Practical Sublinear Algorithms for Node Sampling in Large Networks

Omri Ben-Eliezer
MIT

Talya Eden
BU/MIT -> Bar Ilan U

Joel Oren
General Motors

Dimitris Fotakis
Natl Tech U Athens

Joint with:
The problem: Sampling multiple nodes

Start at single random node

Explore graph through **query access**: querying node reveals its **neighbors**

**Goal**: generate many random nodes with as few queries as possible
The problem: Sampling multiple nodes

Start at single random node

Explore graph through **query access**: querying node reveals its **neighbors**

**Goal**: generate many random nodes with as few queries as possible
The problem: Sampling multiple nodes

Start at single random node

Explore graph through **query access**: querying node reveals its **neighbors**

**Goal**: generate many random nodes with as few queries as possible
The problem: Sampling multiple nodes

Start at single random node

Explore graph through query access: querying node reveals its neighbors

Goal: generate many random nodes with as few queries as possible
The problem: Sampling multiple nodes

Start at single random node

Explore graph through query access: querying node reveals its neighbors

Goal: generate many random nodes with as few queries as possible:

For $\epsilon > 0$ and $k \ll n$, return random $S \in \binom{V}{k}$
where $\Pr(S) \leq \frac{1+\epsilon}{\binom{m}{k}}$ for all $S \in \binom{V}{k}$
Motivation

• Many algorithms (sublinear-time / property testing, data mining, ...) assume access to **random nodes**.
• Exploring many different “parts” of a large network with few queries.
• Queries supported in modern social network APIs.

Retrieved from Twitter API
Solution I: BFS

• This talk: real world graphs (social networks). But let us start with some theoretical observations.

Trivial solution: Query all nodes, $O(n)$ query complexity. Tight (in worst case) even for sampling a single node!
Solution II: Random walks
Solution II: Random walks

- Uniform random walk (+ rejection step) generates one node ($k = 1$) in $O(d_{avg} t_{mix} \cdot \log 1/\epsilon)$ queries [Chierichetti, Dasgupta, Kumar, Lattanzi, Sarlos ’16]

average degree  mixing time of uniform random walk
Solution II: Random walks

- Uniform random walk (+ rejection step) generates one node ($k = 1$) in $O(d_{avg} t_{mix} \cdot \log 1/\epsilon)$ queries [Chierichetti, Dasgupta, Kumar, Lattanzi, Sarlos ’16]

- Essentially optimal: $\Omega(d_{avg} t_{mix})$ lower bound (for some graphs) [Chierichetti, Haddadan ‘18]
Solution II: Random walks

- Uniform random walk (+ rejection step) generates one node \((k = 1)\) in \(O(d_{\text{avg}} t_{\text{mix}} \cdot \log 1/\epsilon)\) queries [Chierichetti, Dasgupta, Kumar, Lattanzi, Sarlos ’16]

- Essentially optimal: \(\Omega(d_{\text{avg}} t_{\text{mix}})\) lower bound (for some graphs) [Chierichetti, Haddadan ‘18]

- Not hard to show \(\Omega(k \cdot t_{\text{mix}})\) lower bounds for sampling \(k\) nodes, for wide classes of realistic graphs.
Solution II: Random walks

- Uniform random walk (+ rejection step) generates one node \((k = 1)\) in \(O(d_{avg} t_{mix} \cdot \log 1/\epsilon)\) queries [Chierichetti, Dasgupta, Kumar, Lattanzi, Sarlos ’16]

- Essentially optimal: \(\Omega(d_{avg} t_{mix})\) lower bound (for some graphs) [Chierichetti, Haddadan ‘18]

- Not hard to show \(\Omega(k \cdot t_{mix})\) lower bounds for sampling \(k\) nodes, for wide classes of realistic graphs.

