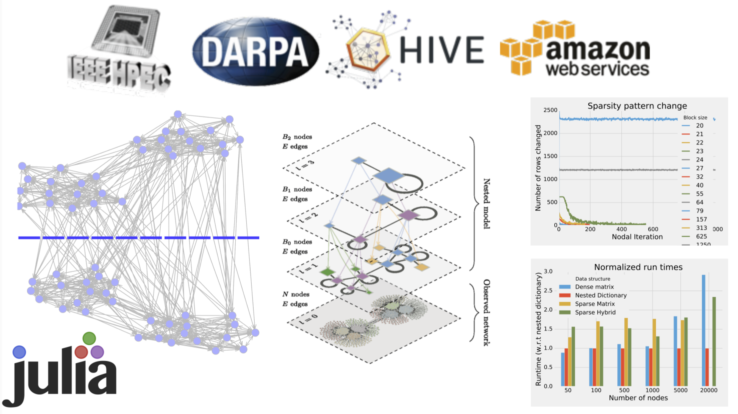
Performance Effects of Dynamic Graph Data Structures in Community Detection Algorithms

Rohit Varkey Thankachan, Brian P. Swenson, and *James P. Fairbanks*
Georgia Tech Research Institute, Atlanta, GA, USA
james.fairbanks@gtri.gatech.edu

Slides available at: http://jpfairbanks.com/publication/hpec2018/

September 26, 2018

Summary


IEEE HPEC
DARPA
HIVE
amazon web services
julia
B_2 nodes
E edges
$l = 3$
B_1 nodes
E edges
$l = 2$
B_0 nodes
E edges
$l = 1$
N nodes
E edges
$l = 0$
Nested model
Observed network
Sparsity pattern change
Number of rows changed
Nodal Iteration
Block size
20
21
22
23
24
27
32
40
55
64
79
157
313
625
Normalized run times
Runtime (w.r.t nested dictionary)
Number of nodes
Data structure
Dense matrix
Nested Dictionary
Sparse Matrix
Sparse Hybrid

Introduction

- Motivated by graph challenge
- Memory representations of graphs are significant for performance
- Many agglomerative community detection algorithms build a *community graph*
- Performance of the community graph data structure dominates runtime
- How can we study the performance of this *inner loop data structure?*
- Conclusions about data structures using the algorithm
- Conclusions about the algorithm using the data structures

Outline

- How do we choose a IBECM datastructure for this algorithm?
- Experimental Performance
- Theoretical cost model
- Hybrid Data Structure
- Sparsity change and entropy decrease set fundamental limits
- Dynamic Graph for IBECM

Community Detection Refresher

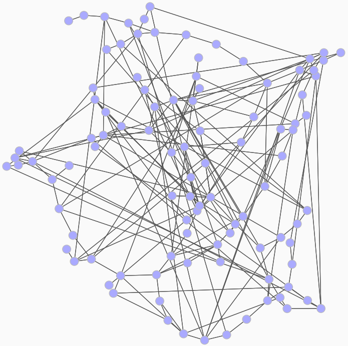


Figure 1: A graph

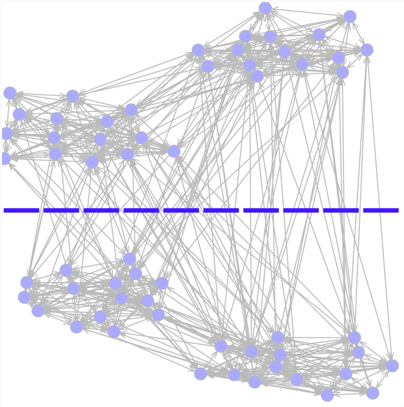


Figure 2: 4 detected communities

Piexoto's Algorithm

- Agglomerative algorithm that produces hierarchical clusters
- Nodal Phase moves vertices between clusters best cluster per vertex
- Merge Phase identifies clusters to merge

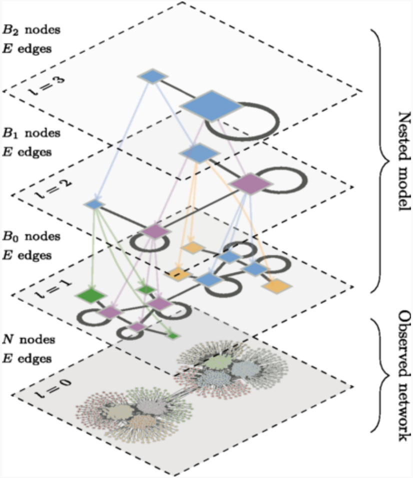


Image Credit: Piexoto 2014 https://doi.org/10.1103/PhysRevX.4.011047

Inter-block Edge Count Matrix Operations

M_{ij} counts number of edges between a vertices in community i and vertices in community j and vertices in community j

1. Insertion: M_{ij}, $0 \mapsto +$, adding an edge $i \to j$
2. Deletion: M_{ij}, $+ \mapsto 0$, removing an edge $i \to j$
3. Updates: M_{ij}, $w_{ij} \mapsto w'_{ij}$, updating the weight of the edge
4. Static structures are faster if you can use them
5. Algorithms that assign vertices to communities only once do not delete

Graph Formats

Memory access dominates graph algorithm performance. For typical graph algorithms like BFS, graphs have poor spatial and temporal locality making them hard to optimize [3].

