [15] on betweenness centrality. A main theme in their work is the fact that for large scale graphs, even algorithms that have cubic time complexity are inefficient in practice and high-quality approximations obtained with high confidence are usually sufficient in real-world applications. The authors observe that keeping track of the exact centrality values, which may change continuously, provides little information gain. So the idea is to sample a subset of all shortest paths in the graph so that, for given $0 < \epsilon, \delta < 1$, they

KDD '20, August 23–27, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7998-4/20/08…$15.00
https://doi.org/10.1145/3394486.3403235

obtain values within ϵ from the exact value with probability $1 - \delta$ [14].

We developed the main results of this paper having in mind both a theoretical and a practical perspective. From the theoretical perspective, for $0 < \epsilon, \delta < 1$ being any fixed constants, we show that the estimation of the percolation centrality can be done in $\tilde{O}(m)$ time. In the particular case of sparse graphs with logarithmic diameter (a very common case in practice), the time complexity is $O(n \log n \log\log n)$. In the practical front, in Section 2.2 we give the relation between these constants and the sample size required for meeting the approximation guarantee and, in fact, our experimental evaluation shows that our algorithm produces results that are orders of magnitude better than the guarantees given by the referred theoretical analysis.

The techniques developed by [14] for the betweenness problem relies on the Vapnik-Chervonenkis (VC) dimension theory and the ϵ-sample theorem. In our work, we use such techniques together with pseudo-dimension (a generalization of the VC-dimension) theory to show that the more general problem of estimating the percolation centrality of every vertex of G can be computed in $\tilde{O}(m)$ time. We note that in the more recent work of Riondato and Upfal [15] they also use pseudo-dimension theory for the betweenness problem, but they obtain different bounds for the sample size and they use pseudo-dimension in order to make use of Rademacher Averages. In our work we need pseudo-dimension theory by the very nature of the problem since percolation functions are real-valued and VC-dimension does not apply in our scenario.

2 PRELIMINARIES

We now introduce the definitions, notation and results we use as the groundwork of our proposed algorithms. In this paper we assume w.l.o.g. that the input graph G is connected, since our algorithm can be applied separately to each of its connected components.

2.1 Graphs and Percolation Centrality

Given a graph $G = (V, E)$ (directed or undirected), the percolation states x_v for each $v \in V$ and $(u, w) \in V^2$, let S_{uw} be the set of all shortest paths from u to w, and $\sigma_{uw} = |S_{uw}|$. For a given path $p_{uw} \in S_{uw}$, let $\text{Int}(p_{uw})$ be the set of internal vertices of p_{uw}, that is, $\text{Int}(p_{uw}) = \{v \in V : v \in p_{uw} \text{ and } u \neq v \neq w\}$. We denote $\sigma_{uw}(v)$ as the number of shortest paths from u to w that $v \in V$ is internal to. Let $P_u(w) = \{s \in V : (s, w) \in E_{p_{uw}}\}$ be the set of (immediate) *predecessors* of w in $p_{uw} \in S_{uw}$, where $E_{p_{uw}}$ is the set of edges of p_{uw}. We call the *diameter* of G, denoted by $\text{diam}(G)$, as the largest shortest path in G. Let $0 \leq x_v \leq 1$ be the percolation state of $v \in V$. We say v is *fully percolated* if $x_v = 1$, *non-percolated* if $x_v = 0$ and *partially percolated* if $0 < x < 1$. We say that a path from u to w is *percolated* if $x_u - x_w > 0$. The percolation centrality is defined below.

Definition 2.1 (Percolation Centrality). Let $R(x) = \max\{x, 0\}$. Given a graph $G = (V, E)$ and percolation states $x_v, \forall v \in V$, the *percolation centrality* of a vertex $v \in V$ is defined as

$$p(v) = \frac{1}{n(n-1)} \sum_{\substack{(u,w) \in V^2 \\ u \neq v \neq w}} \frac{\sigma_{uw}(v)}{\sigma_{uw}} \frac{R(x_u - x_w)}{\sum\limits_{\substack{(f,d) \in V^2 \\ f \neq v \neq d}} R(x_f - x_d)}.$$

The definition originally presented in [13] does not have the normalization factor $\frac{1}{n(n-1)}$, introduced here to allow us to define a proper probability distribution in Section 3. This normalization obviously does not change the relation between the centrality of vertices.

2.2 Sample Complexity and Pseudo-dimension

In sampling algorithms, the sample complexity analysis relates the minimum size of a random sample required to estimate results that are consistent with the desired parameters of quality and confidence (e.g., in our case a minimum number of shortest paths that must be sampled). An upper bound to the Vapnik-Chervonenkis Dimension (VC-dimension) of a class of binary functions especially defined in order to model the particular problem that one is dealing provides an upper bound to sample size respecting such parameters. Generally speaking, the VC-dimension measures the expressiveness of a class of subsets defined on a set of points [14].

For the problem presented in this work, however, the class of functions that we need to deal are not binary. Hence, we use the *pseudo-dimension*, which is a generalization of the VC-dimension for real-valued functions. An in-depth exposition of the definitions and results presented below can be found in the books of Anthony and Bartlett [4], Mohri *et. al* [12] Shalev-Schwartz and Ben-David [17] and Mitzenmacher and Upfal [11].

Empirical averages and ϵ-representative samples. Given a domain U and a set of values of interest $\mathcal{H}$, let $\mathcal{F}$ be the family of functions from U to $\mathbb{R}_+$ such that there is one $f_h \in \mathcal{F}$ for each $h \in \mathcal{H}$. Let S be a collection of r elements from U sampled with respect to a probability distribution π.

Definition 2.2. For each $f_h \in \mathcal{F}$, such that $h \in \mathcal{H}$, we define the expectation of f_h and its empirical average as L_U and L_S, respectively, i.e.,

$$L_U(f_h) = \mathbb{E}_{u \in U}[f_h(u)]$$

and

$$L_S(f_h) = \frac{1}{r} \sum_{s \in S} f_h(s).$$

Definition 2.3. Given $0 < \epsilon, \delta < 1$, a set S is called *ϵ-representative* w.r.t. some domain U, a set $\mathcal{H}$, a family of functions $\mathcal{F}$ and a probability distribution π if $\forall f_h \in \mathcal{F}$,

$$|L_S(f_h) - L_U(f_h)| \leq \epsilon.$$

By the linearity of expectation, the expected value of the empirical average $L_S(f_h)$ corresponds to $L_U(f_h)$. Hence, $|L_S(f_h) - L_U(f_h)| = |L_S(f_h) - \mathbb{E}_{f_h \in \mathcal{F}}[L_S(f_h)]|$, and by the *law of large numbers*, $L_S(f_h)$ converges to its true expectation as r goes to infinity, since $L_S(f_h)$ is the empirical average of r random variables sampled independently and identically w.r.t. π. However, this law provides no information about the value $|L_S(f_h) - L_U(f_h)|$ for any sample size. Thus, we use results from the VC-dimension and pseudo-dimension theory, which provide bounds on the size of the sample

that guarantees that the maximum deviation of $|L_S(f_h) - L_U(f_h)|$ is within ϵ with probability at least $1 - \delta$, for given $0 < \epsilon, \delta < 1$.

