

Percolation Centrality via Rademacher Complexity

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

^aDepartment 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 every vertex in an undirected weighted graph under the light of sample complexity theory. 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 $\text{Diam}_V(G)$ be the maximum number of vertices in a shortest path in G . In this paper we present a $\mathcal{O}(m \log n \text{Diam}_V(G))$ time randomized approximation algorithm for the percolation centrality for every vertex of G . The estimation obtained is within ϵ of the exact value with probability $1 - \delta$, for fixed constants $0 < \epsilon, \delta < 1$. The main idea is a progressive sampling strategy that iteratively builds a sample of shortest paths in G and computes an estimation for the Rademacher Complexity of this sample. In this analysis, we observe that since Rademacher Complexity of a sample is sensitive to the input distribution, the performance of our algorithm also takes advantage of the input distribution (if we assume that G is chosen from a prescribed random distribution). Furthermore, even if we have assumption about such distribution, we show that in the worst case our algorithm performs as well as the best previous algorithm for the same task of estimation of percolation centrality.


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) [6] in the context of the passage of a fluid in a medium. In graphs, percolation centrality was proposed by Piraveenan *et al* (2013) [15], 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 every vertex of a graph depends on computing all its shortest paths [17] and, consequently, the same applies to the computation of percolation centrality. The fastest algorithm for this task for weighted graphs, proposed by Williams (2014) [22], runs in $\mathcal{O}\left(n^3/2^c\sqrt{\log n}\right)$ time, for some constant c [23]. 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 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 present paper follows on steps of Lima *et al.* (2020) [7] and Riondato and Upfal (2018) [19] in field of sample complexity theory applied to estimation problems in graphs. A main theme in both of these previous works is the fact

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

 <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)

that for large scale graphs, even algorithms with time complexity that scales quadratically 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 obtain values within ϵ from the exact value with probability $1 - \delta$ [17].

In this paper we show that the techniques presented in Lima et al. [7] on Pseudo-dimension theory applied to percolation centrality and the work of Riondato and Upfal on Rademacher Averages applied to betweenness centrality can be combined and further developed for giving a randomized algorithm for the approximation of the percolation centrality based on the idea of progressive sampling. More precisely, we show that for any fixed constants $0 < \epsilon, \delta < 1$, the estimation of the percolation centrality can be computed in $\mathcal{O}(m \log n \text{Diam}_V(G))$ time, where $\text{Diam}_V(G)$ 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 $\mathcal{O}(n \log n \log \log n)$.

The main contribution presented in this paper is a progressive sampling algorithm relying on the use of Rademacher Complexity. 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 our 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 use a result from [7] based on Vapnik-Chervonenkis (VC) theory to show that the sample size of our algorithm is always tighter than the ones given by standard Hoeffding and union-bound techniques and never worse than the sample size given the previous algorithm given in [7] for the estimation of percolation centrality.

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 G is connected, since the algorithms can be applied separately to each of its connected components. Since our work build on some of the techniques developed in [7], in Sections 2.1 and 2.2 we maintain similar notation and definitions that appeared in [7].

2.1. Graphs and Percolation Centrality

Given an undirected weighted graph $G = (V, E)$, 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 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 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

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

The definition originally presented in [15] 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 [17].

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 [21] and Mitzenmacher and Upfal [11].

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 $VCDim(\mathcal{R}) = \max\{k : \exists S \subseteq U \text{ such that } |S| = k \text{ and } |\mathcal{I}_S| = 2^k\}$.

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 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 the context where for a given a domain U and a set of values of interest \mathcal{H} , there is 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 from U sampled 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., $L_U(f_h) = \mathbb{E}_{u \in U}[f_h(u)]$ 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 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 $0 < \epsilon, \delta < 1$.

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

Theorem 1 (see [9], Section 1). *Let $\mathcal{R}' = (D, \mathcal{F}^+)$ be a range space ($D = U \times [0, 1]$) with $VC\text{Dim}(\mathcal{R}') \leq d$ 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 1 and 2, stated and proved by Riondato and Upfal (2018), present constraints on the sets that can be shattered by a range set \mathcal{F}^+ .