- Dense Matrix
- Sparse Matrix
- Hash-map based structures
- Dynamic Graphs
- Relational Databases

Parallel Implementation

- Locking for correctness is slow
- MCMC allows you to relax strict ordering of operations [2]
- Parallel phases: read phase then write phase.

Performance

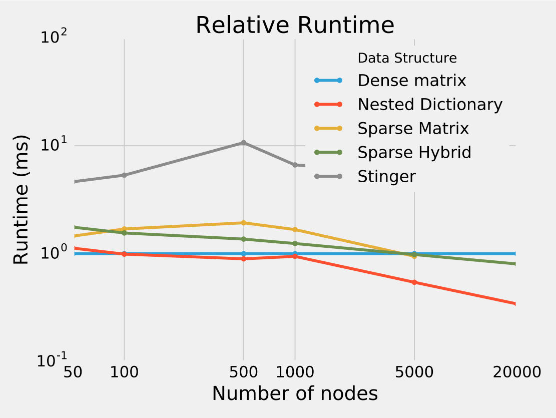


Figure 3: Run time of each data structure as a function of graph size n. The hybrid data structure is faster than the sparse matrix structure after the crossover point at $n \approx 5000$

Algorithm Cost Analysis i

- Piexto algorithm over cost is $O(n\log^2 n)$ [4].
- For HPC applications we need components of the overall runtime bound because the different operations take different amounts of time
- Read operations access M (proposed moves)
- Write operations modify M (accepted moves)
- Proposals per vertex be denoted by N_p
- Proposals accepted per vertex be denoted by N_e

Algorithm Cost Analysis ii

Let the cost of a read operation be α and the cost of a write operation be β. Cost is measured according to the time or cycles used per operation. The runtime formula is given by

$$\alpha N_p V + \beta N_e V \tag{1}$$

- Aggregate operation counts control performance
- Different Data structures show different performance
- Our code uses Julia and multiple dispatch to allow hot-swapping implementations

Sparse Matrix Hybrid

Taking a page from streaming graph algorithms an incremental linear algebra, IBECM M satisfies:

$$M = C'AC \tag{2}$$

Let Δ represent updates to C, such that $C_{new} = C + \Delta$

$$\begin{aligned} M_{new} &= (C+\Delta)'A(C+\Delta) \qquad (3) \\ &= C'AC + \Delta'AC + C'A\Delta + \Delta'A\Delta \qquad (4) \end{aligned}$$

Hybrid Data Structure Approach

From read-write analysis of the algorithm, we derived a threshold on when a hybrid algorithm is an improvement:

$$\frac{2\gamma}{V}\frac{N_c}{N_p} < \frac{N_e}{N_p}(\beta_R - \beta_W) + (\alpha_W - \alpha_R) \tag{5}$$

Basically, single point reads must be constant time for optimal data structure.

Normalized Run Time

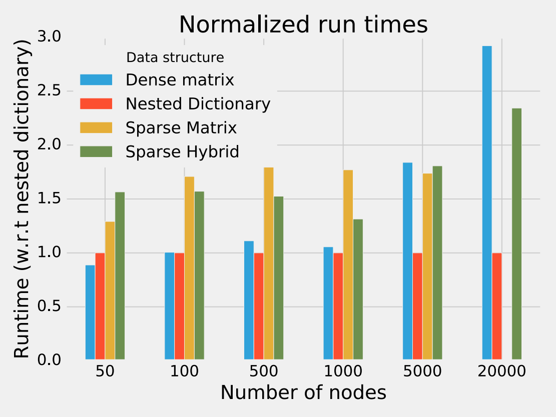


Figure 4: Run time normalized to nested dictionary performance for each graph size n. Nested dictionary is faster in most cases. Performance of sparse hybrid data structure is better than sparse matrix, as predicted.

Memory Usage

Table 1: Average Memory Allocated (Normalized to dense matrix allocation) for 5000 nodes

Name	Memory Allocated (GB)	Normalized Memory
Dense matrix	1996.7	1
Nested Dictionary	311.704	0.156
Sparse Matrix	662.199	0.332
Hybrid	665.545	0.333
Stinger	1225.696	0.614

The Julia Programming Language

- Solves the “two language problem” by offering high performance in a high productivity language
- Generic Programming with multiple dispatch allows for swapping data structures
- A mature graph library LightGraphs.jl [1].
- Building on previous work with STINGER.jl [5].
- Easy to use parallel @threads.