VC-dimension. A *range space* is a pair $\mathcal{R} = (X, \mathcal{I})$, where X is a domain (finite or infinite) and $\mathcal{I}$ is a collection of subsets of X, called *ranges*. For a given $S \subseteq X$, the *projection* of $\mathcal{I}$ on S is the set $\mathcal{I}_S = \{S \cap I : I \in \mathcal{I}\}$. If $|\mathcal{I}_S| = 2^{|S|}$ then we say S is *shattered* by $\mathcal{I}$. The VC-dimension of a range space is the size of the largest subset S that can be shattered by $\mathcal{I}$, as presented by the following definition.

Definition 2.4. The VC-dimension of a range space $\mathcal{R} = (X, \mathcal{I})$, denoted by $VCDim(\mathcal{R})$, is $VCDim(\mathcal{R}) = \max\{d : \exists S \subseteq X \text{ such that } |S| = d \text{ and } |\mathcal{I}_S| = 2^d\}$.

Pseudo-dimension. Let $\mathcal{F}$ be a family of functions from some domain U to the range $[0, 1]$. Consider $D = U \times [0, 1]$. For each $f \in \mathcal{F}$, there is a subset $R_f \subseteq D$ defined as $R_f = \{(x, t) : x \in U \text{ and } t \leq f(x)\}$.

Definition 2.5 (see [4], Section 11.2). Let $\mathcal{R} = (U, \mathcal{F})$ and $\mathcal{R}' = (D, \mathcal{F}^+)$ be range spaces, where $\mathcal{F}^+ = \{R_f : f \in \mathcal{F}\}$. The *pseudo-dimension* of $\mathcal{R}$, denoted by $PD(\mathcal{R})$, corresponds to the VC-dimension of $\mathcal{R}'$, i.e., $PD(\mathcal{R}) = VCDim(\mathcal{R}')$.

Theorem 2.6 states that having an upper bound to the pseudo-dimension of a range space allows to build an ϵ-representative sample.

Theorem 2.6 (see [9], Section 1). *Let $\mathcal{R}' = (D, \mathcal{F}^+)$ be a range space ($D = U \times [0, 1]$) with $VCDim(\mathcal{R}') \leq d$ and a probability distribution π on U. Given $0 < \epsilon, \delta < 1$, let $S \subseteq D$ be a collection of elements sampled w.r.t. π, with $|S| = \frac{c}{\epsilon^2}\left(d + \ln \frac{1}{\delta}\right)$ where c is a universal positive constant. Then S is ϵ-representative with probability at least $1 - \delta$.*

In the work of [10], it has been proven that the constant c is approximately $\frac{1}{2}$. Lemmas 2.7 and 2.8, stated and proved by Riondato and Upfal [15], present constraints on the sets that can be shattered by a range set $\mathcal{F}^+$.

Lemma 2.7 (see [15], Section 3.3). *Let $B \subseteq D$ be a set that is shattered by $\mathcal{F}^+$. Then, B can contain at most one $(d, y) \in D$ for each $d \in U$.*

Lemma 2.8 (see [15], Section 3.3). *Let $B \subseteq D$ be a set that is shattered by $\mathcal{F}^+$. Then, B does not contain any element in the form $(d, 0) \in D$, for each $d \in U$.*

3 PSEUDO-DIMENSION AND PERCOLATED SHORTEST PATHS

In this section we model the percolation centrality in terms of a range set of the percolated shortest paths. That is, for a given a graph $G = (V, E)$ and the percolation states x_v for each $v \in V$, let $\mathcal{H} = V$, with $n = |V|$, and let $U = S_G$, where

$$S_G = \bigcup_{(u,w) \in V^2 : u \neq w} S_{uw}.$$

For each $v \in V$, there is a set $\tau_v = \{p \in U : v \in \text{Int}(p)\}$. For a pair $(u, w) \in V^2$ and a path $p_{uw} \in S_G$, let $f_v : U \to [0, 1]$ be the function

$$f_v(p_{uw}) = \frac{R(x_u - x_w)}{\sum\limits_{(f,d) \in V^2 : f \neq v \neq d} R(x_f - x_d)} \mathbb{1}_{\tau_v}(p_{uw}).$$

The function f_v gives the proportion of the percolation between u and w to the total percolation in the graph if $v \in \text{Int}(p_{uw})$. We define $\mathcal{F} = \{f_v : v \in V\}$.

Let $D = U \times [0, 1]$. For each $f_v \in \mathcal{F}$, there is a range

$$R_v = R_{f_v} = \{(p_{uw}, t) : p_{uw} \in U \text{ and } t \leq f_v(p_{uw})\}.$$

Note that each range R_v contains the pairs (p_{uw}, t), where $0 < t \leq 1$ such that $v \in \text{Int}(p_{uw})$ and

$$t \leq \frac{R(x_u - x_w)}{\sum\limits_{(f,d) \in V^2 : f \neq v \neq d} R(x_f - x_d)}.$$

We define $\mathcal{F}^+ = \{R_v : f_v \in \mathcal{F}\}$.

Each $p_{uw} \in U$ is sampled according to the function (which is a valid probability distribution according to Theorem 3.1)

$$\pi(p_{uw}) = \frac{1}{n(n-1)} \frac{1}{\sigma_{uw}},$$

and $\mathbb{E}[f_v(p_{uw})] = p(v)$ for all $v \in V$, as proved in Theorem 3.2.

