Percolation Centrality via Rademacher Complexity

Alane M. de Lima[a], Murilo V. G. da Silva[a] and André L. Vignatti[a]

[a]*Department of Computer Science, Federal University of Paraná, Curitiba – Paraná – Brazil*

ARTICLE INFO

Keywords:
percolation centrality
approximation algorithm
pseudo-dimension
rademacher averages

ABSTRACT

In this work we investigate the problem of estimating the percolation centrality of all vertices in a weighted graph. The percolation centrality measure quantifies the importance of a vertex in a graph that is going through a contagious process. The fastest exact algorithm for the computation of this measure in a graph G with n vertices and m edges runs in $\mathcal{O}(n^3)$. Let $\mathrm{Diam}_V(G)$ be the maximum number of vertices in a shortest path in G. In this paper we present an expected $\mathcal{O}(m \log n \log \mathrm{Diam}_V(G))$ time approximation algorithm for the estimation of the percolation centrality for all vertices of G. We show in our experimental analysis that in the case of real-world complex networks, the output produced by our algorithm returns approximations that are even closer to the exact values than its guarantee in terms of theoretical worst case analysis.

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 Broadbent and Hammersley (1957) [8] in the context of the passage of a fluid in a medium. In graphs, percolation centrality was proposed by Piraveenan *et al.* (2013) [17], 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 best-known algorithms that exactly compute the betweenness centrality for all vertices of a graph depends on computing all its shortest paths [19] and, consequently, the same applies in the computation of percolation centrality. The fastest algorithm for this task for weighted graphs, proposed by Williams (2014) [24], runs in $\mathcal{O}\left(n^3/2^{c\sqrt{\log n}}\right)$ time, for some constant c. Currently it is a central open problem in graph theory whether this problem can be solved in $\mathcal{O}(n^{3-c})$, 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 many applications, the complexity of the exact computation for the betweenness centrality can be improved to $\mathcal{O}(n^2)$. 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 main contributions of our work are approximations algorithms to estimate the percolation centrality of all vertices of a graph. The present paper is an extended version of a conference paper where we describe an approximation algorithm for the problem using sample complexity theory. In the conference version we designed a fixed-size sample algorithm for this task, while in the current paper we show how the algorithm can be modified so we have a progressive sampling approach. Sections 2.3, 3.2 and 4.1 describe the results and techniques that we use in this extension. We also provide experimental evaluation, in Section 5, for the progressive sampling algorithm. In both works, we follow the

⋆This document is the results of the research project funded by the CAPES and CNPq(Proc. 428941/2016-8).

⋆Partial results in this full paper have been published in the proceedings of ACM KDD'20 at [9].

`http://www.inf.ufpr.br/amlima`, `amlima@inf.ufpr.br` (A.M. de Lima);
`http://www.inf.ufpr.br/murilo`, `murilo@inf.ufpr.br` (M.V.G. da Silva);
`http://www.inf.ufpr.br/vignatti`, `vignatti@inf.ufpr.br` (A.L. Vignatti)
ORCID(s): 0000-0003-4575-2401 (A.M. de Lima); 0000-0002-3392-714X (M.V.G. da Silva); 0000-0001-8268-5215 (A.L. Vignatti)

steps of Riondato, Kornaropoulos and Upfal [19, 20], which designed an approximation algorithm for the betweenness centrality problem under the light of sample complexity theory. A main theme we deal with is the fact that for large scale graphs, even algorithms with time complexity that scales in quadratic time are inefficient in practice and high-quality approximations obtained with high confidence are usually sufficient in real-world applications. In [20], 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 any fixed constants $0 < \epsilon, \delta < 1$, they obtain values within ϵ from the exact value with probability $1 - \delta$.

In this paper we combine techniques presented in Lima *et al.* (2020) [9] on Pseudo-dimension theory applied to percolation centrality, and in the work of Riondato and Upfal on Rademacher Averages applied to betweenness centrality. We show that this combination can be further developed for giving an approximation algorithm for the percolation centrality based on a progressive sampling strategy. The idea is that the algorithm iteratively increases the size of a sample of shortest paths used for the estimation of the percolation centrality until the desired accuracy is achieved. The stop condition depends on the Rademacher Averages of the current sample of shortest paths. One of the consequences of the approach based on Rademacher Averages is that such technique is sensitive to the input distribution, so it can provide tighter bounds for certain inputs. Additionally, even if no assumption is made on the input distribution, we show that with the use of Vapnik–Chervonenkis (VC) theory on the sample analysis we can obtain a sample size that is tighter than the one given by standard Hoeffding and union-bound techniques, and never worse than the sample size given by the fixed-size sample algorithm (in the conference version of this paper we used only fixed-size samples).

We have in mind both a theoretical and a practical perspective. More precisely, we show that the estimation of the percolation centrality can be computed in $\mathcal{O}(m \log n \log \mathrm{Diam}_V(G))$ expected time, where $\mathrm{Diam}_V(G)$ is the maximum number of vertices in a shortest path of G. Note that since many real-world graphs are sparse and have logarithmic diameter, the time complexity of the algorithm for such graphs is $\mathcal{O}(n \log n \log \log n)$. In the practical front, in Section 5, we give the relation between the quality and confidence constants and the sample size required for meeting the approximation guarantee and, in fact, our experimental evaluation shows that our algorithms produce results that are orders of magnitude better than the guarantees given by the referred theoretical analysis.

2. Preliminaries

In this section, we present the definitions, notation and results that are the groundwork of our proposed algorithms. In all results of this paper, we assume w.l.o.g. that the input graph is connected, since the algorithms can be applied separately to each of its connected components.

2.1. Graphs and Percolation Centrality

Given a directed weighted graph $G = (V, E)$, the percolation states $0 \leq x_v \leq 1$ 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 $\mathrm{Int}(p_{uw})$ be the set of internal vertices of p_{uw}, that is, $\mathrm{Int}(p_{uw}) = \{v \in V : v \in p_{uw} \text{ and } u \neq v \neq w\}$. 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 define $\mathrm{Diam}_V(G)$ as the vertex-diameter of G, i.e. the maximum number of vertices in a shortest path in G. We say a vertex 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 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 by Piraveenan *et al.* [17] does not have the normalization factor $\frac{1}{n(n-1)}$, introduced in this paper with the purpose of defining a proper probability distribution in Section 3. This normalization preserves the original relation among the vertices centralities.

2.2. Sample Complexity and Pseudo-dimension

In sampling algorithms, typically the estimation of a certain quantity observing parameters of quality and confidence is desired. The sample complexity analysis relates the minimum size of a random sample required to estimate results that are consistent with such desired parameters (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, a central concept in sample complexity theory, is especially defined in order to model the particular problem that one is dealing. There is, an upper bound to the VC-dimension of the sampling problem at hand is also an upper bound to the sample size which respects the desired quality and confidence parameters. Generally speaking, the VC-dimension measures the expressiveness of a class of subsets defined on a set of points [19].

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 (2009) [4], Mohri *et al.* (2012) [15], Shalev-Shwartz and Ben-David (2014) [23] and Mitzenmacher and Upfal (2017) [14].

Definition 2 (Range Space). *A* range space *is a pair $\mathcal{R} = (U, \mathcal{I})$, where U is a domain (finite or infinite) and $\mathcal{I}$ is a collection of subsets of U, called* ranges.

