Explicit Ordering Refinement for Accelerating Irregular Graph Analysis

Michael Mandulak
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York, USA
mandum@rpi.edu

Ruochen Hu
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York, USA
hur4@rpi.edu

George M. Slota
Department of Computer Science
Rensselaer Polytechnic Institute
Troy, New York, USA
slotag@rpi.edu

***Abstract*—Vertex reordering for efficient memory access in extreme-scale graph-based data analysis shows considerable improvement to the cache efficiency and runtimes of widely used graph analysis algorithms. Despite this, modern efficient ordering methods are often heuristic-based and do not directly optimize some given metrics. Thus, this paper conducts an experimental study into explicit metric-based vertex ordering optimization. We introduce a universal graph partitioning-inspired approach focusing on CPU shared-memory parallelism to the vertex ordering problem through the explicit refinement of low-degree vertices using the Linear Gap Arrangement and Log Gap Arrangement problems as comprehensive metrics for ordering improvement. This degree-based refinement method is evaluated upon a number of initial orderings with timing and cache efficiency results relative to three shared-memory graph analytic algorithms: PageRank, Louvain and the Multistep algorithm. Applying refinement, we observe runtime improvements of up to 15x on the ClueWeb09 graph and up to 4x improvements to cache efficiency on a variety of network types and initial orderings, demonstrating the feasibility of an optimization approach to the vertex ordering problem at a large scale.**

***Index Terms*—graph ordering, cache efficiency, linear gap arrangement, log gap arrangement, graph analysis**

I. Introduction

Graph analysis at a large scale has become increasingly applicable given modern advancements in high-performance computing methods and the sheer size of real-world networks [1, 2]. Despite this, inefficient memory access patterns often limit the performance and feasibility of such analytic algorithms, especially as edge totals surpass the trillions. This problem is generalized to a vertex locality issue, prompting concerns regarding the frequency of vertex access and the order at which this occurs. These concerns are shown to limit the scalability of graph analytic algorithms in parallel [3].

To improve upon vertex locality, ordering methods have been explored to generate hierarchical relationships in large-scale graphs through the consideration of vertex labels relative to cache memory access patterns among certain classes of networks. Layered Label Propagation (LLP) [4], the Shingle ordering heuristic [5] and the Rabbit ordering algorithm [6], among others, all show results towards this goal. In an application setting, these methods demonstrate greater efficiencies within common analysis algorithms such as PageRank and community detection while prompting usage in graph compression and the development of compression-friendly orderings [7, 8]. While showing promising results, such reordering algorithms are often complex and focus on heuristics or greedy methods for approximation.

In this work, we seek to conduct an experimental study on the application of optimization to the vertex ordering problem for the improvement of CPU shared-memory parallel graph analysis methods. Specifically, we take inspiration from graph partitioning methods and propose a novel optimization method focusing on the explicit refinement of low-degree vertices within a graph. While known ordering methods achieve efficient cache access patterns and analysis runtimes using heuristics, we note that our optimization approach is easily applied to these generated orderings and shows promising results for optimization as a solution to the vertex ordering problem.

A. Contributions

Our degree-based refinement method builds upon any input initial ordering and improves ordering quality relative to our locality metrics in the Linear Gap Arrangement (LinGap) and Log Gap Arrangement (LogGap) problem towards improved analysis runtimes and cache efficiencies. We focus on three CPU shared-memory parallel graph analysis algorithms: PageRank, Multistep connectivity [9] and Louvain [10]. We apply and compare our refinement method to three heuristic ordering methods in the LLP, Shingle ordering and Rabbit ordering algorithms. Using an AMD system, we demonstrate runtime improvements of up to 15x on PageRank and over 2x on Multistep relative to the ClueWeb09 graph's natural ordering and 10-15x speedups on a number of graphs for the Louvain algorithm. On algorithm-generated orderings, we observe speedups between 1.1-3x and up to 2x improvement to cache efficiency. From this data, we state the main observations of our experimental study as follows:

- The LinGap and LogGap metrics show a strong positive correlation with PageRank analysis measures and slight correlations for the Multistep and Louvain measures.
- Spikes in the improvement of analysis measures occur at singular points within refinement progression, dependent on the graph structure.

