PROJECT PERIODIC REPORT

Grant Agreement number: 288956

Project acronym: NADINE

Project title: New tools and Algorithms for Directed Network analysis

Funding Scheme: Small or medium-scale focused research project (STREP)

Periodic report: 1st 2nd X

Period covered: from 1.11.2013 to 30.04.2015

Name, title and organisation of the scientific representative of the project's coordinator[1]:

Dr. Dima Shepelyansky

Directeur de recherche au CNRS

Lab de Phys. Theorique, Universite Paul Sabatier, 31062 Toulouse, France

Tel: +331 5 61556068, Fax: +33 5 61556065, Secr.: +33 5 61557572

E-mail: dima@irsamc.ups-tlse.fr; URL: www.quantware.ups-tlse.fr/dima

Project website address: www.quantware.ups-tlse.fr/FETNADINE/

[1] Usually the contact person of the coordinator as specified in Art. 8.1. of the grant agreement

NADINE DELIVERABLE D1.2.

It is based on milestones M10 with deliverable publications (WP1.4)

[17] P2.5 L.Ostroumova, K.Avrachenkov, N.Litvak, "**Quick detection of popular entities in large directed networks**", preprint submitted to Computer Science Conference, Oct 2013 [reported in period 1]

[49] P2.12 N.Litvak and S.Vigna, "**Introduction to Special Issue on Searching and Mining the Web and Social Networks**", Internet Mathematics, v.10(3-4), p.219-221 (2014)

[50] P2.13 K.Avrachenkov, N.Litvak, L.Ostroumova-Prokhorenkova and E.Suyargulova, "**Quick detection of high-degree entities in large directed networks**", IEEE International Conference on Data Mining (ICDM 2014), 14-17 Dec 2014, Shenzhen, China. pp. 20-29. IEEE Computer Society (2014) (arXiv:1410.0571v2[cs.SI])

[

Quick detection of popular entities in large directed networks

ABSTRACT

In this paper, we address a problem of quick detection of popular entities in large online social networks. Practical importance of the problem is attested by a large number of companies that continuously collect and update statistics about popular entities. We suggest an efficient two-stage algorithm for solving this problem. For instance, our algorithm needs only one thousand API requests in order to find the top-50 most popular users in Twitter, a network with more than a billion of registered users. Our algorithm is easy to implement, it outperforms existing methods, and serves many different purposes, such as finding most popular users or most popular interest groups in social networks. An important contribution of this work is the analysis of the proposed algorithm using the Extreme Value Theory – a branch of probability that studies extreme events and properties of largest order statistics in random samples. Using this theory, we derive accurate predictions for the algorithm's performance and show that the number of API requests for finding top-k most popular entities is sublinear in the number of entities. Moreover, we formally show that the high variability among the entities, expressed through heavy-tailed distributions, is the reason for the algorithm's efficiency. We quantify this phenomenon in a rigorous mathematical way.

1. INTRODUCTION

In this paper, we propose a randomized algorithm for quick detection of popular entities in large online social networks. The entities can be, for example, users or interest groups, user categories, geographical locations, etc. For instance, one can be interested in finding out a list of Twitter users with many followers or Facebook interest groups with many members. Practical importance of the problem is attested by a large number of companies that continuously collect and update statistics about popular entities (*twittercounter.com*, *followerwonk.com*, *twitaholic.com*, *www.insidefacebook.com*, *yavkontakte.ru* just to name a few).

The problem at hand may seem trivial, if one assumes that the network structure and the relation between entities are known. However, even then finding, for example, top-k in-degree nodes in a directed graph G of size N takes the time $\mathrm{O}(N)$. For large networks, such linear complexity is already too high. In fact, for any practical purpose, it is much more valuable to find an approximate result in a sublinear time than an exact result in a linear time. Furthermore, the data of current social graphs is typically available only to the owners of the social network, and can be obtained by other interested parties only through API requests. The rate of allowed API requests is usually quite small. For instance, Twitter has the limit of one access per minute for one standard API account. Then, in order to crawl the entire network with more than 500 million users one need more than 950 years. Clearly, we would like to find most popular entities using only a small number of API requests.

Formally, the problem addressed in this paper is as follows. Let V be a set of entities, usually users, that can be accessed using API requests. Let W also be another set of entities (possibly equal to V). We represent V and W as vertices of a bipartite graph (V, W, E), where a directed edge $(v, w) \in E$, with $v \in V$, $w \in W$, represents a relation between v and w. For instance, if $V = W$ is a set of Twitter users, then $(v, w) \in E$ may mean that v follows w, or that v retweeted a tweet from w. Note that any directed graph $G = (V, E)$ can be represented equivalently by the bi-partite graph (V, V, E). One can also suppose that V is a set of users and W is a set of interest groups, while the edge (v, w) represents that user v belongs to group w. Our goal is to quickly find top-k in-degree entities in W. In this setting, throughout the paper, we use the terms 'nodes' and 'entities' interchangeably.

The algorithm proposed in this paper can detect popular entities with high precision using very small number of API requests. Most of our experiments are performed on the Twitter graph, because it is a good example of a huge network (billion of registered users) and very limited rate of requests to API. We use only 1000 API request to find top-50 Twitter users with very high precision. We also demonstrate the efficacy of our approach on the example of online social network (to be specified in the camera-ready version) which had, at the time of article preparation, more than 200 million registered users. We use our algorithm to quickly detect most popular interest groups in this social network. Experiments on random graph models show that our algorithm outperforms the baselines algorithms from [4] and [14]. Moreover, our algorithm can be used in a very general settings for finding most popular entities, while the baseline algorithms can only be use for finding nodes of largest degrees in directed ([14]) or undirected ([4]) graphs.

An important contribution of this work is the novel analysis of proposed algorithm using classical results of the Extreme Value Theory (EVT) – a branch of probability that studies extreme events and properties of largest order statistics in random samples. We refer to [8] for a comprehensive introduction to EVT. Specifically, we treat the largest in-degrees in W as high order statistics of a heavy-tailed distribution, and use EVT to obtain the limiting properties of these order statistics. This way we obtain statistical estimation of the magnitude of the largest in-degrees in W. Using these mathematical tools, we can, for instance, accurately predict the average fraction of correctly identified top-100 most followed users in Twitter using only the knowledge of top-20 degrees, which can be detected by our algorithm very quickly with practically 100% accuracy.

We derive the complexity of our algorithm in terms of the number of entities in W show that the complexity is sublinear if the in-degree distribution in W is heavy tailed. Intuitively, this should be the case because the high variability of the in-degrees implies that the largest entities have extremely large number of in-links and thus are easy to find. We formalize this argument using the EVT results.

The algorithm consists of two stages. The parameters of the algorithm, n_1 and n_2, are the number of API requests used on the first and the second stage, respectively. The performance of the algorithm is very robust with respect of the parameters' values. We find the optimal scaling for n_1 and n_2 with respect to three measures of the algorithm performance: the average fraction of correctly identified top-k entities, the first-error index (the number of the highest statistics within top-k that was not included in the identified top-k list), and the the sum of incoming degrees of identified n_2 entities. Notice that for fixed n, the latter performance measure does not monotonically grows with n_2 because with small n_1 the number of links received from n_1 random users is a poor indication of the node's actual degree. This can be clearly seen in Figure 2 for the Twitter graph.

The rest of the paper is organized as follows. In Section 2, we give a short overview of the related work. In Section 3, we formally describe our algorithm. We empirically show the efficiency of our algorithm and compare it to baseline strategies in Section 4. We present a detailed analysis of the algorithm in Section 5 and evaluate its optimal parameters with respect to the above mentioned performance characteristics. Section 6 concludes the paper.

2. RELATED WORK

Over the last years data sets have become increasingly massive. For such large data any complexity higher than linear (in dataset size) is unacceptable, and even linear complexity may be too high. It is also well understood that an algorithm, which runs in sublinear time, cannot return an exact answer. In fact, such algorithms often use randomization, and then errors occur with positive probability. Nevertheless, in practice, a rough but quick answer is often more valuable than exact but computationally demanding solution. Therefore, sublinear time algorithms become increasingly important, and many studies of such algorithms have appeared in recent years (see, e.g., [10, 13, 15, 16]).

An essential assumption of this work is that the network structure is not available, and has to be discovered using the API requests. This setting is similar to on-line computations, when information is obtained and immediately processed while crawling the network graph (for instance the World Wide Web). There is a large body of literature where such on-line algorithms are developed and analyzed. Many of these algorithms are developed for computing and updating the PageRank vector [1, 6]. In particular, the algorithm recently proposed in [6] computes the PageRank vector in sublinear time. Furthermore, the probabilistic Monte Carlo methods [2, 11] allow to continuously update the PageRank as the structure of the Web changes.

Randomized algorithms are also used for discovering the structure of social networks. Often random walks are designed in such a way that the desired nodes are easily found. For example, in [12] an unbiased random walk, where each node is visited with equal probability, is constructed in order to find the degree distribution on Facebook. A different random walk is designed in [4] for finding nodes with largest degrees in undirected graphs. This random walk has jumps, so that it does not get stuck around just one hub, but unlike PageRank, its a stationary distribution completely defined by the nodes' degrees.

The problem of finding the most popular entities in large networks has been analyzed in several papers. In Section 4.3 we show that our algorithm outperforms two baselines: the random walk algorithm in [4], and the crawling algorithm in [14]. The latter algorithm [14] is designed to efficiently discover the correct set of pages with largest incoming degrees in a fixed network, and to track these pages over time when the network is changing. Their setting is different from ours in several aspects. For example, in our case we can use API to get indegree of any given item, while in the World Wide Web this information is not available. On the other hand, the algorithm in [14] is designed to discover the graph structure, and cannot be easily adopted for other tasks, such as finding most popular use categories or interest groups.

To the best our knowledge, this is the first work that presents and analyzes an efficient algorithm for retrieving most popular entities under realistic API constraints.

3. ALGORITHM DESCRIPTION

Recall that we consider a bipartite graph (V, W, E), where V and W are sets of entities, and $(v, w) \in E$ represents a relation between the entities.

Let $n = n_1 + n_2$. Our algorithm has two stages, described below. See Algorithm 1 for the pseudocode.

First stage. We start by sampling uniformly at random a set A of n_1 entities (users, or nodes) $v_1, \dots, v_{n_1} \in V$. The nodes are sampled independently, so the same node may appear in A more than once, in which case we regard each copy of this node as a different node. Since multiplicities occur with very small probability this does not affect the efficiency of the algorithm but simplifies the implementation. For each node in A we record its out-neighbors in W. In practice, we bound the number of recorded out-links by the maximal number of id's that can be retrieved within one API request, thus the first stage uses exactly n_1 API requests.

Second stage. Let S_w, $w \in W$, be the number of nodes in A that have a (recorded) edge to w, and let w_i be the node in W with i-th largest values of S_w, so that $S_{w_1} \geqslant S_{w_2} \geqslant \dots \geqslant S_{w_N}$. Then we use another n_2 API requests to retrieve the actual in-degrees of the n_2 top-nodes $w_1, \dots, w_{n_2}$.

The set $\{w_1, w_2, \dots, w_{n_2}\}$ is supposed to contain nodes from W with large in-degrees. For example, if we are interested in top-k in-degree nodes in a directed graph, we hope to identify these nodes with high precision if k is significantly smaller than n_2.

```
Algorithm 1: Find entities with large incoming degrees
input : Set of entities $V$ of size $M$, set of entities
        $W$ of size $N$, number of random nodes $n_1$,
        number of candidate nodes $n_2$
output: Nodes $w_1, \dots w_{n_2} \in W$, their degrees
        $d_1, \dots, d_{n_2}$
for $i \leftarrow 1$ to $N$ do
    $S[i] \leftarrow 0$;
for $i \leftarrow 1$ to $n_1$ do
    $v \leftarrow random(M)$;
    $F \leftarrow OutNeighbors(v)$;
    foreach $j$ in $F$ do
        $S[j] \leftarrow S[j] + 1$;
$w_1, \dots, w_{n_2} \leftarrow Top\_n_2(S)$ // $S[w_1], \dots, S[w_{n_2}]$ are
top $n_2$ values in $S$;
for $i \leftarrow 1$ to $n_2$ do
    $d_i \leftarrow InDegree(w_i)$;
```

4. EXPERIMENTS

4.1 Twitter graph

First, we show that our algorithm quickly finds the most popular users in Twitter graph. Formally, V is a set of Twitter users, $W = V$, and $(v, w) \in E$ iff v is a follower of w. Twitter is an example of a huge network with limited access to its structure. Information on the Twitter graph can be obtained via Twitter API. The standard rate of requests to API is one per minute. Every vertex has an id, which is an integer number starting from 12. The largest id of a user is $\sim$ 1460M (at the time when we performed the experiments). Due to such id assignment, a random user in Twitter can be easily chosen. The only problem is that some users in this range have been deleted, some are suspended, and therefore errors occur when addressing the id's of these pages. In our implementation we usually skip errors and assume that we do not spend resources on such nodes. The fraction of errors is $\approx 20\%$.

Given an id of a user, a request to API can return one of the following: i) the number of followers (indegree), ii) the number of followees (outdegree), or iii) at most 5000 id's of followers or followees. If a user has more than 5000 followees, then all their id's can be retrieved only by using several API requests. Instead, as described above, we record only the first 5000 of the followees and ignore the rest. This does not affect the performance of the algorithm because we record followees of randomly sampled users, and the fraction of Twitter users with more than 5000 followees, is small.

In order to obtain the ground truth, we first took $n_1 = n_2 = 500\,000$ and found top-1000 users with a very high precision. We used the obtained list for evaluating the performance of our algorithm.

Figure 1 shows the average fraction of correctly identified users from top-k for different k over 100 experiments, as a function of n_2, when $n = 1000$. Remarkably, we can find top-50 users with very high precision.