For a given $S \subseteq U$, 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 3 (VC-dimension). *The VC-dimension of a range space $\mathcal{R} = (U, \mathcal{I})$, denoted by VCDim($\mathcal{R}$), is VCDim($\mathcal{R}$) = $\max\{k : \exists S \subseteq U$ such that $|S| = k$ and $|\mathcal{I}_S| = 2^k\}$.*

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 4 (Pseudo-dimension (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($\mathcal{R}'$).*

The following combinatorial object, called ϵ-sample, is useful when one wants to intersect ranges of a sufficient size with respect to the right relative frequency of each range in $\mathcal{I}$ within the sample S.

Definition 5 (ϵ-sample). *Given $0 < \epsilon < 1$, a set $S \subseteq U$ is called ϵ-sample w.r.t. a range space $\mathcal{R} = (U, \mathcal{I})$ and a probability distribution π on U if $\forall I \in \mathcal{I}$, $\left|\Pr_\pi(I) - \frac{|S \cap I|}{|S|}\right| \leq \epsilon$.*

A more general definition of ϵ-sample (called ϵ-representative) is given below for a given a domain U, a set of values of interest $\mathcal{H}$, and a family of functions $\mathcal{F}$ 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 sampled independently from U with respect to a probability distribution π.

Definition 6. *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. $L_U(f_h) = \mathbb{E}_{u\sim\pi}[f_h(u)]$ and $L_S(f_h) = \frac{1}{r}\sum_{s\in S} f_h(s)$.*

Definition 7. *Given $0 < \epsilon, \delta < 1$, a set $S \subseteq$ 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 $\forall f_h \in \mathcal{F}$, $|L_S(f_h) - L_U(f_h)| \leq \epsilon$.*

As observed by [23], by the linearity of expectation we have that 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*, $L_S(f_h)$ almost surely 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. For any sample size, though, no information about the value $|L_S(f_h) - L_U(f_h)|$ is available by the application of the referred law. 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 $0 < \epsilon, \delta < 1$.

Theorem 1 states that having an upper bound to the Pseudo-dimension of a range space allows to build an ϵ-sample.

Theorem 1 (see [10], Theorem 2.12). *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 an ϵ-sample with probability at least $1 - \delta$.*

Löffler and Phillips [13] observe empirically that the constant c is approximately $\frac{1}{2}$. Lemmas 1 and 2, stated and proved by [20], present constraints on the sets that can be shattered by a range set $\mathcal{F}^+$.

Lemma 1 (see [20], 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 $d \in U$ and for some $y \in [0, 1]$.*

Lemma 2 (see [20], 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 $d \in U$.*

2.3. Progressive Sampling and Rademacher Complexity

In some problems, finding a bound for the sample size that is tight may be a complicated task. An alternative to this issue relies on the use of progressive sampling, in which the process starts with a small sample size which progressively increases until the accuracy can be improved [18]. The use of an appropriate scheduling for the sample increase combined with an efficient-to-evaluate stopping condition (i.e. knowing when the sample is large enough) leads to a greater improvement in time for the estimation of the value of interest [20]. A key idea is that the stopping condition takes into consideration the input distribution, which can be extracted by the use of Rademacher Complexity (see [14], chapter 14). The main results from this theory that is in the core of statistical learning theory and that we apply in our algorithms are presented below.

Consider a sample S and the computation of the maximum deviation of $L_S(f_h)$ from the true expectation of f_h, for all $f_h \in \mathcal{F}$, that is, $\sup_{f_h \in \mathcal{F}} |L_S(f_h) - L_U(f_h)|$. The *empirical Rademacher average* of $\mathcal{F}$ is defined as follows.

Definition 8. *Consider a sample $S = \{z_1, \dots, z_r\}$ and a distribution of r independent Rademacher random variables $\sigma = (\sigma_1, \dots, \sigma_r)$, i.e. $\Pr(\sigma_i = 1) = \Pr(\sigma_i = -1) = 1/2$ for $1 \leq i \leq r$. The empirical Rademacher average of a family of functions $\mathcal{F}$ w.r.t. to S is defined as*

$$\tilde{R}_r(\mathcal{F}, S) = \mathbb{E}_\sigma \left[\sup_{f_h \in \mathcal{F}} \frac{1}{r} \sum_{i=1}^{r} \sigma_i f_h(z_i) \right].$$

At the heart of our algorithm, the stopping condition for the progressive sampling depends on the Rademacher Complexity of the sample. For the connection of the empirical Rademacher average with the value of $\sup_{f_h \in \mathcal{F}} |L_S(f_h) - L_U(f_h)|$, we use the bound of [20], which extended the bound of [16] to the supremum of its absolute value to functions with codomain in $[0, 1]$ and uniform probability distribution on the input.

Theorem 2. *(see [20], Theorem 3.3) With probability at least $1 - \delta$,*

$$\sup_{f_h \in \mathcal{F}} \left| L_S(f_h) - L_U(f_h) \right| \leq 2\tilde{R}_r(\mathcal{F}, S) + \frac{\ln\frac{3}{\delta} + \sqrt{(\ln\frac{3}{\delta} + 4r\tilde{R}_r(\mathcal{F}, S))\ln\frac{3}{\delta}}}{r} + \sqrt{\frac{\frac{3}{\delta}}{2r}}.$$

The exact computation of $\tilde{R}_r(\mathcal{F}, S)$ depends on an extreme value, i.e. the supremum of deviations for all functions in $\mathcal{F}$, which can be expensive and not straight-forward to compute over a large (or infinite) set of functions [14]. For this reason, we use the bound given by Theorem 3, which is a variant of the Massart's Lemma (see Theorem 14.22, [14]) that is convex, continuous in $\mathbb{R}^+$ and can be efficiently minimized by standard convex optimization methods.

Consider the vector $v_{f_h} = (f_h(z_1), \dots, f_h(z_r))$ for a given sample of r elements, denoted by $S = \{z_1, \dots, z_r\}$, and let $\mathcal{V}_S = \{v_{f_h} : f_h \in \mathcal{F}\}$.

Theorem 3. *(see [20], Theorem 3.4) Let $w : \mathbb{R}^+ \to \mathbb{R}^+$ be the function*

$$w(s) = \frac{1}{s} \ln \sum_{v_{f_h} \in \mathcal{V}_S} \exp\left(\frac{s^2 \|v_{f_h}\|_2^2}{2r^2}\right).$$

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

3. Pseudo-dimension and percolated shortest paths

In this section we model the percolation centrality estimation problem in terms of a range set of the percolated shortest paths. In the conference version [9], our algorithm uses a sample of fixed size and a range set where the points of the domain are shortest paths. So, as one of the contributions of the current paper with respect to the conference version, in Section 3.2 we present a range space where the domain corresponds to the pairs of vertices of G. This modification is necessary in the progressive sampling algorithm.

3.1. Range Space defined for the Fixed-Size Sample Algorithm

For a given 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 $\tau_v = \{p \in U : v \in \mathrm{Int}(p)\}$. 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 $f_v(p_{uw}) = \frac{R(x_u - x_w)}{\sum_{(f,d)\in V^2 : f\neq v\neq d} R(x_f - x_d)} \mathbb{1}_{\tau_v}(p_{uw})$, where $\mathbb{1}_{\tau_v}(p_{uw})$ is the indicator function that returns 1 if $v \in \mathrm{Int}(p_{uw})$ (and hence, p_{uw} is in the interval τ_v of vertex v) and 0 otherwise. The function f_v gives the proportion of the percolation between u and w to the total percolation in the graph if $v \in \mathrm{Int}(p_{uw})$. 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 $v \in \mathrm{Int}(p_{uw})$ and $t \leq \frac{R(x_u - x_w)}{\sum_{(f,d)\in V^2 : f\neq v\neq d} R(x_f - x_d)}$. We define $\mathcal{F}^+ = \{R_v : f_v \in \mathcal{F}\}$.