Lemma 1 (see [19], 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 a $y \in [0, 1]$.*

Lemma 2 (see [19], 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 to 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 [16]. 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 [19]. A key idea that lies in the core of statistical learning theory – although having applications that extend the context of learning algorithms [18] – is that the stopping condition takes into consideration the input distribution, which can be extracted by the use of Rademacher Complexity (see [11], chapter 14). The main results from this theory that are applied 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 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. However, the exact computation of this measure is expensive, so we use the bound of Boucheron et al. [5] to relate the empirical Rademacher average with the value of $\sup_{f_h \in \mathcal{F}} |L_S(f_h) - L_U(f_h)|$. We are aware that more refined bounds than the one used in this work are available, as the ones presented by Oneto et al. [13, 14] and Riondato and Upfal [19], if the sample is created under a uniform distribution. We note that such adaptation is possible in the current work, since we could modify our sample space from shortest paths to pair of vertices (note that sampling pair of vertices uniformly can be easily done), but that would lead to a looser VC-Dimension bound.

Theorem 2. (see [5], Theorem 3.2) *With probability at least $1 - \delta$,*

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

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, as we have mentioned previously, expensive over a large (or infinite) set of functions [11]. Even a Monte Carlo simulation to estimating $\tilde{R}_r(\mathcal{F}, S)$ is expensive to be extracted in this case; hence, we use the bound given by Theorem 3, which is a variant of the Massart's Lemma (see Theorem 14.22, [11]) that is convex, continuous in \mathbb{R}^+ and can be efficiently minimized by standard convex optimization methods.

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

Theorem 3. (Riondato and Upfal [19]) Let $w : \mathbb{R}^+ \rightarrow \mathbb{R}^+$ be the function

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

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

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 $f_v : U \rightarrow [0, 1]$ be the function $f_v(p_{uw}) = \frac{R(x_u - x_w)}{\sum_{(f,d) \in V^2: f \neq v \neq d} R(x_f - x_d)} \mathbb{1}_{\tau_v}(p_{uw})$.

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

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

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

In [7], Lima et al. showed that $\mathbb{E}[f_v(p_{uw})] = p(v)$ for all $v \in V$. We state this result in the next theorem.

Theorem 4. (Lima et al. [7]) 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 $\pi(p_{uw})$, $\mathbb{E}[f_v(p_{uw})] = p(v)$.

Let $S = \{(p_{u_i w_i}, 1 \leq i \leq r)\}$ 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 6:

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

For each $v \in V$, the value $\tilde{p}(v)$ can be defined as $\|\mathbf{v}_v\|_1 / r$, where

$$\mathbf{v}_v = (f_v(p_{u_1 w_1}), \dots, f_v(p_{u_r w_r})).$$

We denote the set $\mathcal{V} = \{\mathbf{v}_v, v \in V\}$. Note that $|\mathcal{V}| \leq |V|$, since there may be different vertices u' and v' with $\mathbf{v}_{u'} = \mathbf{v}_{v'}$.

4. Estimation for the percolation centrality

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

Theorem 5 states an upper bound to the VC-Dimension of the range space \mathcal{R} defined in Section 3 in order to bound the fixed sample size that guarantees $|\tilde{p}(v) - p(v)| \geq \epsilon$ for each $v \in V$ with probability at least $1 - \delta$. In remainder of this paper let $\text{Diam}_V(G)$ be the vertex-diameter of G , i.e., the maximum number of vertices in a shortest path of G .

Theorem 5. (Lima et al. [7]) *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 $\text{Diam}_V(G)$ be the vertex-diameter of G . We have $PD(\mathcal{R}) = VCDim(\mathcal{R}') \leq \lceil \lg \text{Diam}_V(G) - 2 \rceil + 1$.*

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 $\lceil \frac{\epsilon}{\epsilon^2} (\lceil \lg \text{Diam}_V(G) - 2 \rceil + 1 + \ln \delta) \rceil$ 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 $2\exp(-2r^2)n \geq \delta$, which leads to $r \geq \frac{1}{2\epsilon^2}(\ln 2 + \ln n + \ln(1/\delta))$. Even though $\text{Diam}_V(G)$ 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 $\text{Diam}_V(G) = \ln n$ (which is common in many real-world graphs, in particular power-law graphs), we have that $VCDim(\mathcal{R}) \leq \lceil \lg(\ln n) - 2 \rceil + 1$. 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 $\text{Diam}_V(G)$ is enough and it can be efficiently computed [19].