Theorem 3.1. *The function $\pi(p_{uw})$, for each $p_{uw} \in U$, is a valid probability distribution.*

Proof. Let S_{uw} be the set of shortest paths from u to w, where $u \neq w$. Then,

$$\begin{aligned}
\sum_{p_{uw} \in U} \pi(p_{uw}) &= \sum_{p_{uw} \in U} \frac{1}{n(n-1)} \frac{1}{\sigma_{uw}} \\
&= \sum_{u \in V} \sum_{\substack{w \in V \\ w \neq u}} \sum_{p \in S_{uw}} \frac{1}{n(n-1)} \frac{1}{\sigma_{uw}} \\
&= \sum_{u \in V} \sum_{\substack{w \in V \\ w \neq u}} \frac{1}{n(n-1)} \frac{\sigma_{uw}}{\sigma_{uw}} \\
&= \frac{1}{n(n-1)} \sum_{u \in V} \sum_{\substack{w \in V \\ w \neq u}} 1 \\
&= \frac{n(n-1)}{n(n-1)} \\
&= 1.
\end{aligned}$$

□

Theorem 3.2. *For $f_v \in \mathcal{F}$ and for all $p_{uw} \in U$, such that each p_{uw} is sampled according to the probability function $\pi(p_{uw})$,*

$$\mathbb{E}[f_v(p_{uw})] = p(v).$$

Proof. For a given graph $G = (V, E)$ and for all $v \in V$, we have from Definition 2.2

$$\begin{aligned}
L_U(f_v) &= \mathbb{E}_{p_{uw}\in U}[f_v(p_{uw})] \\
&= \sum_{p_{uw}\in U} \pi(p_{uw}) f_v(p_{uw}) \\
&= \sum_{p_{uw}\in U} \frac{1}{n(n-1)} \frac{1}{\sigma_{uw}} \frac{R(x_u - x_w)}{\sum\limits_{\substack{(f,d)\in V^2 \\ f\neq v\neq d}} R(x_f - x_d)} \mathbb{1}_{\tau_v}(p_{uw}) \\
&= \frac{1}{n(n-1)} \sum_{\substack{u\in V \\ u\neq v}} \sum_{\substack{w\in V \\ w\neq v\neq u}} \sum_{p\in S_{uw}} \frac{1}{\sigma_{uw}} \frac{R(x_u - x_w)}{\sum\limits_{\substack{(f,d)\in V^2 \\ f\neq v\neq d}} R(x_f - x_d)} \mathbb{1}_{\tau_v}(p) \\
&= \frac{1}{n(n-1)} \sum_{\substack{u\in V \\ u\neq v}} \sum_{\substack{w\in V \\ w\neq v\neq u}} \frac{\sigma_{uw}(v)}{\sigma_{uw}} \frac{R(x_u - x_w)}{\sum\limits_{\substack{(f,d)\in V^2 \\ f\neq v\neq d}} R(x_f - x_d)} \\
&= \frac{1}{n(n-1)} \sum_{\substack{(u,w)\in V^2 \\ u\neq v\neq w}} \frac{\sigma_{uw}(v)}{\sigma_{uw}} \frac{R(x_u - x_w)}{\sum\limits_{\substack{(f,d)\in V^2 \\ f\neq v\neq d}} R(x_f - x_d)} \\
&= p(v).
\end{aligned}$$

□

Let $S = \{(p_{u_i w_i}, 1 \le i \le r)\}$ be a collection of r shortest paths sampled independently and identically from U. Next, we define $\tilde{p}(v)$, the estimation to be computed by the algorithm, as the empirical average from Definition 2.2:

$$\begin{aligned}
\tilde{p}(v) = L_S(f_v) &= \frac{1}{r} \sum_{p_{u_i w_i}\in S} f_v(p_{u_i w_i}) \\
&= \frac{1}{r} \sum_{p_{u_i w_i}\in S} \frac{R(x_{u_i} - x_{w_i})}{\sum\limits_{\substack{(f,d)\in V^2 \\ f\neq v\neq d}} R(x_f - x_d)} \mathbb{1}_{\tau_v}(p_{u_i w_i}).
\end{aligned}$$

4 APPROXIMATION TO THE PERCOLATION CENTRALITY

In this section we present a randomized algorithm for the approximation of the percolation centrality for every vertex of a graph. The correctness and running time of the algorithm relies on the sample size given by Theorem 2.6. In order to bound the sample size, in Theorem 4.1, we prove an upper bound to the range space $\mathcal{R}$. We are aware that the main idea in the proof is similar to the proof of a result for a different range space on the shortest paths obtained in [14] in their work using VC-dimension. For the sake of clarity, instead of trying to fit their definition to our model and use their result, we found it easier stating and proving the theorem directly for our range space.

Theorem 4.1. *Let $\mathcal{R} = (U, \mathcal{F})$ and $\mathcal{R}' = (D, \mathcal{F}^+)$ be the corresponding range spaces for the domain and range sets defined in Section 3, and let $Diam_V(G)$ be the vertex-diameter of G. We have*

$$PD(\mathcal{R}) = VCDim(\mathcal{R}') \le \lfloor \lg Diam_V(G) - 2 \rfloor + 1.$$

Proof. Let $VCDim(\mathcal{R}') = k$, where $k \in \mathbb{N}$. Then, there is $S \subseteq D$ such that $|S| = k$ and S is shattered by $\mathcal{F}^+$. From Lemmas 2.7 and 2.8, we know that for each $p_{uw} \in U$, there is at most one pair (p_{uw}, t) in S for some $t \in (0, 1]$ and there is no pair in the form $(p_{uw}, 0)$. By the definition of shattering, each $(p_{uw}, t) \in S$ must appear in 2^{k-1} different ranges in $\mathcal{F}^+$. On the other hand, each pair (p_{uw}, t) is in at most $|p_{uw}| - 2$ ranges in $\mathcal{F}^+$, since $(p_{uw}, t) \notin R_v$ either when $t > f_v(p_{uw})$ or $v \notin \mathrm{Int}(p_{uw})$. Considering that $|p_{uw}| - 2 \le \mathrm{Diam}_V(G) - 2$, we have

$$2^{k-1} \le |p_{uw}| - 2 \le \mathrm{Diam}_V(G) - 2$$

$$k - 1 \le \lg(\mathrm{Diam}_V(G) - 2).$$

Since k is an integer,

$$k \le \lfloor \lg \mathrm{Diam}_V(G) - 2 \rfloor + 1 \le \lg(\mathrm{Diam}_V(G) - 2) + 1.$$

Therefore,

$$PD(\mathcal{F}) = VCDim(\mathcal{F}^+) = k \le \lfloor \lg \mathrm{Diam}_V(G) - 2 \rfloor + 1.$$

□

By Theorem 3.2 and Definition 2.3, $L_U(f_v) = p(v)$ and $L_S(f_v) = \tilde{p}(v)$, respectively, for each $v \in V$ and $f_v \in \mathcal{F}$. Thus, $|L_S(f_v) - L_U(f_v)| = |\tilde{p}(v) - p(v)|$, and by Theorems 2.6 and 4.1, we have that a sample of size $\lceil \frac{c}{\epsilon^2} (\lfloor \lg \mathrm{Diam}_V(G) - 2 \rfloor + 1 - \ln \delta) \rceil$ suffices to our algorithm, for given $0 < \epsilon, \delta < 1$.