Each $p_{uw} \in U$ is sampled according to the function $\pi(p_{uw}) = \frac{1}{n(n-1)} \frac{1}{\sigma_{uw}}$. In order to see that this is a valid probability distribution, note that

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

We state in the next theorem that $\mathbb{E}[f_v(p_{uw})] = p(v)$ for all $v \in V$.

Theorem 4. *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 6

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

Let $S = \{(p_{u_i w_i}, 1 \le i \le r)\}$ be a collection of r shortest paths sampled independently from U. Next, we define $\tilde{p}(v)$, the estimation to be computed, as the empirical average from Definition 6:

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

3.2. Range Space defined for the Progressive Sampling Algorithm

In this section we describe a modification in the range space so that we can use the bound of [20] in our progressive sampling algorithm, since an uniform probability distribution in the points of the domain is required in this case.

For a given graph $G = (V, E)$ and the percolation states x_v for each $v \in V$, let $U' = V \times V$. Let $f_v : U' \to [0,1]$ be the function

$$f_v(u,w) = \frac{R(x_u - x_w)}{\sum_{(f,d) \in V \times V : u \neq w} R(x_f - x_d)} \frac{\sigma_{u,w}(v)}{\sigma_{u,w}}.$$

We define $\mathcal{F}' = \{f_v : v \in V\}$. There is one range $R_v = \{((u,w),t) : (u,w) \in U' \text{ and } t \le f_v(u,w)\}$ for each $f_v \in \mathcal{F}'$. We define $\mathcal{F}'^{+} = \{R_v : f_v \in \mathcal{F}'\}$. Each $(u,w) \in U'$ is sampled with probability $\pi(u,w) = \frac{1}{n(n-1)}$, which is a valid probability distribution.

For a collection $S' = \{(u_i, w_i), 1 \le i \le r)\}$ of r pairs of vertices sampled independently and identically from U', the estimation $\tilde{p}(v)$ to be computed is the empirical average of f_v, according to Definition 6:

$$\tilde{p}(v) = L_{S'}(f_v) = \frac{1}{r} \sum_{(u_i,w_i) \in S'} f_v(u_i, w_i) = \frac{1}{r} \sum_{(u_i,w_i) \in S'} \frac{R(x_{u_i} - x_{w_i})}{\sum\limits_{\substack{(f,d) \in V \times V \\ f \neq v \neq d}} R(x_f - x_d)} \frac{\sigma_{uw}(v)}{\sigma_{uw}}.$$

4. Estimation for the Percolation Centrality

We present the approximation algorithms for the estimation of the percolation centrality of all vertices of a graph. We give the outline of the algorithm that runs in a fixed-size sample, which we compare with the progressive sampling approach, and that is described in more detail in the conference version of this paper [9]. We chose to present only the idea of the conference version algorithm because the progressive sampling algorithm presented in this paper ends up superseding the previous version. In Section 4.1, we describe the necessary modifications in the fixed-size sample algorithm of the conference version so we obtain a progressive sampling approach.

We first define the problem in terms of range spaces and then we present the algorithms which take as input a directed weighted graph $G = (V, E)$ with n vertices and m edges, the percolation states x_v for each $v \in V$, a sample schedule $(|S_i|)_{i \ge 1}$ (in the case of the progressive sampling approach) and the quality and confidence parameters $0 < \epsilon, \delta < 1$, assumed to be constants (they do not depend on the size of G).

Theorems 1 and 5 state an upper bound for the VC-Dimension of the range space $\mathcal{R} = (U, \mathcal{F})$ defined in Section 3, respectively, in order to bound the fixed sample size that guarantees $|\tilde{p}(v) - p(v)| \le \epsilon$ for each $v \in V$ with probability at least $1 - \delta$.

Theorem 5. *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 $\text{VCDim}(\mathcal{R}') = k$, where $k \in \mathbb{N}$. Then, there is $S \subseteq D$ such that $|S| = k$ and S is shattered by $\mathcal{F}^{+}$. From Lemmas 1 and 2, 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 $v \notin \text{Int}(p_{uw})$. Considering that $|p_{uw}| - 2 \le \text{Diam}_V(G) - 2$, we have

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

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

Since k must be integer, $k \leq \lfloor \lg \mathrm{Diam}_V(G) - 2 \rfloor + 1 \leq \lg \mathrm{Diam}_V(G) - 2 + 1$. Finally, $\mathrm{PD}(\mathcal{F}) = \mathrm{VCDim}(\mathcal{F}^+) = k \leq \lfloor \lg \mathrm{Diam}_V(G) - 2 \rfloor + 1$. □

By Theorem 4 and Definition 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 1 and 5, a sample of size $\lceil \frac{c}{\epsilon^2} \left(\lfloor \lg \mathrm{Diam}_V(G) - 2 \rfloor + 1 + \ln \delta \right) \rceil$ suffices to our algorithm, for given $0 < \epsilon, \delta < 1$.

If we had used a Hoeffding bound, we would have $\Pr(|\tilde{p}(v) - p(v)| \geq \epsilon) \leq 2\exp(-2r\epsilon^2)$ for a sample of size r and for each $v \in V$. Applying the union bound for all $v \in V$, the value of r must be $2\exp(-2r^2\epsilon^2)n \geq \delta$, which leads to $r \leq \frac{1}{2\epsilon^2}(\ln 2 + \ln n + \ln(1/\delta))$. Even though $\mathrm{Diam}_V(G)$ might be as large as n, we note that the bound given in Theorem 5 is tighter since it depends on the combinatorial structure of G, which gives a sample size tailored for it. For instance, if $\mathrm{Diam}_V(G) = \ln n$ (which is common in many real-world graphs, in particular power-law graphs), we have that $\mathrm{VCDim}(\mathcal{R}) \leq \lfloor \lg(\ln n) - 2 \rfloor + 1$. In particular, 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], however, a bound on $\mathrm{Diam}_V(G)$ is enough and it can be efficiently computed [20].

The main idea of the fixed-size sample algorithm presented in the conference version [9] is shown below.

step 1. Sample a pair of vertices $(u, w) \in V \times V$ uniformly and independently at random;

step 2. Compute the set S_{uw} of shortest paths from u to w;

step 3. Sample a shortest path $p_{uw} \in S_{uw}$ independently with probability $1/\sigma_{uw}$. In this step, start a backward traversing from w as follows. Do $t \leftarrow w$, and while $t \neq u$, sample a predecessor z of t with probability σ_{uz}/σ_{ut}; increase the estimation for the percolation centrality $\tilde{p}(z)$ by $\frac{1}{r}\frac{R(x_u - x_w)}{\sum_{(f,d)\in V\times V : f \neq v \neq d} R(x_d - x_f)}$, and do $t \leftarrow z$;

step 4. Repeat the steps above k times, where $k = \frac{c}{\epsilon^2}\left(\lfloor \lg \mathrm{Diam}_V(G) - 2 \rfloor + 1 + \ln \frac{1}{\delta} \right)$;

step 5. Return the set $\{\tilde{p}(v) = \tilde{p}(v), v \in V \text{ and } \tilde{p}(v) > 0\}$.