The main idea of the algorithm presented in [7] is that an upper bound for $\lceil \lg(\ln n) - 2 \rceil + 1$ can efficiently be computed. In the current paper, instead of running the sampling algorithm directly in a sample such fixed size, we build a sampling schedule that is used in a progressive sampling approach. This 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 complexity 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

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

There is no fixed strategy for scheduling. Provost et al. [16] conjecture that a geometric sampling schedule is optimal (although we do not need such assumption), i.e., the one that $S_i = c^i S_1$, for each $i \geq 1$ and for a constant $c > 1$. In our algorithm we follow this results, as previously indicated by Riondato and Upfal [19].

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 algorithm for estimating the percolation centrality with probability $1 - \delta$. Consider the table \tilde{p} with the centrality estimation and the set \mathcal{V} , that stores the values of \mathbf{v}_v , for each $v \in V$, without repetition.

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^2 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 . Sample a shortest path p_{uw} from S_{uw} with probability $1/\sigma_{uw}$. For each vertex $z \in \text{Int}(p_{uw})$, increase $\tilde{p}(z)$ by $\frac{R(x_u - x_w)}{\sum_{(f,d) \in V^2: f \neq v \neq d} R(x_d - x_f)}$;

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 and step 2 can be performed by running Dijkstra's Algorithm in time $\mathcal{O}(m + n \log n)$ in the input graph G for each sampled pair of vertices (u, w) .

The value $minus_s[v] = \sum_{(f,d) \in V^2: f \neq v \neq d} R(x_f - x_d)$ for each $v \in V$, which are necessary to compute $\bar{p}(v)$, is obtained in line 4 by the linear time dynamic programming strategy presented in Algorithm 1. The proof of correctness of this algorithm appeared originally in [7]. For the sake of completeness we show the proof in Theorem 6.

The storage of each value in \mathcal{V} in a sparse way and without repetition and the updating of the map $\mathbf{v}[v]$, for each $v \in V$, are computed by Algorithm 3 in line 19, following the steps of Riondato and Upfal [19].

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 $\sum_{i=1}^n \sum_{j=1}^n R(A[j] - A[i])$ and $\sum_{i=1}^n \sum_{\substack{j=1 \\ i \neq k, j \neq k}}^n R(A[j] - A[i])$, for each $k \in \{1, \dots, n\}$.*

Proof. By the definition of sum , we have that

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

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

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

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.

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

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

The solutions to the above recurrences are, respectively,

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

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

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

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

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

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

□

Algorithm 1: GETPERCOLATIONDIFFERENCES(A, n)

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

Result: The value $\text{sum} = \sum_{i=1}^n \sum_{j=1}^n R(A[j] - A[i])$ and the array

$$\left\{ \text{minus_sum}[k] = \sum_{\substack{i=1 \\ i \neq k}}^n \sum_{\substack{j=1 \\ j \neq k}}^n R(A[j] - A[i]), \forall k \in \{1, \dots, n\} \right\}, \text{ such that } R(z) = \max\{z, 0\}.$$

```

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
    
```

Theorem 7. Consider a sample $S_r = \{p_{u_1 w_1}, \dots, p_{u_r w_r}\}$ of size r , where each $p_{u_i w_i}$ is a shortest path in S_G between the vertices u_i and w_i , for $1 \leq i \leq r$, and let η_i be the value obtained in line 23 on the i -th iteration. Then $\eta_r = 2w_s + \sqrt{\frac{2 \ln(2/\delta_r)}{|S_r|}}$, where $\delta_r = \delta/2^r$, is the value where r is the minimal $i \geq 1$ such that $\eta_r \leq \epsilon$ for the input graph $G = (V, E)$ and for fixed constants $0 < \epsilon, \delta < 1$. 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.

