Scalable Community Detection
Benchmark Generation

Jonathan Berry¹
Siva Rajamanickam¹
Cynthia Phillips¹
George M. Slota²

¹Sandia National Labs, ²Rensselaer Polytechnic Institute
jberry@sandia.gov, caphill@sandia.gov, srajama@sandia.gov, slotag@rpi.edu

SIAM CSC 6 June 2018
Motivation

Better evaluate community detection algorithms processing $O$(Billion)-sized graphs on HPC resources

- Small-scale state-of-the-art: “LFR”
  - Lancichinetti, Fortunato, Radicchi, 2008
  - With $>1600$ citations, this is a de facto standard
  - Generates ground truth to test against
  - Has a tunable parameter for community coherence: $\mu$
  - Limited scalability: best implementation takes $\sim17$hrs to generate $O$(1B) edges (Hamann et al., 2017)

- Large-scale state-of-the-art
  - Without a reliable ground truth, parallel algorithms test with modularity or similar measures
  - This approach is flawed in several ways

Goal: evaluate at HPC scale against ground truth
Overview

Primary results of this work:

- We develop a novel method for generating large-scale graphs with a tunable ground truth community structure.
- We utilize the scalable BTER generator (Kolda et al., 2014) as a core step.
- Our approach generates large-scale community benchmarking graphs at a rate of 1B edge/minute on KNL.
  - Orders-of-magnitude faster than state-of-the-art.
Step 0: Input degree ($n_d$) and clustering coefficient ($c_d$) distributions

```
   d   1   2   3   4   5   6   7
n_d  9   5   4   2   2   1   1
   0   0.6  0.4  0.2  0.1  0.1  0.1
```
BTER
Block Two-level Erdös-Rényi Graph Generator

- Step 0: Input degree \( n_d \) and clustering coefficient \( c_d \) distributions
- Step 1: With ordered degree sequence, group \( d + 1 \) vertices \( v \) of degree \( d(v) \geq d \) into affinity blocks

<table>
<thead>
<tr>
<th>d</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n_d )</td>
<td>9</td>
<td>5</td>
<td>4</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>( c_d )</td>
<td>--</td>
<td>0.6</td>
<td>0.4</td>
<td>0.2</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
</tr>
</tbody>
</table>
BTER
Block Two-level Erdös-Rényi Graph Generator

- Step 0: Input degree \((n_d)\) and clustering coefficient \((c_d)\) distributions
- Step 1: With ordered degree sequence, group \(d + 1\) vertices \(v\) of degree \(d(v) \geq d\) into affinity blocks
BTER
Block Two-level Erdős-Rényi Graph Generator

- Step 0: Input degree \((n_d)\) and clustering coefficient \((c_d)\) distributions
- Step 1: With ordered degree sequence, group \(d + 1\) vertices \(v\) of degree \(d(v) \geq d\) into **affinity blocks**
- Step 2: Use Erdős-Rényi probability \(p_d = \frac{3}{\sqrt{c_d}}\) to create intra-block edges via \(G(n, p)\) process
BTER
Block Two-level Erdös-Rényi Graph Generator

- Step 0: Input degree \((n_d)\) and clustering coefficient \((c_d)\) distributions
- Step 1: With ordered degree sequence, group \(d + 1\) vertices \(v\) of degree \(d(v) \geq d\) into affinity blocks
- Step 2: Use Erdös-Rényi probability \(p_d = \frac{3}{\sqrt{c_d}}\) to create intra-block edges via \(G(n, p)\) process
- Step 3: Create inter-block edges via Chung-Lu process
How we wrap the baseline BTER process for generating graphs for community detection benchmarking:

- Treat affinity blocks as ground truth communities
- We have a native $\mu_n$, based on ratio of inter- to intra-block edges generated from the original distributions
- Can shift $\mu_n$ to some target goal $\mu_g$ via a Linear Program solve (to be described) – we use Pyomo and CBC
- Our BTER implementation: fully-parallelized in shared-memory with OpenMP/C++
Linear Program
Shifting the native $\mu$ of a graph’s CC distribution