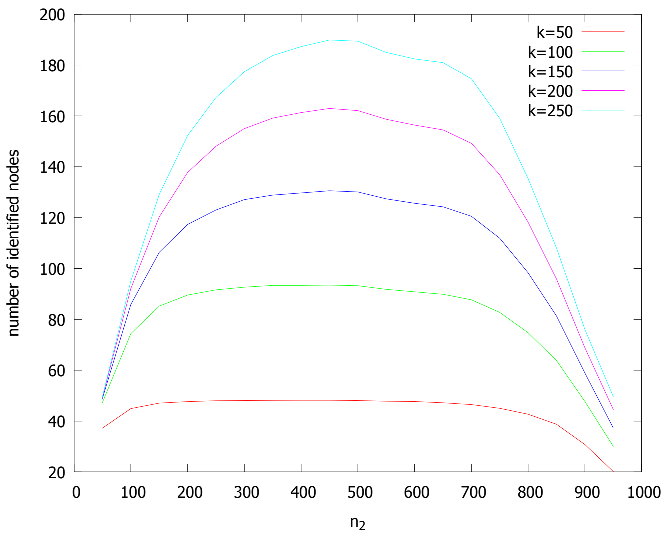

Figure 1: The number of correctly identified top-k most followed Twitter users as a function of n_2, with $n = 1000$.

We have also looked at the first-error index – the position of the first mistake in the top-k list. Formally, if we correctly identified top-$(i-1)$ users, but did not find the ith user, then the first-error index is i. Again, we have averaged the results over 100 experiments. Results are shown in Figure 4 below (red line). Note that with only 1000 API requests we can correctly identify more than 50 users without any omission.

The sums of the degrees of the identified top-n_2 entities, with $n = 1000$, are depicted in Figure 2. Observe that here the optimal value of n_2 is larger than in two previously discussed metrics. Thus, in order to to discover as many true in-links as possible, we may want to check more incoming degrees in the second stage of the algorithm, so that we have a large output list, but with less precision. We will discuss this in more detail in Section 5.3.

4.2 Finding largest interest groups

Let V be a set of users, W be a set of interest groups, and $(v, w) \in E$ iff v is a member of w.

We will demonstrate that our algorithm can find the most popular groups in a large social network with more than 200M registered users (to be specified in the camera-ready version). As for Twitter, information on the network under consideration can be obtained via API. Again, all users have ids: integer numbers starting from 1. Due to this id assignment, a random user in this network can be easily chosen. In addition, all interest groups also have their own id's.

We are interested in the following requests to API: i) given id of a user, return his or her interest groups, ii) given id of a group return its number of members. If a user decide to hide the list of groups, then an error occurs. The portion of such errors is $\approx 30\%$.

As before, first we used our algorithm with $n_1 = n_2 = 50000$ in order to find the most popular groups with high precision. Table 1 presents some statistics on the most popular groups. Then, we took $n_1 = 700$, $n_2 = 300$ and com-

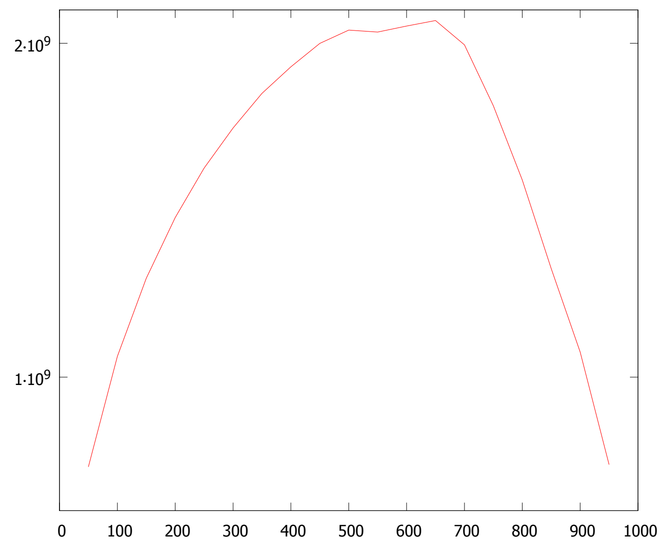

Figure 2: The sum of incoming degrees of identified users as a function of n_2, $n = 1000$.

Table 1: The most popular groups

Rank	Number of participants	Topic
1	4,35M	humor
2	4,1M	humor
3	3,76M	movies
4	3,69M	humor
5	3,59M	humor
6	3,58M	facts
7	3,36M	cookery
8	3,31M	humor
9	3,14M	humor
10	3,14M	movies
100	1,65M	success

puted the fraction of correctly identified groups from top-100. Using only 1000 API requests, our algorithm identifies on average 73.2 from the top-100 interest groups (averaged over 25 experiments). The standard deviation is 4.6.

4.3 Comparison with baseline algorithms

In this section we compare our algorithm with the algorithms suggested in [4] and [14]. We start with the description of these algorithms.

Random walk based algorithm [4]. The algorithm in [4] is a randomized algorithm for undirected graphs that finds a top-k list of nodes with largest degrees in sublinear time. It is based on the random walk with uniform jumps, described by the following transition probabilities [5]:

$$p_{ij} = \begin{cases} \frac{\alpha/N+1}{d_i+\alpha}, & \text{if } i \text{ has a link to } j, \\ \frac{\alpha/N}{d_i+\alpha}, & \text{if } i \text{ does not have a link to } j, \end{cases} \qquad (1)$$

where N is the number of nodes in the graph and d_i is the degree of node i. The parameter α controls how often the random walk makes an artificial jump. In [4] it is suggested to take the parameter α equal to the average degree in order to maximize the number of independent samples. Interestingly, this implies that the random walk, in stationarity, makes on average just one step between the jumps. With such choice of α the random walk method of [4] mimics most closely the suggested algorithm with independent sampling and exactly one step from entity in V to entity in W. We should note that the random walk method could be very valuable when the independent uniform sampling is expensive, for example, when the id space is very sparse.

The random walk keeps a candidate list of k nodes. Once a new node is discovered according to the transition probability (1), we check its degree and compare it with degrees of the nodes in the candidate list. If this newly discovered node has a degree larger than degrees of some nodes in the candidate list, the newly discovered node is inserted in the candidate list and a node with the smallest degree in the candidate list is pushed out. See Algorithm 2 for more detailed description. The algorithm can be run for a predefined number of steps or can be terminated according to one of the stopping criteria provided in [4].

Algorithm 2: Random walk based algorithm

```
input  : Graph $G$ with $N$ nodes, $E$ edges, number
         of steps $n$, size of output list $k$, parameter $\alpha$
output: Nodes $v_1, \dots v_k$, their degrees $d_1, \dots, d_k$
$v \leftarrow random(N)$;
$F \leftarrow Neighbors(v)$;
$D[v] \leftarrow size(F)$;
for $i \leftarrow 2$ to $n$ do
    $r \xleftarrow{\text{sample}} U[0,1]$;
    if $r < \frac{D[v]}{D[v]+\alpha}$ then
        $v \leftarrow$ random from $F$;
    else
        $v \leftarrow random(N)$;
    $F \leftarrow Neighbors(v)$;
    $D[v] \leftarrow size(F)$;
$v_1, \dots, v_k \leftarrow Top\_k(D)$ // $D[v_1], , D[v_k]$ are top $k$ values in $D$;
```

Crawl-Al and Crawl-GAI [14]. At each step we consider one node and ask for its outgoing edges. At step n any node j has its *apparent indegree* S_j, $j = 1, \dots, N$: the number of discovered edges pointing to this node. In Crawl-Al the next node to consider is a random node, with probability proportional to the apparent indegree. In Crawl-GAI, the next node is the node with the highest apparent indegree. After n steps we get a list of nodes with largest apparent indegrees. See Algorithm 3 for the pseudocode of the algorithm Crawl-GAI.

In the experiments of the present paper we take the same budget for all tested algorithms to compare their performance.

Note that we cannot compare the algorithms on the Twitter graph for several reasons. First, Algorithm 2 works only on undirected graphs. Second, in order to choose a random edge of a node, we need at least two request to API, to ask for followees and followers. Also, the random walk often hits nodes of high degree, and then many additional requests are needed to retrieve their followers and followees, because the

Algorithm 3: Crawl-GAI

```
input  : Graph $G$ with $N$ nodes, number of steps $n$,
         size of output list $k$
output: Nodes $v_1, \dots v_k$
for $i \leftarrow 1$ to $N$ do
    $S[i] \leftarrow 0$;
for $i \leftarrow 1$ to $N$ do
    $v \leftarrow \text{argmax}(S[i])$;
    $F \leftarrow OutNeighbors(v)$;
    foreach $j$ in $F$ do
        $S[j] \leftarrow S[j] + 1$;
$v_1, \dots, v_k \leftarrow Top\_k(S)$ // $S[w_1], \dots, S[w_k]$ are top $k$
values in $S$;
```

Table 2: Number of correctly identified nodes from top-100 averaged over 100 experiments, $n = 1000$.

Algorithm	mean	standard deviation
Our (directed)	91.9	4.88
Crawl GAI (directed)	81.9	2.42
Crawl AI (directed)	82.9	2.38
Our (undirected)	97.9	1.71
Random walk (undirected)	60.7	4.76

number of id's that can be obtained in one request is limited (5000 in Twitter). For example, we need 6K request to get the followers of a user with 30M followers. Algorithm 3 crawls only out-degrees, that are usually much smaller, but it can potentially suffer from the API constraints, for example, when in-degrees and out-degrees are dependent.

Therefore, in order to compare Algorithms 1–3, we have generated a random directed graph according to the configuration model (see [7]). Our artificial graph has 1M nodes, 6M edges, and the parameter of the power law degree distribution is 2. This directed graph is used to compare our algorithm to Crawl-Al and Crawl-GAI. In order to compare our method to the random walk based algorithm, we treat the generated graph as undirected. As prescribed by [4], we took α slightly smaller than the average degree in the graph (in our case $\alpha = 10$) and we considered a random walk with 1000 steps.

For the algorithm suggested in this paper we took $n_1 = 700$, $n_2 = 300$. The results of comparison can be seen in Table 2.

We expect our algorithm with $n_1 = 1000$ to be close to Crawl-GAI. Indeed, in the directed case our algorithm with $n_1 = 1000$ identifies 81.4 nodes from top-100 on average (this number is not presented in the table). Further improvement of our algorithm over the baselines is obtained because of the right balance between n_1 and n_2.

5. ANALYSIS OF THE ALGORITHM

In this section, we present the theoretical analysis of Algorithm 1. The goal of this analysis is: 1) to mathematically justify our suggested two-steps procedure; 2) to prove that the total number of API requests, n, scales sublinearly with the network size, N; 3) to find the optimal scaling of n_1 and n_2 (the number of API requests in the first and the second stage of the algorithm) with respect to n.

We number the nodes in W by $1, 2, \dots, N$ according to the number of incoming links, from most popular to least popular. As prescribed by Algorithm 1, we pick n_1 nodes in V uniformly at random. The first important observation is that S_j follows a binomial distribution. Indeed, let F_j be the unknown random in-degree of node $j \in W$, so that $F_1 \geqslant F_2 \geqslant \dots \geqslant F_N$. Then, if we label all nodes from V that have a edge to j (we call such nodes followers of j), then S_j is exactly the number of labeled nodes in a random sample of n_1 nodes, so its distribution is $Binomial\Big(n_1, \frac{F_j}{N}\Big)$. Hence, we have

$$\mathbb{E}(S_j|F_j) = n_1\,\frac{F_j}{N}, \quad \text{Var}(S_j) = n_1\,\frac{F_j}{N}\left(1 - \frac{F_j}{N}\right). \qquad (2)$$

For the top nodes with large F_j this distribution can be approximated with the Poisson distribution $Poisson\Big(\frac{n_1 F_j}{N}\Big)$.

5.1 Candidate list

The quality of the top-k lists produced by Algorithm 1 is defined by the events whether or not the value of S_j, $j = 1, \dots, k$, is among the top-n_2 values of $S_1, S_2, \dots, S_N$, obtained in the first stage of the algorithm. This is justified by the intuition that if $F_j > F_l$, then we are likely to see $S_j > S_l$. Note, however, that the case when S_j is as small as 1, the event $1 = S_j > S_l = 0$ is not informative.

Example 1. *Let us take $n_1 = n_2 = 500$ in the case of the Twitter graph. Then the average number of nodes i among the top-10000 with $S_i = 1$ is already*

$$\sum_{i=1}^{10^4} P(S_i = 1) \approx \sum_{i=1}^{10^4} \frac{500F_i}{5 \cdot 10^8} e^{-500F_i/5\cdot 10^8} = 2539.1,$$

hence, many more than n_2 nodes will have $S_i = 1$ and can make it to the top n_2 values of $S_1, S_2, \dots, S_N$ only with a small probability.

Motivated by the above considerations, we formulate our approach in terms of a statistical test as follows. Let our data be $S_1, S_2, \dots, S_N$. We assume that the observations are realizations of independent Poisson random variables with parameters $n_1F_1/N, n_1F_2/N, \dots, n_1F_N/N$. For the two numbers $j, l \in 1, \dots, N$, we test the null-hypothesis $H_0 : F_j \leqslant F_l$ against the alternative $H_1 : F_j > F_l$. Let $S_{i_1} \geqslant S_{i_2} \geqslant \cdots \geqslant S_{i_{n_2}}$ be the top-n_2 order statistics of $S_1, \dots, S_N$ obtained by Algorithm 1. Then the first stage of the algorithm is equivalent to rejecting $H_0 : F_{i_j} \leqslant F_{i_{n_2}}$ for $j = 1, \dots, n_2 - 1$ such that

$$S_{i_j} > \max\{S_{i_{n_2}}, 1\}. \qquad (3)$$

Here the strict inequality is necessary to guarantee that i_j is on the top-n_2 list after the first stage of the algorithm. If H_0 is rejected, then the actual degree of entity i_j will be retrieved in the second stage of the algorithm.

Note that in contrast to the classical hypothesis testing, here we do not draw the conclusions solely from the observed random data $S_1, S_2, \dots, S_N$ but we obtain the true values of the parameters in the second stage of the algorithm. Hence, if we use $S_{i_{n_2}}$ as a proxy for S_{n_2}, then, given $F_1, F_2, \dots, F_N$, the quality of the top-k list is expressed as the power of the test as follows:

$$
\begin{aligned}
&P(\text{node } j \text{ is found}|F_j, F_{n_2}) \\
&\quad = P(S_j > \max\{S_{i_{n_2}}, 1\}|F_j, F_{i_{n_2}}) \qquad (4) \\
&\quad \approx P(S_j > \max\{S_{n_2}, 1\}|F_j, F_{n_2}) \\
&\quad \approx \sum_{s=0}^{\infty} e^{-\frac{n_1 F_{n_2}}{N}} \frac{(n_1 F_{n_2})^s}{N^s s!} \sum_{r>\max\{s,1\}} e^{-\frac{n_1 F_j}{N}} \frac{(n_1 F_j)^r}{N^r r!} \\
&\quad =: P_j(n_1), \quad j = 1, \dots, k. \qquad (5)
\end{aligned}
$$

5.2 Performance criteria

The main constraint of Algorithm 1 is the number of API requests that we can use. In order to measure the performance of the algorithm, we propose three objectives, described formally in this section.