4.1. Description of the Progressive Sampling Algorithm

When comparing the fixed-size sample algorithm of [9] with a progressive sampling approach, we can build an algorithm using the range space defined by $\mathcal{R} = (U, \mathcal{F})$ in Section 3 and the bound in Theorem 3.2 of [6] as the stopping condition for sampling. However, the corresponding initial size to the sample schedule obtained by this bound is greater than the value given by Theorems 1 and 5 for a fixed-size sample approach, so we use the range space defined by $\mathcal{R}' = (U', \mathcal{F}')$ in Section 3.2, which has an uniform probability distribution on the domain U'. We note that if there is only one shortest path between any pair of vertices, then we can guarantee $\mathrm{PD}(\mathcal{R}') \leq \lfloor \lg \mathrm{Diam}_V(G) - 2 \rfloor + 1$; otherwise, the Pseudo-dimension of $\mathcal{R}'$ is the same as the one obtained by Hoeffding and union bounds (as proven in Lemma 4.5 of [20]). Despite this issue, we show in the experimental evaluation that in practice, this range space can lead to improvements on the running time of an approximation algorithm that uses a progressive sampling schedule.

The schedule is defined as follows: let S_1 be the initial sample size and $\delta_1 = \delta/2$. At this point, the only information available about the empirical Rademacher average of S_1 is that $\tilde{R}_r(\mathcal{F}', S_1) \geq 0$. Plugging this with the r.h.s. of the bound in Theorem 11, which has to be at most ϵ, we have

$$\frac{2\ln(6/\delta)}{|S_1|} + \sqrt{\frac{\ln(6/\delta)}{2|S_1|}} \leq \epsilon \Rightarrow \frac{4\ln^2(6/\delta)}{|S_1|^2} - \frac{4\ln(6/\delta)\epsilon}{|S_1|} + \epsilon^2 \leq \frac{\ln(6/\delta)}{2|S_1|} \Rightarrow |S_1| \geq \frac{(1 + 8\epsilon + \sqrt{1+16\epsilon})\ln(6/\delta)}{4\epsilon^2}. \quad (1)$$

There is no fixed strategy for scheduling. Provost *et al.* (1999) [18] conjecture that a geometric sampling schedule is optimal (although we do not need such assumption), i.e. the one that $S_i = g^i S_1$, for each $i \geq 1$ and for constant $g > 1$. In our algorithm we follow this results, as previously indicated in [20].

Given $0 < \epsilon, \delta < 1$, let $(|S_i|)_{i\geq 1}$ be a geometric sampling schedule with starting sample size defined in (1). We present the outline of the progressive sampling algorithm for estimating the percolation centrality with probability $1 - \delta$. Consider the table $\tilde{p}$ with the centrality estimation.

The following steps are repeated for each $i \geq 1$. For the sake of clarity, $S_0 = \emptyset$.

step 1. Create a sample of $k = |S_i| - |S_{i-1}|$ elements of $V \times V$ chosen uniformly and independently (with replacement) at random;

step 2. For each pair of vertices $(u, w) \in \{S_i - S_{i-1}\}$, compute the set S_{uw} of shortest paths from u to w. Let z be an internal vertex of some shortest path between u and w. Increase the value $\tilde{p}(z)$ by $\frac{R(x_u - x_w)}{\sum_{(f,d)\in V\times V : f\neq v\neq d} R(x_d - x_f)} \frac{\sigma_{uw}(z)}{\sigma_{uw}}$;

step 3. Compute the bound to $\tilde{R}_r(\mathcal{F}', S_i)$ by minimizing the function defined in Theorem 3. If it satisfies the stopping condition defined in Theorem 2, then return the set $\{\tilde{p}(v) = \tilde{p}(v)/|S_i|, v \in V \text{ and } \tilde{p}(v) > 0\}$. Otherwise, increase the size of S_i until it has size $|S_{i+1}|$, increase i and return to step 1.

Step 1 is trivial. Step 2 requires the computation and update of internal vertices to some shortest path from u to w and the computation of minus_$s[v] = \sum\limits_{(f,d)\in V^2 : f\neq v\neq d} R(x_f - x_d)$ for each $v \in V$. The former task can be computed in time $\mathcal{O}(m + n \log n)$ following the steps of [20]: we run a modified Dijkstra's Algorithm in $G = (V, E)$, for each sampled pair of vertices (u, w). The modification, discussed in Lemma 3 of Brandes (2001) [7], works as follows. Let z be an internal vertex of some shortest path from u to w. The modified Dijkstra stores the distance $d(u, z)$ from u to z in a shortest path from u to w. After S_{uw} is computed, the set of internal vertices in some path $p \in S_{uw}$ is sorted in inverse order of $d(u, z)$. The value $\sigma_{uw}(z)$ corresponds to $\sigma_{uz}\sigma_{zw}$, where σ_{uz} is returned by the modified Dijkstra algorithm, and $\sigma_{zw} = \sum_{y: z\in P_u(y)}$, where $P_u(y)$ is the set of immediate predecessors of y in a shortest path from u to y.

In the calculation of minus_$s[v] = \sum\limits_{(f,d)\in V^2 : f\neq v\neq d} R(x_f - x_d)$ for each $v \in V$, which are necessary to compute $\tilde{p}(v)$, we perform a linear time dynamic programming strategy presented in Algorithm 1. The proof of correctness of this algorithm is stated in Theorem 6.

On Step 3, let $S_i = \{(u_1, w_1, \dots, u_j, w_j)\}$ be the sample of size j obtained after the execution of the i-th iteration of the progressive sampling algorithm and let $\mathbf{v}_v$ be the vector $\mathbf{v}_v = \big(f_v(u_1, w_1), \dots, f_v(u_j, w_j)\big)$, for all $v \in V$. The ℓ_1 and ℓ_2 norms of each $\mathbf{v}_v$ are stored on the hash tables $\mathcal{V}_1$ and $\mathcal{V}_2$, respectively. The set $\mathcal{V}$, represented as a hash table, keeps the values of $\mathcal{V}_2$ with no repetition to the computation of ω_s in line 21, which is the bound for the empirical Rademacher average of S_i obtained by the function defined in Theorem 3. For each internal vertex v to be updated, Algorithm 3 checks if the value associated to $\mathcal{V}_2[v]$ in $\mathcal{V}$ is greater than zero. If yes, the value in $\mathcal{V}[\mathcal{V}_2[v]]$ is increased by one; otherwise, a new key with $\mathcal{V}_2[v]$ is created in $\mathcal{V}$. The value of $\tilde{p}(v)$ corresponds to $\mathcal{V}_1[v]/|S_i|$.

We prove the correctness and running time of Algorithm 2 in Theorems 7 and 8, respectively.

Theorem 6. *For an array A of size n, sorted in non-decreasing order, Algorithm 1 returns for sum and minus_sum*$[k]$*, respectively, the values* $\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} R(A[j] - A[i])$ *and* $\sum\limits_{\substack{i=1\\ i\neq k}}^{n}\sum\limits_{\substack{j=1\\ j\neq k}}^{n} R(A[j] - A[i])$*, for each* $k \in \{1, \dots, n\}$.

Proof. By the definition of sum, we have that

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

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 $k \in \{1, ..., n\}$,

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

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.