Sparsity Change Analysis

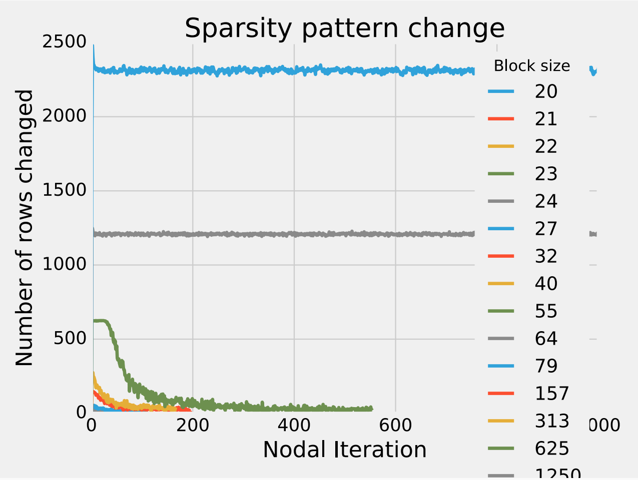


Figure 5: Number of rows changed in the nodal iteration phase (V=5000). Sparsity changes are stable for iterations of sizes 2500 and 1250 with almost all rows touched every time. As the existing partition

Community Detection Quality

Table 2: Average Detection Quality

Name	Accuracy	Pairwise precision	Pairwise recall
Dense matrix	0.94	1	0.95
Nested Dictionary	0.93	0.99	0.94
Sparse Matrix	0.96	1	0.97
Sparse Hybrid	0.93	1	0.94
Stinger	0.97	1	0.97

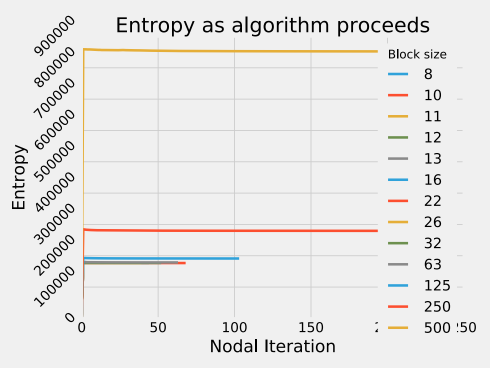
- Detection quality is similar across all data structures
- Variation due to parallel benign races

Entropy Decrease as a Stopping Criterion

Entropy of nodal iterations for a 1000 node graph.

The nodal phase doesn't decrease entropy.

Entropy measured as description length [2]



Entropy change is not a good proxy for stopping criterion.

Conclusion

- Our theoretical analysis allows you to choose between data structures (or hybrids) a priori.
- Entropy analysis fails as a stopping criteria
- Large sparsity churn in this algorithm sets a limit on performance improvement
- Hard Problem: developing dynamic graph data structures for large sparsity churn

Acknowledgments

Rohit Varkey Thankachan

Geoff Sanders

Edward Kao

David Bader

Eric Hein

and the PACE team at Georgia Tech

Strong Scaling

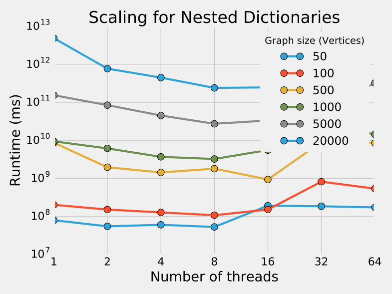


Figure 6: Strong Scaling: Run time as a function of thread count. Scaling is better for larger values of *n* where there is more work to be done. Also, hyperthreading (16 – 64 threads) is not substantially helpful for this problem.

References

- Seth Bromberger, James Fairbanks, and other contributors.
  JuliaGraphs/LightGraphs.jl: LightGraphs v0.13.1, Sep 2017.
- Edward Kao, Vijay Gadepally, Michael Hurley, Michael Jones, Jeremy Kepner, Sanjeev Mohindra, Paul Monticciolo, Albert Reuther, Siddharth Samsi, William Song, et al.
  Streaming graph challenge: Stochastic block partition.
  arXiv preprint arXiv:1708.07883, 2017.
- Andrew Lumsdaine, Douglas Gregor, Bruce Hendrickson, and Jonathan Berry.
  Challenges in parallel graph processing.
  Parallel Processing Letters, 17(01):5–20, 2007.
- Tiago P Peixoto.
  Efficient monte carlo and greedy heuristic for the