Minimally shift the input clustering coefficient (CC) distribution such that the output graph has a desired goal $\mu_g$ considering both definitions:

$$\mu_g = \frac{1}{N} \sum_d d_{inter} \frac{d}{d} \quad \mu_g = \frac{1}{2M} \sum_d n_d d_{inter}$$

minimize $\sum_d |\hat{p}_d - p_d|$

subject to $\sum_d n_d \hat{p}_d = N(1 - \mu_g)$

$$\sum_d d n_d \hat{p}_d = 2M (1 - \mu_g)$$

$0 \leq \hat{p}_d \leq 1$

output $\hat{c}_d = \hat{p}_d^3$

- $p_d$ is $G(n, p)$ probabilities per degree from CC distribution $c_d$, $p_d = \sqrt[3]{c_d}$
- $\hat{p}_d$ is output probabilities to get new CC distribution $\hat{c}_d$, $\hat{c}_d = \hat{p}_d^3$
- $n_d$ is degree distribution, $n$ vertices of $d$ degree
- $d_{inter}$ is expected number of inter-community edges for vertex of degree $d$
- $N$ is number of vertices in graph, $M$ is number of edges
Experimental Setup
Test system and test graphs

**Test System**: *Bowman* at Sandia Labs – each node has a KNL with 68 cores, 96 GB DDR, and 16 GB MCDRAM

**Test Graphs**:

<table>
<thead>
<tr>
<th>Network</th>
<th>(n)</th>
<th>(m)</th>
<th>(d_{avg})</th>
<th>(d_{max})</th>
<th>(\tilde{D})</th>
</tr>
</thead>
<tbody>
<tr>
<td>LJ-fp</td>
<td>4.2 M</td>
<td>27 M</td>
<td>18</td>
<td>20 K</td>
<td>18</td>
</tr>
<tr>
<td>uk-2002</td>
<td>18 M</td>
<td>261 M</td>
<td>28</td>
<td>195 K</td>
<td>28</td>
</tr>
<tr>
<td>Wikilinks</td>
<td>26 M</td>
<td>332 M</td>
<td>23</td>
<td>39 K</td>
<td>170</td>
</tr>
<tr>
<td>RMAT_26</td>
<td>67 M</td>
<td>1.1 B</td>
<td>16</td>
<td>6.7 K</td>
<td>8</td>
</tr>
<tr>
<td>Friendster</td>
<td>66 M</td>
<td>1.8 B</td>
<td>27</td>
<td>5.2 K</td>
<td>34</td>
</tr>
</tbody>
</table>

Graphs are from the SNAP, Koblenz, and LAW databases. LiveJournal-fp is a parsed version of LiveJournal from SNAP.
Shifting Distribution
How the CC distribution shifts for varying $\mu$

- Only every $5^{th}$ value plotted for better visualization
- Generally, distribution is most “accurate” near native $\mu$
- Better smoothing of distribution via LP constraints is future work

![Graph showing distribution shifts for varying $\mu$. The x-axis represents vertex degree, and the y-axis represents clustering coefficient. Different markers and colors represent different values of $\mu$.](graph.png)
Hitting Target $\mu$
Accuracy of LP for generating desired $\mu$

- Generation accuracy is comparable to LFR
- Less than 5% error in most instances
- Error is greatest at lower $\mu$ targets
Strong scaling generally good up to 2 threads/core
- Time decreases with increasing $\mu$, due to *coupon collectors* edge generation scaling - higher CC requires more attempts for each edge
- Generation time a function of scale and complexity (max degree)
- Average $\sim 2$ minutes for 1.8B unique edges
  - Original BTER code: $\sim 4$ min. for 1.2B edges on 32 node Hadoop cluster
  - Fastest LFR implementation: 17 hours for 1B edges in shared-memory
A Note on BTER Assortativity

An issue with our approach so far is the degree homogeneity of communities.

We propose the following addition:

- Consider intra-comm edge count of each vertex
- Permute community assignments of all vertices with same count

**Observation:** won’t affect $\mu$, de-homogenizes communities in terms of degree

This approach might also be applied to baseline BTER generation.
Timing Breakdown
Full wBTER approach with community permutation, $\mu = 0.5$

- Time costs of major wBTER steps with community assignment permutation

**Work Complexity:**
\[ d = D_{max}, n = |V|, m = |E| \]

- LP: expected to scale as $O(d \log d)$
- EdgeGen: $O(m \log d)$
- Finalize: $O(n + m)$
- CSR: $O(n + m)$
- Swap: $O(n \log n + m)$
Conclusions
and future work

- We shift a graph’s CCD to fit a $\mu$ generated by BTER
- Our approach can output graphs for community detection order-of-magnitudes faster than commonly-used generators, e.g., LFR
- Our approach can output graphs with more realistic degree and CC distributions than commonly-used generators

Future Work:
- Better develop LP to reduce noise in output CC distribution
- Shift graph scale – i.e., output equivalent distributions for a graph with $2 \times, \frac{1}{2} \times$ original scale
- Develop generation methods for hierarchical communities

www.gmslota.com, slotag@rpi.edu