The problem of computing the diameter of G is not known to be easier than the problem of computing all of its shortest paths [3], so obtaining an exact value for the diameter would defeat the whole purpose of using a sampling strategy that avoids computing all shortest paths. Hence, we use a 2-approximation for the diameter described in [14] for undirected graphs. If G is directed, we use this strategy in the underlying undirected graph. Note that this method does not guarantee the desired approximation factor in the directed case, however, in practice the algorithm gives very tight bounds as we show in our experimental evaluation.

We note that the diameter can be approximated within smaller factors, but even for a $(\frac{3}{2}, \frac{3}{2})$-approximation algorithm (i.e., an algorithm that outputs a solution of size at most $\frac{3}{2} \cdot \mathrm{diam}(G) + \frac{3}{2}$) the complexity is $\tilde{O}(m\sqrt{n} + n^2)$ [3], what would also be a bottleneck to our algorithm. Furthermore, since in our case we do not need the largest shortest path, but simply the value of the diameter, and we take logarithm of this value, the approximation of [14] is sufficient.

4.1 Algorithm description and analysis

Given a directed weighted graph $G = (V, E)$ and the percolation states x_v for each $v \in V$ as well as the quality and confidence parameters $0 < \epsilon, \delta < 1$, assumed to be constants (they do not depend on the size of G) respectively, the Algorithm 2 works as follows. At the beginning of the execution the approximated value $\mathrm{Diam}_V(G)$ for the vertex-diameter of G is obtained, in line 2, by a 2-approximation described in [14], if the graph is undirected, as previously mentioned. If the input graph is directed, we can use the same approximation algorithm, ignoring the edges directions, which may not guarantee the approximation factor of 2, but it is good in practice as shown in the experimental evaluation (Section 5). According to Theorem 4.1, this value is used to determine the sample size, denoted by r, in line 3.

The value $minus\_s[v] = \sum_{(f,d)\in V^2 : f\neq v\neq d} R(x_f - x_d)$ for each vertex $v \in V$, which are necessary to compute $\tilde{p}(v)$, is obtained in line 5 by the linear time dynamic programming strategy presented in Algorithm 1. The correctness of Algorithm 1 is not self evident, so we provide a proof of its correctness in Theorem 4.2.

A pair $(u, w) \in V^2$ is sampled uniformly and independently, and then a shortest path p_{uw} between (u, w) is sampled independently in S_{uw} in lines 10–16. For a vertex $z \in \mathrm{Int}(p_{uw})$, the value $\frac{1}{r}\frac{R(x_u - x_w)}{minus\_s[z]}$ is added to $\tilde{p}(z)$.

Theorem 4.2. *For an array A of size n, sorted in non-decreasing order, Algorithm 1 returns for sum and $minus\_sum[k]$, respectively, the values*

$$\sum_{i=1}^{n}\sum_{j=1}^{n} R(A[j] - A[i])$$

and

$$\sum_{\substack{i=1\\ i\neq k}}^{n}\sum_{\substack{j=1\\ j\neq k}}^{n} R(A[j] - A[i]),$$

for each $k \in \{1, \ldots, n\}$.

Proof. By the definition of *sum*, we have that

$$\begin{aligned} sum &= \sum_{i=1}^{n}\sum_{j=1}^{n} R(A[i] - A[j]) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} R(A[j] - A[i]) \\ &= \sum_{i=1}^{n}\sum_{j=1}^{n} \max\{A[j] - A[i], 0\}. \end{aligned}$$

Since A is sorted, then $\max\{A[j] - A[i], 0\} = 0$ if $j < i$. Hence, if we consider only the $j \geq i$, this value becomes

$$sum = \sum_{i=1}^{n}\sum_{j=i}^{n}(A[j] - A[i]).$$

A similar step can be applied to the values of the array *minus_sum*, and then for all indices $k \in \{1, \ldots, n\}$,

$$\begin{aligned} minus\_sum[k] &= \sum_{\substack{i=1\\ i\neq k}}^{n}\sum_{\substack{j=1\\ j\neq k}}^{n} \max\{A[j] - A[i], 0\} \\ &= \sum_{\substack{i=1\\ i\neq k}}^{n}\sum_{\substack{j=i\\ j\neq k}}^{n}(A[j] - A[i]). \end{aligned}$$

The recurrences below follow directly from lines 5 and 6, where sum_k denotes the value of *sum* at the beginning of the k-th iteration of the algorithm.

$$svp[k] = \begin{cases} 0, & \text{if } k = 1 \\ svp[k-1] + A[k-1], & \text{otherwise.} \end{cases}$$

$$sum_k = \begin{cases} 0, & \text{if } k = 1 \\ sum_{k-1} + (k-1)A[k] - svp[k], & \text{otherwise.} \end{cases}$$

The solutions to the above recurrences are, respectively,

$$svp[k] = \sum_{i=1}^{k-1} A[i]$$

and

$$sum_k = \sum_{i=1}^{k} ((i-1)A[i] - svp[i]).$$

The value *sum* is then correctly computed in lines 4–6, since

$$\begin{aligned} sum &= \sum_{i=1}^{n}\sum_{j=i}^{n}(A[j] - A[i]) = \sum_{i=1}^{n}\sum_{j=i}^{n} A[j] - \sum_{i=1}^{n}\sum_{j=i}^{n} A[i] \\ &= \sum_{i=1}^{n}\sum_{j=i}^{n} A[j] - \sum_{i=1}^{n}(n-i+1)A[i] \\ &= \sum_{j=1}^{n}\sum_{i=1}^{j} A[j] - \sum_{i=1}^{n}(n-i+1)A[i] \\ &= \sum_{j=1}^{n} jA[j] - \sum_{i=1}^{n}(n-i+1)A[i] = \sum_{i=1}^{n} iA[i] - \sum_{i=1}^{n}(n-i+1)A[i] \\ &= \sum_{i=1}^{n}(i-1)A[i] - \sum_{i=1}^{n}(n-i)A[i] = \sum_{i=1}^{n}(i-1)A[i] - \sum_{i=1}^{n}\sum_{j=1}^{i-1} A[j] \\ &= \sum_{i=1}^{n}\left((i-1)A[i] - \sum_{j=1}^{i-1} A[j]\right) = \sum_{i=1}^{n}((i-1)A[i] - svp[i]). \end{aligned}$$