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

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

The solutions to the above recurrences are, respectively,

$$\mathrm{svp}[k] = \sum_{i=1}^{k-1} A[i] \quad \text{and} \quad sum_k = \sum_{i=1}^{k} ((i-1)A[i] - \mathrm{svp}[i])\,.$$

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

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

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

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

□

Theorem 7. *Algorithm 2 returns with probability at least $1-\delta$ an approximation $\tilde{p}(v)$ to $p(v)$, for each $v \in V$, such that $\tilde{p}(v)$ is within ϵ error.*

Proof. Let l be the number of iterations of the loop in lines 7–24 of Algorithm 2. Consider the sample $S_l = \{(u_1, w_1), \ldots, (u_r, w_r)\}$ of size r obtained after the last iteration of such loop where the stopping condition is satisfied, where each (u_j, w_j) is a pair in $V \times V$, for $1 \leq j \leq r$. Let η_i be the value obtained in line 23 on the i-th iteration, where $1 \leq i \leq l$, and let ω_s be the optimum value of the function defined in Theorem 3, which is an upper bound to

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
    $\{\text{minus\_sum}[k] = \sum_{\substack{i=1\\ i\neq k}}^{n}\sum_{\substack{j=1\\ j\neq k}}^{n} R(A[j] - A[i]), \forall k \in \{1,...,n\}\}$, such that $R(z) = \max\{z, 0\}$.
1  sum ← 0
2  minus_sum[i] ← $0, \forall i \in \{1,\dots,n\}$
3  svp ← $(0, 0, \dots, 0)$
4  for $i \leftarrow 2$ 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 $i \leftarrow 1$ 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 $0 < \delta < 1$, sample scheduling $(S_i)_{i\geq 1}$.
Result: Approximation $\tilde{p}(v)$ for the percolation centrality of all vertices $v \in V$.
1  $\mathcal{V}, \mathcal{V}_1, \mathcal{V}_2$ ← hash tables
2  $\mathcal{V}_1[v] \leftarrow 0, \mathcal{V}_2[v] \leftarrow 0$, minus_$s[v] \leftarrow 0, \forall v \in V$
3  sort $x$                                   /* after sorted, $x = (x_1, x_2, \dots, x_n)$ */
4  minus_$s$ ← GETPERCOLATIONDIFFERENCES$(x, n)$
5  $|S_0| \leftarrow 0$
6  $i \leftarrow 0$
7  do
8      $i \leftarrow i + 1$
9      for $l \leftarrow 1$ to $|S_i| - |S_{i+1}|$ do
10         sample $u \in V$ with probability $1/n$
11         sample $w \in V$ with probability $1/(n-1)$
12         $S_{uw}$ ← ALLSHORTESTPATHS$(u, w)$
13         if $S_{uw} \neq \emptyset$ then
14             for $z \in P_u(w)$ do
15                 $\sigma_{zw} \leftarrow 1$
16             for each z internal to some shortest path from u to w in inverse order of $d(u, w)$ do
17                 $\sigma_{uw}(z) \leftarrow \sigma_{uz}\sigma_{zw}$
18                 UPDATESETV$\left(\mathcal{V}, z, \mathcal{V}_1, \mathcal{V}_2, \frac{R(x_u - x_w)}{\text{minus\_}s[z]}\frac{\sigma_{uw}(z)}{\sigma_{uw}}\right)$
19                 for $y \in P_u(z)$ do
20                     $\sigma_{yw} \leftarrow \sigma_{yz} + \sigma_{zw}$
21     $\omega_s \leftarrow \min_{s\in\mathbb{R}^+} \frac{1}{s} \ln \sum_{q\in\mathcal{V}} \exp \frac{s^2 q}{2|S_i|^2}$
22     $\delta_i \leftarrow \delta/2^i$
23     $\eta \leftarrow 2\omega_s + \frac{\ln(3/\delta_i)+\sqrt{(\ln(3/\delta_i)+4|S_i|\omega_s)\ln(3/\delta_i)}}{|S_i|} + \sqrt{\frac{\ln(3/\delta_i)}{2|S_i|}}$
24 while $\eta > \epsilon$
25 $\tilde{p}[v] \leftarrow \mathcal{V}_1[v]/|S_i|, \forall v \in V$
26 return $\tilde{p}[v], \forall v \in V$
```

Algorithm 3: UpdateSetV($\mathcal{V}$,z,$\mathcal{V}_1$,$\mathcal{V}_2$,r_z)

