

Random Walk Based Algorithms for Complex Network Analysis

Konstantin Avrachenkov
Inria Sophia Antipolis

Rescom 2014, 12-16 May, Furiani, Corse

Main features of complex networks:

- ▶ Sparse topology;
- ▶ Heavy-tail degree distribution;
- ▶ Small average distance;
- ▶ Many triangles.

Complex networks

Many complex networks are very large. For instance,

- ▶ The static part of the web graph has more than 10 billion pages. With an average number of 38 hyper-links per page, the total number of hyper-links is 380 billion.
- ▶ Twitter has more than 500 million users. On average a user follows about 100 other users. Thus, the number of "following"-type social relations is about 50 billion.

Complex network analysis

Often the topology of a complex network is not known or/and constantly changing.

And crawling networks is often subject to a limit on the number of requests per minute.

For instance, a standard Twitter account can make no more than one request per minute.

With this rate, we would crawl the entire Twitter social network in 950 years...

Complex network analysis

Thus, for the analysis of complex networks, it is just essential to use methods with **linear** or even **sub-linear** complexity.

Complex network analysis

In this tutorial we answer the following questions:

- ▶ How to estimate quickly the size of a large network?
- ▶ How to count the number of network motifs?
- ▶ How to detect quickly most central nodes?
- ▶ How to partition network in clusters/communities?

And we answer these questions by
random walk based methods with low complexity.

How to estimate quickly the number of nodes?

Suppose that we can only crawl the network.

And we would like to estimate quickly the total number of nodes in the network.

The first element of our method is
[the inverse birthday paradox](#).

How to estimate quickly the number of nodes?

In a class of 23 students, the probability of having at least one pair of students with the same birthday is more than 50%!

A closely related the [inverse birthday paradox](#) says:

If we sample repeatedly with replacement, [independently and uniformly](#), from a population of size n , the number of trials required for the first repetition has expectation $\sqrt{2n}$ and variance $\Theta(\sqrt{n})$.

How to estimate quickly the number of nodes?

Let L be the number of node samples until a repetition occurs. Then, an obvious estimator of the network size is just

$$\hat{n} = \frac{L^2}{2}.$$

Since the variance is quite high, we need to perform and average several experiments.

Theorem

Denote by k the number of samples and let $\hat{n}_k = 1/k \sum_{i=1}^k L_i^2/2$. Then, the relative error $|\hat{n}_k - n|/n$ is less than ε with high probability if we take $\Theta(1/\varepsilon^2)$ samples.

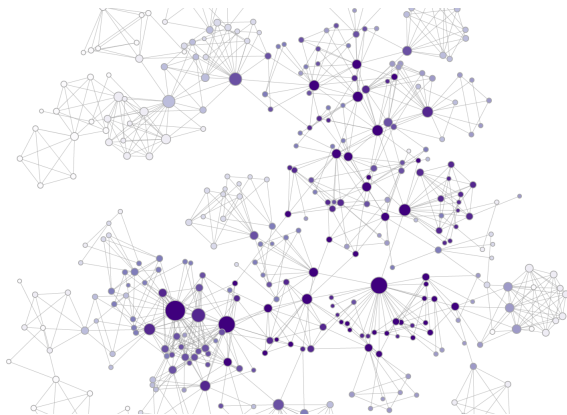
How to estimate quickly the number of nodes?

In many complex networks, generating samples from the uniform distribution is costly or even infeasible.

To obtain a sample, which is very close to the uniformly random, we can use either discrete-time or continuous-time random walks.

How to estimate quickly the number of nodes?

Let us first consider the discrete-time random walk.



How to estimate quickly the number of nodes?

Denote by d_i the degree of node i . Then, the stationary distribution of the random walk is given by

$$\pi_i = P\{S_t = i\} = \frac{d_i}{2m},$$

where m is the number of links.

We can unbiased the RW sampling by retaining a sample with probability $1/d_i$.

How to estimate quickly the number of nodes?

Alternatively, we can use a continuous time random walk also choosing uniformly from the list of neighbours and waiting an exponentially distributed time with the mean duration of $1/d_i$.

In such a case, the stationary distribution is described by the differential equation

$$\dot{\pi}(t) = \pi(t)(A - D),$$

where $D = \text{diag}\{d_i\}$ and A is the adjacency matrix

$$A_{ij} = \begin{cases} 1, & \text{if } (i, j) \in E, \\ 0, & \text{otherwise.} \end{cases}$$

How to estimate quickly the number of nodes?