Can we do better than \(O(k \cdot t_{mix})\) for large \(k\)?
Real-worlds social networks

- $t_{mix}$ can be pretty large: several 100’s or more [DR’09, MYK’10, QXZZ’20],
- Some small-world models have $\Theta(\log^2 n)$ mixing time, e.g., Newman-Watts [Dur’10, AL’12, KRS’15].

⇒ Issue: High amortized query complexity for random walk based algorithms!
Real-worlds social networks

- **Power law** degree distribution
- Highly expanding “core”, isolated “periphery” components [BE’99, LLDM ’09, RPFM’14, ZMN’15, BK’19, ...]

[Krebs-Holley, ‘06]

<table>
<thead>
<tr>
<th>PROFILE</th>
<th>FOLLOWERS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barack Obama</td>
<td>132,382,271</td>
</tr>
<tr>
<td>Justin Bieber</td>
<td>114,149,758</td>
</tr>
<tr>
<td>KATY PERRY</td>
<td>108,919,460</td>
</tr>
<tr>
<td>Rihanna</td>
<td>107,003,013</td>
</tr>
<tr>
<td>Cristiano Ronaldo</td>
<td>102,284,851</td>
</tr>
<tr>
<td>Elon Musk</td>
<td>102,170,738</td>
</tr>
</tbody>
</table>

[socialtracker.io]
Let’s use core-periphery structure!

Can you reach a random node in less than $t_{mix} = O(d)$ queries?
HEURISTICS AHEAD
SampLayer [BEFO’22]: New node sampling algorithm

• **Preprocessing**: Greedily search for “most influential” nodes in network, $L_0$.

• **Layering & Calibrating**: implicitly partition network into three layers: $L_0$, $L_1$, and the periphery $L_{\geq 2}$.

• **Sampling** by length 2 walks from $L_0$ to $L_{\geq 2}$ + local BFS in $L_{\geq 2}$ + rejection.
Phase 1: Greedy core construction

Starting from single node, construct $L_0$ by repeatedly adding node $v$ with highest “perceived degree” and querying $v$. 
Phase 2: Structural layering

$L_1$ : all neighbors of $L_0$,
$L_{\geq 2}$ : all other nodes in network.

**Key observation**: sublinear-sized $L_0$ can decompose $L_{\geq 2}$ into tiny components!
Phase 2: Structural layering

“Preparations” for sampling:
• Estimate $L_{\geq 2}$ size ($|L_0|$, $|L_1|$ known).
• Find a “reachability baseline” for $L_{\geq 2}$
  • Generated distribution will be uniform except for “low reachability” nodes.
Phase 3: Sampling

- Sampling from $L_0 \cup L_1$ straightforward.
- Sampling from $L_{\geq 2}$ by **length-2 walk** between $L_0$ and $L_{\geq 2}$, then **BFS** in reached $L_{\geq 2}$ component. Finally, **rejection step** to ensure uniform probabilities.
Empirical results: SampLayer vs random walks

• Sina Weibo [ZYLX’14], social network with \( \approx 60M \) nodes, 260M edges
Empirical results: SampLayer vs random walks

- Other social & information networks
Empirical results: SampLayer vs random walks

<table>
<thead>
<tr>
<th>Dataset</th>
<th>n</th>
<th>m</th>
<th>$d_{\text{avg}}$</th>
<th>$L_0$ size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>SL</td>
</tr>
<tr>
<td>Epinions [47]</td>
<td>76K</td>
<td>509K</td>
<td>13.4</td>
<td>3K</td>
</tr>
<tr>
<td>Slashdot [38]</td>
<td>82K</td>
<td>948K</td>
<td>23.1</td>
<td>3K</td>
</tr>
<tr>
<td>DBLP [56]</td>
<td>317K</td>
<td>1.05M</td>
<td>6.62</td>
<td>30K</td>
</tr>
<tr>
<td>Forest Fire [36, 37]</td>
<td>1M</td>
<td>6.75M</td>
<td>13.5</td>
<td>10K</td>
</tr>
<tr>
<td>Youtube [56]</td>
<td>1.1M</td>
<td>2.99M</td>
<td>5.27</td>
<td>30K</td>
</tr>
<tr>
<td>Pokec [54]</td>
<td>1.6M</td>
<td>30.6M</td>
<td>37.5</td>
<td>200K</td>
</tr>
<tr>
<td>SinaWeibo [58]</td>
<td>58.7M</td>
<td>261M</td>
<td>8.91</td>
<td>500K</td>
</tr>
</tbody>
</table>