Finally, *minus_sum* is correctly computed in lines 8 and 9, since

$$\begin{aligned} minus\_sum[k] &= \sum_{\substack{i=1\\ i\neq k}}^{n}\sum_{\substack{j=i\\ j\neq k}}^{n}(A[j] - A[i]) \\ &= \sum_{i=1}^{n}\sum_{j=i}^{n}(A[j] - A[i]) - \left(\sum_{j=1}^{k-1}(A[k] - A[j]) + \sum_{j=k+1}^{n}(A[j] - A[k])\right) \\ &= sum - \left(\sum_{j=1}^{k-1} A[k] - \sum_{j=k+1}^{n} A[k] - \sum_{j=1}^{k-1} A[j] + \sum_{j=k+1}^{n} A[j]\right) \\ &= sum - \left((k-1)A[k] - (n-(k+1)+1)A[k] - \sum_{j=1}^{k-1} A[j] + \sum_{j=k+1}^{n} A[j]\right) \\ &= sum - \left((2k-n-1)A[k] + \sum_{j=1}^{n} A[j] - \sum_{j=1}^{k-1} A[j] - A[k] - \sum_{j=1}^{k-1} A[j]\right) \\ &= sum - \left((2k-n-2)A[k] + \sum_{j=1}^{n} A[j] - 2\sum_{j=1}^{k-1} A[j]\right) \\ &= sum - (2k-n-2)A[k] - svp[n+1] + 2svp[k]. \end{aligned}$$

□

Theorem 4.3. *Let $S = \{p_{u_i w_i}, \ldots, p_{u_r w_r}\}$ be a sample of size*

$$r = \frac{c}{\epsilon^2}(\lfloor \lg Diam_V(G) - 2 \rfloor + 1) - \ln\delta)$$

for a given directed weighted graph $G = (V, E)$ and for given $0 < \epsilon, \delta < 1$. Algorithm 2 returns with probability at least $1 - \delta$ an

Algorithm 1: getPercolationDifferences(A, n)

Data: Array A, sorted in non-decreasing order, and $n = |A|$.

Result: The value $sum = \sum_{i=1}^{n}\sum_{j=1}^{n} R(A[j] - A[i])$ and the array $\{minus\_sum[k] = \sum_{\substack{i=1 \\ i\neq k}}^{n}\sum_{\substack{j=1 \\ j\neq k}}^{n} R(A[j] - A[i]), \forall k \in \{1, ..., n\}\}$, such that $R(z) = \max\{z, 0\}$.

```
1  sum ← 0
2  minus_sum[i] ← 0, ∀i ∈ {1, ..., n}
3  svp ← (0, 0, ..., 0)
4  for i ← 2 to n do
5      svp[i] ← svp[i − 1] + A[i − 1]
6      sum ← sum + (i − 1)A[i] − svp[i]
7  svp[n + 1] ← svp[n] + A[n]
8  for i ← 1 to n do
9      minus_sum[i] ←
         sum − A[i](2i − n − 2) − svp[n + 1] + 2svp[i]
10 return sum, minus_sum
```

Algorithm 2: PercolationCentralityApproximation(G, x, ϵ, δ)

Data: Graph $G = (V, E)$ with $n = |V|$, percolation states x, accuracy parameter $0 < \epsilon < 1$, confidence parameter $0 < \delta < 1$.

Result: Approximation $\tilde{p}(v)$ for the percolation centrality of all vertices $v \in V$.

```
1  p̃[v] ← 0, minus_s[v] ← 0, ∀v ∈ V
2  Diam_V(G) ← getVertexDiameter(G)
3  r ← ⌈(c/ε²)(⌊lg Diam_V(G) − 2⌋ + 1 − ln δ)⌉
4  sort x            /* after sorted, x = (x_1, x_2, ..., x_n) */
5  minus_s ← getPercolationDifferences(x, n)
6  for i ← 1 to r do
7      sample u ∈ V with probability 1/n
8      sample w ∈ V with probability 1/(n − 1)
9      S_uw ← allShortestPaths(u, w)
10     if S_uw ≠ ∅ then
11         t ← w
12         while t ≠ u do
13             sample z ∈ P_u(t) with probability σ_uz/σ_ut
14             if z ≠ u then
15                 p̃[z] ← p̃[z] + (1/r)(R(x_u − x_w)/minus_s[z])
16             t ← z
17 return p̃[v], ∀v ∈ V
```

approximation $\tilde{p}(v)$ to $p(v)$, for each $v \in V$, such that $\tilde{p}(v)$ is within ϵ error.

Proof. Each pair (u_i, w_i) is sampled with probability $\frac{1}{n(n-1)}$ in lines 7 and 8, and for each pair, the set $S_{u_i w_i}$ is computed by Dijkstra algorithm (line 9). A shortest path $p_{u_i w_i}$ is sampled independently in $S_{u_i w_i}$ (lines 10–16), i.e., with probability $\frac{1}{\sigma_{u_i w_i}}$, by a backward traversing starting from w_i (Lemma 5 in [14], Section 5.1). Therefore, $p_{u_i w_i}$ is sampled with probability $\frac{1}{n(n-1)}\frac{1}{\sigma_{u_i w_i}}$.

In lines 12–16, each $z \in p_{u_i w_i}$ reached by the backward traversing has its value increased by $\frac{1}{r}\frac{R(x_{u_i} - x_{w_i})}{minus\_s[z]}$. The value of $minus\_s[z]$ is correctly computed as shown in Theorem 4.2. Let $S' \subseteq S$ be the set of shortest paths that z is an internal vertex. Then, at the end of the r-th iteration,

$$\tilde{p}(z) = \frac{1}{r}\sum_{p_{gh}\in S'} \frac{R(x_g - x_h)}{\sum\limits_{\substack{(f,d)\in V^2 \\ f\neq z\neq d}} R(x_f - x_d)} = \frac{1}{r}\sum_{p_{u_i w_i}\in S} \frac{R(x_{u_i} - x_{w_i})}{\sum\limits_{\substack{(f,d)\in V^2 \\ f\neq z\neq d}} R(x_f - x_d)} \mathbb{1}_{\tau_v}(p_{u_i w_i})$$

which corresponds to

$$\tilde{p}(z) = \frac{1}{r}\sum_{p_{u_i w_i}\in S} f_z(p_{u_i w_i}).$$

Since $L_S(f_v) = \tilde{p}(v)$ and $L_U(f_v) = p(v)$ (Theorem 3.2) for all $v \in V$ and $f_v \in \mathcal{F}$, and S is a sample such that $|S| = r$, then $\Pr(|\tilde{p}(v) - p(v)| \leq \epsilon) \geq 1 - \delta$ for each $v \in V$ (Theorem 2.6). □

Theorem 4.4. *Given a directed weighted graph $G = (V, E)$ with $n = |V|$ and $m = |E|$ and a sample of size*

$$r = \frac{c}{\epsilon^2}(\lfloor \lg Diam_V(G) - 2 \rfloor + 1) - \ln \delta),$$

Algorithm 2 has running time $O(m \log^2 n)$.