The first objective is the average number of correctly identified top-k nodes. This is defined in the same way as in [3]:

$$
\begin{aligned}
&E[\text{fraction of correctly indentified top-}k \text{ entities}] \\
&\quad = \frac{1}{k}\sum_{j=1}^{k} P(\text{node } j \text{ is found}|F_j, F_{n_2}) \approx \frac{1}{k}\sum_{j=1}^{k} P_j(n_1). \qquad (6)
\end{aligned}
$$

The second objective is the first-error index, which is equal to i if the top $(i-1)$ entities are identified correctly, but the top-i entity is not identified. If all top-n_2 entities are identified correctly, we set the first-error index equal to $n+1$. Using that for a discrete random variable X with values $1, 2, \dots, k+1$ holds $E(X) = \sum_{l=0}^{k} P(X > l)$, we obtain the average first-error index as follows:

$$
\begin{aligned}
&E[\text{1st-error index}] = \sum_{l=0}^{n_2} P(\text{1st-error index} > l) \\
&\quad = \sum_{j=1}^{n_2+1} \prod_{l=1}^{j-1} P(S_j > \max\{S_{i_{n_2}}, 1\}|F_j, \dots, F_{i_{n_2}}) \\
&\quad \approx \sum_{j=1}^{n_2} \prod_{l=1}^{j-1} P_l(n_1). \qquad (7)
\end{aligned}
$$

Finally, our last objective is the sum of the identified top-n_2 degrees, that can be written in a very simple form:

$$
U := [\text{sum of identified } n_2 \text{ degrees}] = \sum_{l=1}^{n_2} F_{i_l}. \qquad (8)
$$

5.3 EVT performance predictions

In order to compute the values in (6), (7), we need to make assumptions on the top-n_2 in-degrees of entities in W: F_1, $F_2, \dots, F_{n_2}$. To this end, we employ the quantile estimation techniques from the Extreme Value Theory (EVT).

In most social networks the degrees of the entities show a great variability. This is often modeled using power laws, although it has been often argued that classical Pareto distribution does not always fit the observed data. In our analysis we assume that the incoming degrees of the entities in W are independent random variables following a *regularly varying* distribution G:

$$
1 - G(x) = L(x)x^{-1/\gamma}, \quad x > 0, \qquad (9)
$$

where $L(\cdot)$ is a slowly varying function, that is,

$$
\lim_{x\to\infty} L(tx)/L(x) = 1, \quad t > 0
$$

($L(\cdot)$ can be, for example, a constant or a logarithm). We note that (9) describes a broad class of heavy-tailed distributions, for which the EVT arguments presented below are valid, without imposing the rigid Pareto assumption.

Observe that $F_1, F_2, \dots, F_N$ are the order statistics of G. Assume now that we know the correct values of the top-m nodes, $m < k$. This is plausible because, for instance, in Twitter, with $n = 1000$, the top-50 nodes are identified with a very high precision, see Figure 1. Then, in order to estimate the value of γ, we can use the classical Hill's estimator $\hat{\gamma}$, based on the top-m order statistics:

$$
\hat{\gamma} = \frac{1}{m-1}\sum_{i=1}^{m-1} \log(F_i) - \log(F_m). \qquad (10)
$$

Next, we use the quantile estimator, given by formula (4.3) in [9], but we replace their two-moment estimator by the Hill's estimator in (10). This is possible because both estimators are consistent (under slightly different conditions). Under the assumption $\gamma > 0$, we have the following estimator f_j for the $(j-1)/N$-th quantile of G:

$$
f_j = F_m \left(\frac{m}{j-1}\right)^{\hat{\gamma}}, \qquad j > 1, j << N. \qquad (11)
$$

We propose to use f_j as a prediction of F_j.

Note that our argument is inspired but not entirely justified by [9] because the consistency of the proposed quantile estimator (11) is only proved for $j < m$, while we want to use it for $j > m$. However, in the experiments we observe that expressions (6) and (7) are very robust with respect to the estimated values $F_1, \dots, F_{n_2}$. Moreover, $\hat{\gamma}$ increases with m, and it is easy to see that with smaller $\hat{\gamma}$ the predictions of the algorithm performance are more conservative.

In Figure 3 we compare the true fraction of the correctly identified top-k followed Twitter users to the performance prediction (6) for $n = 1000$ and $k = 100$. The magenta line shows the prediction for the fraction of correctly identified nodes in (6), where we used the correct values of $F_1, F_2, \dots, F_{n_2}$. The green line represents the results for the estimated values of $F_1, \dots, F_k$ and F_{n_2}, based on the true values of the top-20 degrees. We see that it is very close to the magenta line, which is based on the true values of the degrees.

Similarly, we use formula (7) and the estimator (11) in order to provide the prediction of the first-error index. The results are given in Figure 4. We see again that the EVT predictions are more pessimistic than the experimental results, so we find the lower bound for the algorithm's actual performance. Note also that the shape of the plot and the optimal value of n_2 have been captures correctly by both predictors.

It is also clear that there is a principal difficulty in finding similar analytical predictions for the objective U in (8) because is it is based not on the *actual* degrees $F_1, F_2, \dots$, but on the degrees $F_{i_1}, F_{i_2}, \dots, F_{i_{n_2}}$, where $S_{i_1} \geqslant S_{i_2} \geqslant \dots \geqslant S_{i_N}$ are the order statistics of the S_j's. The exact expressions for such order statistics are rather messy. However, we can get some insight in the behavior of U in Figure 2. Indeed, clearly, the sum of correct top-n_2 degrees, $\sum_{i=1}^{n_2} F_i$, is an increasing function of n_2. Moreover, if we use the estimator (11), then we observe that the largest values of F_j's

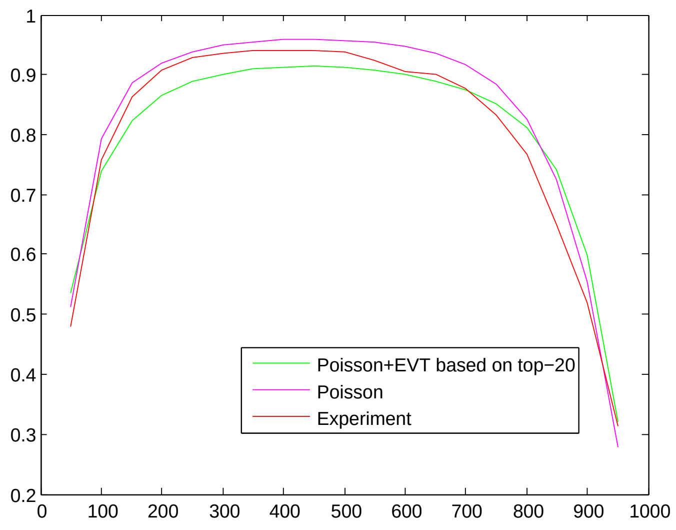

Figure 3: Fraction of correctly predicted nodes out of top-100 as a function of n_2, with $n = 1000$: experiments (red); prediction (5) based on the true values of the degrees (magenta); prediction (5) based on top-m degrees and estimator (11) with $m = 20$, $\hat{\gamma} = 2.2$ (green).

are of the same order of magnitude:

$$F_j/F_l \approx \left(\frac{l-1}{j-1}\right)^{\hat{\gamma}}.$$

Thus, as long as n_1 large enough so that a large entity j receives large S_j, we have that U is comparable to $\sum_{i=1}^{n_2} F_i$, and hence U increases in n_2. However, as n_1 becomes smaller, then small entities will constitute a large proportion of the set $\{i_1, i_2, \dots, i_{n_2}$. For example, if $n_2 = 800$, $n_1 = 200$, then we obtain, for the true values of in-degrees in Twitter graph with $N \approx 500$M:

$$\sum_{i=1}^{800} P(S_i > 1) \approx 280.9,$$

thus on average about 520 out of the top-800 nodes will be undistinguishable from other, much smaller nodes (see Example 1). Moreover, in this case

$$\sum_{i=1}^{10^5} P(S_i > 1) \approx 485.18,$$

thus, on average, more than 300 nodes will be included into $\{i_1, \dots, i_{800}\}$ essentially on a random basis. Since large majority of the nodes has very small degrees, this will drastically affect the magnitude of U. This is exactly what we observe in Figure 2.

5.4 Optimal scaling for n_1 and n_2

In this section our goal is to find the ratio n_2 to n_1 which maximizes the performance of the Algorithm 1. For simplicity, as a performance criterion we consider the fraction of correctly identified nodes from top-k in (6):

$$\frac{1}{k}\sum_{j=1}^{k} P_j(n_1) \to \max.$$

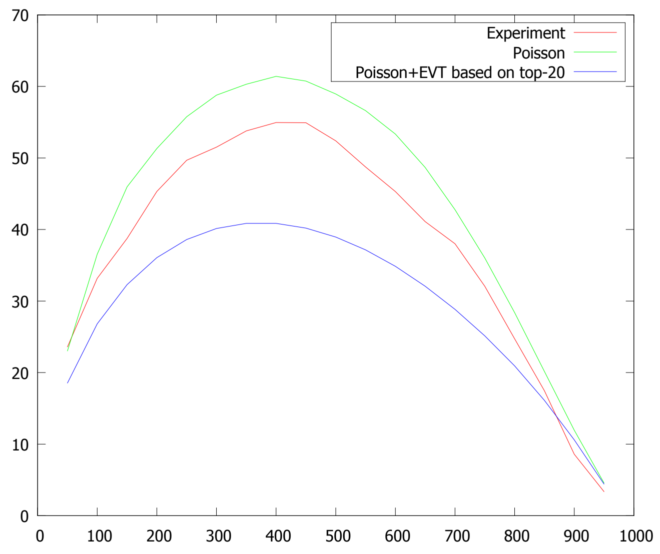

Figure 4: The position of first error as a function of n_2, with $n = 1000$.

We start with analyzing the optimal scaling for n_1. Intuitively, after the first stage of the algorithm, only $O(n_1)$ nodes j will have $S_j > 1$, and thus there is no need to check more than $n_2 = O(n_1)$ nodes in the second stage, which implies that n_1 should grow at least proportionally to n. This is formalized in the next proposition.

Proposition 1. *It is optimal to choose $n = O(n_1)$.*

Proof. Let J be a randomly chosen node, and J_l, $l = 1, \dots, n_1$ be independent realizations of J in the first stage of Algorithm 1. Denote by M the maximal number of neighbors that a given API allows to retrieve. The first stage of the algorithm returns a list of candidate nodes, for which we require $S_j > 1$. Observe that the number of such nodes is bounded by

$$U := \frac{1}{2}\sum_{l=1}^{n_1} \max\{M, \text{out-degree}(J_l)\}.$$

Assuming that the out-degrees of each node are independent, we obtain that

$$E(U) = \frac{1}{2} n_1 E(\max\{M, \text{out-degree}(J)\}),$$
$$Var(U) = \frac{1}{4} n_1 \text{Var}(\max\{M, \text{out-degree}(J)\}).$$

Note that the API restriction simplifies the derivation because the variance of $\max\{M, \text{out-degree}(J)\}$ is finite. The formal argument for $M = \infty$ and infinite variance of out-degrees will be similar but requires some more work. Using, e.g. Chernoff bound or Chebyshev bound we obtain that $P(U > E(U)(1 + \varepsilon)) \to 0$ as $n_1 \to \infty$. Thus, the number of nodes j with $S_j > 1$ is at most $O(n_1)$ with high probability, so we choose $n_2 = O(n_1)$ which results in $n = O(n_1)$. ☐

Note that if n is large enough, then the top nodes (first, second, etc.) can be found with very high probability. Figure 1 shows that if $n = 1000$, then for a wide range of n_2 the fraction of correctly identified nodes from top-50 is the same. As k grows, the optimization becomes much more important. Motivated by this observation, we maximize the value $P_k(n_1)$. We prove the following theorem.

THEOREM 1. *Assume that $k = o(n)$ as $n \to \infty$. The maximizer n_2^* of probability $P_k(n - n_2)$ is close to the maximal root of the equation*

$$\frac{1}{3\gamma k^\gamma} x^{\gamma+1} + x - n = 0, \qquad (12)$$

that is,

$$n_2^* = x(1 + o(1)), \quad \text{as} \quad k/n_2^* \to 0.$$

If in addition $n_2^* = o(n)$ as $n \to \infty$, then n_2^* can be given in a closed-form asymptotic expression

$$n_2 = (3\gamma k^\gamma n)^{\frac{1}{\gamma+1}} + o(n^{\frac{1}{\gamma+1}}).$$

PROOF. Consider first an extreme regime: $x = \mathrm{O}(k)$. Thus, we exclude the regime $n - x = \mathrm{o}(n)$. Consequently, $n_1 \to \infty$ as $n \to \infty$ and we can apply the following normal approximation

$$P_k(n_1) \approx P\left(N\left(\frac{n_1(F_k - F_{n_2})}{N}, \frac{n_1(F_k + F_{n_2})}{N}\right) > 0\right)$$

$$= P\left(N(0,1) > -\sqrt{\frac{n_1}{N}}\frac{F_k - F_{n_2}}{\sqrt{F_k + F_{n_2}}}\right). \qquad (13)$$

(A completely formal justification can be given by the Berry-Esseen theorem.) Thus, in order to maximize the above probability, we need to maximize $\sqrt{\frac{n_1}{N}}\frac{F_k - F_{n_2}}{\sqrt{F_k+F_{n_2}}}$. From EVT it follows that F_k decays as $k^{-\gamma}$. So, we can maximize

$$\frac{\sqrt{n_1}\left(k^{-\gamma} - n_2^{-\gamma}\right)}{\sqrt{k^{-\gamma} + n_2^{-\gamma}}}. \qquad (14)$$

Now if $x = \mathrm{O}(k)$, $\sqrt{n-x} = \sqrt{n}(1 + o(1))$, and the maximization of (14) mainly depends on the remaining term in the product, which is an increasing function of n_2. This suggests that n_2 has to be chosen considerably greater than k. Hence, we proceed assuming the only interesting asymptotic regime where $k = o(n_2)$. In this asymptotic regime, we can simplify (14) as follows:

$$\frac{\sqrt{n-x}\left(k^{-\gamma} - x^{-\gamma}\right)}{\sqrt{k^{-\gamma} + x^{-\gamma}}} =$$

$$\frac{1}{k^{\gamma/2}}\sqrt{n-x}\left(1 - \frac{3}{2}\left(\frac{k}{x}\right)^\gamma\right) + \mathrm{o}\left(\left(\frac{k}{x}\right)^\gamma\right).$$

Next, we differentiate the function

$$f(x) := \sqrt{n-x}\left(1 - \frac{3}{2}\left(\frac{k}{x}\right)^\gamma\right)$$

and set the derivative to zero. This results in equation (12). If we assume further that $n_2^* = \mathrm{o}(n)$, then only the highest order term will remain in (12) and we immediately obtain the following approximation

$$n_2 = (3\gamma k^\gamma n)^{\frac{1}{\gamma+1}} + \mathrm{o}(n^{\frac{1}{\gamma+1}}).$$

□

For example, for $n = 1000$, $k = 100$, and $\gamma = 0.35$ we get $n_2 \approx 570$.

