Rensselaer

Explicit Ordering Refinement for Accelerating Irregular Graph Analysis

Michael Mandulak[1] Ruochen Hu[1] George M. Slota[1]

[1]Rensselaer Polytechnic Institute

IEEE HPEC 2022- Graph Analytics & Network Science

20 September 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 find permutation

to minimize a metric.

Metrics:

- Linear Gap Arrangement (LinGap) problem:

- Log Gap Arrangement (LogGap) problem:

Experimental Study

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)

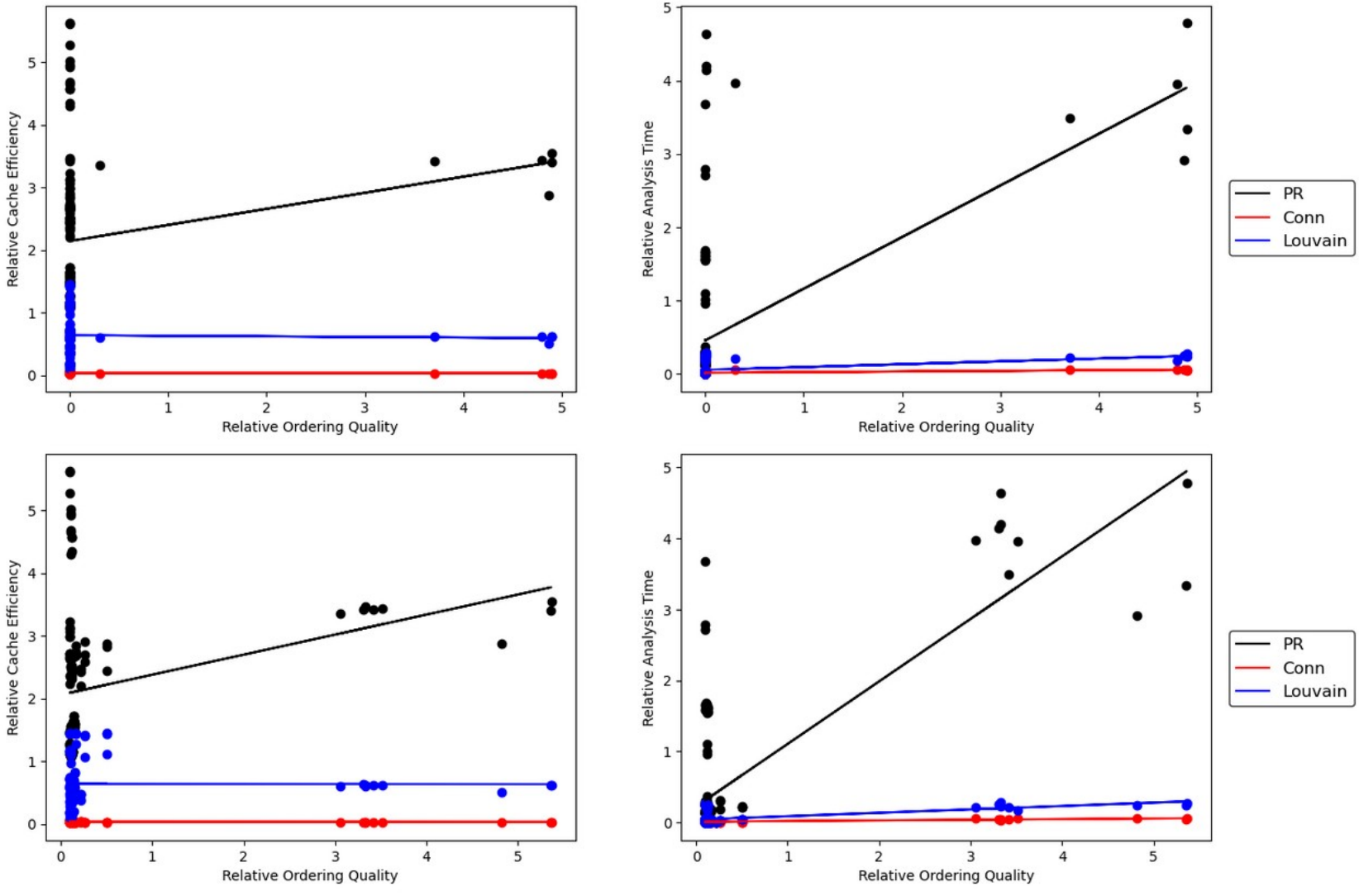
- Community detection through label propagation.
- Considers global distribution of labels.
- Optimizes for compression.