Table 1: The list of networks we considered with numbers of nodes ($n$), edges ($m$), their average degrees ($d_{\text{avg}}$), and $L_0$ sizes we selected for SampLayer and SampLayer+. 
Empirical results: SampLayer vs random walks

- Forest Fire network model [LKF’05] with $p_f = 0.37, p_b = 0.3$
Why does it work?

• Algorithm provably converges to uniformity:

**Theorem 3.1.** If our size estimation for $L_{\geq 2}$ is in $(1 \pm o(1))|L_{\geq 2}|$, and if the baseline reachability $r_{s_0}$ used in our algorithm is the $o(1)$-percentile in the reachability distribution, then the output node distribution of SAMPLE is $o(1)$-close to uniform in total variation distance. Furthermore, the sampling probability of any node is at most $\frac{1+o(1)}{n}$. 
Why does it work?

**Key observation**: sublinear-sized $L_0$ can decompose $L_{\geq 2}$ into *tiny components*!
Why does it work?

- **Sublinear “almost domination”:** Most nodes with, say, (out-)degree \( \geq 10 \) have a neighbor in top 0.1%-1% highest degrees.

![Graph showing distribution of node degrees across various datasets](image)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>No. Nodes</th>
<th>No. Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epinions [39]</td>
<td>76K</td>
<td>509K</td>
</tr>
<tr>
<td>Slashdot [28]</td>
<td>82K</td>
<td>948K</td>
</tr>
<tr>
<td>DBLP [48]</td>
<td>317K</td>
<td>1.05M</td>
</tr>
<tr>
<td>Twitter-Higgs [18]</td>
<td>457K</td>
<td>14.9M</td>
</tr>
<tr>
<td>Youtube [48]</td>
<td>1.1M</td>
<td>2.99M</td>
</tr>
<tr>
<td>Pokec [45]</td>
<td>1.6M</td>
<td>30.6M</td>
</tr>
<tr>
<td>Wiki-topcats [49]</td>
<td>1.8M</td>
<td>28.5M</td>
</tr>
<tr>
<td>Orkut [48]</td>
<td>3.1M</td>
<td>117M</td>
</tr>
<tr>
<td>LiveJournal [48]</td>
<td>4.8M</td>
<td>69M</td>
</tr>
<tr>
<td>Twitter-MPI-SWS [9]</td>
<td>53M</td>
<td>2.0B</td>
</tr>
<tr>
<td><strong>Tumblr</strong></td>
<td><strong>247M</strong></td>
<td><strong>14.5B</strong></td>
</tr>
</tbody>
</table>
Why does it work?

• (Weak) theoretical **bounds on query complexity**:

\[ \text{Theorem 3.2. The expected query complexity of sampling a single node using} \text{ \textit{SAMPLAYER} is } O \left( c \cdot \left( \frac{1}{\alpha} + wd \right) \right). \]
Open Questions

• More explanations and applications for “sublinear almost domination”? [BLMPP’15, MSSK’13, NA’12]
• Efficient node sampling in the random walk query model? (e.g., [PS’21])
• Other practical algorithms based on core-periphery? [ASK’12, AIY’13, BK’19]
  • Also, better theoretical guarantees for our algorithm?
• Learning-augmented models for algorithms on large networks? (e.g., [CEILNRSWZ’22])

Thank you!