5.5 Sublinear complexity

The normal approximation (13) immediately implies the following proposition.

PROPOSITION 2. *For large enough n_1, the inequality*

$$\sqrt{\frac{n_1}{N}}\frac{F_k - F_{n_2}}{\sqrt{F_k + F_{n_2}}} \geqslant x_{1-\varepsilon}$$

guarantees that on average we can find the fraction $1 - \varepsilon$ of top-k nodes in W.

For the inequality in (2) to hold, it is necessary that $\sqrt{n_1}(F_k - F_{n_2})$ is at least of the same order of magnitude as $N\sqrt{F_k + F_{n_2}}$. Moreover, it follows from Proposition 1 that $n = O(n_1)$, and thus the complexity n of the algorithm is defined by n_1. In the theorem below we use the results from Extreme Value Theory to show that n_1 scales sublinearly with N.

Theorem 1, and estimator (11), we can already provide a rough indication of the number of API request we need to use. Indeed, $k > m$, rough estimation with $n - n_2 \approx n$ and $F_k >> F_{n_2}$ gives

$$n \geqslant \frac{N x_{1-\varepsilon}^2 k^{\hat{\gamma}}}{F_m m^\gamma}. \qquad (15)$$

For finding top-100 most followed users on Twitter with good precision, this will result in about 5000 of API requests (with $N = 500M$, $m = 20$, $k = 100$, $x_{1-\varepsilon} \approx 2$, $\hat{\gamma} = 2.2$).

For a better result, we may take into account the value of n_2, and substitute the value $n_2 = \left(3k^{\hat{\gamma}} n\hat{\gamma}\right)^{\frac{1}{\gamma+1}}$ obtained in Proposition 2:

$$\frac{k^{-\hat{\gamma}/2}}{2\sqrt{n}}\left(2n - \left(3k^{\hat{\gamma}} n\hat{\gamma}\right)^{\frac{1}{\gamma+1}}\right)\left(1 - \frac{3}{2}\left(3k^{\hat{\gamma}} n\hat{\gamma}\right)^{\frac{-\hat{\gamma}}{\gamma+1}} k^{\hat{\gamma}}\right)$$

$$\geqslant x_{1-\varepsilon}\sqrt{\frac{N}{F_m m^\gamma}}.$$

From (15) we can also already anticipate that n is sublinear in N because $F_m m^\gamma$ grows with N. This argument is formalized in Theorem 2 below.

Notice that, interestingly, the obtained complexity is in terms of the cardinality of W, not V. In particular, this makes the problem of finding popular groups easier than the problem of finding popular users.

THEOREM 2. *If the in-degrees of the nodes are independent realizations of a regularly varying distribution G with exponent $1/\gamma$ as defined in (9), and $F_1 \geqslant F_2 \geqslant \cdots \geqslant F_N$ are their order statistics. Let $(a_N)_{N\geqslant 1}$, $(b_N)_{N\geqslant 1}$ be sequences such that*

$$\lim_{N\to\infty} N(1 - G(a_N x + b_N)) = (1 + \gamma x)^{-1/\gamma}.$$

Then Algorithm 1 finds $(1-\varepsilon)$ of the top-k nodes with high probability in

$$n_1 = O(N/a_N),$$

of API requests. In particular, n scales sublinearly in N, and

$$\log(n_1) = (1 - \gamma)\log(N).$$

PROOF. For a regularly varying G, Theorem 2.1.1 in [8] can be applied, and thus for any finite m

$$\left(\frac{F_1 - b_N}{a_N}, \cdots \frac{F_m - b_N}{a_N}\right)$$

converges in distribution, as $N \to \infty$, to

$$\left(\frac{E_1^{-\gamma} - 1}{\gamma}, \cdots, \frac{(E_1 + \cdots + E_m)^{-\gamma} - 1}{\gamma}\right),$$

where E_i's are independent exponential random variables with parameter 1. This implies, in particular, that $a_N/b_N = \mathrm{O}(1)$ and that for large enough N and any $\varepsilon > 0$, there exist l_i, u_i such that $P[l_i a_N \leqslant F_i \leqslant u_i a_N] > 1-\varepsilon$. It follows that for fixed k

$$\sqrt{\frac{n_1}{N}}\sqrt{F_k} = O(1)$$

with high probability when $n_1 = O(N/a_N)$, and the first statement of the theorem follows because $k = o(n_2)$ implying that $F_{n_2} = \mathrm{o}(F_k)$. In particular, if G is a Pareto distribution, $1 - G(x) = Cx^{-1/\gamma}$, $x \geqslant x_0$, then

$$a_N = \gamma C^\gamma N^\gamma, \quad b_N = C^\gamma n^\gamma.$$

For a general regularly varying distribution in (9) the slowly varying function will influence a_N but the logarithmic asymptotics of a_N will be still determined by the power law:

$$\log(a_N) = \gamma \log(N),$$

which gives the result. $\square$

6. CONCLUSION

We proposed a randomized algorithm for quick detection of popular entities in large online social networks whose architecture has underlying directed graphs. Examples of social network entities are users and interest groups. We have analyzed the algorithm with respect to three criteria and compared with two baseline methods. Our analysis demonstrates that the algorithm has nonlinear complexity on networks with heavy-tailed in-degree distribution and that the performance of the algorithm is robust with respect to the values of its few parameters. The algorithm outperforms the two baseline methods and has much wider applicability. An important ingredient of our analysis is substantial use of the extreme value theory. The extreme value theory is not so well know in computer science and sociology but appears to be a very useful tool in the analysis of social networks. We feel that our work could be a good reference point for other researchers to start applying EVT in social network analysis. We have validated our theoretical results on two very large online social networks.

We see several extensions of the present work. A top list of popular entities is just one type of properties of social networks. We expect that our approach based on extreme value theory and using referral links can be extended to infer and to analyze other properties such as power law index and the tail, network functions and network motifs, degree-degree correlation. It will be very interesting and useful to develop quick and effective statistical tests to check for network assortativity and presence of heavy tails.

Since our approach requires very small numbers of API accesses, we believe that it will trace well network changes. Of course, a formal justification of the algorithm applicability for dynamic networks is needed.

7. REFERENCES

[1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. *Proceedings of the 12-th International World Wide Web Conference*, 2003.

[2] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM J. Numer. Anal.*, 45(2):890–904, 2007.

[3] K. Avrachenkov, N. Litvak, D. Nemirovsky, E. Smirnova, and M. Sokol. Quick detection of top-k personalized pagerank lists. In *Proceeding on the 8th Workshop on Algorithms and Models for the Web Graph, WAW 2011*, pages 50–61. Springer, 2011.

[4] K. Avrachenkov, N. Litvak, M. Sokol, and D. Towsley. Quick detection of nodes with large degrees. In *Proceeding on the 9th Workshop on Algorithms and Models for the Web Graph*, pages 54–65. Springer, 2012.

[5] K. Avrachenkov, B. Ribeiro, and D. Towsley. Improving random walk estimation accuracy with uniform restarts. In *Proceeding on the 7th Workshop on Algorithms and Models for the Web Graph, WAW 2010*, pages 98–109. Springer, 2010.

[6] C. Borgs, M. Brautbar, J. Chayes, and S.-H. Teng. A sublinear time algorithm for pagerank computations. *Lecture Notes in Computer Science*, 7323:41–53, 2012.

[7] T. Britton, M. Deijfen, and A. Martin-Löf. Generating simple random graphs with prescribed degree distribution. *J. Stat. Phys.*, 124(6):1377–1397, 2006.

[8] L. De Haan and A. Ferreira. *Extreme value theory.* Springer, 2006.

[9] A. L. M. Dekkers, J. H. J. Einmahl, and L. de Haan. A moment estimator for the index of an extreme-value distribution. *The Annals of Statistics*, 17(4):1833–1855, 1989.

[10] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS*, 75:97–126, 2001.

[11] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlósa. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358, 2005.

[12] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. *Proceedings of IEEE INFOCOM'10*, 2010.

[13] O. Goldreich. Combinatorial property testingŮa survey. *Randomization Methods in Algorithm Design, DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 45–60, 1998.

[14] R. Kumar, K. Lang, C. Marlow, and A. Tomkins. Efficient discovery of authoritative resources. *IEEE 24th International Conference on Data Engineering*, pages 1495–1497, 2008.

[15] R. Rubinfeld and A. Shapira. Sublinear time algorithms. *SIAM J. Discrete Math.*, 25(4):1562–1588, 2011.

[16] M. Sudan. Invariance in property testing. *Property Testing: Current Research and Surveys, O. Goldreich, ed., Lecture Notes in Comput. Sci.*, pages 211–227, 2010.

Preface

The past few decades have seen the rise of online social networks as a worldwide phenomenon with a high impact on our society. Beyond the obvious exposure phenomena, with obvious implications on security and privacy, people have started to become acquainted—even married!—in online social networks. In parallel, we have seen an enormous growth in terms of the number of published papers in computer science, mathematics and physics that study the organization of such networks. The availability of large free databases of friendships, collaborations and quotations have made possible to study social networks at a scale and with a precision previously unknown.

This issue of *Internet Mathematics*, titled *'Searching and mining the Web and social networks'*, was born out of the interest of the editors for the problem of searching and analyzing not only the web, but also social networks in a broad sense. In particular, we aimed to publish a collection of papers that take a rigorous mathematical viewpoint on problems most important and common in network applications. The general topics represented in this special issue cover ranking of the nodes, network measurements, and adversarial behavior. Each of these topics received a large attention in the literature. We believe however that the originality of the papers presented in this volume is in a high level of mathematical rigor.

All submitted articles have been thoroughly reviewed in accordance with the usual high standards of *Internet Mathematics*. Each paper received reviews from at least two experts: one in the field of application, and one in the in the relevant branch of mathematics. Some high quality submissions have been rejected because they did not sufficiently satisfied either the criterion of practical importance for social networks or did not have a sufficient mathematical depth.

The first two papers deal with the problem of detecting interesting properties of the nodes using only the structure of a social network (i.e., the underlying graph). In the paper *Axioms for centrality* by Paolo Boldi and Sebastiano Vigna the authors try to understand the inner works of *centrality measures*, which are designed to identify which nodes in a social networks are more important than others. The paper *Towards quantifying vertex similarity in networks* by Charalampos Tsourakakis proposes (somewhat dually) new techniques to identiy *similar* nodes in large networks.

The next two papers address statistical measurements in social networks, including the in-depth mathematical analysis of the proposed estimators. The paper *Degree-degree dependencies in random graphs with heavy tailed degrees* by Remco van der Hofstad and Nelly Litvak is the first rigorous study of statistical estimators for correlations between degrees of neighbouring nodes in general social networks, and in common random graph models for them. The paper *Estimating sizes of social networks via biased sampling* by Liran Katzir, Edo Liberty, Oren Somekh, and Ioana A. Cosma proposes a new random walk sampling techniques for estimating the network size – the basic network characteristic that is often important and unavailable in practice.

Finally, the last two papers are devoted to identifying, predicting, and preventing an adversarial behaviour in Web and social networks. The paper *Communities, random walks and social sybil defense* by Lorenzo Alvisi, Allen Clement, Alessandro Epasto, Silvio Lattanzi, Alessandro Panconesi addresses a question whether a sybil attack, when an adversary introduces fake nodes and links in the graph, can be identified, based solely on the graph structure. To this end, they study analytically the change of the graph characteristics under a specified model of the sybil attack. The paper *The classification power of Web features* by Miklós Erdélyi, András A. Benczúr, Bálint Daróczy, András Garzó, Tamás Kiss and Dávid Siklósi analyzes in a rigorous experimental setting a wide range of signals used to detect *spam* pages. Both these papers also provide an excellent review on their respective topics.

We would like to thank the authors of all submissions for their high quality contributions. It has been our pleasure to receive and handle the very interesting papers submitted to this volume. We hope that it will give rise to new fascinating research on the topic.

Nelly Litvak, Sebastiano Vigna, guest editors.

Quick Detection of High-degree Entities in Large Directed Networks

K. Avrachenkov
Inria
k.avrachenkov@inria.fr

N. Litvak
University of Twente
n.litvak@utwente.nl

L. Ostroumova Prokhorenkova
Yandex
ostroumova-la@yandex.ru

E. Suyargulova
Yandex
siyargul@yandex.ua

***Abstract*—In this paper we address the problem of quick detection of high-degree entities in large online social networks. Practical importance of this problem is attested by a large number of companies that continuously collect and update statistics about popular entities, usually using the degree of an entity as an approximation of its popularity. We suggest a simple, efficient, and easy to implement two-stage randomized algorithm that provides highly accurate solutions to this problem. For instance, our algorithm needs only one thousand API requests in order to find the top-100 most followed users, with more than 90% precision, in the online social network Twitter with approximately a billion of registered users. Our algorithm significantly outperforms existing methods and serves many different purposes such as finding the most popular users or the most popular interest groups in social networks. An important contribution of this work is the analysis of the proposed algorithm using Extreme Value Theory — a branch of probability that studies extreme events and properties of largest order statistics in random samples. Using this theory we derive an accurate prediction for the algorithm's performance and show that the number of API requests for finding the top-*k* most popular entities is sublinear in the number of entities. Moreover, we formally show that the high variability of the entities, expressed through heavy-tailed distributions, is the reason for the algorithm's efficiency. We quantify this phenomenon in a rigorous mathematical way.**

I. Introduction

In this paper we propose a randomized algorithm for quick detection of high-degree entities in large online social networks. The entities can be, for example, users, interest groups, user categories, geographical locations, etc. For instance, one can be interested in finding a list of Twitter users with many followers or Facebook interest groups with many members. The importance of this problem is attested by a large number of companies that continuously collect and update statistics about popular entities in online social networks (*twittercounter.com*, *followerwonk.com*, *twitaholic.com*, *www.insidefacebook.com*, *yavkontakte.ru* just to name a few).