```
Data: Table $\mathcal{V}$, vertex $z$, table $\mathcal{V}_1$, table $\mathcal{V}_2$, real value $r_z$.
1  $v \leftarrow \mathcal{V}_2[z]$
2  $v' \leftarrow v + r_z^2$
3  if $v' \notin \mathcal{V}$ then
4      $\mathcal{V}[v'] \leftarrow 1$
5  else
6      $\mathcal{V}[v'] \leftarrow \mathcal{V}[v'] + 1$
7  if $v > 0$ and $\mathcal{V}[v] \geq 1$ then
8      $\mathcal{V}[v] \leftarrow \mathcal{V}[v] - 1$
9  if $v > 0$ and $\mathcal{V}[v] = 0$ then
10     remove $\mathcal{V}[v]$
11 $\mathcal{V}_1[z] \leftarrow \mathcal{V}_1[z] + r_z$
12 $\mathcal{V}_2[z] \leftarrow \mathcal{V}_2[z] + r_z^2$
```

the empirical Rademacher average of the sample S_l and which is computed by a linear-time procedure of [11]. Then $\eta_l = 2\omega_s + \frac{\ln(3/\delta_l)+\sqrt{(\ln(3/\delta_l)+4|S_l|\omega_s)\ln(3/\delta_l)}}{|S_l|} + \sqrt{\frac{\ln(3/\delta_l)}{2|S_l|}}$ is the value such that $\eta_l \leq \epsilon$ for the input graph $G = (V, E)$ and for fixed constants $0 < \epsilon, \delta < 1$.

Let E_i be the event where $\sup_{v\in V} |\tilde{p}(v) - p(v)| > \eta_i$ in iteration i. We need the event E_i occurring with probability at most δ for some iteration i. That is, we need

$$\Pr(\exists i \geq 1 \text{ s.t. } E_i \text{ occurs}) \leq \sum_{i=1}^{\infty} \Pr(E_i) \leq \delta$$

where the inequality comes from union bound. Setting $\Pr(E_i) = \delta/2^i$, we have

$$\sum_{i=1}^{\infty} \Pr(E_i) = \delta \sum_{i=1}^{\infty} \frac{1}{2^i} = \delta.$$

For each iteration i in 7–24, the pair (u_j, w_j) is sampled with probability $\frac{1}{n(n-1)}$ in lines 10 and 11, for $1 \leq j \leq r$, and the set $S_{u_j w_j}$ is computed by Dijkstra algorithm (line 12).

The value of $\frac{R(x_{u_j} - x_{w_j})}{\text{minus\_}s[z]} \frac{\sigma_{u_j w_j}(z)}{\sigma_{u_j w_j}}$, for each internal vertex z of a shortest path $p \in S_{u_j w_j}$ found on the backtracking procedure (lines 16–20) is added to $\mathcal{V}_1[z]$ by Algorithm 3 in line 18. The correctness of the procedure in lines 19–20 can be checked in Lemma 3 of [7].

The value of minus_$s[z]$ is correctly computed in line 4 as shown in Theorem 6. Then, at the end of Algorithm 2,

$$\tilde{p}(z) = \frac{1}{r} \sum_{p_{uw} \in S_l} \frac{R(x_u - x_w)}{\sum\limits_{\substack{(f,d)\in V\times V \\ f\neq z\neq d}} R(x_f - x_d)} \frac{\sigma_{uw}(z)}{\sigma_{uw}}$$

which corresponds to $\tilde{p}(z) = \frac{1}{r} \sum\limits_{p_{uw}\in S_l} f_z(p_{uw})$.

Since $\eta_l \leq \epsilon$, $L_{S_l}(f_v) = \tilde{p}(v)$ and $L_{U'}(f_v) = p(v)$ (Theorem 4) for all $v \in V$ and $f_v \in \mathcal{F}$, then $\Pr(|\tilde{p}(v) - p(v)| \leq \epsilon) \geq 1 - \delta$ (Theorem 2). ☐

Theorem 8. *Given a 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 expected running time $\mathcal{O}(m \log^2 n)$.*

Proof. We sample the vertices u and w in lines 10 and 11, respectively, in linear time.

Sorting the percolation states array x (line 3) can be done in $\mathcal{O}(n \log n)$ time and the execution of Algorithm 1 on the sorted array x (line 4) has running time $\mathcal{O}(n)$. Before the loop in lines 16–20 start, the vertices in G are sorted according to $d(u, w)$ in reverse order, which takes $\mathcal{O}(n \log n)$. The complexity analysis of the procedure in 16–20 proceeds as follows. Once $|P_u(z)| \leq d_G(z)$, where $d_G(z)$ denotes the degree of z in G and $P_u(z)$ is the set of predecessors of z in the shortest paths from u to w, and since this loop is executed at most n times if all the vertices of G are internal to some shortest path between u and w, the total running time of these steps corresponds to $\sum_{v \in V} d_G(v) = 2m = \mathcal{O}(m)$.

The execution of Algorithm 3 in line 18 has $\mathcal{O}(1)$ expected running time, since the sets $\mathcal{V}$, $\mathcal{V}_1$ and $\mathcal{V}_2$ are stored as hash tables and operations of insertion, deletion and search on these structures take $\mathcal{O}(1)$ time in average. Line 21 is executed by an algorithm that is linear in the size of the sample [11]. The loop in lines 7–24 runs at most r times, since $|\tilde{p}(v) - p(v)| \leq \epsilon$ for all $v \in V$ with probability $1 - \delta$ when the sample has size r (Theorem 1). The Dijkstra algorithm which is executed in line 12 has running time $\mathcal{O}(m \log n)$, so the total expected running time of Algorithm 2 is $\mathcal{O}(n \log n + r \max(m, m \log n)) = \mathcal{O}(n \log n + r(m \log n)) = \mathcal{O}(r(m \log n)) = \mathcal{O}(m \log^2 n)$. □

Corollary 1. *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 expected running time $\mathcal{O}((m + n) \log n)$.*

Proof. The proof is analogous to the one of Theorem 8, with the difference that the shortest paths between a sampled pair $(u, w) \in V \times V$ will be computed by the breadth-first search (BFS) algorithm, which has running time $\mathcal{O}(m + n)$. □

We observe that, even though it is an open problem whether there is a $\mathcal{O}(n^{3-c})$ algorithm for computing all shortest paths in weighted graphs, in the unweighted case there is a $\mathcal{O}(n^{2.38})$ (non-combinatorial) algorithm for this problem [22]. However, even if this algorithm could be adapted to compute betweenness/percolation centrality (what is not clear), our algorithm obtained in Corollary 1 is still faster.

5. Experimental Evaluation

We perform our experimental evaluation on publicly available real-world graph datasets from Stanford Large Network Dataset Collection [12] and Network Repository [21], described in Table 1. These graphs spam from social, peer-to-peer and citations networks. We compare the running time and the accuracy of our progressive sampling algorithm with the fixed-size sample approach. Regarding to the exact algorithm, we compare our results with the best know algorithm for computing the same measure in the exact case, since our approach is the first estimation algorithm for percolation centrality, as far as we know. A main advantage of our algorithms is that both of them output estimations with a very small error. In fact, for all networks used in our experiments, the average estimation error are kept below the quality parameter by orders of magnitude.

Graph	Type	|V|	|E|	Approximated $\text{Diam}_V(G)$	Exact Algorithm Running Time (in secs.)
p2p-Gnutella04	Directed	10876	39994	39	257.8975
Cit-HepPh	Directed	34546	421578	42	4455.429
p2p-Gnutella31	Directed	62586	147892	42	9692.5041
socfb-Berkeley13	Undirected	22900	852419	33	15271.9312

Table 1
Dataset details for the real-world graphs