Proof. We use the linear time algorithm of [18] for the sampling step in lines 7, 8 and 13. The upper bound to the vertex-diameter, computed in line 2 and denoted by $\mathrm{Diam}_V(G)$, is obtained by the approximation of [14], which runs in time $O(m \log n)$.

Sorting the percolation states array x (line 4) can be done in $O(n \log n)$ time and the execution of Algorithm 1 on the sorted array x (line 5) has running time $O(n)$. As for the loop in lines 12–16, the complexity analysis is as follows. Once $|P_u(w)| \leq d_G(w)$, where $d_G(w)$ denotes the degree of w in G, and since this loop is executed at most n times if the sampled path traverses all the vertices of G, the total running time of these steps corresponds to $\sum_{v\in V} d_G(v) = 2m = O(m)$.

The loop in lines 6–16 runs r times and the Dijkstra algorithm which is executed in line 9 has running time $O(m + n \log n) = O(m \log n)$, so the total running time of Algorithm 2 is $O(n \log n + r \max(m, m \log n)) = O(n \log n + r(m \log n)) = O(r(m \log n)) = O(\log n(m \log n)) = O(m \log^2 n)$. □

Corollary 4.5. *Given an unweighted graph $G = (V, E)$ with $n = |V|$ and $m = |E|$ and a sample of size $r = \frac{c}{\epsilon^2}(\lfloor \lg Diam_V(G) - 2 \rfloor + 1) - \ln \delta)$, Algorithm 2 has running time $O(\log n(m + n))$.*

Proof. The proof is analogous to the one of Theorem 4.4, with the difference that the shortest paths between a sampled pair $(u, w) \in V^2$ are computed by a BFS, which has running time $O(r(m + n)) = O(\log n(m + n))$. □

We observe that, even though it is an open problem whether there is a $O(n^{3-c})$ algorithm for computing all shortest paths in weighted graphs, in the unweighted case there is a $O(n^{2.38})$ (non-combinatorial) algorithm for this problem [16]. However, even if this algorithm could be adapted to compute betweenness/percolation centrality (what is not clear), our algorithm obtained in Corollary 4.5 is still faster.

5 EXPERIMENTAL EVALUATION

In Section 5.1 we perform an experimental evaluation of our approach using real-world networks. Since, as far as we know, our algorithm is the first estimation algorithm for percolation centrality, we compare our results with the best known algorithm for computing the same measure in the exact case. As expected, our algorithm is many times faster than the exact algorithm, since the exact computation of the percolation centrality takes $O(n^3)$ time, which is extremely time consuming. Additionally, a main advantage of our algorithm is that it outputs an estimation with a very small error. In fact, for every one of the networks used in our experiment, the average estimation error are kept below the quality parameter ϵ by many orders of magnitude. For instance, even for $\epsilon = 0.1$ (the largest in our experiments), the average estimation error is in the order of 10^{-11} and maximum estimation error of any one of the centrality measure is in the order of 10^{-9}.

Additionally, we also run experiments using synthetic graphs in order to validate the scalability of our algorithm, since for this task we need a battery of "similar" graphs of increasing size. We use power-law graphs generated by the Barabási-Albert model [5] for such experiments.

We use Python 3.7 language in our implementation[1] and, for graph manipulation, we use the NetworkX library [7]. The implementation of the exact algorithm for the percolation centrality is the one available in referred NetworkX library. The experiments were performed on a 2.8 Mhz Intel i7-4900MQ quad core with 12GB of RAM and Windows 10 64-bit operating system.

In all experiments, we set the percolation state x_v, for each $v \in V$, as a random number between 0 and 1, and the weights of each edge $e \in E$ as a random number between 1 and 100. The parameters δ and c remained fixed. We set $\delta = 0.1$ for all experiments and $c = 0.5$ as suggested by Löffler and Phillips (2009) [10].

5.1 Real-world graphs

In this section we describe the experimental results obtained in our work. We use graphs from the *Stanford Large Network Dataset Collection* [8], which are publicly available real-world graph datasets. These spans graphs from social, peer-to-peer, autonomous systems and collaboration networks. The details of the datasets can be seen in Table 1. In this table, $\widetilde{VD}(G)$ is the upper bound for the vertex-diameter of the graph obtained in the execution of our algorithm. The last two columns in the same table show by which factor our algorithm is faster than the algorithm for computing the exact centrality. We compute this factor by dividing the running time of the exact method by the running time of our algorithm. We run our algorithm five times and show in the table minimum and the maximum factors. Note that for the largest graph our algorithm is around 30 times faster than the exact method. For this table, the running time of our algorithm is taken for $\epsilon = 0.04$.

In our experimental evaluation, for every graph, we also run our algorithm for different values of ϵ, more precisely set to 0.04, 0.06, 0.08 and 0.1. In Table 2 we show how many times our algorithm is faster than the exact method in the average of the five runs.

The error of the estimation computed by our algorithm is within ϵ for every vertex of every graph even though this guarantee could possibly fail with probability $\delta = 0.1$. However, in addition to the better confidence results than the theoretical guarantee, the most surprising fact is that for every graph used in the experiment the maximum error among the error for the estimation of every vertex is around 10^{-9} and the average error among all vertices is around 10^{-11}, even when we set the quality guarantee to $\epsilon = 10^{-1}$. We run our algorithm five times for every graph for each value of ϵ. Them maximum error is taken among all five runs. These results are shown in Figures 1, 2 and 3.