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 do
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 and as < desiredSwapVal_u then
8:                 desiredSwap_u = v
9:                 desiredSwapVal_u = as
10:            end if
11:        end for
12:    end for
13:    for each vertex u in the first p percent of S do
14:        bs = evalLogGapArr(G)
15:        swap(G, u, desiredSwap_u)
16:        as = evalLogGapArr(G)
17:        if bs < as then
18:            swap(G, u, desiredSwap_u)
19:        end if
20:    end for
21: end function
```

Results Collection

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.
- Tens of thousands range of vertices to refine (experimentation).

Architecture:

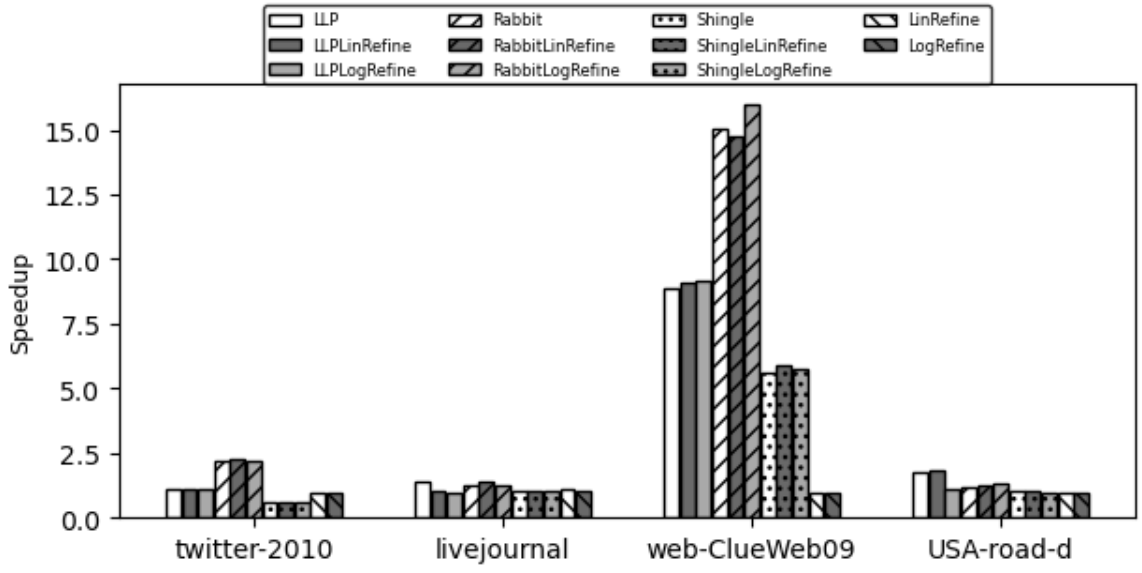
- AMD system – 2TB DDR4 RAM.
- Cache per (3 shared L3 per socket.

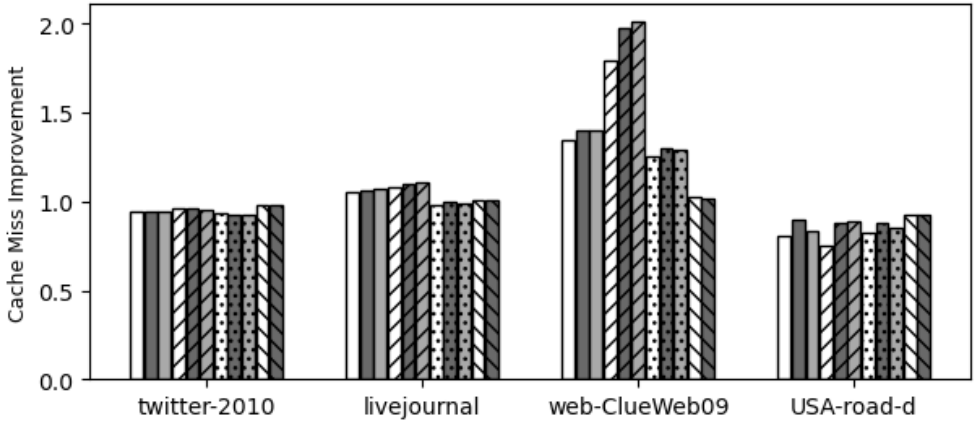
TABLE I
BASIC GRAPH PROPERTIES

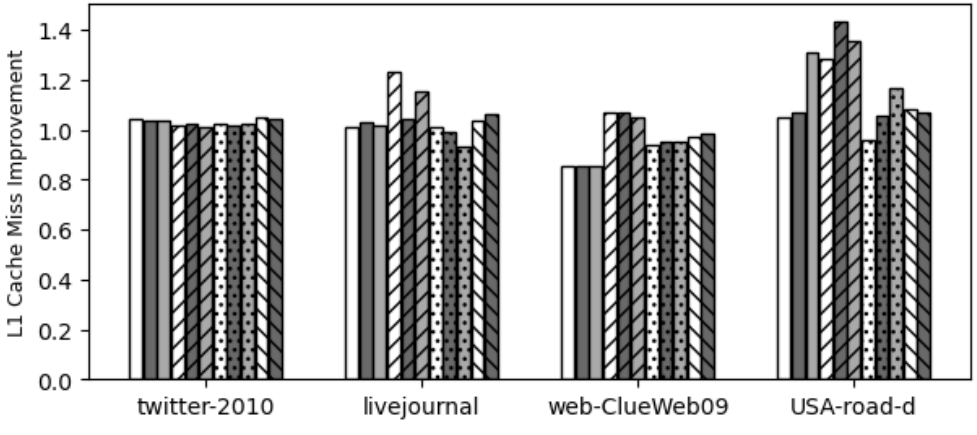
Graph	Class	#Vertices	#Edges
com-Friendster	Social	66 M	1.8 B
twitter-2010	Social	41.7 M	1.5 B
LiveJournal	Social	4.8 M	69 M
web-ClueWeb09	Web Graph	1.7 B	7.9 B
enwiki-2013	Web Graph	4.2 M	101.3 M
web-BerkStan	Web Graph	685 K	7.6 M
it-2004	Web Graph	41.3M	1.2 B
ant1km	Mesh	13.5 M	53.8 M
trianglemesh1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

Results