The problem under consideration may seem trivial if one assumes that the network structure and the relation between entities are known. However, even then finding for example the top-k in-degree nodes in a directed graph G of size N takes the time $\mathrm{O}(N)$. For very large networks, even linear complexity is too high cost to pay. Furthermore, the data of current social networks is typically available only to managers of social networks and can be obtained by other interested parties only through API (Application Programming Interface) requests. API is a set of request messages, along with a definition of the structure of response messages. Using one API request it is usually possible to discover either friends of one given user, or his/her interest groups, or the date when his/her account was created, etc. The rate of allowed API requests is usually very limited. For instance, Twitter has the limit of one access per minute for one standard API account (see *dev.twitter.com*). Then, in order to crawl the entire network with a billion users, using one standard API account, one needs more than 1900 years.

Hence currently, there is a rapidly growing interest in algorithms that evaluate specific network properties, using only local information (e.g., the degree of a node and its neighbors), and give a good approximate answer in the number of steps that is sublinear in the network size. Recently, such algorithms have been proposed for PageRank evaluation [3], [9], [10], for finding high-degree nodes in graphs [4], [11], [12], [20], and for finding the root of a preferential attachment tree [8].

In this paper, we propose a new two-stage method for finding high-degree nodes in large directed networks with highly skewed in-degree distribution. We demonstrate that our algorithm outperforms other known methods by a large margin and has a better precision than the for-profit Twitter statistics *twittercounter.com*.

II. Problem formulation and our contribution

Let V be a set of N entities, typically users, that can be accessed using API requests. Let W be another set of M entities (possibly equal to V). We consider a bipartite graph (V, W, E), where a directed edge $(v, w) \in E$, with $v \in V$, and $w \in W$, represents a relation between v and w. In our particular model of the Twitter graph V is a set of Twitter users, $W = V$, and $(v, w) \in E$ means that v follows w or that v retweeted a tweet of w. Note that any directed graph $G = (V, E)$ can be represented equivalently by the bipartite graph (V, V, E). One can also suppose that V is a set of users and W is a set of interest groups, while the edge (v, w) represents that the user v belongs to the group w.

Our goal is to quickly find the top in-degree entities in W. In this setting, throughout the paper, we use the terms 'nodes', 'vertices', and 'entities' interchangeably.

We propose a very simple and easy-to-implement algorithm that detects popular entities with high precision using a surprisingly small number of API requests. Most of our experiments are performed on the Twitter graph, because it is a good example of a huge network (approximately a billion of registered users) with a very limited rate of requests to API. We use only 1000 API requests to find the top-100

[0]The authors are given in alphabetical order. L. Ostroumova Prokhorenkova is the principal author.

Twitter users with a very high precision. We also demonstrate the efficacy of our approach on the popular Russian online social network VKontakte (*vk.com*) with more than 200 million registered users. We use our algorithm to quickly detect the most popular interest groups in this social network. Our experimental analysis shows that despite of its simplicity, our algorithm significantly outperforms existing approaches, e.g., [4], [11], [20]. Moreover, our algorithm can be used in a very general setting for finding the most popular entities, while some baseline algorithms can only be used for finding nodes of largest degrees in directed [20] or undirected [4] graphs.

In most social networks the degrees of entities show great variability. This is often modeled using power laws, although it has been often argued that the classical Pareto distribution does not always fit the observed data. In our analysis we assume that the incoming degrees of the entities in W are independent random variables following a *regularly varying* distribution G:

$$1 - G(x) = L(x)x^{-1/\gamma}, \quad x > 0,\ \gamma > 0, \tag{1}$$

where $L(\cdot)$ is a slowly varying function, that is,

$$\lim_{x\to\infty} L(tx)/L(x) = 1, \quad t > 0.$$

$L(\cdot)$ can be, for example, a constant or logarithmic function. We note that (1) describes a broad class of heavy-tailed distributions without imposing the rigid Pareto assumption.

An important contribution of this work is a novel analysis of the proposed algorithm that uses powerful results of the Extreme Value Theory (EVT) — a branch of probability that studies extreme events and properties of high order statistics in random samples. We refer to [13] for a comprehensive introduction to EVT. Using EVT we can accurately predict the average fraction of correctly identified top-k nodes and obtain the algorithm's complexity in terms of the number of nodes in V. We show that the complexity is sublinear if the in-degree distribution of the entities in W is heavy tailed, which is usually the case in real networks.

The rest of the paper is organized as follows. In Section III, we give a short overview of related work. We formally describe our algorithm in Section IV, then we introduce two performance measures in Section V. Section VI contains extensive experimental results that demonstrate the efficiency of our algorithm and compare it to baseline strategies. In Sections VII-IX we present a detailed analysis of the algorithm and evaluate its optimal parameters with respect to the two performance measures. Section X concludes the paper.

III. Related work

Over the last years data sets have become increasingly massive. For algorithms on such large data any complexity higher than linear (in dataset size) is unacceptable and even linear complexity may be too high. It is also well understood that an algorithm which runs in sublinear time cannot return an exact answer. In fact, such algorithms often use randomization, and then errors occur with positive probability. Nevertheless, in practice, a rough but quick answer is often more valuable than the exact but computationally demanding solution. Therefore, sublinear time algorithms become increasingly important and many studies of such algorithms appeared in recent years (see, e.g., [15], [18], [24], [25]).

An essential assumption of this work is that the network structure is not available and has to be discovered using API requests. This setting is similar to on-line computations, where information is obtained and immediately processed while crawling the network graph (for instance the World Wide Web). There is a large body of literature where such on-line algorithms are developed and analyzed. Many of these algorithms are developed for computing and updating the PageRank vector [1], [2], [9], [16]. In particular, the algorithm recently proposed in [9] computes the PageRank vector in sub-linear time. Furthermore, probabilistic Monte Carlo methods [2], [6], [16] allow to continuously update the PageRank as the structure of the Web changes.

Randomized algorithms are also used for discovering the structure of social networks. In [21] random walk methods are proposed to obtain a graph sample with similar properties as a whole graph. In [17] an unbiased random walk, where each node is visited with equal probability, is constructed in order to find the degree distribution on Facebook. Random walk based methods are also used to analyse Peer-to-Peer networks [22]. In [8] traceroute algorithms are proposed to find the root node and to approximate several other characteristics in a preferential attachment graph.

The problem of finding the most popular entities in large networks based only on the knowledge of a neighborhood of a current node has been analyzed in several papers. A random walk algorithm is suggested in [12] to quickly find the nodes with high degrees in a preferential attachment graph. In this case, transitions along undirected edges x, y are proportional to $(d(x)d(y))^b$, where $d(x)$ is the degree of a vertex x and $b > 0$ is some parameter.

In [4] a random walk with restart that uses only the information on the degree of a currently visited node was suggested for finding large degree nodes in undirected graphs. In [11] a local algorithm for general networks, power law networks, and preferential attachment graphs is proposed for finding a node with degree, which is smaller than the maximal by a factor at most c. Another crawling algorithm [20] is proposed to efficiently discover the correct set of web pages with largest incoming degrees in a fixed network and to track these pages over time when the network is changing. Note that the setting in [20] is different from ours in several aspects. For example, in our case we can use API to inquire the in-degree of any given item, while in the World Wide Web the information on in-links is not available, the crawler can only observe the in-links that come from the pages already crawled.

In Section VI-B we show that our algorithm outperforms the existing methods by a large margin. Besides, several of the existing methods such as the ones in [4] and [20] are designed specifically to discover the high degree nodes, and they cannot be easily adapted for other tasks, such as finding the most popular user categories or interest groups, while the algorithm proposed in this paper is simpler, much faster, and more generic.

To the best of our knowledge, this is the first work that presents and analyzes an efficient algorithm for retrieving the most popular entities under realistic API constraints.

IV. Algorithm description

Recall that we consider a bipartite graph (V, W, E), where V and W are sets of entities and $(v, w) \in E$ represents a relation between the entities.

Let n be the allowed number of requests to API. Our algorithm consists of two steps. We spend n_1 API requests on the first step and n_2 API requests on the second step, with $n_1 + n_2 = n$. See Algorithm 1 for the pseudocode.

Algorithm 1: Two-stage algorithm

```
input : Set of entities $V$ of size $N$, set of entities
        $W$ of size $M$, number of random nodes $n_1$
        to select from $V$, number of candidate nodes
        $n_2$ from $W$
output: Nodes $w_1, \dots w_{n_2} \in W$, their degrees
        $d_1, \dots, d_{n_2}$
for $w$ in $W$ do
    $S[w] \leftarrow 0$;
for $i \leftarrow 1$ to $n_1$ do
    $v \leftarrow random(N)$;
    foreach $w$ in $OutNeighbors(v) \subset W$ do
        $S[w] \leftarrow S[w] + 1$;
$w_1, \dots, w_{n_2} \leftarrow Top\_n_2(S)$ // $S[w_1], \dots, S[w_{n_2}]$ are
the top $n_2$ maximum values in $S$;
for $i \leftarrow 1$ to $n_2$ do
    $d_i \leftarrow InDegree(w_i)$;
```

First stage. We start by sampling uniformly at random a set A of n_1 nodes $v_1, \dots, v_{n_1} \in V$. The nodes are sampled independently, so the same node may appear in A more than once, in which case we regard each copy of this node as a different node. Note that multiplicities occur with a very small probability, approximately $1 - e^{-n_1^2/(2N)}$. For each node in A we record its out-neighbors in W. In practice, we bound the number of recorded out-links by the maximal number of IDs that can be retrieved within one API request, thus the first stage uses exactly n_1 API requests. For each $w \in W$ we identify $S[w]$, which is the number of nodes in A that have a (recorded) edge to w.

Second stage. We use n_2 API requests to retrieve the actual in-degrees of the n_2 nodes with the highest values of $S[w]$. The idea is that the nodes with the largest in-degrees in W are likely to be among the n_2 nodes with the largest $S[w]$. For example, if we are interested in the top-k in-degree nodes in a directed graph, we hope to identify these nodes with high precision if k is significantly smaller than n_2.

V. Performance metrics

The main constraint of Algorithm 1 is the number of API requests we can use. Below we propose two performance metrics: the average fraction of correctly identified top-k nodes and the first-error index.

We number the nodes in W in the deceasing order of their in-degrees and denote the corresponding in-degrees by $F_1 \geqslant F_2 \geqslant \dots \geqslant F_M$. We refer to F_j as the j-th order statistic of the in-degrees in W. Further, let S_j be the number of neighbors of a node j, $1 \leqslant j \leqslant M$, among the n_1 randomly chosen nodes in V, as described in Algorithm 1. Finally, let $S_{i_1} \geqslant S_{i_2} \geqslant \ldots \geqslant S_{i_M}$ be the order statistics of $S_1, \dots, S_M$. For example, i_1 is the node with the largest number of neighbors among n_1 randomly chosen nodes, although i_1 may not have the largest degree. Clearly, node j is identified if it is in the set $\{i_1, i_2, \dots, i_{n_2}\}$. We denote the corresponding probability by

$$P_j(n_1) := \mathrm{P}(j \in \{i_1, \dots, i_{n_2}\}) . \tag{2}$$

The first performance measure is the average fraction of correctly identified top-k nodes. This is defined in the same way as in [3]:

$$\mathbb{E}[\text{fraction of correctly identified top-}k\text{ entities}] = \frac{1}{k} \sum_{j=1}^{k} P_j(n_1). \tag{3}$$

The second performance measure is the first-error index, which is equal to i if the top $(i-1)$ entities are identified correctly, but the top-ith entity is not identified. If all top-n_2 entities are identified correctly, we set the first-error index equal to $n_2 + 1$. Using the fact that for a discrete random variable X with values $1, 2, \dots, K+1$ holds $\mathbb{E}(X) = \sum_{j=1}^{K+1} \mathrm{P}(X \geqslant j)$, we obtain the average first-error index as follows:

$$\begin{aligned}\mathbb{E}[\text{1st-error index}] &= \sum_{j=1}^{n_2+1} \mathrm{P}(\text{1st-error index} \geqslant j) \\ &= \sum_{j=1}^{n_2+1} \prod_{l=1}^{j-1} P_l(n_1).\end{aligned} \tag{4}$$

If the number n of API requests is fixed, then the metrics (3) and (4) involve an interesting trade-off between n_1 and n_2. On the one hand, n_1 should be large enough so that the values S_i's are sufficiently informative for filtering out important nodes. On the other hand, when n_2 is too small we expect a poor performance because the algorithm returns a top-k list based mainly on the highest values of S_i's, which have rather high random fluctuations. For example, on Figure 1, when $n_2 = k = 100$, the algorithm returns the nodes $\{i_1, \dots, i_{100}\}$, of which only 75% belong to the true top-100. Hence we need to find the balance between n_1 and n_2. This is especially important when n is not very large compared to k (see Figure 1 with $n = 1000$ and $k = 250$).

VI. Experiments

This section is organized as follows. First, we analyze the performance of our algorithm (most of the experiments are performed on the Twitter graph, but we also present some results on the CNR-2000 graph). Then we compare our algorithm with baseline strategies on the Twitter graph and show that the algorithm proposed in this paper significantly outperforms existing approaches. Finally, we demonstrate another application of our algorithm by identifying the most popular interest groups in the large online social network VKontakte.

All our experiments are reproducible: we use public APIs of online social networks and publicly available sample of a web graph.

