Rensselaer

Vertex Ordering Refinement and Coarsening Methods for Accelerated Graph Analysis

Michael Mandulak[1] Christopher Brissette[1]
George M. Slota[1]

[1]Rensselaer Polytechnic Institute

SIAM Conference on Computational Science and Engineering (CSE23)

3 March 2023

Outline

- Motivation & Background - Vertex Ordering and Coarsening
- Experimental Study - Metrics and Analysis Methods
- Experimental Study - Considered Ordering Methods
- Explicit Ordering Refinement - Previous Work
- Applications Within Coarsening - Algebraic Distance Refinement
- Experimental Results - Improvements and Variability
- Conclusions & Future Works

Vertex Ordering Problem - Coarsening

Motivation

Goal: Develop ordering refinement methods within coarsening

- Improve analysis runtime and cache efficiency.
- Integrate ordering refinement with coarsening methods.

Why?

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

Focus:

- Improve vertex locality for improved memory access patterns.
- Consider vertex ordering in the context of coarsening.
- Experimental study – is optimization viable?
  - Apply partitioning methods to vertex ordering.

Vertex Ordering Problem

Background

Problem:

- Undirected graph $\mathrm{G} = (\mathrm{V}[0, \mathrm{n}), \mathrm{E} \subseteq \mathrm{V} \times \mathrm{V})$, find permutation

$\pi : \mathrm{V} \rightarrow \mathrm{N}$ to minimize a metric.

Metrics:

- Linear Gap Arrangement (LinGap) problem:

$$LinGap(G, \pi) = \sum_{u \in N} \sum_{v_i \in sN(u)} |\pi(u) - \pi(v)| .$$

- Log Gap Arrangement (LogGap) problem:

$$LogGap(G, \pi) = \sum_{u \in N} \sum_{v_i \in sN(u)} \log(|\pi(u) - \pi(v)|) .$$

Graph Coarsening Problem

Background

Problem:

- Undirected graph $G = (V[0, n), E \subseteq V \times V)$, find permutation
- Find representation $G_c = (V_c, E_c)$ where $|V_c| = n_c < n$.

Metric:

- Similarity - Algebraic Distance

Algorithm 1 Algebraic Distance

Input: Parameter ω, initial random vector x^r

1: **for** $k = 1, 2, \ldots$ **do**
2: $\quad \tilde{x}_i^{(k)} \leftarrow \sum_j w_{ij} x_j^{(k-1)} / \sum_j w_{ij} \quad \forall i$
3: $\quad x^{(k)} \leftarrow (1 - \omega) x^{(k-1)} + \omega \tilde{x}^{(k)}$
4: **end for**

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? How to include coarsening?

Analysis Algorithms

Memory Access

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

PageRank

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

Louvain

- Community detection 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.

Previous Refinement Work

Highlights:

- Positive refinement metric and analysis measure correlations.
- Degree-based refinement method.
- Experimental results - high improvement with initial Rabbit ordering.

Relative Analysis Improvements

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

Degree-based Refinement Method

Algorithm

Algorithm 2 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
```

Integrate With Coarsening

Motivation:

- Ordering and coarsening framework - maintain locality.
- Relate coarsening metrics to analysis measures.
- Predict ordering vertex set from coarsening.

Considerations:

- Can we predict an ordering using our metrics?
- Can we consider coarsening metrics in ordering refinement?
- Can we develop a refinement method to integrate with coarsening?

Coarsening Method

Algebraic Distance

Vertex Similarity Metric: Algebraic Distance

Algorithm 4 Algebraic Distance Coarsening

```
1: function ALGDIST COARSEN(G,V,p)                ▷ GPU Parallel
2:     V_ad ← algDist(G, V)
3:     SetEdgeWeights(V_ad, G)
4:     M = SuitorMatching(G)
5:     algDistRefine(G, V_ad, p)                  ▷ Refine Ordering
6:     G = merge(G, M)
7: end function
```

Algorithm 5 GPU-Suitor Algorithm