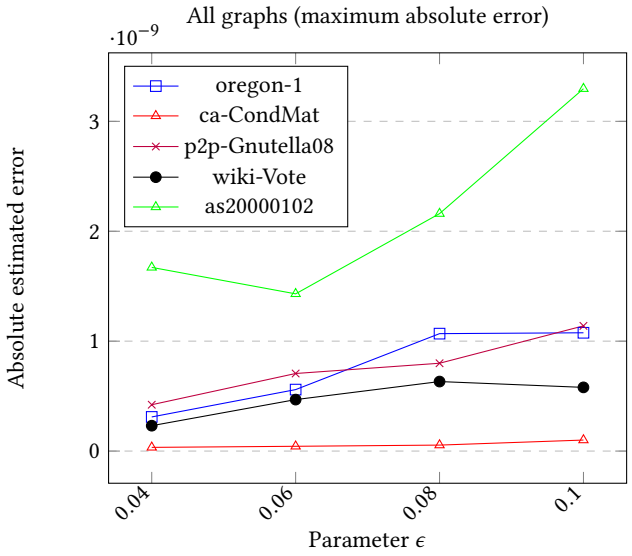

Figure 1: Percolation centrality absolute maximum error estimation.

5.2 Synthetic graphs

In the scalability experiments, we used a sequence of synthetic graphs increasing in size and compared the execution time of our estimating algorithm with the algorithm for the computation of the exact centrality provided by NetworkX library.

We also use the same library for generating random power-law graphs, by the Barabási-Albert model [5] with each vertex creating two edges, obtaining power-law graphs with average degree of 2. In the experiments we use graphs with the number of vertices n in $\{100, 250, 500, 1000, 2000, 4000, 8000, 16000\}$. The value of ϵ is fixed at 0.05. The results are shown in Figure 4.

[1]The implementation of our algorithm can be found at https://github.com/alanemarie/percolationApproximation.

Graph	Type	$\lvert V\rvert$	$\lvert E\rvert$	$\widetilde{VD}(G)$	$\frac{\text{exact time}}{\text{approx. time}}$ Min	$\frac{\text{exact time}}{\text{approx. time}}$ Max
wiki-Vote	Directed	7115	103689	11	10.21	10.49
p2p-Gnutella08	Directed	6301	20777	14	10.23	9.5
oregon1-010331	Undirected	10670	22002	14	17.72	17.6
ca-CondMat	Undirected	23133	93497	21	31.19	29.44
asas20000102	Undirected	6474	13895	14	9.18	9.72

Table 1: Datasets details for the real-world graphs.

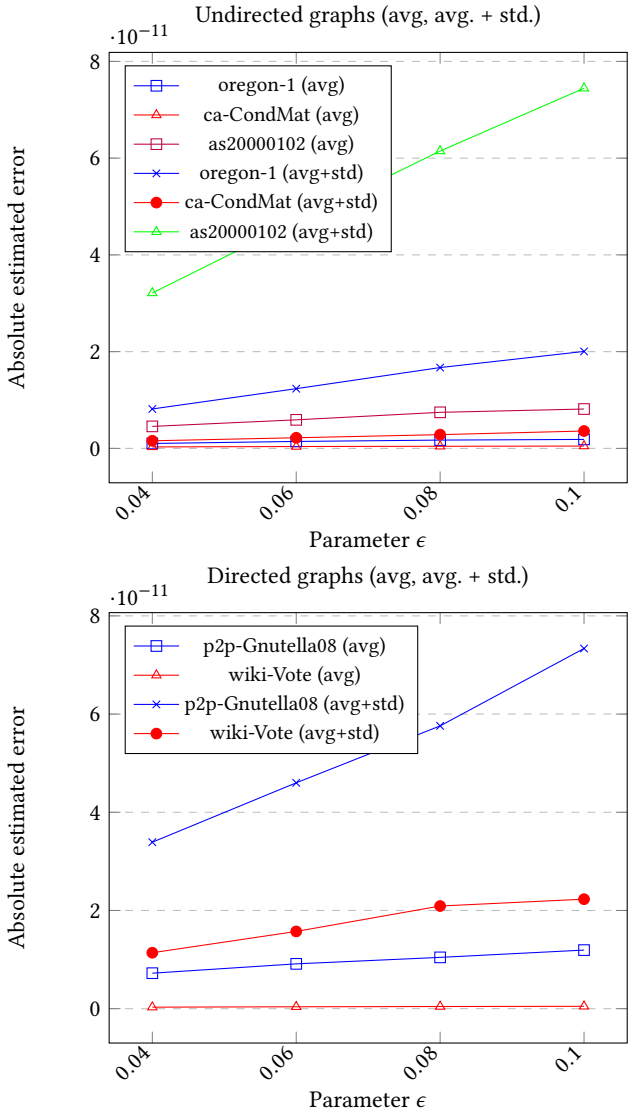

Figure 2: Percolation centrality absolute error estimation.