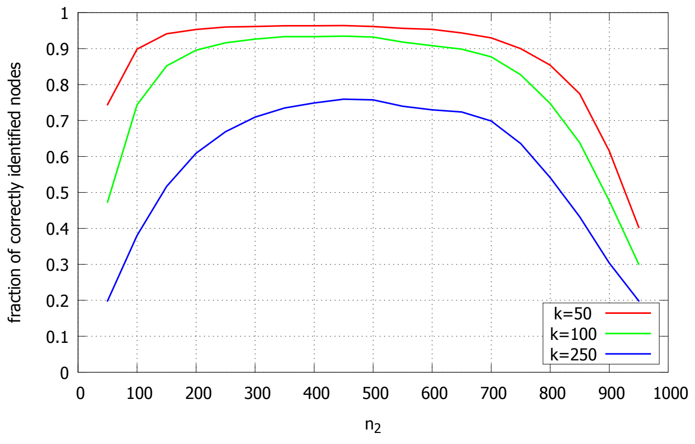

Fig. 1. The fraction of correctly identified top-k most followed Twitter users as a function of n_2, with $n = 1000$.

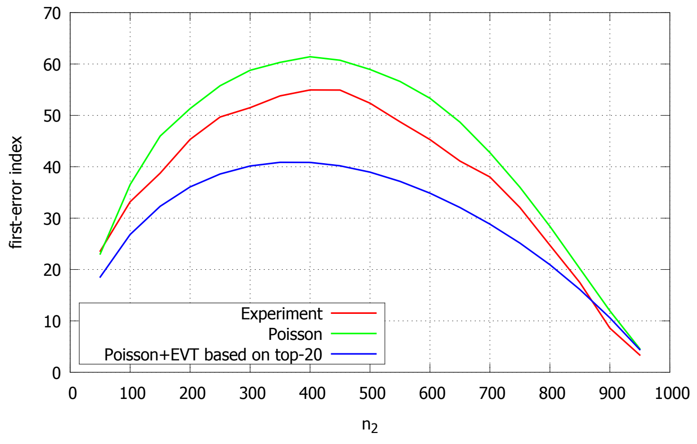

Fig. 2. The first-error index as a function of n_2, with $n = 1000$, on Twitter.

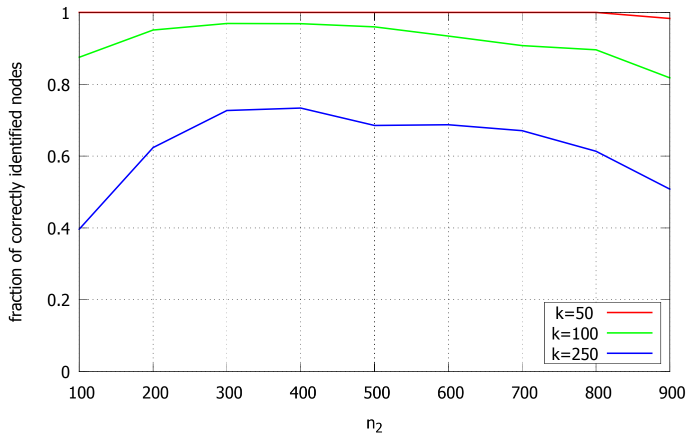

Fig. 3. The fraction of correctly identified top-k in-degree nodes in the CNR-2000 graph as a function of n_2, with $n = 1000$.

A. *Performance of the proposed algorithm*

First, we show that our algorithm quickly finds the most popular users in Twitter. Formally, V is a set of Twitter users, $W = V$, and $(v, w) \in E$ iff v is a follower of w. Twitter is an example of a huge network with a very limited access to its structure. Information on the Twitter graph can be obtained via Twitter public API. The standard rate of requests to API is one per minute (see *dev.twitter.com*). Every vertex has an ID, which is an integer number starting from 12. The largest ID of a user is $\sim$ 1500M (at the time when we performed the experiments). Due to such ID assignment, a random user in Twitter can be easily chosen. Some users in this range have been deleted, some are suspended, and therefore errors occur when addressing the IDs of these pages. In our implementation we skip errors and assume that we do not spend resources on such nodes. The fraction of errors is approximately 30%. In some online social networks the ID space can be very sparse and this makes problematic the execution of uniform sampling in the first stage of our algorithm. In such situation we suggest to use random walk based methods (e.g., Metropolis-Hastings random walk from [17] or continuous-time random walk from [22]) that produce approximately uniform sampling after a burn-in period. To remove the effect of correlation, one can use a combination of restart [5] and thinning [4], [17].

Given an ID of a user, a request to API can return one of the following: i) the number of followers (in-degree), ii) the number of followees (out-degree), or iii) at most 5000 IDs of followers or followees. If a user has more than 5000 followees, then all their IDs can be retrieved only by using several API requests. Instead, as described above, we record only the first 5000 of the followees and ignore the rest. This does not affect the performance of the algorithm because we record followees of randomly sampled users, and the fraction of Twitter users with more than 5000 followees is very small.

In order to obtain the ground truth on the Twitter graph, we started with a top-1000 list from the publicly available source *twittercounter.com*. Next, we obtained a top-1000 list by running our algorithm with $n_1 = n_2 = 20\,000$. We noticed that 1) our algorithm discovers all top-1000 users from *twittercounter.com*, 2) some top users identified by our algorithm are not presented in the top-1000 list on *twittercounter.com*. Then, we obtained the ground truth for top-1000 users by running our algorithm with ample number of API requests: $n_1 = n_2 = 500\,000$.

First we analyzed the fraction of correctly identified top-k nodes (see Equation (3)). Figure 1 shows the average fraction of correctly identified top-k users for different k over 100 experiments, as a function of n_2, when $n = 1000$, which is very small compared to the total number of users. Remarkably we can find the top-50 users with very high precision. Note that, especially for small k, the algorithm has a high precision in a large range of parameters.

We also looked at the first-error index (see Equation (4)), i.e., the position of the first error in the top list. Again, we averaged the results over 100 experiments. Results are shown on Figure 2 (red line). Note that with only 1000 API requests we can (on average) correctly identify more than 50 users without any omission.

Although in this paper we mostly focus on the Twitter graph (since it is a huge network with a very limited rate of requests to API), we also demonstrated the performance of our algorithm on CNR-2000 graph (*law.di.unimi.it/webdata/cnr-2000*). This graph is a sample of the Italian CNR domain. It is much smaller and there are no difficulties in obtaining the ground truth here. We get very similar results for this graph (see Figure 3). Interestingly, the performance of the algorithm is almost insensitive to the network size: the algorithm performs similarly on the network with a billion nodes as on the network with half a million nodes.

B. Comparison with baseline algorithms

Literature suggests several solutions for the problem studied here. Not every solution is feasible in the setting of a large unknown realistic network. For example, random-walk-based algorithms that require the knowledge of the degrees of all neighbors of a currently visited node, such as the one in [12], are not applicable. Indeed if we want to make a transition from a vertex of degree d, we need at least d requests to decide where to go. So once the random walk hits a vertex of high degree, we may spend all the allowed resources on just one transition of the random walk. In this section, we compare our algorithm with the algorithms suggested in [4], [11], and [20]. We start with the description of these algorithms.

RandomWalk [4].

The algorithm in [4] is a randomized algorithm for undirected graphs that finds a top-k list of nodes with largest degrees in sublinear time. This algorithm is based on a random walk with uniform jumps, described by the following transition probabilities [5]:

$$p_{ij} = \begin{cases} \frac{\alpha/N+1}{d_i+\alpha}, & \text{if } i \text{ has a link to } j, \\ \frac{\alpha/N}{d_i+\alpha}, & \text{if } i \text{ does not have a link to } j, \end{cases} \quad (5)$$

where N is the number of nodes in the graph and d_i is the degree of node i. The parameter α controls how often the random walk makes an artificial jump. In [4] it is suggested to take α equal to the average degree in order to maximize the number of independent samples, where the probability of sampling a node is proportional to its degree. After n steps of the random walk, the algorithm returns top-k degree nodes from the set of all visited nodes. See Algorithm 2 for formal description.

Note that Algorithm 2 works only on undirected graphs. In our implementation on Twitter, all links in the Twitter graph are treated as undirected, and the algorithm returns the top-k in-degree visited vertices. The idea behind this is that the random walk will often find users with large total number of followers plus followees, and since the number of followers of popular users is usually much larger than the number of followees, the most followed users will be found. Another problem of Algorithm 2 in our experimental settings is that it needs to request IDs of all neighbors of a visited node in order to follow a randomly chosen link, while only limited number of IDs can be obtained per one API request (5000 in Twitter). For example, the random walk will quickly find a node with 30M followers, and we will need 6K requests to obtain IDs of all its neighbors. Therefore, an honest implementation of Algorithm 2 usually finds not more than one vertex from top-100. Thus, we have implemented two versions of this algorithm: strict and relaxed. One step of the strict version is one API request, one step of the relaxed version is one considered vertex. Relaxed algorithm runs much longer but shows better results. For both algorithms we took $\alpha = 100$, which is close to twice the average out-degree in Twitter.

Algorithm 2: RandomWalk

```
input : Undirected graph $G$ with $N$ nodes, number
        of steps $n$, size of output list $k$, parameter $\alpha$
output: Nodes $v_1, \dots v_k$, their degrees $d_1, \dots, d_k$
$v \leftarrow random(N)$;
$A \leftarrow Neighbors(v)$;
$D[v] \leftarrow size(A)$;
for $i \leftarrow 2$ to $n$ do
    $r \xleftarrow{\text{sample}} U[0,1]$;
    if $r < \frac{D[v]}{D[v]+\alpha}$ then
        $v \leftarrow$ random from $A$;
    else
        $v \leftarrow random(N)$;
    $A \leftarrow Neighbors(v)$;
    $D[v] \leftarrow size(A)$;
$v_1, \dots, v_k \leftarrow Top\_k(D)$ // $D[v_1], \dots, D[v_k]$ are the
top $k$ maximum values in $D$;
```

Crawl-Al and Crawl-GAI [20].

We are given a directed graph G with N nodes. At each step we consider one node and ask for its outgoing edges. At every step all nodes have their *apparent in-degrees* S_j, $j = 1, \dots, N$: the number of discovered edges pointing to this node. In Crawl-Al the next node to consider is a random node, chosen with probability proportional to its apparent in-degree. In Crawl-GAI, the next node is the node with the highest apparent in-degree. After n steps we get a list of nodes with largest apparent in-degrees. See Algorithm 3 for the pseudocode of Crawl-GAI.

Algorithm 3: Crawl-GAI

```
input : Directed graph $G$ with $N$ nodes, number of
        steps $n$, size of output list $k$
output: Nodes $v_1, \dots v_k$
for $i \leftarrow 1$ to $N$ do
    $S[i] \leftarrow 0$;
for $i \leftarrow 1$ to $n$ do
    $v \leftarrow \text{argmax}(S[i])$;
    $A \leftarrow OutNeighbors(v)$;
    foreach $j$ in $A$ do
        $S[j] \leftarrow S[j] + 1$;
$v_1, \dots, v_k \leftarrow Top\_k(S)$ // $S[v_1], \dots, S[v_k]$ are the
top $k$ maximum values in $S$;
```

HighestDegree [11].

A strategy which aims at finding the vertex with largest degree is suggested in [11]. In our experimental setting with a limited number of API requests this algorithm can be presented as follows. While we have spare resources we choose random vertices one by one and then check the degrees of their neighbors. If the graph is directed, then we check the incoming degrees of out-neighbors of random vertices. See Algorithm 4 for the pseudocode of the directed version of this algorithm.

```
Algorithm 4: HighestDegree
input : Directed graph $G$ with $N$ nodes, number of
        steps $n$, size of output list $k$
output: Nodes $v_1, \dots v_k$, their degrees $d_1, \dots, d_k$
$s \leftarrow 0$;
for $i \leftarrow 1$ to $n$ do
    if $s = 0$ then
        $v \leftarrow random(N)$;
        $A \leftarrow OutNeighbors(v)$;
        $s \leftarrow size(A)$;
    else
        $D[A[s]] \leftarrow InDeg(A[s])$;
        $s \leftarrow s - 1$;
$v_1, \dots, v_k \leftarrow Top\_k(D)$ // $D[v_1], \dots, D[v_k]$ are the
top $k$ maximum values in $D$;
```

The algorithms Crawl-AI, Crawl-GAI and HighestDegree find nodes of large in-degrees, but crawl only out-degrees that are usually much smaller. Yet these algorithms can potentially suffer from the API constraints, for example, when in-degrees and out-degrees are positively dependent so that large in-degree nodes tend have high number of out-links to be crawled. In order to avoid this problem on Twitter, we limit the number of considered out-neighbors by 5000 for these algorithms.

In the remainder of this section we compare our Algorithm 1 to the baselines on the Twitter follower graph.

The first set of results is presented in Table I, where we take the same budget (number of request to API) $n = 1000$ for all tested algorithms to compare their performance. If the standard rate of requests to Twitter API (one per minute) is used, then 1000 requests can be made in 17 hours. For the algorithm suggested in this paper we took $n_1 = 700$, $n_2 = 300$.

As it can be seen from Table I, Crawl-GAI algorithm, that always follows existing links, seems to get stuck in some densely connected cluster. Note that Crawl-AI, which uses randomization, shows much better results. Both Crawl-GAI and Crawl-AI base their results only on apparent in-degrees. The low precision indicates that due to randomness apparent in-degrees of highest in-degree nodes are often not high enough. Clearly, the weakness of these algorithms is that the actual degrees of the crawled nodes remain unknown. Algorithm 2, based on a random walk with jumps, uses API requests to retrieve IDs of all neighbors of a visited node, but only uses these IDs to choose randomly the next node to visit. Thus, this algorithm very inefficiently spends the limited budget for API requests. Finally, HighestDegree uses a large number of API requests to check in-degrees of all neighbors of random nodes, so it spends a lot of resources on unpopular entities.

Our Algorithm 1 greatly outperforms the baselines. The reason is that it has several important advantages: 1) it is insensitive to correlations between degrees; 2) when we retrieve IDs of the neighbors of a random node (at the first stage of the algorithm), we increase their count of S, hence we do not lose any information; 3) sorting by $S[w]$ prevents the waste of resources on checking the degrees of unpopular nodes at the second stage; 4) the second stage of the algorithm returns the exact degrees of nodes, thus, to a large extent, we eliminate the randomness in the values of S.