```
1: function GPU-SUITOR(G(V, E), mate)
2:     while there are vertices to process do
3:         for each V_i in parallel do
4:             for each v ∈ V_i do
5:                 Process adj(v) in parallel
6:                 Determine best candidate for v in parallel
7:             end for
8:             Set suitor for each candidate of V_i in parallel
9:             Store self or displaced vertices
10:            Synchronize across warps; load balance
11:
```

Algebraic Distance Refinement Method

Algorithm

Algorithm 3 Algebraic Distance Refinement

```
1: function ALGEBRAIC DISTANCE REFINE(G,p)
2:     V_ad = algDist(G)
3:     V_s = degreeSortAndChoose(G, p)
4:     for u ∈ V_s in parallel do
5:         for each vertex v in u's adjacency list do
6:             localAlgDist = |V_ad[u] − V_ad[v]|
7:             if localAlgDist is maximum then
8:                 maxPair = v
9:             end if
10:        end for
11:        minV = smallestLabel(u, maxPair)
12:        maxV = largestLabel(u, maxPair)
13:        for each vertex v in minV's adjancency list do
14:            if |label[maxV] − label[v]| is minimum then
15:                desiredSwap_minV = v
16:            end if
17:        end for
18:    end for
19:    for each desiredSwap_minV do
20:        bs = evalLogGapArr(G)
