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
September 26, 2018
Summary

[Image of a network diagram with labels for nodes and edges, and graphs showing sparsity pattern change and normalized run times.]

- DARPA
- HIVE
- Amazon Web Services
- Julia

[Graph showing sparsity pattern change with x-axis labeled 'Nodal iteration' and y-axis labeled 'Number of rows changed'.]

- Normalized run times with data structures: Dense matrix, Nested Dictionary, Sparse Matrix, Sparse Hybrid.

[Graph showing normalized run times with x-axis labeled 'Number of nodes' and y-axis labeled 'Runtime (w.r.t. nested dictionary)'.]
• 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 vertices in community $i$ and vertices in community $j$ and vertices in community $j$

1. Insertion: $M_{ij}$, $0 \rightarrow +$, adding an edge $i \rightarrow j$
2. Deletion: $M_{ij}$, $+ \rightarrow 0$, removing an edge $i \rightarrow j$
3. Updates: $M_{ij}$, $w_{ij} \rightarrow w'_{ij}$, 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
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.
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 $n \approx 5000$
• 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 $N_e$
Let the cost of a read operation be $\alpha$ and the cost of a write operation be $\beta$. Cost is measured according to the time or cycles used per operation. The runtime formula is given by

$$\alpha N_p V + \beta N_e V$$

(1)

- Aggregate operation counts control performance
- Different Data structures show different performance
- Our code uses Julia and multiple dispatch to allow hot-swapping implementations
Taking a page from streaming graph algorithms an incremental linear algebra, IBECM $M$ satisfies:

$$M = C'AC$$  \hspace{1cm} (2)$$

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

$$M_{new} = (C + \Delta)'A(C + \Delta)$$  \hspace{1cm} (3)$$

$$= C'AC + \Delta'AC + C'A\Delta + \Delta'A\Delta$$  \hspace{1cm} (4)$$
From read-write analysis of the algorithm, we derived a threshold on when a hybrid algorithm is an improvement:

\[
\frac{2\gamma}{V} \frac{N_c}{N_p} < \frac{N_e}{N_p} (\beta_R - \beta_W) + (\alpha_W - \alpha_R)
\]  

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Memory Allocated (GB)</th>
<th>Normalized Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense matrix</td>
<td>1996.7</td>
<td>1</td>
</tr>
<tr>
<td>Nested Dictionary</td>
<td>311.704</td>
<td>0.156</td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td>662.199</td>
<td>0.332</td>
</tr>
<tr>
<td>Hybrid</td>
<td>665.545</td>
<td>0.333</td>
</tr>
<tr>
<td>Stinger</td>
<td>1225.696</td>
<td>0.614</td>
</tr>
</tbody>
</table>
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.
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 becomes more modular, the sparsity changes due to a nodal move become smaller. Each series is the initial number of vertices \(n\).
### Community Detection Quality

<table>
<thead>
<tr>
<th>Name</th>
<th>Accuracy</th>
<th>Pairwise precision</th>
<th>Pairwise recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dense matrix</td>
<td>0.94</td>
<td>1</td>
<td>0.95</td>
</tr>
<tr>
<td>Nested Dictionary</td>
<td>0.93</td>
<td>0.99</td>
<td>0.94</td>
</tr>
<tr>
<td>Sparse Matrix</td>
<td>0.96</td>
<td>1</td>
<td>0.97</td>
</tr>
<tr>
<td>Sparse Hybrid</td>
<td>0.93</td>
<td>1</td>
<td>0.94</td>
</tr>
<tr>
<td>Stinger</td>
<td>0.97</td>
<td>1</td>
<td>0.97</td>
</tr>
</tbody>
</table>

- 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
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.
Seth Bromberger, James Fairbanks, and other contributors. *JuliaGraphs/LightGraphs.jl: LightGraphs v0.13.1, Sep 2017.*


Tiago P Peixoto. *Efficient monte carlo and greedy heuristic for the inference of stochastic block models.*