- Refinement upon an initial Rabbit ordering shows the most overall improvement of analysis measures within our test data.
- The application of optimization methods to the vertex ordering problem shows promising improvements upon heuristic methods.

II. Related Works and Background

A. Related Works

A number of existing vertex ordering algorithms take intuitive approaches to reordering vertices towards efficient graph analysis in main memory. Presented in [5], the Shingle ordering heuristic focuses on ordering vertices relative to the measured commonality between two vertices' neighborhoods. This ordering scheme shows high compression rates relative to the natural ordering for social networks in particular. The Layered Label Propagation ordering method in [4] applies typical label propagation methods for compression optimization while considering global label states. This algorithm shows noticeable improvements to the compression rates of web graphs compared to the Apostolico and Drovandi algorithm in [11].

Alternatively, the Rabbit Order algorithm proposed in [6] focuses on runtime efficiency by optimizing towards cache efficiency directly rather than compression rates. This algorithm focuses on developing a hierarchy within real-world graphs and connecting it to cache hierarchies for an intuitive sense of cache locality. Communities are then developed and extracted from this hierarchy and a new vertex ordering is generated. This method shows notable speedup in parallel relative to compression-based ordering schemes while improving the runtimes of sparse matrix-vector multiplication (SpMV) analytic algorithms, such as PageRank and Label Propagation [12]. This focus on SpMV extends generally into matrix ordering contexts. Specifically, we see applications to the similar fill-reducing ordering problem, where methods attempt to apply heuristics in the fill-reduction of matrices for graph structures, partitioning and general matrix orderings [13, 14].

B. Graph Ordering Problem

Defining the ordering problem as in [5], we consider an undirected graph $G = (V[0, n), E \subseteq V \times V)$ and seek a permutation $\pi : V \rightarrow N$ such that the chosen metric is minimized. To set our optimization goal, we reference some accepted metrics in graph ordering that, while not robust, provide a means of judging ordering quality based on a notion of vertex label locality.

C. Metrics

Considering the focus on vertex-centric models later justified in Section IV-A, we use this approach in our choice of metrics for refinement: the Linear Gap Arrangement and Log Gap Arrangement problems. We choose these metrics due to their demonstrated correlation with the runtimes and cache efficiencies of our parallel graph analytic algorithms. We discuss this in detail in Section V-A.

First, we reference the Minimum Linear Arrangement (MinLA) problem in [5], which considers the sum of the vertex label differences across G. This is defined formally as $LA(G, \pi) = \sum_{u,v \in E} |\pi(u) - \pi(v)|$. We use the extension of this, the Linear Gap Arrangement (LinGap) problem, which considers the sum of the differences of each vertex's sorted neighborhood, excluding itself. Given a vertex u and its sorted neighborhood $sN(u) = \{v_1, v_2, ..., v_d\}$, this is defined as $LinGap(G, \pi) = \sum_{u \in N} \sum_{v_i \in sN(u)} |v_i - v_{i+1}|$.

Similarly, we reference the Minimum Log Arrangement (MinLogA) problem in [5], which considers the log of the sum of the vertex label differences across G, formally defined as $LA(G, \pi) = \sum_{u,v \in E} log(|\pi(u) - \pi(v)|)$. We again reference the Log Gap Arrangement (LogGap) problem, which considers the log of the sum of the differences of each vertex's sorted neighborhood, excluding itself. Under the same conditions as LinGap, LogGap is defined as $LogGap(G, \pi) = \sum_{u \in N} \sum_{v_i \in sN(u)} log(|v_i - v_{i+1}|)$.

Shown in [15] and furthered in [5], both MinLA and MinLogA are determined to be NP-hard over most types of graphs, warranting the usage of either heuristics or greedy approaches [5, 16] to the problems in application. This is extended to the LinGap and LogGap problems. Graph compression schemes have also been studied for the approximation of ordering quality [17]. Approximation methods have also been explored for the MinLA problem and related metrics, such as the Minimum Containing Interval Graph and the Minimum Storage-Time Product in [18].

