Direct Graph Ordering Optimization for Cache-Efficient Graph Analysis

Michael Mandulak¹  Ruochen Hu¹  George M. Slota¹

¹Rensselaer Polytechnic Institute

SIAM Annual Meeting - Graph and Matrix Ordering and Coarsening

15 July 2022
Graph Ordering Problem

Motivation

**Goal:** Improve Analysis Measures using Refinement
- Analysis Runtime
- Analysis Cache Efficiency

**Why?**
- Faster graph analysis – growing network sizes.
- Memory access pattern concerns on HPC systems.

**Focus:**
- Improve vertex locality for improved memory access patterns.
- NP-hard – heuristic and approximation solutions.
- Experimental study – is optimization viable?
Graph Ordering Problem

Background

Problem:

- Undirected graph $G = (V[0, n), E \subseteq V \times V)$, find permutation $\pi : V \rightarrow \mathbb{N}$ to minimize a metric.

Metrics:

- Linear Gap Arrangement (LinGap) problem:
  $$\text{LinGap}(G, \pi) = \sum_{u \in \mathbb{N}} \sum_{v_i \in sN(u)} |\pi(u) - \pi(v_i)|.$$  

- Log Gap Arrangement (LogGap) problem:
  $$\text{LogGap}(G, \pi) = \sum_{u \in \mathbb{N}} \sum_{v_i \in sN(u)} \log(|\pi(u) - \pi(v_i)|).$$
Considerations:

▶ What analysis algorithms can we test with?
▶ What ordering methods can we compare with?
▶ How do our metrics relate to analysis measures?
▶ How should we refine?
Analysis Algorithms
Memory Access

**Focus:** Vertex-centric approaches with CPU-based shared memory parallelism.

**PageRank**
- Sparse matrix-vector multiplication.
- Compressed Sparse Row locality.

**Louvain**
- Coarsening through edge density.
- Ordering dependent within neighborhoods.

**Multistep**
- Traversal and propagation connectivity.
- BFS-based vertex access.
Ordering Methods

Natural Ordering
Rabbit
  ▶ Community generation and mapping to cache-hierarchies.
  ▶ Optimizes for cache efficiency.
Layered Label Propagation (LLP)
  ▶ Community detection through label propagation.
  ▶ Considers global distribution of labels.
  ▶ Optimizes for compression.
Shingle
  ▶ Order by neighborhood commonalities.
  ▶ Optimizes for compression.
Metric Correlation
LinGap (Top) & LogGap (Bottom)
Refinement Method

Algorithm

Algorithm 1 Log Gap Arrangement Refinement by Degree

1: function LOGGAP DEGREE REFINE(G,p)
2:     S = sort(V) ascending by degree
3:     for each vertex u in the first p percent of S in parallel
4:         for each vertex v in u’s adjacency list do
5:             bs = evalLogGapArrLocal(G, u, v)
6:             as = evalLogGapArrLocalSwap(G, u, v)
7:             if as < bs then
8:                 desiredSwap_u = v
9:             end if
10:         end for
11:     end for
12:     for each vertex u in the first p percent of S do
13:         bs = evalLogGapArr(G)
14:         swap(u, desiredSwap_u)
15:         as = evalLogGapArr(G)
16:         if bs < as then
17:             swap(u, desiredSwap_u)
18:         end if
19:     end for
20: end function
Considerations:

- What graphs to test on?
- How much refinement should we conduct?
- What observations can be drawn from results?
- How are these results to be used?
Experimentation