TABLE I. PERCENTAGE OF CORRECTLY IDENTIFIED NODES FROM TOP-100 IN TWITTER AVERAGED OVER 30 EXPERIMENTS, $n = 1000$

Algorithm	mean	standard deviation
Two-stage algorithm	92.6	4.7
RandomWalk (strict)	0.43	0.63
RandomWalk (relaxed)	8.7	2.4
Crawl-GAI	4.1	5.9
Crawl-AI	23.9	20.2
HighestDegree	24.7	11.8

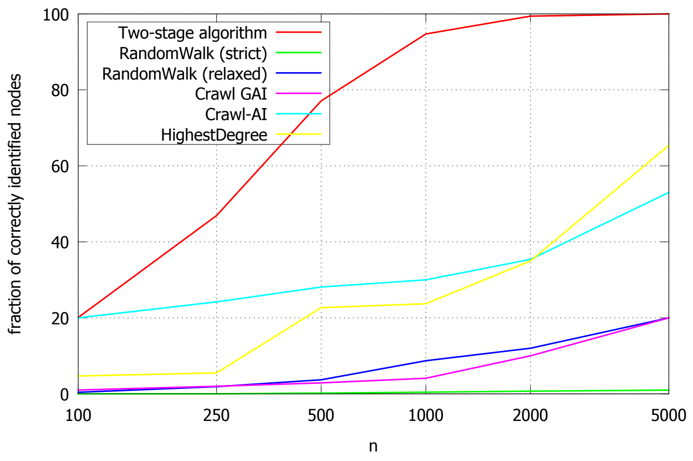

Fig. 4. The fraction of correctly identified top-100 most followed Twitter users as a function of n averaged over 10 experiments.

On Figure 4 we compare the average performance of our algorithm with the average performance of the baseline strategies for different values of n (from 100 to 5000 API requests). For all values of n our algorithm outperforms other strategies.

C. Finding the largest interest groups

In this section, we demonstrate another application of our algorithm: finding the largest interest groups in online social networks. In some social networks there are millions of interest groups and crawling all of them may not be possible. Using the algorithm proposed in this paper, the most popular groups may be discovered with a very small number of requests to API. In this case, let V be a set of users, W be a set of interest groups, and $(v, w) \in E$ iff v is a member of w.

Let us demonstrate that our algorithm allows to find the most popular interest groups in the large social network VKontakte with more than 200M registered users. As in the case of Twitter, information on the VKontakte graph can be obtained via API. Again, all users have IDs: integer numbers starting from 1. Due to this ID assignment, a random user in this network can be easily chosen. In addition, all interest groups also have their own IDs.

We are interested in the following requests to API: i) given an ID of a user, return his or her interest groups, ii) given an ID of a group return its number of members. If for some ID there is no user or a user decides to hide his or her list of groups, then an error occurs. The portion of such errors is again approximately 30%.

As before, first we used our algorithm with $n_1 = n_2 = 50\,000$ in order to obtain the ground truth for the top-100 most popular groups (publicly available sources give the same top-100). Table II presents some statistics on the most popular groups.

TABLE II. THE MOST POPULAR GROUPS FOR VKONTAKTE

Rank	Number of participants	Topic
1	4,35M	humor
2	4,10M	humor
3	3,76M	movies
4	3,69M	humor
5	3,59M	humor
6	3,58M	facts
7	3,36M	cookery
8	3,31M	humor
9	3,14M	humor
10	3,14M	movies
100	1,65M	success stories

Then, we took $n_1 = 700$, $n_2 = 300$ and computed the fraction of correctly identified groups from top-100. Using only 1000 API requests, our algorithm identifies on average 73.2 groups from the top-100 interest groups (averaged over 25 experiments). The standard deviation is 4.6.

VII. PERFORMANCE PREDICTIONS

In this section, we evaluate the performance of Algorithm 1 with respect to the metrics (3) and (4) as a function of the algorithm's parameters n_1 and n_2.

Recall that without loss of generality the nodes in W can be numbered $1, 2, \ldots, M$ in the decreasing order of their in-degrees, F_j is the unknown in-degree of a node j, and S_j is the number of followers of a node j among the randomly chosen n_1 nodes in V.

As prescribed by Algorithm 1, we pick n_1 nodes in V independently and uniformly at random with replacement. If we label all nodes from V that have an edge to $j \in W$, then S_j is exactly the number of labeled nodes in a random sample of n_1 nodes, so its distribution is $Binomial\left(n_1, \frac{F_j}{N}\right)$. Hence we have

$$\mathbb{E}(S_j) = n_1 \frac{F_j}{N}, \quad \mathrm{Var}(S_j) = n_1 \frac{F_j}{N}\left(1 - \frac{F_j}{N}\right). \qquad (6)$$

We are interested in predictions for the metrics (3) and (4). These metrics are completely determined by the probabilities $P_j(n_1)$, $j = 1, \ldots, k$, in (2). The expressions for $P_j(n_1)$, $j = 1, \ldots, k$, can be written in a closed form, but they are computationally intractable because they involve the order statistics of $S_1, S_2, \ldots, S_M$. Moreover, these expressions depend on the unknown in-degrees $F_1, F_2, \ldots, F_M$.

We suggest two predictions for (3) and (4). First, we give a *Poisson* prediction that is based on the unrealistic assumption that the degrees $F_1, \ldots, F_{n_2}$ are known, and replaces the resulting expression for (3) and (4) by an alternative expression, which is easy to compute. Next, we suggest an *Extreme Value Theory* (EVT) prediction that does not require any preliminary knowledge of unknown degrees but uses the top-m values of highest degrees obtained by the algorithm, where m is much smaller than k.

A. Poisson predictions

First, for $j = 1, \ldots, k$ we write

$$\begin{aligned} &P_j(n_1) = \\ &\quad = \mathrm{P}(S_j > S_{i_{n_2}}) + \mathrm{P}(S_j = S_{i_{n_2}}, j \in \{i_1, \ldots, i_{n_2}\}). \end{aligned} \qquad (7)$$

Note that if $[S_j > S_{i_{n_2}}]$ then the node j will be selected by the algorithm, but if $[S_j = S_{i_{n_2}}]$, then this is not guaranteed and even unlikely. This observation is illustrated by the following example.

Example 1. *Consider the Twitter graph and take $n_1 = 700$, $n_2 = 300$. Then the average number of nodes i with $S_i = 1$ among the top-l nodes is*

$$\sum_{i=1}^{l} \mathrm{P}(S_i = 1) = \sum_{i=1}^{l} 700 \frac{F_i}{10^9}\left(1 - \frac{F_i}{10^9}\right)^{699},$$

which is 223.3 for $l = 1000$, and it is 19.93 for $l = n_2 = 300$. Hence, in this example, we usually see $[S_{i_{300}} = 1]$, however, only a small fraction of nodes with $[S_i = 1]$ is selected (on a random basis) into the set $\{i_1, \ldots, i_{300}\}$.

Motivated by the above example, we suggest to approximate $P_j(n_1)$ in (7) by its first term $\mathrm{P}(S_j > S_{i_{n_2}})$.

Next, we employ the fact that S_{n_2} has the n_2-th highest average value among $S_1, \ldots, S_M$, and we suggest to use S_{n_2} as a proxy for the order statistic $S_{i_{n_2}}$. However, we cannot replace $\mathrm{P}(S_j > S_{i_{n_2}})$ directly by $\mathrm{P}(S_j > S_{n_2})$ because the latter includes the case $[S_j > S_{n_2} = 0]$, while with a reasonable choice of parameters it is unlikely to observe $[S_{i_{n_2}} = 0]$. This is not negligible as, e.g., in Example 1 we have $\mathrm{P}(S_{n_2} = 0) \approx 0.06$. Hence, we propose to approximate $\mathrm{P}(S_j > S_{i_{n_2}})$ by $P(S_j > \max\{S_{n_2}, 1\})$, $j = 1, \ldots, n_2$.

As the last simplification, we approximate the binomial random variables S_j's by independent Poisson random variables. The Poisson approximation is justified because even for $j = 1, \ldots, k$ the value F_j/N is small enough. For instance, in Example 1 we have $F_1/N \approx 0.04$, so $n_1 F_1/N$ is $700 \cdot 0.04 = 28$.

Thus, summarizing the above considerations, we propose to replace $P_j(n_1)$ in (3) and (4) by

$$\hat{P}_j(n_1) = \mathrm{P}(\hat{S}_j > \max\{\hat{S}_{n_2}, 1\}), \; j = 1, \ldots, n_2, \qquad (8)$$

where $\hat{S}_1, \ldots, \hat{S}_{n_2}$ are independent Poisson random variables with parameters $n_1 F_1/N, \ldots, n_1 F_{n_2}/N$. We call this method a Poisson prediction for (3) and (4).

On Figures 2 and 5 the results of the Poisson prediction are shown by the green line. We see that these predictions closely follow the experimental results (red line).

B. EVT predictions

Denote by $\hat{F}_1 > \hat{F}_2 > \cdots > \hat{F}_k$ the top-k values obtained by the algorithm.

Assume that the actual in-degrees in W are randomly sampled from the distribution G that satisfies (1). Then $F_1 > F_2 > \cdots > F_M$ are the order statistics of G. The EVT techniques allow to predict high quantiles of G using the top values of F_i's [14]. However, since the correct values of F_i's are not known, we instead use the obtained top-m values $\hat{F}_1, \hat{F}_2, \ldots, \hat{F}_m$, where m is much smaller than k. This is justified for two reasons. First, given F_j, $j < k$, the estimate $\hat{F}_j$ converges to F_j almost surely as $n_1 \to \infty$, because, in the limit, the degrees can be ordered correctly using S_i's only according to the strong law of large numbers. Second, when m is small, the top-m list can be found with high precision even when n is very modest. For example, as we saw on Figure 1, we find 50 the most followed Twitter users with very high precision using only 1000 API requests.

Our goal is to estimate $\hat{P}_j(n_1)$, $j = 1, \ldots, k$, using only the values $\hat{F}_1, \ldots, \hat{F}_m$, $m < k$. To this end, we suggest to first estimate the value of γ using the classical Hill's estimator $\hat{\gamma}$ [19] based on the top-m order statistics:

$$\hat{\gamma} = \frac{1}{m-1} \sum_{i=1}^{m-1} (\log(\hat{F}_i) - \log(\hat{F}_m)). \tag{9}$$

Next, we use the quantile estimator, given by formula (4.3) in [14], but we replace their two-moment estimator by the Hill's estimator in (9). This is possible because both estimators are consistent (under slightly different conditions). Under the assumption $\gamma > 0$, we have the following estimator $\hat{f}_j$ for the $(j-1)/M$-th quantile of G:

$$\hat{f}_j = \hat{F}_m \left(\frac{m}{j-1}\right)^{\hat{\gamma}}, \qquad j > 1, j << M. \tag{10}$$

We propose to use $\hat{f}_j$ as a prediction of the correct values F_j, $j = m+1, \ldots, n_2$.

Summarising the above, we suggest the following prediction procedure, which we call EVT prediction.

1) Use Algorithm 1 to find the top-m list, $m << k$.
2) Substitute the identified m highest degrees $\hat{F}_1, \hat{F}_2, \ldots, \hat{F}_m$ in (9) and (10) in order to compute, respectively, $\hat{\gamma}$ and $\hat{f}_j$, $j = m+1, \ldots, n_2$.
3) Use the Poisson prediction (8) substituting the values $F_1, \ldots, F_{n_2}$ by $\hat{F}_1, \ldots, \hat{F}_m$, $\hat{f}_{m+1}, \ldots, \hat{f}_{n_2}$.

On Figures 5 and 2 the blue lines represent the EVT predictions, with $k = 100$, $m = 20$ and different values of n_2. For the average fraction of correctly identified nodes, depicted on Figure 5, we see that the EVT prediction is very close to the Poisson prediction and the experimental results. The predictions for the first error index on Figure 2 are less accurate but the shape of the curve and the optimal value of n_2 is captured correctly by both predictors. Note that the EVT prediction tends to underestimate the performance of the algorithm for a large range of parameters. This is because in Twitter the highest degrees are closer to each other than the order statistics of a regularly varying distribution would normally be, which results in an underestimation of γ in (9) if only a few top-degrees are used.

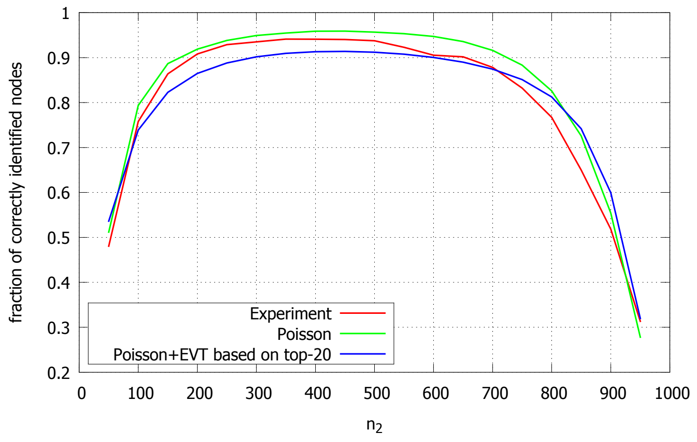

Fig. 5. Fraction of correctly identified nodes out of top-100 most followed users in Twitter as a function of n_2, with $n = 1000$.

Note that the estimation (10) is inspired but not entirely justified by [14] because the consistency of the proposed quantile estimator (10) is only proved for $j < m$, while we want to use it for $j > m$. However, we see that this estimator agrees well with the data.

VIII. Optimal scaling for algorithm parameters

In this section, our goal is to find the ratio n_2 to n_1 which maximizes the performance of Algorithm 1. For simplicity, as a performance criterion we consider the expected fraction of correctly identified nodes from the top-k list (see Equation (3)):

$$\underset{n_1, n_2 : n_1 + n_2 = n}{\text{maximize}} \; \frac{1}{k} \sum_{j=1}^{k} P_j(n_1).$$

We start with stating the optimal scaling for n_1. Let us consider the number of nodes with $S_j > 0$ after the first stage of the algorithm. Assuming that the out-degrees of randomly chosen nodes in V are independent, by the strong law of large numbers we have

$$\limsup_{n_1 \to \infty} \frac{1}{n_1} \sum_{j=1}^{M} I\{S_j > 0\} \leqslant \mu \quad \text{with probability } 1,$$

where μ is the average out-degree in V and $I\{A\}$ is an indicator of the event A. Thus, there is no need to check more than $n_2 = O(n_1)$ nodes on the second stage, which directly implies the next proposition.