Our implementation uses Python 3.7 language, the NetworkX library for graph manipulation, and the NLOpt library for the computation of the minimization function in Theorem 3. The NetworkX library provides an exact algorithm for the percolation centrality which we use to compare with our approximation algorithms. The experiments were performed on a 2.6 MHz Intel Xeon E5-2650v2 octa core with 48GB of RAM and Ubuntu 14.04 64-bit operating system.

In all experiments in graphs of Table 1, 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 an integer random number between 1 and 100. We set the parameters $\delta = 0.1$ and $c = 0.5$ (as suggested by [13]). These parameters remain fixed. We set the accuracy parameter as $\epsilon = \{0.1, 0.08, 0.06, 0.04, 0.02\}$ and the constant for the geometric sampling schedule $g = \{1.2, 1.5, 2.0\}$. We perform five experiments in the combination of parameters ϵ and g. We report the results of the running time and the size of the sample attained by the progressive sampling schedule in Tables 2, 3, 4 and 5, where each table is associated to one of the real-world graphs described in Table 1.

ϵ	Schedule constant	Sample Size (Initial Sample Final Sample)	$\epsilon - \eta$	Final η	Progressive Sampling Running Time (in secs.)	Fixed Sample Running Time (in secs.)	Fixed Sample Size
0.1	1.2	350	-0.00001166	0.098426091	11.76664	10.62395	416
		420	0.001573909				
	1.5	350	-0.000012202	0.0858796514	13.38731		
		524	0.0141203486				
	2	350	-0.00000943	0.0722524692	17.53773		
		699	0.027747531				
0.08	**1.2**	504	-0.000009637	0.078772866	**15.04645**	16.61353	649
		605	0.001227134				
	1.5	504	-0.00000974	0.068962175	21.94075		
		756	0.011037825				
	2	504	-0.000009834	0.058254291	25.03958		
		1008	0.021745709				
0.06	**1.2**	819	-0.000009936	0.059110204	**27.16701**	29.54515	1154
		983	0.000889796				
	1.5	819	-0.000009997	0.051950778	31.63912		
		1229	0.008049222				
	2	819	-0.000010057	0.044087666	43.98850		
		1628	0.015912334				
0.04	**1.2**	1664	-0.000010195	0.039432553	**53.29527**	65.21671	2595
		1997	0.000567447				
	1.5	1664	-0.000010281	0.034817851	**61.23428**		
		2496	0.00518215				
	2	1664	-0.00001037	0.029709455	82.76411		
		3328	0.010290545				
0.02	**1.2**	5909	-0.000010685	0.019735529	**176.70750**	257.03656	10379
		7091	0.000264471				
	1.5	5909	-0.000010748	0.017525662	**219.01537**		
		8863	0.002474338				
	2	5909	-0.000010807	0.015053813	300.53190		
		11817	0.004946187				

Table 2
p2p-Gnutella04 Graph

The highlighted entries in the tables are the cases where the progressive sampling schedule achieved smaller samples in comparison with the fixed-size sample approach. In all graphs and all values of ϵ, the improvement on the sample size where obtained by the smallest geometric schedule constant, i.e. $g = 1.2$. For $\epsilon = 0.02$ and $\epsilon = 0.04$, the progressive sampling algorithm also acquired smaller samples for $g = 1.5$, except for the socfb-Berkeley13, which is the most dense graph among the dataset. The main reason that the fixed-size sample algorithm outperforms the progressive sampling one when $g = 2.0$ is that much more pairs of vertices are sampled than necessary, leading to a value of η that is much smaller than the desired ϵ.

In both approaches, the error of the estimation is within ϵ for all vertices 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 all graphs used in the experiments the maximum error among the error for the estimation of each vertex is around 10^{-10} and the average error among all vertices is around 10^{-11}, even when we set the quality guarantee to $\epsilon = 0.1$. We observe that the differences between the error obtained in the progressive sampling approach and the fixed-size sample approach are within 10^{-14}. These results are shown in Figures 1 and 2.

ϵ	Schedule constant	Sample Size (Initial Sample Final Sample)	$\epsilon - \eta$	Final η	Progressive Sampling Running Time (in secs.)	Fixed Sample Running Time (in secs.)	Fixed Sample Size
	1.2	350 420	-0.000011661 0.001573908	0.098426093	**77.43216**		
0.1	1.5	350 524	-0.000012194 0.014123353	0.0858766471	91.91335	80.69897	416
	2	350 699	-0.000009476 0.027747498	0.072252502	125.93146		
	1.2	504 **605**	-0.000009677 0.001227094	0.078772906	**111.36629**		
0.08	1.5	504 756	-0.000009769 0.011037804	0.068962197	159.96246	129.91781	649
	2	504 1008	-0.000009847 0.021745687	0.058254314	188.53688		
	1.2	819 **983**	-0.000009956 0.000889777	0.059110223	**183.27423**		
0.06	1.5	819 1229	-0.000010018 0.0080492	0.0519508002	245.29033	222.75616	1154
	2	819 1628	-0.000010088 0.015912295	0.044087705	302.89113		
	1.2	1664 **1997**	-0.000010235 0.000567401	0.039432599	**360.51697**		
0.04	**1.5**	1664 **2496**	-0.000010337 0.005182085	0.034817915	**486.70175**	493.08546	2595
	2	1664 3328	-0.000010431 0.010290473	0.029709528	616.52572		
	1.2	5909 **7091**	-0.000010808 0.000264345	0.019735656	**1327.98216**		
0.02	**1.5**	5909 **8863**	-0.000010883 0.002474196	0.017525804	**1734.40681**	2029.62927	10379
	2	5909 11817	-0.000010954 0.004946029	0.015053971	2192.27154		

Table 3
Cit-HepPh graph

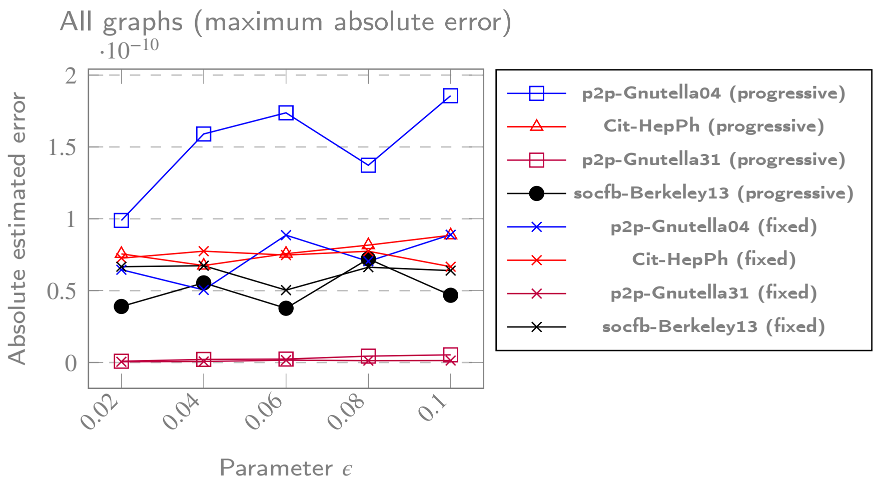

Figure 1: Percolation centrality absolute maximum error estimation for $g = 1.2$.

ϵ	Schedule constant	Sample Size (Initial Sample Final Sample)	ϵ - η	Final η	Progressive Sampling Running Time (in secs.)	Fixed Sample Running Time (in secs.)	Fixed Sample Size
0.1	1.2	350 420	-0.000011326 0.001574200	0.098425800	**35.22300**	40.36586	416
	1.5	350 524	-0.000012007 0.014120532	0.085879468	44.06515		
	2	350 699	-0.000012293 0.027747634	0.072252366	55.88123		
0.08	**1.2**	504 **605**	-0.000009545 0.001227216	0.078772784	**54.85581**	55.63046	649
	1.5	504 756	-0.000009653 0.011037899	0.068962101	63.78716		
	2	504 1008	-0.000009766 0.021745775	0.058254225	86.09895		
0.06	**1.2**	819 **983**	-0.000009873 0.000889857	0.059110143	**2.28515**	101.97880	1154
	1.5	819 1229	-0.000009939 0.008049265	0.051950735	100.22565		
	2	819 1628	-0.000010012 0.015912386	0.044087614	141.78238		
0.04	**1.2**	1664 **1997**	-0.000010155 0.000567474	0.039432526	**182.54564**	225.51281	2595
	1.5	1664 **2496**	-0.000010273 0.005182146	0.034817854	**209.05493**		
	2	1664 3328	-0.000010379 0.010290510	0.029709490	288.22010		
0.02	**1.2**	5909 **7091**	-0.000010795 0.000264357	0.019735643	**614.76785**	903.45608	10379
	1.5	5909 **8863**	-0.000010875 0.002474201	0.017525799	**772.12881**		
	2	5909 11817	-0.000010956 0.004946023	0.015053977	1037.35445		

Table 4
p2p-Gnutella31 Graph

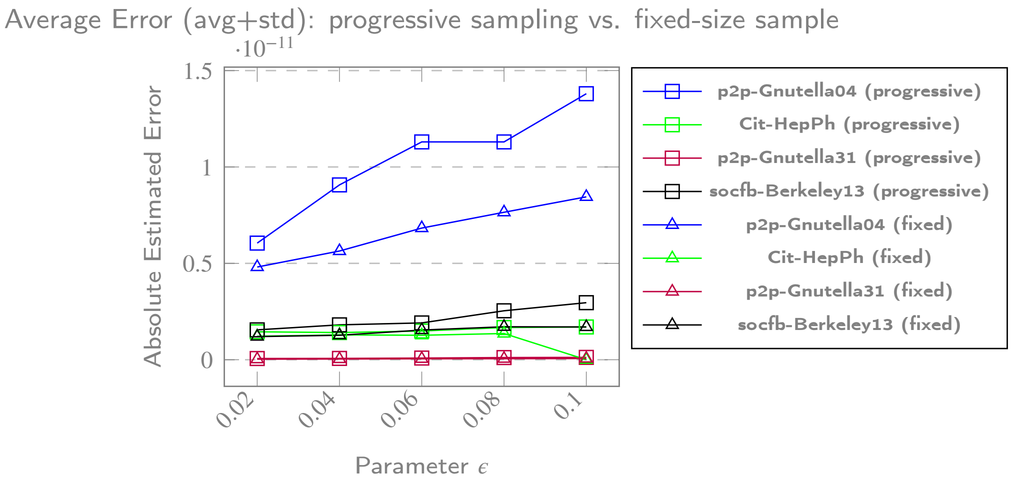