For two distributions p and q , let $d(p, q)$ denotes the **total variation distance**:

$$d(p, q) = \frac{1}{2} \sum_{i=1}^n |p_i - q_i|.$$

The next interpretation is useful: A random sample from distribution p coincides with a random sample from distribution q with probability $1 - d(p, q)$.

How to estimate quickly the number of nodes?

Theorem

Let $\lambda_2 = \min\{\lambda : (D - A)x = \lambda x \text{ \& } \lambda > 0\}$ and let $\pi_i(t)$ be the distribution of the continuous-time random walk when the process starts at node i . Then, we have

$$d(\pi_i(t), \pi) \leq \frac{1}{2\sqrt{\pi_i}} e^{-\lambda_2 t},$$

where π is the stationary distribution.

In our case, $\pi_i = 1/n$. Next, taking $t = 3/2 \log(n)/\lambda_2$ we obtain

$$d(\pi_i(t), \pi) \leq \frac{1}{2n}.$$

How to estimate quickly the number of nodes?

Thus, we can conclude that the complexity of the continuous-time random walk method on expander-type networks is $O(\sqrt{n} \log(n))$, which is **sub-linear complexity**.

How to estimate quickly the number of links?

To estimate the number of edges, we take a different point of view on the random walk.

Consider the first return time to node i

$$T_i^+ = \min\{t > 0 : S_t = i \text{ \& } S_0 = i\}.$$

The expected value of the first return time is given by

$$E[T_i^+] = \frac{1}{\pi_i} = \frac{2m}{d_i}.$$

How to estimate quickly the number of links?

Let $R_k = \sum_{j=1}^k T_k$ be the time of the k -th return to node i . Then, we can use the following estimator for the number of links

$$\hat{m} = \frac{R_k d_i}{2k}.$$

To estimate the required complexity, we need to have an idea about the variance of T_i^+ . We can use the following formula

$$\text{Var}[T_i^+] = E[(T_i^+)^2] - (E[T_i^+])^2 = \frac{2Z_{ii} + \pi_i}{\pi_i^2} - \frac{1}{\pi_i^2}$$

with

$$Z_{ii} = \sum_{t=0}^{\infty} (P\{S_t = i | S_0 = i\} - \pi_i).$$

How to estimate quickly the number of links?

Next, we note that

$$Z_{ii} = \sum_{t=0}^{\infty} (P\{S_t = i | S_0 = i\} - \pi_i) \leq \sum_{t=0}^{\infty} |P\{S_t = i | S_0 = i\} - \pi_i|$$

and using $|P\{S_t = i | S_0 = i\} - \pi_i| \leq \tilde{\lambda}_2^t$, we obtain

$$Z_{ii} \leq \frac{1}{1 - \tilde{\lambda}_2},$$

and hence,

$$\text{Var}[T_i^+] \lesssim \frac{2}{(1 - \tilde{\lambda}_2)\pi_i^2}$$

or, in our context,

$$\text{Var}[T_i^+] \lesssim \frac{8m^2}{(1 - \tilde{\lambda}_2)d_i^2}.$$

Twitter as example

The screenshot shows a web browser displaying a page titled "The 5 most popular Twitter users". The page lists the following data:

Rank	User Name	Followers	Following	Tweets
1	Katy Perry Twitter Stats (@katyperry)	49,887,267	130	5,360
2	Justin Bieber Twitter Stats (@jbs Bieber)	48,102,270	123,687	25,951
3	Barack Obama Twitter Stats (@BarackObama)	41,162,919	653,879	10,806
4	GodBless of Love Twitter Stats (@gblgblgbl)	41,030,092	135,128	4,321
5	YouTube Twitter Stats (@YouTube)	38,779,436	568	9,654

Below the table is a button labeled "Show top 100 users". At the bottom of the page, there is a section titled "Check your own Twitter Stats for Free!" with a "Sign in with Twitter" button. The footer contains navigation links for "TWITTER COUNTER", "FREE TOOLS", "PAID TOOLS", "MY TWITTER COUNTER", and "OTHER SERVICES BY US".

Twitter as example

Assuming that a rough estimation of the number of users is $500 \cdot 10^6$ and the average number of followers per user is 10, the expected return time from the nodes like “Katy Perry” or “Justin Bieber” is about $2 \cdot 10 \cdot 500 \cdot 10^6 / 50 \cdot 10^6 = 200$.

To obtain a decent error ($\leq 5\%$), we need about 1000 samples, and hence in total about 200000 operations. This is **orders of magnitude less** than the size of the Twitter follower graph!

How to estimate quickly the number of triangles?

To evaluate the degree of clustering in a network, we need to estimate the number of triangle.