III. Methods

As our primary method to evaluate explicit order refinement, we propose a degree-based refinement method for the explicit optimization of the vertex labels of low-degree vertices across a graph, given some initial vertex ordering. We consider our approach as a proof-of-concept that helps motivate, via our experimental study, further study into explicit order refinement algorithms. A more in-depth development of highly scalable and efficient methods for optimization is reserved for future work.

A. Degree-based Refinement

Based on our optimization approach, we employ a degree-based refinement method in parallel to improve an initial ordering towards one of the chosen metrics. Note that the initial ordering can simply be the natural ordering or any algorithm-generated ordering.

1) Metric Computation: For the explicit refinement of a chosen metric, we utilize a global and local calculation of metrics.

In the global case, we proceed in parallel through G's vertex set V with each vertex calculating its gap metric among its sorted adjacency list. Note that the gap metric and the sorting of the adjacency list proceeds relative to the input label map, determined as the initial ordering. Since we perform this calculation among vertices in parallel, each vertex reduces its local metric value into a global metric value for the ordering.

Local metric calculations focus on the two-hop neighborhood of two input vertices for performance improvements.

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
```

Considering the Log Gap Arrangement Refinement algorithm in Algorithm 1, the algorithm proceeds in parallel as follows: given an undirected graph G defined as previously, we first sort V in ascending order based on each vertices' degree. We then focus on two major operations: the Desired Swap (DS) step and the Swap Completion (SC) step. Note that the translation to a LinGap-based implementation for each step is trivial.

2) Desired Swaps: In the DS step, we have each vertex within an input fraction of G compute its most preferred label swap along each edge among its neighbors, decided by calculated improvement upon the LogGap metric. Thus, each vertex, in parallel, simulates a swap with each of its neighbors and compares the metric value before and after the simulated swap. Each vertex then saves its desired swap and continues onto the SC step.

3) Swap Completion: In the SC step, we sequentially iterate through the list of desired swaps and perform the swap if the metric improvement still holds. We are required to check this condition due to the possible infringement of previous confirmed swaps on future desired swaps. Note that the we perform the metric test locally. This continues up to the set fraction of low-degree vertices in G.

IV. Experimental Setup

With the goal of comparing our refinement algorithm's performance relative to metric improvement, analysis runtimes and cache efficiency, we measure cache efficiency utilizing the Linux perf tool on runs of three shared-memory parallel graph analytic algorithms: PageRank, Louvain [10] and the Multistep[1] connectivity algorithm [28]. We define cache-efficiency relative to cache miss percentages of L1 and L3 cache in relation to overall cache accesses for each. Towards the calculation of these measures, we proceed to describe our experimental setup.

TABLE I
Basic graph properties

Graph	Class	#Vertices	#Edges	Cite
com-Friendster	Social	66 M	1.8 B	[19]
twitter-2010	Social	41.7 M	1.5 B	[20]
LiveJournal	Social	4.8 M	69 M	[21]
web-ClueWeb09	Web Graph	1.7 B	7.9 B	[22]
enwiki-2013	Web Graph	4.2 M	101.3 M	[4]