**Data:** Diverse classes and sizes
- SNAP, DIMACS, WebGraph

**Collection:**
- Ten runs per analysis algorithm per initial ordering per refinement method.

**Architecture:**
- AMD system – 2TB DDR4 RAM.
- Cache per core: 4MiB L1, 64 MiB L2, 256MiB shared L3 per socket.

<table>
<thead>
<tr>
<th>Graph</th>
<th>Class</th>
<th>#Vertices</th>
<th>#Edges</th>
</tr>
</thead>
<tbody>
<tr>
<td>com-Friendster</td>
<td>Social</td>
<td>66 M</td>
<td>1.8 B</td>
</tr>
<tr>
<td>twitter-2010</td>
<td>Social</td>
<td>41.7 M</td>
<td>1.5 B</td>
</tr>
<tr>
<td>LiveJournal</td>
<td>Social</td>
<td>4.8 M</td>
<td>69 M</td>
</tr>
<tr>
<td>web-ClueWeb09</td>
<td>Web Graph</td>
<td>1.7 B</td>
<td>7.9 B</td>
</tr>
<tr>
<td>enwiki-2013</td>
<td>Web Graph</td>
<td>4.2 M</td>
<td>101.3 M</td>
</tr>
<tr>
<td>web-BerkStan</td>
<td>Web Graph</td>
<td>685 K</td>
<td>7.6 M</td>
</tr>
<tr>
<td>it-2004</td>
<td>Web Graph</td>
<td>41.3 M</td>
<td>1.2 B</td>
</tr>
<tr>
<td>ant1km</td>
<td>Mesh</td>
<td>13.5 M</td>
<td>53.8 M</td>
</tr>
<tr>
<td>trianglemesh1</td>
<td>Mesh</td>
<td>1.9 M</td>
<td>1.9 M</td>
</tr>
<tr>
<td>USA-road-d</td>
<td>Road</td>
<td>24 M</td>
<td>58.3 M</td>
</tr>
</tbody>
</table>
Refinement Percent

LiveJournal

The graph shows the improvement in refinement percentage for two methods: LinRefine (red line) and LogRefine (black line). The x-axis represents the refinement percentage ranging from 0.1% to 5.0%, while the y-axis shows the improvement ranging from 1.0000 to 1.0030. The LinRefine method shows a more gradual increase in improvement compared to the LogRefine method, which exhibits a steeper increase especially at higher refinement percentages.
Results

PageRank

![Graphs showing speedup and cache miss improvement for different datasets and algorithms.](image)
Results

Louvain
Results

Multistep
Results Summary

Observations:

▶ High impact from an initial Rabbit ordering.
▶ Refinement with a Rabbit ordering promising.
▶ Improvement trends upon algorithm-generated orderings with refinement.

<table>
<thead>
<tr>
<th>Ordering</th>
<th>Cache</th>
<th>L1 Cache</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>LLP</td>
<td>0.991</td>
<td>1.002</td>
<td>1.637</td>
</tr>
<tr>
<td>LLPLinRefine</td>
<td>1.053</td>
<td>1.005</td>
<td>1.623</td>
</tr>
<tr>
<td>LLPLLogRefine</td>
<td>1.056</td>
<td>1.007</td>
<td>1.593</td>
</tr>
<tr>
<td>Rabbit</td>
<td>1.002</td>
<td>0.999</td>
<td>1.933</td>
</tr>
<tr>
<td>RabbitLinRefine</td>
<td><strong>1.144</strong></td>
<td>1.017</td>
<td><strong>2.025</strong></td>
</tr>
<tr>
<td>RabbitLogRefine</td>
<td>1.137</td>
<td><strong>1.031</strong></td>
<td>1.973</td>
</tr>
<tr>
<td>Shingle</td>
<td>1.017</td>
<td>1.018</td>
<td>1.317</td>
</tr>
<tr>
<td>ShingleLinRefine</td>
<td>1.043</td>
<td>0.986</td>
<td>1.336</td>
</tr>
<tr>
<td>ShingleLogRefine</td>
<td>1.050</td>
<td>1.026</td>
<td>1.340</td>
</tr>
<tr>
<td>LinRefine</td>
<td>1.054</td>
<td>1.007</td>
<td>1.479</td>
</tr>
<tr>
<td>LogRefine</td>
<td>1.058</td>
<td>0.992</td>
<td>1.458</td>
</tr>
</tbody>
</table>
Contributions

- Experimental study into the explicit refinement of vertex orderings.
- LinGap and LogGap metrics show promising correlations with PageRank analysis measures.
- Metrics improve in spikes throughout refinement.
- Refinement is most effective on an initial Rabbit ordering.
- Optimization shows promising improvements to heuristic methods.
Future Works

Refinement Testing
- Further testing of degree-based refinement – more graph classes and sizes.
- More diverse analysis algorithms – not TLAV.

Improved Refinement
- Explicit refinement on subgraphs.
- Alternate partitioning methods for refinement.
- Apply spectral and multi-level methods to the refinement process.
Future Works

Cont.

Optimization

- Apply linear and non-linear programming models to adjacency lists for our metrics.
- Address runtime growth concerns with such optimization models.
- Apply to subgraphs for easily distributed computing?

Metrics

- Single metric that is memory access pattern-agnostic?
- Further experimentation with current metrics – determine properties of each for a new metric?
- Spectral considerations in metrics?
Acknowledgement & Contact

Acknowledgement

▶ This work is supported by the National Science Foundation under Grant No. 2047821.

Contact

▶ Michael Mandulak: mandum@rpi.edu
▶ George Slota: slotag@rpi.edu