Towards this goal, we consider a random walk on weighted network where for each link (i, j) we assign a weight $1 + t(i, j)$, with $t(i, j)$ being the number of triangles containing (i, j) .

The stationary distribution of the random walk on such weighted network is given by

$$\pi_i = \frac{d_i + \sum_{j \in N(i)} t(i, j)}{2m + 6t(G)}.$$

How to estimate quickly the number of triangles?

Thus, if $R_k = \sum_{j=1}^k T_k$ is the time of the k -th return to node i , we can use the following estimator

$$\hat{t}(G) = \max \left\{ 0, \frac{(d_i + \sum_{j \in N(i)} t(i, j)) R_k}{6k} - \frac{m}{3} \right\},$$

where m is the number of links which we already know how to estimate.

Example of the Web graph with 855802 nodes, 5066842 links and 31356298 triangles: Starting from the node with 53371 triangles, the expected return time is 1753. For a good accuracy it was needed to make about 100 returns.

Quick detection of top-k largest degree nodes

What if we would like to find quickly in a network
top-k nodes with largest degrees?

Some applications:

- ▶ Routing via large degree nodes
- ▶ Finding influential users in OSN
- ▶ Proxy for various centrality measures
- ▶ Node clustering and classification
- ▶ Epidemic processes on networks

Top-k largest degree nodes

Even IF the adjacency list of the network is known...

the top-k list of nodes can be found by the HeapSort with complexity $O(n + k \log(n))$, where n is the total number of nodes.

Even this modest complexity can be quite demanding for large networks (i.e., 950 years for Twitter graph).

Random walk approach

Let us again try a random walk approach.

We actually recommend the **random walk with jumps** with the following transition probabilities:

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

where d_i is the degree of node i and α is a parameter.

Random walk approach

This modification can again be viewed as a random walk on weighted graph.

Since the weight of link is $1 + \alpha/n$, the stationary distribution of the random walk is given by a simple formula

$$\pi_i(\alpha) = \frac{d_i + \alpha}{2|E| + n\alpha} \quad \forall i \in V. \quad (2)$$

Random walk approach

Example:

If we run a random walk on the web graph of the UK domain (about 18 500 000 nodes), the random walk spends on average only about 5 800 steps to detect the largest degree node.

Three order of magnitude faster than HeapSort!

Random walk approach

We propose the following algorithm for detecting the top k list of largest degree nodes:

1. Set k , α and m .
2. Execute a random walk step according to (1). If it is the first step, start from the uniform distribution.
3. Check if the current node has a larger degree than one of the nodes in the current top k **candidate list**. If it is the case, insert the new node in the top- k candidate list and remove the worst node out of the list.
4. If the number of random walk steps is less than m , return to Step 2 of the algorithm. Stop, otherwise.

Random walk approach

Let us investigate how the performance of the algorithm depends on parameters α and m .

Let us first discuss the choice of α .

The choice of α

We calculate

$$\begin{aligned} P_\pi[W_t = i | \text{jump}] &= \frac{P_\pi[W_t = i, \text{jump}]}{P_\pi[\text{jump}]} \\ &= \frac{P_\pi[W_t = i]P_\pi[\text{jump} | W_t = i]}{\sum_{j=1}^n P_\pi[W_t = j]P_\pi[\text{jump} | W_t = j]} = \frac{\frac{d_i + \alpha}{2|E| + n\alpha} \frac{\alpha}{d_i + \alpha}}{\sum_{j=1}^n \frac{d_j + \alpha}{2|E| + n\alpha} \frac{\alpha}{d_j + \alpha}} = \frac{1}{n}, \end{aligned}$$

and, similarly,

$$P_\pi[W_t = i | \text{no jump}] = \frac{d_i}{2|E|} = \pi_i(0), \quad i = 1, 2, \dots, n.$$

The choice of α

There is a trade off for α : we would like to maximize the long-run fraction of independent observations from $\pi(0)$.

To this end, we note that given m' cycles, the mean total number of steps is

$$m' E[\text{cycle length}] = m' (P_\pi[\text{jump}])^{-1}.$$

On average $m' P_\pi[\text{jump}]$ observations coincide with a jump.

$$\frac{m' - m' P_\pi[\text{jump}]}{m' (P_\pi[\text{jump}])^{-1}} = P_\pi[\text{jump}] (1 - P_\pi[\text{jump}]) \rightarrow \max.$$

Obviously, the maximum is achieved when

$$P_\pi[\text{jump}] = \frac{1}{2}.$$

The choice of α

It remains to rewrite $P_\pi[\text{jump}]$ in terms of the algorithm parameters:

$$\begin{aligned} P_\pi[\text{jump}] &= \sum_{j=1}^n P_\pi[W_t = j] P_\pi[\text{jump} | W_t = j] \\ &= \sum_{j=1}^n \frac{d_j + \alpha}{2|E| + n\alpha} \frac{\alpha}{d_j + \alpha} = \frac{n\alpha}{2|E| + n\alpha} = \frac{\alpha}{\bar{d} + \alpha}, \end{aligned} \quad (3)$$

where $\bar{d} := 2|E|/n$ is the average degree.