PageRank







Results Summary

Observations:

- High impact from an initial Rabbit ordering.
- Experimental results show promise using baseline methods.
- In
orde

TABLE III
IMPROVEMENT AND SPEEDUP RESULTS FOR EACH ORDERING METHOD TAKEN AS THE GEOMETRIC AVERAGE ACROSS ALL GRAPHS AND ACROSS ALL ANALYTIC ALGORITHMS USING THE AMD SYSTEM.

Ordering	Cache	L1 Cache	Time
LLP	0.991	1.002	1.637
LLPLinRefine	1.053	1.005	1.623
LLPLogRefine	1.056	1.007	1.593
Rabbit	1.002	0.999	1.933
RabbitLinRefine	**1.144**	1.017	**2.025**
RabbitLogRefine	1.137	**1.031**	1.973
Shingle	1.017	1.018	1.317
ShingleLinRefine	1.043	0.986	1.336
ShingleLogRefine	1.050	1.026	1.340
LinRefine	1.054	1.007	1.479
LogRefine	1.058	0.992	1.458

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 given improved refinement methods.

Future Works

Refinement Testing

- More diverse analysis algorithms – not TLAV / graph classes.

Improved Refinement

- Explicit refinement on subgraphs.
- Alternate partitioning/spectral/multi-level methods for refinement.

Optimization

- Apply linear and non-linear solver models to adjacency lists for our metrics.
- Apply to subgraphs for easily distributed computing?

Metrics

- Single metric that is memory access pattern-agnostic?

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