web-BerkStan	Web Graph	685 K	7.6 M	[23]
it-2004	Web Graph	41.3M	1.2 B	[24]
ant1km	Mesh	13.5 M	53.8 M	[25]
trianglemesh1	Mesh	1.9 M	1.9 M	[26]
USA-road-d	Road	24 M	58.3 M	[27]

A. Memory Access

In terms of memory access patterns in graph analysis, we consider general patterns alongside our employed algorithms: PageRank, Multistep connectivity and Louvain. One of the more general approaches considers the "think like a vertex" (TLAV) framework for patterns of vertex-centric access. Such a method, by nature, improves vertex locality while allowing for scalable means of processing. A comprehensive survey of TLAV frameworks and methods is included in [29]. For our analysis algorithms, we note the SpMV basis of the PageRank algorithm yields cache misses due to poor locality within the compressed sparse row (CSR) format of adjacency storage. Further definition of CSR locality within SpMV is mentioned in [6]. For the Multistep connectivity algorithm, focus lies in the initial traversal-based approach with a secondary propagation phase, utilizing BFS-based patterns of memory access among label propagation schemes. The Louvain algorithm accesses the neighbor information for all vertices in a graph and focuses on graph coarsening, which is relative to vertex locality and ordering dependent.

Within our experimental study, we note that all of our graph analysis algorithms function under a vertex-centric approach with CPU-based shared-memory parallelism. We make this distinction to denote the focus of our study on such parallel models of graph analysis methods despite algorithmic differences.

B. Data

We present results using a variety of network topologies, specifically social networks, web graphs, meshes and a road network. The properties of each network are included in Table I. Our graphs are collected from well known data sources such as SNAP, DIMACS and WebGraph.

[1]https://github.com/HPCGraphAnalysis/Connectivity

C. Architecture

We use an AMD system for the collection of cache miss and runtime results. The AMD system has 2TB of DDR4 RAM and dual-socket NUMA AMD EPYC 7742 64-core processors and 2 threads per core with a clock speed of 1500MHz. Each core has a 4MiB L1 instruction cache, 4MiB L1 data cache, a 64MiB L2 cache and 256MiB of shared L3 cache per socket. The AMD system operates on Ubuntu version 20.04.4 LTS.

TABLE II
RELATIVE METRIC CORRELATION COEFFICIENTS FOR ALGORITHM ANALYSIS METRICS

Metric	LinGap	LogGap
PageRank Cache	0.2568	0.3196
PageRank Timing	0.7041	0.8796
Multistep Cache	-0.0005	-0.0006
Multistep Timing	0.0088	0.0090
Louvain Cache	-0.0093	-0.0007
Louvain Timing	0.0387	0.0477

D. Experimentation

Our degree-based refinement method, alternative ordering algorithms (Layered Label Propagation, Shingle and Rabbit[2]) and our parallel graph analytic algorithms (PageRank, Multistep connectivity[1] and Louvain) are implemented in C++ and compiled using GCC version 9.4.0 and OpenMP version 4.5 for shared-memory parallelism. The Linux perf tool is used for collecting L1 cache and overall cache miss rates with all reported results being the arithmetic average of ten runs per graph analytic algorithm. The perf tool events used are cache-references, cache-misses, L1-dcache-load and L1-dcache-load-miss. We run PageRank on each graph for twenty iterations and we set the max iterations for our Louvain runs at five. Mesh generation for the trianglemesh1 graph occurs on Python3 version 3.8.10 and Pygalmesh version 0.10.6.

V. RESULTS