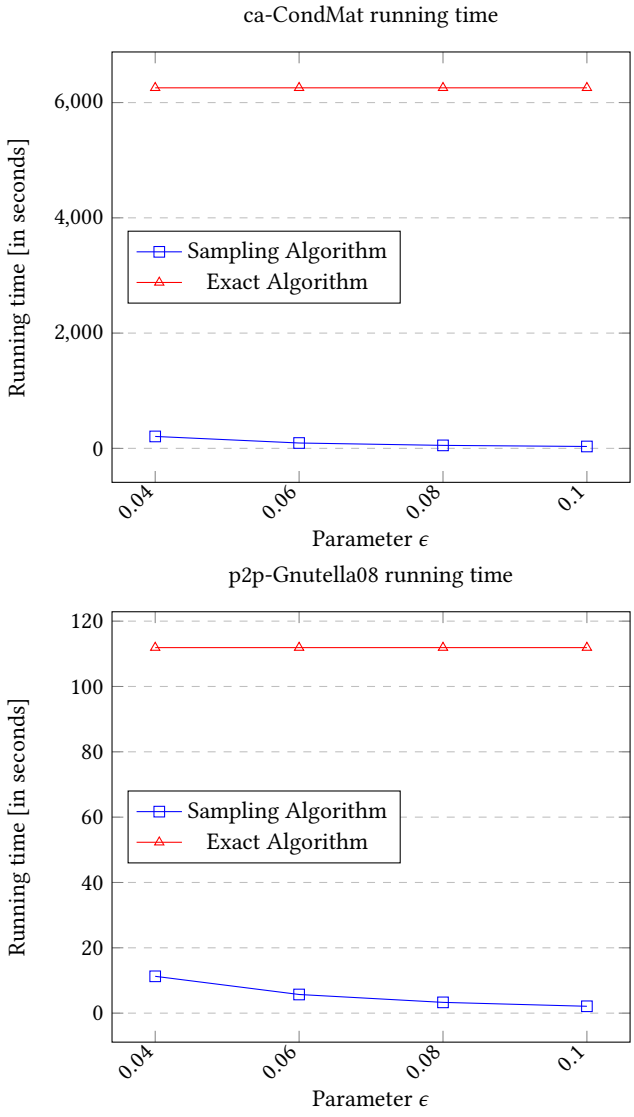

Figure 3: Average running time for 5 runs of Algorithm 2

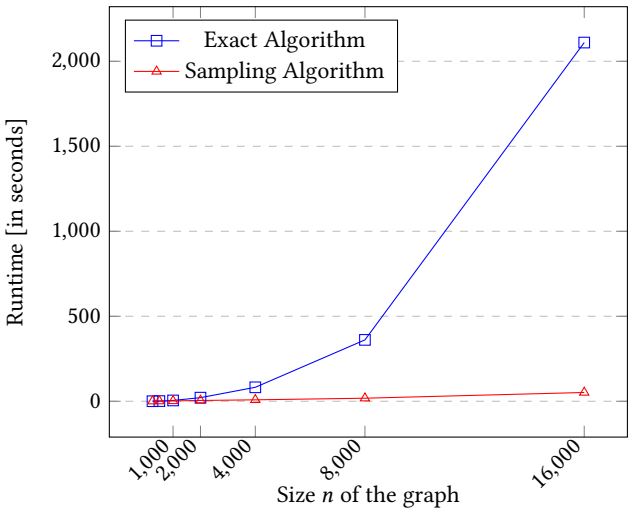

Figure 4: Scalability experiments, with $\epsilon = 0.05$.

exact/approx. time average ratio				
Graph	$\epsilon = 0.04$	$\epsilon = 0.06$	$\epsilon = 0.08$	$\epsilon = 0.1$
wiki-Vote	10.36	22.17	36.93	52.71
p2p-Gnutella	9.92	19.59	33.91	53.21
oregon1-010331	17.66	39.7	70.14	108.71
ca-CondMat	30.25	67.32	120.98	191.1
as20000102	9.61	20.67	38.81	55.4

Table 2: Ratio for average running time after 5 runs of the percolation centrality estimation algorithm and the exact algorithm.

6 CONCLUSION

We presented an algorithm with running time $\tilde{O}(m)$ for estimating the percolation centrality for every vertex of a weighted graph. The estimation obtained by our algorithm is within ϵ of the exact value with probability $1 - \delta$, for *fixed* constants $0 < \epsilon, \delta < 1$.

Since many large scale graphs are sparse and have small diameter (typically of size $\log n$), our algorithm provides a fast approximation for such graphs (more precisely running in $O(n \log n \log \log n)$ time). We validate our proposed method performing experiments on real-world networks. As expected, our algorithm is much faster than the exact algorithm (around 30 times for the largest graph) and, furthermore, in practice the estimation error is many orders of magnitude smaller than the theoretical worst case guarantee for graphs of a variety of sizes.

ACKNOWLEDGMENTS

This work was partially funded by CNPq(Proc. 428941/2016-8) and CAPES.

REFERENCES

[1] A. Abboud and V. Williams. 2014. Popular Conjectures Imply Strong Lower Bounds for Dynamic Problems. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science*. 434–443. https://doi.org/10.1109/FOCS.2014.53
[2] A. Abboud, V. Williams, and H. Yu. 2018. Matching Triangles and Basing Hardness on an Extremely Popular Conjecture. *SIAM J. Comput.* 47, 3 (2018), 1098–1122. https://doi.org/10.1137/15M1050987 arXiv:https://doi.org/10.1137/15M1050987
[3] D. Aingworth, C. Chekuri, and R. Motwani. 1996. Fast Estimation of Diameter and Shortest Paths (Without Matrix Multiplication). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '96)*. 547–553.
[4] M. Anthony and P. L. Bartlett. 2009. *Neural Network Learning: Theoretical Foundations* (1st ed.). Cambridge University Press, New York, NY, USA.
[5] Albert-László Barabási and Réka Albert. 1999. Emergence of Scaling in Random Networks. *Science* 286, 5439 (1999), 509–512.
[6] S. R. Broadbent and J. M. Hammersley. 1957. Percolation processes: I. Crystals and mazes. *Math. Proc. of the Cambridge Philosophical Society* 53, 3 (1957), 629–641.
[7] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. 2008. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)*. Pasadena, CA USA, 11–15.
[8] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.
[9] Yi Li, Philip M. Long, and Aravind Srinivasan. 2001. Improved Bounds on the Sample Complexity of Learning. *J. Comput. System Sci.* 62, 3 (2001), 516 – 527. https://doi.org/10.1006/jcss.2000.1741
[10] M. Löffler and J. M. Phillips. 2009. Shape Fitting on Point Sets with Probability Distributions. In *Algorithms - ESA 2009, 17th Annual European Symposium, Copenhagen, Denmark, September 7-9, 2009. Proceedings*. Springer Berlin Heidelberg, 313–324.
[11] M. Mitzenmacher and E. Upfal. 2017. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis* (2nd ed.). Cambridge University Press.
[12] M. Mohri, A. Rostamizadeh, and A. Talwalkar. 2012. *Foundations of Machine Learning*. The MIT Press.
[13] M. Piraveenan, M. Prokopenko, and L. Hossain. 2013. Percolation Centrality: Quantifying Graph-Theoretic Impact of Nodes during Percolation in Networks. *PLOS ONE* 8, 1 (2013), 1–14.
[14] M. Riondato and E. M. Kornaropoulos. 2016. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery* 30, 2 (2016), 438–475.
[15] M. Riondato and E. Upfal. 2018. ABRA: Approximating Betweenness Centrality in Static and Dynamic Graphs with Rademacher Averages. *ACM Trans. Knowl. Discov. Data* 12, 5, Article 61 (July 2018), 38 pages.
[16] R. Seidel. 1995. On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. *J. Comput. System Sci.* 51, 3 (1995), 400 – 403.
[17] S. Shalev-Shwartz and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
[18] M. D. Vose. 1991. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on software engineering* 17, 9 (1991), 972–975.
[19] R. Williams. 2018. Faster All-Pairs Shortest Paths via Circuit Complexity. *SIAM J. Comput.* 47, 5 (2018), 1965–1985. https://doi.org/10.1137/15M1024524 arXiv:https://doi.org/10.1137/15M1024524