21:        swap(G, minV, desiredSwap_minV)
22:        as = evalLogGapArr(G)
23:        if bs < as then
24:            swap(G, minV, desiredSwap_minV)
25:        end if
26:    end for
27: end function
```

Algebraic Distance Refinement Method

Desired Swaps

- Find least similar (maximum) neighbor from metric.
- Find the closest label among the smallest label vertex's neighbors.

2: $V_{ad} = algDist(G)$
3: $V_s = degreeSortAndChoose(G, p)$
4: **for** $u \in V_s$ in parallel **do**
5: **for** each vertex v in u's adjacency list **do**
6: $localAlgDist = |V_{ad}[u] - V_{ad}[v]|$
7: **if** $localAlgDist$ is maximum **then**
8: $maxPair = v$
9: **end if**
10: **end for**
11: $minV = smallestLabel(u, maxPair)$
12: $maxV = largestLabel(u, maxPair)$
13: **for** each vertex v in $minV$'s adjancency list **do**
14: **if** $|label[maxV] - label[v]|$ is minimum **then**
15: $desiredSwap_{minV} = v$
16: **end if**
17: **end for**
18: **end for**

Algebraic Distance Refinement Method

Swap Completion

- Sequentially consider $p \times n$ swaps.
- Commit swap if the metric still holds - changes per swap.

19: **for each** $desiredSwap_{minV}$ **do**
20: $bs = \text{evalLogGapArr}(G)$
21: $\text{swap}(G, minV, desiredSwap_{minV})$
22: $as = \text{evalLogGapArr}(G)$
23: **if** $bs < as$ **then**