In this section, we present and evaluate experimental results. Our contributions here are two-fold: (1) an experimental study into LinGap and LogGap as metrics for cache efficiency and analysis times; (2) a comparative evaluation of our presented degree-based refinement method to improve ordering quality relative to these metrics. We first discuss results pertaining to the relationship between the amount of degree-based refinement and graph analytic efficiencies. We then perform a comprehensive suite of tests using our refinement method and the ordering algorithms discussed in Section II-A and discuss the results.

A. A Motivating Case for LinGap and LogGap

We begin evaluation by considering the LinGap and LogGap problems as our metrics for refinement. The choice is intuitive considering their quantified notion of vertex locality through the measured label gaps within a neighborhood. Similarly, each problem provides an explicit refinement focus for optimization and serves as a general measure for how "different" our refined ordering is from an input initial ordering. Thus, we consider the correlation between the quality of an ordering relative to the measures we are interested in: cache efficiency and analytic runtime. We take the relative measure of ordering quality relative to that graph's worst metric while relative time and cache efficiency follow. Results are collected on the AMD system and are taken across all graphs and orderings. The results for LinGap and LogGap correlation are shown in Table II. The PageRank results, alongside the intuitive notion of the LinGap and LogGap metrics as a vertex locality measure, show promise in the exploration and application, even if the Multistep and Louvain algorithm results suggest a potentially complex relationship.

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

B. Degree-based Approach

Building on the correlation in Section V-A, we collect results using each degree-based refinement method to observe metric growth progression within refinement. Specifically, we conduct degree-based refinement for each metric using the LiveJournal graph on intervals between 0.1% and 5.0% of the low-degree vertices and analyze the metric improvement for each interval. A similar trend in these results exists for both metrics in improvement with significant improvement starting around the 1% marker, or approximately 48,000 low-degree vertices refined. We use this marker to collect our comprehensive results. From this, we draw one main observation: the largest improvement in analytic measures tend to occur directly at a single threshold percentage, with minimal improvement otherwise. We attribute this to the existence of key vertex-label pairings within an ordering, where the refinement of such pairings yields high analysis metric improvements, suggesting potential in the identification of such pairings before refinement.

C. Analytic Performance

First, we describe our refinement process for experimentation. We perform degree-based refinement on each graph in Table I using between 10% and 0.001% of the graph's low-degree vertices, determined based on the threshold marker

[2]https://github.com/araij/rabbit_order

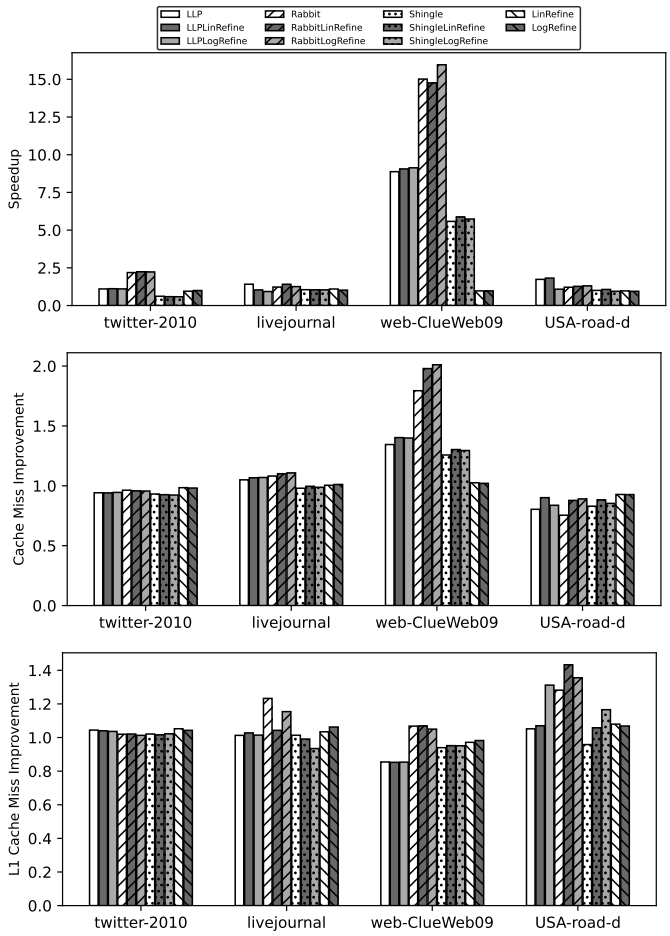

Fig. 1. Cache miss improvement and algorithm speedup relative to the natural ordering for the PageRank algorithm. Improvement and speedup is taken as a fraction of the performance of the natural ordering over each ordering method's performance as recorded using the AMD system.