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  $\mathbf{v}[v] \leftarrow \emptyset, \text{minus}_s[v] \leftarrow 0, \forall v \in V$ 
2  $\text{count} \leftarrow$  array of size  $n$ , initialized with zeros
3  $\text{sort } x$  /* after sorted,  $x = (x_1, x_2, \dots, x_n)$  */
4  $\text{minus}_s \leftarrow \text{GETPERCOLATIONDIFFERENCES}(x, n)$ 
5  $|S_0| \leftarrow 0$ 
6  $\mathcal{V} \leftarrow \emptyset$ 
7  $i \leftarrow 0$ 
8 do
9    $i \leftarrow i + 1$ 
10  for  $l \leftarrow 1$  to  $|S_i| - |S_{i+1}|$  do
11    sample  $u \in V$  with probability  $1/n$ 
12    sample  $w \in V$  with probability  $1/(n-1)$ 
13     $S_{uw} \leftarrow \text{ALLSHORTESTPATHS}(u, w)$ 
14    if  $S_{uw} \neq \emptyset$  then
15       $t \leftarrow w$ 
16      while  $t \neq u$  do
17        sample  $z \in P_u(t)$  with probability  $\frac{\sigma_{uz}}{\sigma_{ut}}$ 
18        if  $z \neq u$  then
19           $\text{UPDATESETV}(\mathcal{V}, z, \mathbf{v}, \frac{R(x_u - x_w)}{\text{minus}_s[z]}, \text{count})$ 
20           $t \leftarrow z$ 
21     $w_s \leftarrow \min_{s \in \mathbb{R}^+} \frac{1}{s} \ln \sum_{v \in \mathcal{V}} \exp \frac{s^2 \|\mathbf{v}\|_2^2}{2|S_i|^2}$ 
22     $\delta_i \leftarrow \delta/2^i$ 
23     $\eta \leftarrow 2w_s + \sqrt{\frac{2 \ln(2/\delta_i)}{|S_i|}}$ 
24  while  $\eta > \epsilon$ 
25   $\tilde{p}[v] \leftarrow \|\mathbf{v}[v]\| / |S_i|, \forall v \in V$ 
26 return  $\tilde{p}[v], \forall v \in V$ 

```

Algorithm 3: UPDATESETV($\mathcal{V}, z, \mathbf{v}, \text{FRAC}, \text{COUNT}$)

Data: Set \mathcal{V} , vertex z , vector \mathbf{v} , real value r_z , counter array count .