24: $\text{swap}(G, minV, desiredSwap_{minV})$
25: **end if**
26: **end for**

Experimentation

Data: Diverse classes and sizes

- SNAP, DIMACS, WebGraph

Collection:

- Ten runs per analysis algorithm per initial ordering per refinement method.

Architecture:

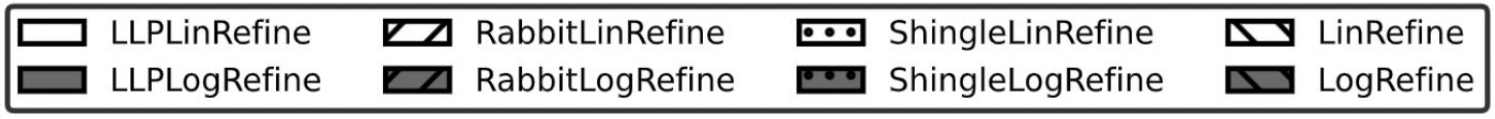
- AMD system – 2TB DDR4 RAM.
- Cache per core: 4MiB L1, 64 MiB L2, 256MiB shared L3 per socket.

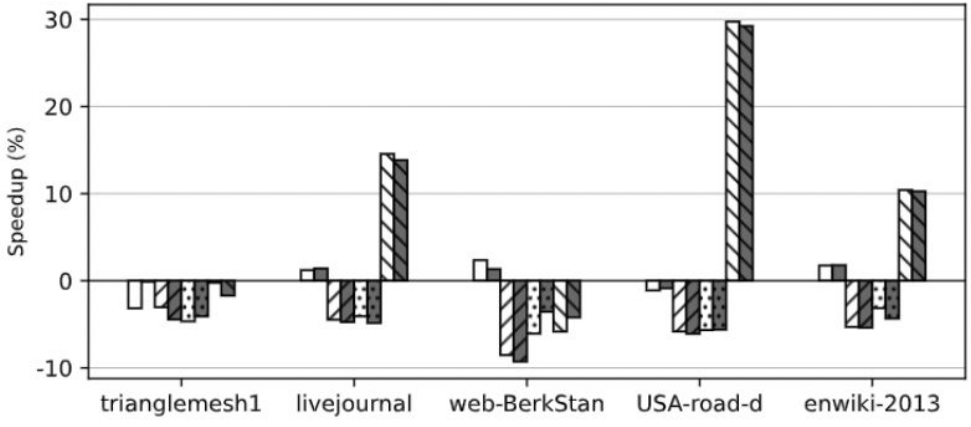
TABLE 1
Basic ordering graph properties

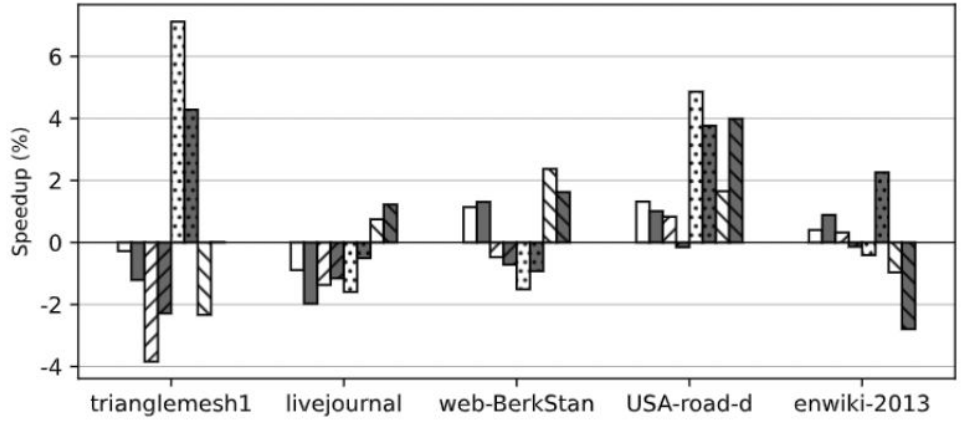
Graph	Class	#Vertices	#Edges	Cite
com-Friendster	Social	66 M	1.8 B	[27]
twitter-2010	Social	41.7 M	1.5 B	[28]
LiveJournal	Social	4.8 M	69 M	[29]
web-ClueWeb09	Web Graph	1.7 B	7.9 B	[30]
enwiki-2013	Web Graph	4.2 M	101.3 M	[4]
web-BerkStan	Web Graph	685 K	7.6 M	[31]
it-2004	Web Graph	41.3M	1.2 B	[32]
ant1km	Mesh	13.5 M	53.8 M	[33]
trianglemesh1	Mesh	1.9 M	1.9 M	[34]
bunny	Mesh	34.8 K	69.6 K[1]	[35]
USA-road-d	Road	24 M	58.3 M	[36]

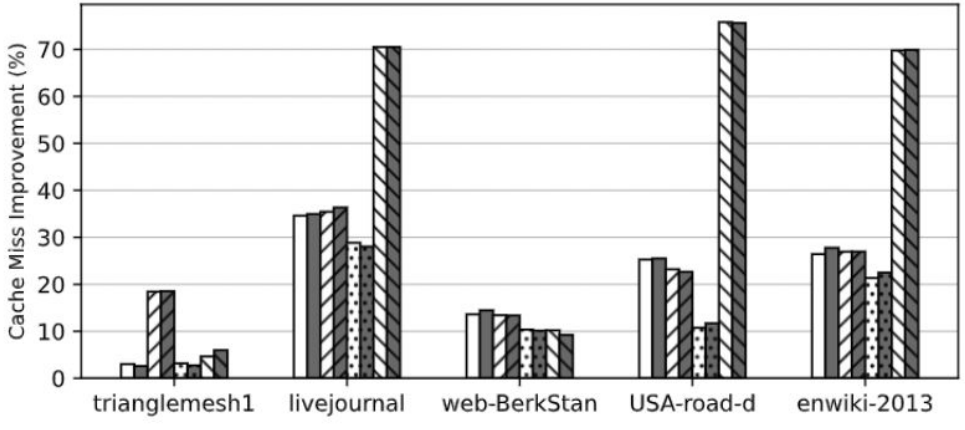
Results - Louvain (L) & Multistep (R)

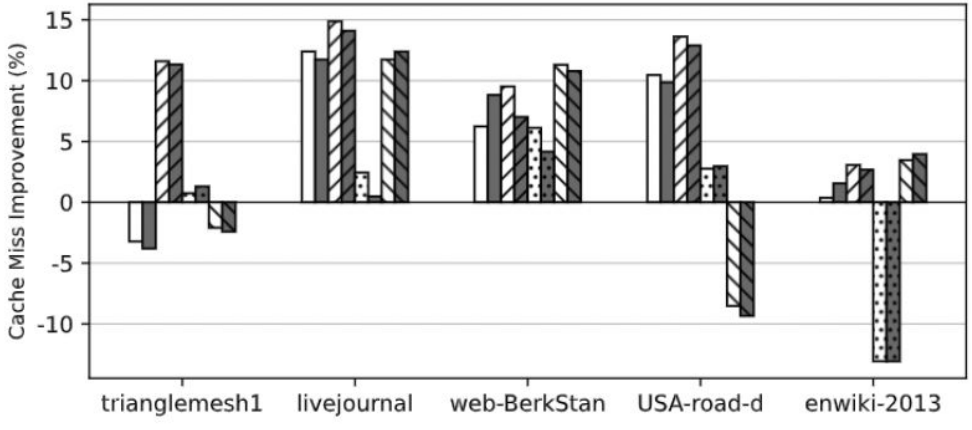
Algebraic Distance Refinement





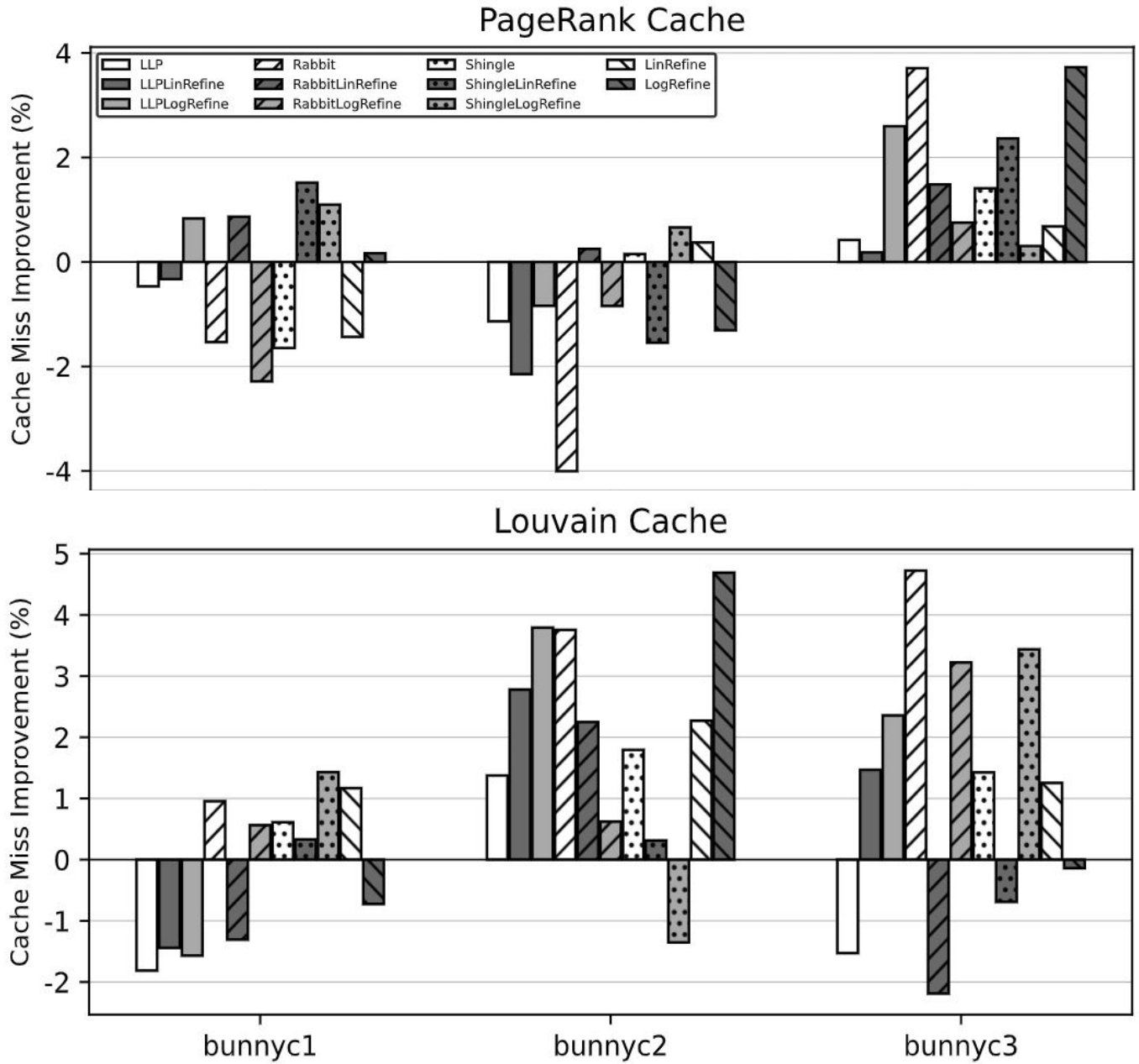






Results - Ordering per Coarsening Level

Algebraic Distance Refinement



Results Summary

- Overall slight analysis improvements for Louvain and Multistep using algebraic distance refinement.
- Algebraic distance refinement on an initial natural ordering can show high improvement.
- Initial results for refinement per coarsening level show variable improvement trends.
- Optimization shows potential for improvements to heuristic methods.

Contributions & Conclusions

Contributions

- Experimental study into integrated ordering within coarsening.
- Refinement framework for similarity metric-based ordering refinement.
- Experimental results for the improvement of analysis measures.

Conclusions:

- Explicit ordering methods are complex!
- Algebraic distance refinement shows similar improvements to degree-based refinement.
- Not currently competitive with heuristics - can show high improvement with a carefully chosen refinement set.

Future Works

Refinement Testing

- Further testing of algebraic distance refinement – more graph classes and sizes.
- More diverse analysis algorithms – not TLAV.
- Alternative similarity metrics.

Framework

- Runtime analysis per coarsening level and refinement amount.
- Active prediction of ordering based on similarity - omit portions of refinement from predicted coarsening/ordering.
- Apply spectral and multi-level methods to the refinement process.

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