Proposition 1. *It is optimal to choose n_1 such that $n = O(n_1)$.*

As we noted before (see, e.g., Figure 1), for small k the algorithm has a high precision in a large range of parameters. However, for not too small values of k, the optimization becomes important. In particular, we want to maximize the value $P_k(n_1)$. We prove the following theorem.

Theorem 1. *Assume that $k = o(n)$ as $n \to \infty$, then the maximizer of the probability $P_k(n - n_2)$ is*

$$n_2 = (3\gamma k^\gamma n)^{\frac{1}{\gamma+1}} (1 + o(1)),$$

with γ as in (1).

Proof: It follows from Proposition 1 that $n_1 \to \infty$ as $n \to \infty$, so we can apply the following normal approximation

$$P_k(n_1) \approx P\left(N\left(\frac{n_1(F_k - F_{n_2})}{N}, \frac{n_1(F_k + F_{n_2})}{N}\right) > 0\right)$$
$$= P\left(N(0,1) > -\sqrt{\frac{n_1}{N}}\frac{F_k - F_{n_2}}{\sqrt{F_k + F_{n_2}}}\right). \quad (11)$$

The validity of the normal approximation follows from the Berry-Esseen theorem. In order to maximize the above probability, we need to maximize $\sqrt{\frac{n_1}{N}}\frac{F_k - F_{n_2}}{\sqrt{F_k + F_{n_2}}}$. It follows from EVT that F_k decays as $k^{-\gamma}$. So, we can maximize

$$\frac{\sqrt{n - n_2}\left(k^{-\gamma} - n_2^{-\gamma}\right)}{\sqrt{k^{-\gamma} + n_2^{-\gamma}}}. \quad (12)$$

Now if $n_2 = \mathrm{O}(k)$, then $\sqrt{n - n_2} = \sqrt{n}(1 + o(1))$ and the maximization of (12) mainly depends on the remaining term in the product, which is an increasing function of n_2. This suggests that n_2 has to be chosen considerably greater than k. Also note that it is optimal to choose $n_2 = o(n)$ since only in this case the main term in (12) amounts to $\sqrt{n}$. Hence, we proceed assuming the only interesting asymptotic regime where $k = o(n_2)$ and $n_2 = o(n)$. In this asymptotic regime, we can simplify (12) as follows:

$$\frac{\sqrt{n - n_2}\left(k^{-\gamma} - n_2^{-\gamma}\right)}{\sqrt{k^{-\gamma} + n_2^{-\gamma}}} =$$

$$\frac{1}{k^{\gamma/2}}\sqrt{n - n_2}\left(1 - \frac{3}{2}\left(\frac{k}{n_2}\right)^\gamma + \mathrm{O}\left(\left(\frac{k}{n_2}\right)^{2\gamma}\right)\right).$$

Next, we differentiate the function

$$f(n_2) := \sqrt{n - n_2}\left(1 - \frac{3}{2}\left(\frac{k}{n_2}\right)^\gamma\right)$$

and set the derivative to zero. This results in the following equation:

$$\frac{1}{3\gamma k^\gamma} n_2^{\gamma+1} + n_2 - n = 0. \quad (13)$$

Since $n_2 = \mathrm{o}(n)$, then only the highest order term remains in (13) and we immediately obtain the following approximation

$$n_2 = (3\gamma k^\gamma n)^{\frac{1}{\gamma+1}} (1 + \mathrm{o}(1)).$$

■

IX. Sublinear complexity

The normal approximation (11) implies the following proposition.

Proposition 2. *For large enough n_1, the inequality*

$$Z_k(n_1) := \sqrt{\frac{n_1}{N}}\frac{F_k - F_{n_2}}{\sqrt{F_k + F_{n_2}}} \geqslant z_{1-\varepsilon}, \quad (14)$$

where $z_{1-\varepsilon}$ is the $(1-\varepsilon)$-quantile of a standard normal distribution, guarantees that the mean fraction of top-k nodes in W identified by Algorithm 1 is at least $1 - \varepsilon$.

Using (10), the estimated lower bound for n_1 in (14) is:

$$n_1 \geqslant \frac{N z_{1-\varepsilon}^2 (k^{-\hat{\gamma}} + n_2^{-\hat{\gamma}})}{\hat{F}_m m^{\hat{\gamma}} (k^{-\hat{\gamma}} - n_2^{-\hat{\gamma}})^2}. \quad (15)$$

In the case of the Twitter graph with $N = 10^9$, $m = 20$, $\hat{F}_{20} = 18,825,829$, $k = 100$, $n_2 = 300$, $z_{0.9} \approx 1.28$, $\hat{\gamma} = 0.4510$, this will result in $n_1 \geqslant 1302$, which is more pessimistic than $n_1 = 700$ but is sufficiently close to reality. Note that Proposition 2 is expected to provide a pessimistic estimator for n_1, since it uses the k-th highest degree, which is much smaller than, e.g., the first or the second highest degree.

We will now express the complexity of our algorithm in terms of M and N, assuming that the degrees in W follow a regularly varying distribution G defined in (1). In a special case, when our goal is to find the highest in-degree nodes in a directed graph, we have $N = M$. If M is, e.g., the number of interest groups, then it is natural to assume that M scales with N and $M \to \infty$ as $N \to \infty$. Our results specify the role of N, M, and G in the complexity of Algorithm 1.

From (15) we can already anticipate that n is of the order smaller than N because F_m grows with M. This argument is formalized in Theorem 2 below.

Theorem 2. *Let the in-degrees of the entities in W be independent realizations of a regularly varying distribution G with exponent $1/\gamma$ as defined in (1), and $F_1 \geqslant F_2 \geqslant \cdots \geqslant F_M$ be their order statistics. Then for any fixed $\varepsilon, \delta > 0$, Algorithm 1 finds the fraction $1 - \varepsilon$ of top-k nodes with probability $1 - \delta$ in*

$$n = O(N/a(M))$$

API requests, as $M, N \to \infty$, where $a(M) = l(M)M^\gamma$ and $l(\cdot)$ is some slowly varying function.

Proof: Let $a(\cdot)$ be a left-continuous inverse function of $1/(1 - G(x))$. Then $a(\cdot)$ is a regularly varying function with index γ (see, e.g., [7]), that is, $a(y) = l(y)y^\gamma$ for some slowly varying function $l(\cdot)$. Furthermore, repeating verbatim the proof of Theorem 2.1.1 in [13], we obtain that for a fixed m

$$\left(\frac{F_1}{a(M)}, \cdots, \frac{F_m}{a(M)}\right) \xrightarrow{d} \left(E_1^{-\gamma}, \cdots, (E_1 + \cdots + E_m)^{-\gamma}\right),$$

where E_i are independent exponential random variables with mean 1 and $\xrightarrow{d}$ denotes the convergence in distribution. Now for fixed k, choose n_2 as in Theorem 1. It follows that if $n_1 = CN/a(M)$ for some constant $C > 0$ then $Z_k(n_1) \xrightarrow{d} \sqrt{C(E_1 + \cdots + E_k)^{-\gamma}}$ as $M, N \to \infty$. Hence, we can choose

C, M, N large enough so that $P(Z_k(n_1) > z_{1-\varepsilon}) > 1-\delta$. We conclude that $n_1 = \mathrm{O}(N/a(M))$ for fixed k, as $N, M \to \infty$. Together with Proposition 1, this gives the result. ■

In the case $M = N$, as in our experiments on Twitter, Theorem 2 states that the complexity of the algorithm is roughly of the order $N^{1-\gamma}$, which is much smaller than linear in realistic networks, where we often observe $\gamma \in (0.3, 1)$ [23]. The slowly varying term $l(N)$ does not have much effect since it grows slower than any power of N. In particular, if G is a pure Pareto distribution, $1 - G(x) = Cx^{-1/\gamma}$, $x \geqslant x_0$, then $a(N) = C^\gamma N^\gamma$.

X. Conclusion

In this paper, we proposed a randomized algorithm for quick detection of popular entities in large online social networks whose architecture has underlying directed graphs. Examples of social network entities are users, interest groups, user categories, etc. We analyzed the algorithm with respect to two performance criteria and compared it with several baseline methods. Our analysis demonstrates that the algorithm has sublinear complexity on networks with heavy-tailed in-degree distribution and that the performance of the algorithm is robust with respect to the values of its few parameters. Our algorithm significantly outperforms the baseline methods and has much wider applicability.

An important ingredient of our theoretical analysis is the substantial use of the extreme value theory. The extreme value theory is not so widely used in computer science and sociology but appears to be a very useful tool in the analysis of social networks. We feel that our work could provide a good motivation for wider applications of EVT in social network analysis. We validated our theoretical results on two very large online social networks by detecting the most popular users and interest groups.

We see several extensions of the present work. A top list of popular entities is just one type of properties of social networks. We expect that both our theoretical analysis, which is based on the extreme value theory, and our two-stage randomized algorithm can be extended to infer and to analyze other properties such as the power law index and the tail, network functions and network motifs, degree-degree correlations, etc. It would be very interesting and useful to develop quick and effective statistical tests to check for the network assortativity and the presence of heavy tails.

Since our approach requires very small number of API requests, we believe that it can be used for tracing network changes. Of course, we need formal and empirical justifications of the algorithm applicability for dynamic networks.

Acknowledgment

This work is partially supported by the EU-FET Open grant NADINE (288956) and CONGAS EU project FP7-ICT-2011-8-317672.

References

[1] S. Abiteboul, M. Preda, and G. Cobena. Adaptive on-line page importance computation. *Proceedings of the 12-th International World Wide Web Conference*, 2003.

[2] K. Avrachenkov, N. Litvak, D. Nemirovsky, and N. Osipova. Monte carlo methods in pagerank computation: When one iteration is sufficient. *SIAM J. Numer. Anal.*, 45(2):890–904, 2007.

[3] K. Avrachenkov, N. Litvak, D. Nemirovsky, E. Smirnova, and M. Sokol. Quick detection of top-k personalized pagerank lists. In *Proc. 8th Workshop on Algorithms and Models for the Web Graph*, pages 50–61. 2011.

[4] K. Avrachenkov, N. Litvak, M. Sokol, and D. Towsley. Quick detection of nodes with large degrees. In *Proc. 9th Workshop on Algorithms and Models for the Web Graph*, pages 54–65. 2012, Extended version appears in *Internet Mathematics*, v.10(1-2), 2014.

[5] K. Avrachenkov, B. Ribeiro, and D. Towsley. Improving random walk estimation accuracy with uniform restarts. In *Proc. 7th Workshop on Algorithms and Models for the Web Graph*, pages 98–109. 2010.

[6] B. Bahmani, A. Chowdhury, and A. Goel. Fast incremental and personalized pagerank. *Proceedings of the VLDB Endowment*, 4(3):173–184, 2010.

[7] N. H. Bingham, C. M. Goldie, and J. L. Teugels. *Regular variation*, volume 27 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, Cambridge, 1989.

[8] C. Borgs, M. Brautbar, J. Chayes, S. Khanna, and B. Lucier. The power of local information in social networks. In *Internet and Network Economics*, pages 406–419. Springer, 2012.

[9] C. Borgs, M. Brautbar, J. Chayes, and S.-H. Teng. A sublinear time algorithm for pagerank computations. *Lecture Notes in Computer Science*, 7323:41–53, 2012.

[10] C. Borgs, M. Brautbar, J. Chayes, and S.-H. Teng. Multiscale matrix sampling and sublinear-time pagerank computation. *Internet Mathematics*, 10(1-2):20–48, 2014.

[11] M. Brautbar and M. Kearns. Local algorithms for finding interesting individuals in large networks. *Proceeding on the Innovations in Computer Science, ICS 2010*, pages 188–199, 2010.

[12] C. Cooper, T. Radzik, and Y. Siantos. A fast algorithm to find all high degree vertices in power law graphs. In *Proceedings of the 21st international conference companion on World Wide Web*, pages 1007–1016. ACM, 2012.

[13] L. De Haan and A. Ferreira. *Extreme value theory*. Springer, 2006.

[14] A. L. M. Dekkers, J. H. J. Einmahl, and L. de Haan. A moment estimator for the index of an extreme-value distribution. *The Annals of Statistics*, 17(4):1833–1855, 1989.

[15] E. Fischer. The art of uninformed decisions: A primer to property testing. *Bull. Eur. Assoc. Theor. Comput. Sci. EATCS*, 75:97–126, 2001.

[16] D. Fogaras, B. Rácz, K. Csalogány, and T. Sarlsa. Towards scaling fully personalized pagerank: Algorithms, lower bounds, and experiments. *Internet Mathematics*, 2(3):333–358, 2005.

[17] M. Gjoka, M. Kurant, C. T. Butts, and A. Markopoulou. Walking in facebook: A case study of unbiased sampling of osns. *Proceedings of IEEE INFOCOM'10*, 2010.

[18] O. Goldreich. Combinatorial property testinga survey. *Randomization Methods in Algorithm Design, DIMACS Ser. Discrete Math. Theoret. Comput. Sci.*, pages 45–60, 1998.

[19] B. M. Hill. A simple general approach to inference about the tail of a distribution. *Ann. Statist.*, 3:1031–1188, 1975.

[20] R. Kumar, K. Lang, C. Marlow, and A. Tomkins. Efficient discovery of authoritative resources. *IEEE 24th International Conference on Data Engineering*, pages 1495–1497, 2008.

[21] J. Leskovec and C. Faloutsos. Sampling from large graphs. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 631–636. ACM, 2006.

[22] L. Massoulié, E. Le Merrer, A.-M. Kermarrec, and A. Ganesh. Peer counting and sampling in overlay networks: Random walk methods. In *Proc. of the PODC*, pages 123–132. ACM, 2006.

[23] M. Newman. *Networks: an introduction*. Oxford University Press, Inc., 2010.

[24] R. Rubinfeld and A. Shapira. Sublinear time algorithms. *SIAM J. Discrete Math.*, 25(4):1562–1588, 2011.

[25] M. Sudan. Invariance in property testing. *Property Testing: Current Research and Surveys, O. Goldreich, ed., Lecture Notes in Comput. Sci.*, pages 211–227, 2010.