referenced in Section V-B. This method is applied to each graph once per initial ordering and once per metric. We collect improvement and speedup results relative to the natural ordering of each ordering method for each of the graph analytic algorithms. The results using the AMD system for the PageRank, Multistep connectivity and Louvain algorithms are presented in Figures 1 and 2. For conciseness, we provide results of improvement taken as the geometric mean across all of our analytic algorithms in Table III. We also omit comprehensive results in Figures 1 and 2 for conciseness and instead display the most notable results.

Examining the results for the PageRank algorithm in Figure 1, we notice cache miss improvements remaining relatively consistent with that of the natural ordering for all graphs except for ClueWeb09, where a near 2x improvement is reached using LogGap refinement on a Rabbit ordering. Similarly, PageRank timing shows up to a 15x speedup. We note that ClueWeb09 is the largest graph in our study and attribute such an improvement to the limitations of graph access in main memory, resulting in the large impact of improvement upon ordering quality. Similarly, the application of the hierarchical method of a Rabbit ordering alongside LogGap refinement showing significant results for the PageRank algorithm is likely due to the optimization of outliers within the hierarchical communities towards the centralized adjacency matrix under a SpMV-focused analysis. The Multistep connectivity algorithm results in Figure 2 shows similar trends, but with timing and cache improvements closer to 2x. An interesting distinction from the PageRank results occurs in the maximum improvement through the LinGap refined Rabbit ordering instead of the LogGap refinement. This is likely due to the traversal-based nature of the Multistep connectivity algorithm, which focuses on the gaps among neighboring vertices rather than a scaled measure. In terms of refinement efficacy, we again see a fair distribution of maximized cases that include an applied refinement across the analytic metrics.

The Louvain algorithm's results in Figure 2 show distinct trends compared to those of the previous analytic algorithms, yielding higher levels of cache miss improvement - up to 3x-4x. L1 cache misses show significantly higher miss rates for the graphs that demonstrate high overall cache miss improvement. Speedup values achieve around 15x for our Twitter graph while including 10x speedups for a mesh and the road network. We can observe that the focus on refining low-degree

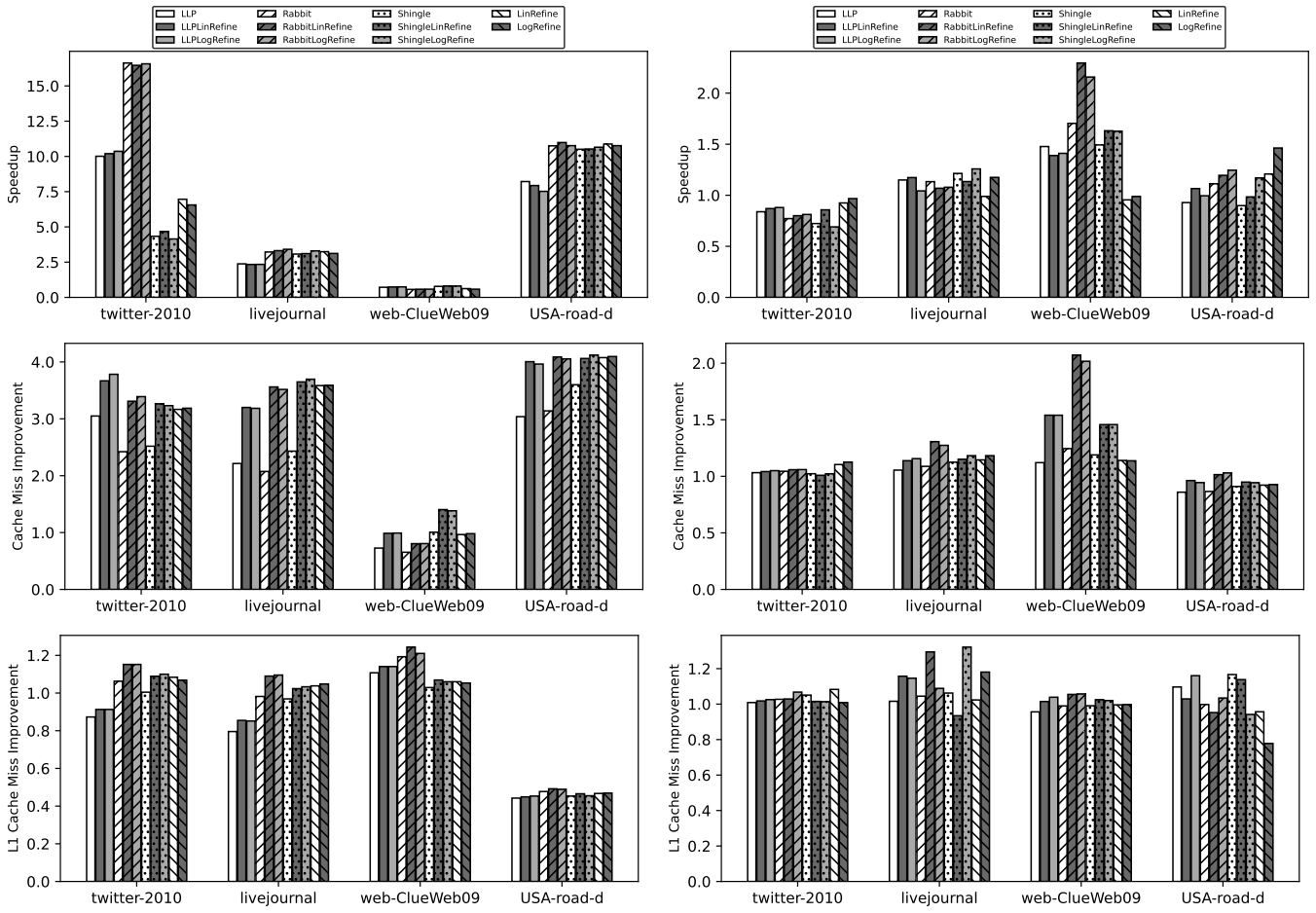

Fig. 2. Cache miss improvement and algorithm speedup relative to the natural ordering for the Louvain (L) and Multistep (R) algorithms. Improvement and speedup is taken as a fraction of the performance of the natural ordering over each ordering method's performance as recorded using the AMD system.

vertices under a coarsening model shows high levels of cache miss improvement for graphs of moderate density and can result in analytic speedups.

To obtain a general measure for overall performance per analytic metric, we take the geometric mean of improvement values across all graphs and graph analytic algorithms for each ordering method. The results from runs on the AMD system are compiled in Table III. Highlighted are the highest improvement totals across each of analysis metrics. We observe that applying refinement achieves the highest overall improvement for all cases across the three analytic measures, especially under a Rabbit ordering. Since our analysis algorithms all consider a vertex-centric approach through SpMV, traversal and coarsening, it holds that a community-based approach with explicit refinement within communities would show locality improvements.

VI. Conclusion