Figure 2: Percolation centrality average error estimation for $g = 1.2$

ϵ	Schedule constant	Sample Size (Initial Sample Final Sample)	$\epsilon - \eta$	Final η	Progressive Sampling Running Time (in secs.)	Fixed Sample Running Time (in secs.)	Fixed Sample Size
	1.2	350 420	-0.000012282 0.001576370	0.098423630	**790.94315**		
0.1	1.5	350 524	-0.000009636 0.014122944	0.085877056	979.45032	695.12026	366
	2	350 699	-0.000009850 0.027747135	0.072252865	1315.06880		
	1.2	504 **605**	-0.000010015 0.001226759	0.078773241	**1150.53222**		
0.08	1.5	504 756	-0.000010129 0.011037425	0.068962575	1429.07482	1064.03995	571
	2	504 1008	-0.000010262 0.021745247	0.058254753	1854.15828		
	1.2	819 **983**	-0.000010430 0.000889299	0.059110701	**1832.42449**		
0.06	1.5	819 1229	-0.000010541 0.008048669	0.051951331	2279.52619	1876.60577	1015
	2	819 1628	-0.000010637 0.015911753	0.044088247	3063.31744		
	1.2	1664 **1997**	-0.000010767 0.000566874	0.039433126	**3668.25732**		
0.04	1.5	1664 2496	-0.000010851 0.005181580	0.034818420	4573.80160	4222.63659	2283
	2	1664 3328	-0.000010931 0.010289983	0.029710017	6170.42000		
	1.2	5909 **7091**	-0.000011230 0.000263926	0.019736074	**13529.57006**		
0.02	1.5	5909 8863	-0.000011290 0.002473796	0.017526204	16329.16631	16781.54561	9129
	2	5909 11817	-0.000011347 0.004945649	0.015054351	21674.77987		

Table 5
socfb-Berkeley13 Graph

5.1. Scalability

We 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 a sequence of synthetic graphs increasing in size and compared the execution time of our progressive sampling algorithm with the exact algorithm provided by NetworkX library and with the fixed-size sample algorithm.

We also use the NetworkX library for generating random power-law graphs by the Barabási-Albert model with each vertex creating two edges, obtaining undirected unweighted power-law graphs with average degree of 2. We set the percolation states of each vertex as a random number between 0 and 1.

In the experiments we use graphs with the number of vertices n in $\{2000, 4000, 8000, 16000, 32000, 64000\}$. The values of ϵ and g are fixed at 0.03 and 1.2, respectively. The results are shown in Figure 3.

6. Conclusion

We presented a sampling-based algorithm to accurately estimate the percolation centrality of all vertices of a weighted graph with high probability. The proposed algorithm has expected running time $\mathcal{O}(m \log^2 n)$, and the estimation is within ϵ of the exact value with probability $1 - \delta$, for *fixed* constants $0 < \epsilon, \delta < 1$. The running time of the algorithm is reduced to $\mathcal{O}((m + n) \log n)$ if the input graph is unweighted. Since many large scale graphs, in practical applications, are sparse and have small vertex-diameter (typically of size $\log n$), our algorithm provides a fast

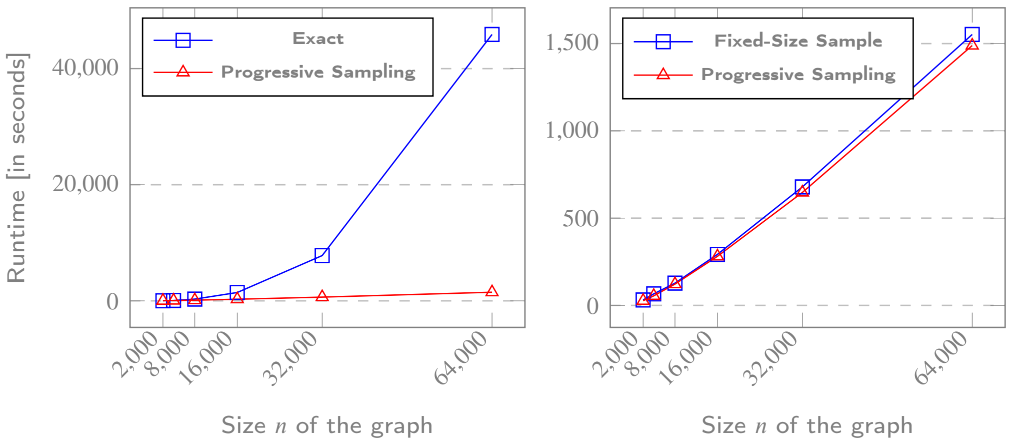

Figure 3: Scalability experiments, with $\epsilon = 0.03$ and $g = 1.2$.

approximation for such graphs (more precisely running in $\mathcal{O}(n \log n \log \log n)$ time).

Our results indicate that the proposed approach is practical in real-world graphs, as validated by our experimental evaluation. The returned estimation errors are many orders of magnitude smaller than the theoretical worst case guarantee for graphs of a variety of sizes. As expected, the fixed-size sample and the progressive sampling algorithms are much faster than the exact algorithm. The progressive sampling approach is around 14 times faster for the largest real-world graph and 42 times faster for the largest synthetic graph. Furthermore, the progressive sampling algorithm returned good quality estimations using a smaller sample set in comparison to the one obtained by the fixed-size sample algorithm, leading to improvements on the running time.

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] Stéphane Boucheron, Olivier Bousquet, and Gábor Lugosi. 2005. Theory of classification: A survey of some recent advances. *ESAIM: probability and statistics* 9 (2005), 323–375.

[7] U. Brandes. 2001. A faster algorithm for betweenness centrality. *Journal of Mathematical Sociology* 25, 163 (2001), 163–177.

[8] 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.

[9] 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 International Conference on Knowledge Discovery and Data Mining (KDD '20)*. Association for Computing Machinery, New York, NY, USA, 1839–1847. https://doi.org/10.1145/3394486.3403235

[10] Sariel Har-Peled and Micha Sharir. 2011. Relative (p, ϵ)-approximations in geometry. *Discrete & Computational Geometry* 45, 3 (2011), 462–496.

[11] Steven G. Johnson. 2014. The NLopt nonlinear-optimization package. http://github.com/stevengj/nlopt

[12] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[13] 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.

[14] M. Mitzenmacher and E. Upfal. 2017. *Probability and Computing: Randomization and Probabilistic Techniques in Algorithms and Data Analysis* (2nd ed.). Cambridge University Press.

[15] M. Mohri, A. Rostamizadeh, and A. Talwalkar. 2012. *Foundations of Machine Learning*. The MIT Press.

[16] Luca Oneto, Alessandro Ghio, Davide Anguita, and Sandro Ridella. 2013. An improved analysis of the Rademacher data-dependent bound using its self bounding property. *Neural Networks* 44 (2013), 107 – 111. https://doi.org/10.1016/j.neunet.2013.03.017

[17] 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.

[18] Foster Provost, David Jensen, and Tim Oates. 1999. Efficient Progressive Sampling. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '99)*. Association for Computing Machinery, New York, NY, USA, 23–32. https://doi.org/10.1145/312129.312188

[19] M. Riondato and E. M. Kornaropoulos. 2016. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery* 30, 2 (2016), 438–475.

[20] 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.

[21] Ryan A. Rossi and Nesreen K. Ahmed. 2015. The Network Data Repository with Interactive Graph Analytics and Visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*. http://networkrepository.com

[22] R. Seidel. 1995. On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. *J. Comput. System Sci.* 51, 3 (1995), 400 – 403.

[23] S. Shalev-Shwartz and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.

[24] Ryan Williams. 2014. Faster All-pairs Shortest Paths via Circuit Complexity. In *Proceedings of the Forty-sixth Annual ACM Symposium on Theory of Computing (STOC '14)*. ACM, New York, NY, USA, 664–673. https://doi.org/10.1145/2591796.2591811