```

1  $\mathbf{v}' \leftarrow \mathbf{v}[z]$ 
2  $\mathbf{v} \leftarrow \{\mathbf{v}[z] \cup r_z\}$ 
3 if  $v \notin \mathcal{V}$  then
4    $\text{count}[\mathbf{v}] \leftarrow 1$ 
5    $\mathcal{V} \leftarrow \mathcal{V} \cup \{\mathbf{v}\}$ 
6 else
7    $\text{count}[\mathbf{v}] \leftarrow \text{count}[\mathbf{v}] + 1$ 
8 if  $\text{count}[\mathbf{v}'] \geq 1$  then
9    $\text{count}[\mathbf{v}'] \leftarrow \text{count}[\mathbf{v}'] - 1$ 
10 if  $\text{count}[\mathbf{v}'] = 0$  then
11    $\mathcal{V} \leftarrow \mathcal{V} \setminus \{\mathbf{v}'\}$ 
12  $\mathbf{v}[z] \leftarrow \mathbf{v}[z] \cup r_z$ 

```

Proof. Let $i \geq 1$ be an iteration of the loop in 8–24 and let E_i be the event where $\sup_{v \in V} |\tilde{p}(v) - p(v)| > \eta_i$ in this iteration. 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

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

Let $S_r = \{p_{u_1 w_1}, \dots, p_{u_r w_r}\}$ be the final sample obtained after the iteration r in the loop 8–24 where the stopping condition is satisfied, i.e., $\eta_r \leq \epsilon$. For each iteration i in 8–24, where $1 \leq i \leq r$, each pair (u_i, w_i) is sampled with probability $\frac{1}{n(n-1)}$ in lines 11 and 12, and for each pair, the set $S_{u_i w_i}$ is computed by Dijkstra algorithm (line 13). A shortest path $p_{u_i w_i}$ is sampled independently in $S_{u_i w_i}$ (lines 16–20), i.e., with probability $\frac{1}{\sigma_{u_i w_i}}$, by a backward traversing starting from w_i (Lemma 5 in [17], Section 5.1). Therefore, $p_{u_i w_i}$ is sampled with probability $\frac{1}{n(n-1)} \frac{1}{\sigma_{u_i w_i}}$.

The value $\frac{R(x_{u_i} - x_{w_i})}{\text{minus}_s[z]}$ is added to $\mathbf{v}[z]$ (Algorithm 3, line 19), for each $z \in p_{u_i w_i}$ reached by the backward traversing (lines 16–20). To keep track of the set \mathcal{V} , Algorithm 3 keeps an array count of size n , such that each value in the array contains the amount of vertices having the same value in the map \mathbf{v} .

The value of $\text{minus}_s[z]$ is correctly computed in line 4 as shown in Theorem 6. Let $S' \subseteq S$ be the set of shortest paths that z is an internal vertex. Then, at the end of Algorithm 2,

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

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 $\eta_r \leq \epsilon$, $L_S(f_v) = \tilde{p}(v)$ 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). \square

Theorem 8. *Given a weighted graph $G = (V, E)$ with $n = |V|$ and $m = |E|$ and a sample of size $r = \frac{c}{\epsilon^2} (\lceil \lg \text{diam}(G) \rceil - 2) + 1 - \ln \delta$, Algorithm 2 has running time $\mathcal{O}(m \log^2 n)$.*

Proof. We sample the vertices u and w and the shortest path p_{uw} in lines 11, 12 and 17, 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 $\mathcal{O}(n)$. As for the loop in lines 16–20, the complexity analysis proceeds as follows. Once $|P_u(w)| \leq d_G(w)$, where $d_G(w)$ denotes the degree of w in G and $P_u(w)$ is the set of predecessors of w in the shortest paths from u to w , 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 $\sum_{v \in V} d_G(v) = 2m = \mathcal{O}(m)$.

Line 21 is executed by an algorithm that is linear in the size of the sample [8]. The loop in lines 8–24 runs at most r times and the Dijkstra algorithm which is executed in line 13 has running time $\mathcal{O}(m \log n)$, so the total running time of Algorithm 2 is $\mathcal{O}(n \log n + r \max(m, m \log n)) = \mathcal{O}(n \log n + r(m \log n)) = \mathcal{O}(r(m \log n)) = \mathcal{O}(m \log^2 n)$. \square

Corollary 1. *Given an unweighted graph $G = (V, E)$ with $n = |V|$ and $m = |E|$ and a sample of size $r = \frac{c}{\epsilon^2} (\lceil \lg \text{diam}(G) \rceil - 2) + 1 - \ln \delta$, Algorithm 2 has 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 $(u, w) \in V^2$ will be computed by the BFS algorithm, which has running time $\mathcal{O}(m + n)$. \square

We observe that, even though it is an open problem whether there is a $\mathcal{O}(n^{3-\epsilon})$ 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 [20]. 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. Conclusion

We presented a sampling based algorithm to accurately estimate the percolation centrality of every vertex of a weighted graph with high probability. The proposed algorithm has 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 diameter (typically of size $\log n$), our algorithm provides a fast approximation for such graphs (more precisely running in $\mathcal{O}(n \log n \log \log n)$ 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] 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.
- [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] Alane M. de Lima, Murilo V. G. da Silva, and André L. Vignatti. 2019. Estimating the Percolation Centrality of Large Networks through Pseudo-dimension Theory. arXiv:cs.DS/1910.00494
- [8] Steven G. Johnson. 2014. The NLOpt nonlinear-optimization package. <http://github.com/stevengj/nlopt>
- [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] 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>
- [14] Luca Oneto, Alessandro Ghio, Sandro Ridella, and Davide Anguita. 2016. Global rademacher complexity bounds: From slow to fast convergence rates. *Neural Processing Letters* 43, 2 (2016), 567–602.
- [15] 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.
- [16] 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>
- [17] M. Riondato and E. M. Kornaropoulos. 2016. Fast approximation of betweenness centrality through sampling. *Data Mining and Knowledge Discovery* 30, 2 (2016), 438–475.
- [18] Matteo Riondato and Eli Upfal. 2015. Mining Frequent Itemsets through Progressive Sampling with Rademacher Averages. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '15)*. Association for Computing Machinery, New York, NY, USA, 1005–1014. <https://doi.org/10.1145/2783258.2783265>
- [19] 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.
- [20] R. Seidel. 1995. On the All-Pairs-Shortest-Path Problem in Unweighted Undirected Graphs. *J. Comput. System Sci.* 51, 3 (1995), 400 – 403.
- [21] S. Shalev-Shwartz and S. Ben-David. 2014. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press.
- [22] 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>
- [23] 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>