This paper has explored a new area within vertex ordering that considers the explicit optimization of the Linear Gap Arrangement and Log Gap Arrangement problem metrics. We analyze this optimization relative to the analytic metrics of cache efficiency and analysis runtime for the CPU shared-memory parallel PageRank, Multistep connectivity and Louvain graph analysis algorithms. Specifically, we demonstrated our two-fold results: (1) the evaluation of the LinGap and LogGap metrics to cache efficiency and analysis timings; (2) the presentation and performance of a comparative analysis of our degree-based explicit refinement method for optimizing ordering quality. Our focus on the LinGap and LogGap metrics is driven by a positive correlation among PageRank cache and timing results while observing slight correlations for the Multistep connectivity and Louvain algorithms. We follow this with a degree-based refinement method that explicitly refines low-degree vertices. This approach shows promising improvements to cache efficiency and analysis timings compared to ordering heuristics and methods such as Layered Label Propagation, Shingle ordering and Rabbit ordering. These improvements demonstrate the viability of an explicit optimization-based approach under the heuristic-focused vertex ordering problem.

VII. Future Works

Developed as introductory work towards an optimization-based approach to vertex ordering for efficient graph analysis,

a number of directions for progression exist. The degree-based refinement method can be further evaluated through testing on graphs on a similar scale to that of ClueWeb09 or graph sets with diverse class distribution. Similarly, a wider variety of graph analytic algorithms could see improvements from explicit refinement methods, especially under parallel models differing from that of shared-memory. We plan on evaluating the degree-based refinement method using GPU-implemented analytic algorithms and performing a comparison to current results.

Furthermore, we plan to develop alternate explicit refinement methods with similarities drawn from common graph partitioning methods for improved parallel speedup. This naturally leads to the consideration of explicit subgraph optimization in vertex ordering. Such a method would consider optimal edge cuts in partitioning for explicit ordering refinement and the preservation of global ordering schemes within subgraphs. Spectral partitioning methods and multi-level methods similar to that of graph coarsening could also show improvement in application within explicit refinement methods.

Methods within the field of optimization can be applied to a graph-focused model under an explicit mapping of vertex labels. Similar attempts were made using linear and non-linear programming models relative to each metric, but were ultimately runtime infeasible in our specific application. Further work towards this would allow for the efficient application of solver models to subgraphs specifically partitioned for ordering optimization, providing a relative optimal ordering.

VIII. Acknowledgements

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

References

[1] M. Frasca, K. Madduri, and P. Raghavan, "Numa-aware graph mining techniques for performance and energy efficiency," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Washington, DC, USA: IEEE Computer Society Press, 2012.

[2] D. Lasalle and G. Karypis, "Multi-threaded graph partitioning," in *Proceedings of the 2013 IEEE 27th International Symposium on Parallel and Distributed Processing*, ser. IPDPS '13. USA: IEEE Computer Society, 2013, p. 225–236. [Online]. Available: https://doi.org/10.1109/IPDPS.2013.50

