A Parallel LFR-like Benchmark for Evaluating Community Detection Algorithms

George M. Slota and Jack Garbus

Rensselaer Polytechnic Institute

ParSocial 2020
What is community detection?

Community Detection: Basic problem

- We have some real-world interaction network (e.g., Facebook)
- Community detection: identifying *clusters* within the network
- Why: communities are often homogeneous (like-attracts-like) so we can often infer information about community members.
Given some community detection algorithm, how can we determine the quality of its output?

- **Ideally**: Evaluate on real-world datasets with “known” communities
  - Very few such datasets exist, none at HPC/real-world social network scale
- **Measure**: Calculate some global measurement such as modularity (how well-clustered is the solution)
- **Compare**: Generate synthetic networks with an “ground-truth” set of communities
  - This is the approach of the well-known *LFR Benchmark*
  - This work focuses on generating large-scale LFR-like test benchmarks
  - For various reasons, this approach is highly preferred to modularity evaluation
“Lancichinetti–Fortunato–Radicchi” (LFR)\(^1\):  
- With >1600 citations, this is a de facto standard  
- Generates approximate solution to test against  
  - Uses tunable parameter for community coherence: \(\mu\)  
- Limited scalability: best implementation takes \(\sim17\)hrs to generate \(\sim10B\) edges\(^2\)  
  - Original code takes hours for million+ edge graphs

Our recent work: Adapted-BTER\(^3\)  
- Generates graphs that match an input degree distribution, but not a community size distribution  
- However: scales to trillion-edge graph generation (and takes only minutes!)  

---

\(^1\)[Lancichinetti et al., 2008]  
\(^2\)[Hamann et al., 2018]  
\(^3\)[Slota et al., 2019]
LFR Benchmark Graph Generation
Community Detection Algorithms: Evaluating with a ground truth

- Generate a synthetic network with some set of “communities”
- Include a mixing parameter – $\mu$ – that controls the ratio of inter- to intra-community edges: $\mu \approx \frac{\text{inter-comm. edges}}{\text{total edges}}$
  - Effectively, this determines how well-defined the communities are
- Evaluate how well an algorithm’s output matches the defined solution
  - Commonly utilize Normalized Mutual Information (NMI)
  - Compare how well algorithms perform as you increase edge mixing via $\mu$
We implement two hierarchical parallel approaches:
- Shared-Memory OpenMP: Configuration Model Chung-Lu (CMCL)
- Distributed-Memory MPI+OpenMP: Two-level Chung-Lu (TLCL)

Both follow the same general algorithmic approach:

- **Phase 1:** Initialize input distributions
  - Power-law distributions for community sizes and the degree distribution; can be generated in parallel
- **Phase 2:** Parallel assignment of vertices to communities
- **Phase 3:** Parallel internal edge generation
  - Use configuration model or Chung-Lu to generate intra-community edges
- **Phase 4:** External edge generation
  - Use Chung-Lu to generate inter-community edges
We run the Louvain Algorithm [Blondel et al., 2008] on networks generated with CMCL, TLCL, and LFR and evaluate NMI.

- Parameters: Num Vertices = 1024, 4096, 16384; Avg. Degree = 16, 24, 32; $\mu = 0.1 \ldots 0.9$
- We note near-identical outputs from all three benchmarks.
Strong Scaling of CMCL

We run strong scaling experiments with CMCL on single Intel Knight’s Landing node (17-272 threads)

- We run using degree distributions from well-known test instances
- Times given are the sum time for generating 9 benchmark graphs with $\mu = 0.1 \ldots 0.9$
- It takes about 15 minutes in total to generate a full set of instances with over 3 billion edges each from the uk-2007 distribution
Strong Scaling of TLCL

We run strong scaling experiments with TLCL on 16 Intel Knight’s Landing nodes (272 threads each)

- We run using the same instances and setup as the CMCL experiments
- We have on average over $10\times$ speedup vs. shared-memory
- All 9 instances for the 3 billion edge uk-2007 input takes in total only 1.5 minutes to generate
Conclusions and thanks!

Major takeaways:

- We develop a scalable method for generating LFR-like community detection algorithm benchmarking graphs
- This generates test instances at HPC-scale – orders-of-magnitude larger than the serial LFR code and order-of-magnitude faster than recent parallel LFR codes
- Code to be released to 
  https://github.com/HPCGraphAnalysis/SAGE pending copyright approvals

Thank you! Contact below with any questions.

slotag@rpi.edu       www.gmslot.a.com