For the maximal efficiency, the last fraction above must be equal to $1/2$, which gives the optimal value for parameter α

$$\alpha_* = \bar{d}.$$

The choice of m

Let us now discuss the choice of m .

We note that once one of the k nodes with the largest degrees appears in the [candidate list](#), it remains there subsequently.

Thus, we are interested in the [hitting events](#).

Theorem (Adaptation from B. Bollobás)

Let H_1, \dots, H_k denote the hitting times to the top- k nodes with the largest degrees ($d_1 \geq \dots \geq d_k \geq d_{k+1} \geq \dots$). Then, the expected time, $E_u[\tilde{H}]$, for the random walk with transition probabilities (1) and starting from the uniform distribution to detect a fraction β of top- k nodes is bounded by

$$E_u[\tilde{H}] \leq \frac{1}{1 - \beta} E_u[H_k]. \quad (4)$$

The choice of m


Under reasonable technical assumption, we can show that

$$E_u[H_k] \lesssim \frac{1}{\pi_k(\alpha)} = \frac{2|E| + n\alpha}{d_k + \alpha}. \quad (5)$$

In particular, choosing $\alpha = \bar{d}$ in (5) yields

$$E_u[H_k] \lesssim \frac{2\bar{d}n}{d_k + \bar{d}}. \quad (6)$$

Example: From (4) and (5), we have for the Twitter network

$$E_u[\text{time to hit 70\% of top-100 nodes}] \leq \frac{1}{1 - \beta} \frac{2\bar{d}n}{d_{100} + \bar{d}} = 18\text{days}$$


Sublinear complexity for configuration model

Consider a configuration random graph model with power law degree distribution.

We assume that the node degrees D_1, \dots, D_n are i.i.d. random variables with a power law distribution F and finite expectation $E[D]$. That is,

$$\bar{F}(x) = Cx^{-\gamma} \quad \text{for } x > x'. \quad (7)$$

In the configuration model, one can use the quantile $x_{(j-1)/n}$ to approximate the degree $D_{(j)}$ of the top- j node, $j = 2, \dots, k$:

$$D_{(j)} \approx C^{1/\gamma} (j-1)^{-1/\gamma} n^{1/\gamma}. \quad (8)$$

Sublinear complexity for configuration model

Combination of equation (8) and inequalities (4) and (5), and taking $\alpha = \bar{d}$, yields

$$E_u[\tilde{H}] \leq \frac{1}{1-\beta} \left(\frac{2E[D]n}{C^{1/\gamma}(k-1)^{-1/\gamma}n^{1/\gamma} + E[D]} \right) \sim \tilde{C}n^{\frac{\gamma-1}{\gamma}},$$

and consequently

$$E_u[\tilde{H}] = O(n^{\frac{\gamma-1}{\gamma}}),$$

which means that we can find a β fraction of top- k largest degree nodes in sublinear expected time in the configuration model.

Stopping rules

Suppose now that node i can be sampled independently with the stationary probability $\pi_i(0)$.

And let us estimate the probability of detecting correctly the top k list of nodes after m i.i.d. samples from (2).

Denote by X_i the number of hits at node i after m i.i.d. samples.

$$P[X_1 \geq 1, \dots, X_k \geq 1] = \sum_{i_1 \geq 1, \dots, i_k \geq 1} \frac{m!}{i_1! \cdots i_k! (m - i_1 - \dots - i_k)!} \pi_1^{i_1} \cdots \pi_k^{i_k} \left(1 - \sum_{i=1}^k \pi_i\right)^{m - i_1 - \dots - i_k}$$

Stopping rules

We propose to use the **Poissonization technique**.

Let $Y_j, j = 1, \dots, n$ be independent Poisson random variables with means $\pi_j m$.

It is convenient to work with the complementary event of not detecting correctly the top k list.

$$\begin{aligned} P[\{X_1 = 0\} \cup \dots \cup \{X_k = 0\}] &\leq 2P[\{Y_1 = 0\} \cup \dots \cup \{Y_k = 0\}] \\ &= 2(1 - P[\{Y_1 \geq 1\} \cap \dots \cap \{Y_k \geq 1\}]) = 2(1 - \prod_{j=1}^k P[\{Y_j \geq 1\}]) \\ &= 2(1 - \prod_{j=1}^k (1 - P[\{Y_j = 0\}])) = 2(1 - \prod_{j=1}^k (1 - e^{-m\pi_j})) =: a, \end{aligned}$$

Stopping rules

This can be used to design the stopping criteria for our random walk algorithm.

Let $\bar{a} \in (0, 1)$ be the admissible probability of an error in the top k list.

Now the idea is to stop the algorithm after m steps when the estimated value of a for the first time is lower than the critical number \bar{a} .

$$\hat{a}_m = 2\left(1 - \prod_{j=1}^k (1 - e^{-X_j})\right)$$

is the maximum likelihood estimator for a , so we would like to choose m such that $\hat{a}_m \leq \bar{a}$.

Stopping rules

The problem, however, is that we do not know which X_j 's are the realisations of the number of visits to the top k nodes.

Then let X_{j_1}, \dots, X_{j_k} be the number of hits to the current elements in the top k candidate list and consider the estimator

$$\hat{a}_{m,0} = 2\left(1 - \prod_{i=1}^k (1 - e^{-X_{j_i}})\right),$$

which is the maximum likelihood estimator of the quantity

$$2\left(1 - \prod_{i=1}^k (1 - e^{-m\pi_{j_i}})\right) \geq a.$$

Stopping rule: Stop at $m = m_0$, where

$$m_0 = \arg \min \{m : \hat{a}_{m,0} \leq \bar{a}\}.$$

Stopping rules

In the introduced stopping rule we have strived to detect **all nodes in the top k list**. This costs us a lot of steps of the random walk.

We can significantly gain in performance by following a generic “80/20 Pareto rule” that

80% of result can be achieved with 20% of effort.

Stopping rules

Let us calculate the expected number of top k elements observed in the candidate list up to trial m .

$$H_j = \begin{cases} 1, & \text{node } j \text{ has been observed at least once,} \\ 0, & \text{node } j \text{ has not been observed.} \end{cases}$$

Assuming we sample in i.i.d. fashion from the distribution (2), we can write

$$E\left[\sum_{j=1}^k H_j\right] = \sum_{j=1}^k E[H_j] = \sum_{j=1}^k P[X_j \geq 1] =$$

$$\sum_{j=1}^k (1 - P[X_j = 0]) = \sum_{j=1}^k (1 - (1 - \pi_j)^m). \quad (10)$$

Stopping rules

Here again we can use the Poisson approximation

$$E\left[\sum_{j=1}^k H_j\right] \approx \sum_{j=1}^k (1 - e^{-m\pi_j}).$$

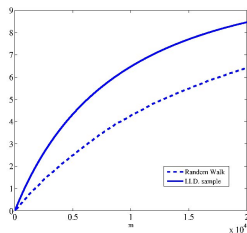
and propose stopping rule. Denote

$$b_m = \sum_{i=1}^k (1 - e^{-X_{ji}}).$$

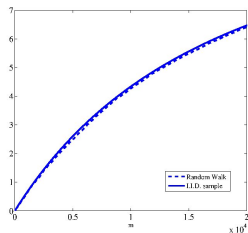
Stopping rule: Stop at $m = m_2$, where

$$m_2 = \arg \min\{m : b_m \geq \bar{b}\}.$$

Stopping rules



(a) $\alpha = 0.001$



(b) $\alpha = 28.6$

Figure: Average number of correctly detected elements in top-10 for UK.

References:

- ▶ Bawa, M., Garcia-Molina, H., Gionis, A., & Motwani, R. (2003). Estimating aggregates on a peer-to-peer network. Stanford Technical Report no.8090/586.
- ▶ Ganesh, A. J., Kermarrec, A. M., Le Merrer, E., & Massouli, L. (2007). Peer counting and sampling in overlay networks based on random walks. *Distributed Computing*, 20(4), 267-278.
- ▶ Cooper, C., Radzik, T., & Siantos, Y. (2013). Fast Low-Cost Estimation of Network Properties Using Random Walks. In *Algorithms and Models for the Web Graph, WAW 2013*, (pp. 130-143).
- ▶ Avrachenkov, K., Litvak, N., Sokol, M., & Towsley, D. (2012). Quick detection of nodes with large degrees. In *Algorithms and Models for the Web Graph, WAW 2012*, (pp. 54-65).

Thank you!

Any questions and suggestions are welcome.