[3] A. Lumsdaine, D. Gregor, B. Hendrickson, and J. Berry, "Challenges in parallel graph processing." *Parallel Processing Letters*, vol. 17, pp. 5–20, 03 2007.

[4] P. Boldi, M. Rosa, M. Santini, and S. Vigna, "Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks," 2010. [Online]. Available: https://arxiv.org/abs/1011.5425

[5] F. Chierichetti, R. Kumar, S. Lattanzi, M. Mitzenmacher, A. Panconesi, and P. Raghavan, "On compressing social networks," in *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 219–228. [Online]. Available: https://doi.org/10.1145/1557019.1557049

[6] J. Arai, H. Shiokawa, T. Yamamuro, M. Onizuka, and S. Iwamura, "Rabbit order: Just-in-time parallel reordering for fast graph analysis," in *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, 2016, pp. 22–31.

[7] Y. Lim, U. Kang, and C. Faloutsos, "Slashburn: Graph compression and mining beyond caveman communities," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 26, pp. 3077–3089, 12 2014.

[8] I. Safro and B. Temkin, "Multiscale approach for the network compression-friendly ordering," *Journal of Discrete Algorithms*, vol. 9, no. 2, pp. 190–202, 2011.

[9] G. M. Slota, S. Rajamanickam, and K. Madduri, "Bfs and coloring-based parallel algorithms for strongly connected components and related problems," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 550–559.

[10] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2008, no. 10, p. 10008, Oct. 2008.

[11] A. Apostolico and G. Drovandi, "Graph compression by bfs," *Algorithms*, vol. 2, no. 3, pp. 1031–1044, 2009, copyright - Copyright MDPI AG 2009; Last updated - 2014-05-20. [Online]. Available: https://www.proquest.com/scholarly-journals/graph-compression-bfs/docview/1525986426/se-2

[12] D. Zhou, O. Bousquet, T. Lal, J. Weston, and B. Olkopf, "Learning with local and global consistency," *Advances in Neural Information Processing Systems 16*, vol. 16, 03 2004.

[13] Ü. V. Çatalyürek, C. Aykanat, and E. Kayaaslan, "Hypergraph partitioning-based fill-reducing ordering for symmetric matrices," *SIAM Journal on Scientific Computing*, vol. 33, no. 4, pp. 1996–2023, 2011.

[14] G. Karypis and V. Kumar, "A parallel algorithm for multilevel graph partitioning and sparse matrix ordering," *Journal of parallel and distributed computing*, vol. 48, no. 1, pp. 71–95, 1998.

[15] M. R. Garey, D. S. Johnson, and L. J. Stockmeyer, "Some simplified np-complete graph problems," *Theor. Comput. Sci.*, vol. 1, no. 3, pp. 237–267, 1976. [Online]. Available: https://doi.org/10.1016/0304-3975(76)90059-1

[16] K. Zhao, Y. Rong, J. X. Yu, W. Huang, J. Huang, and H. Zhang, "Graph ordering: Towards the optimal by learning," in *International Conference on Web Information Systems Engineering*. Springer, 2021, pp. 423–437.

[17] L. Dhulipala, I. Kabiljo, B. Karrer, G. Ottaviano, S. Pupyrev, and A. Shalita, "Compressing graphs and indexes with recursive graph bisection," in *Proceedings of the 22nd ACM SIGKDD International Conference*

on Knowledge Discovery and Data Mining, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1535–1544. [Online]. Available: https://doi.org/10.1145/2939672.2939862

[18] M. Charikar, M. T. Hajiaghayi, H. Karloff, and S. Rao, "L22 spreading metrics for vertex ordering problems," in *Proceedings of the Seventeenth Annual ACM-SIAM Symposium on Discrete Algorithm*, ser. SODA '06. USA: Society for Industrial and Applied Mathematics, 2006, p. 1018–1027.

[19] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," 2012. [Online]. Available: https://arxiv.org/abs/1205.6233

[20] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proceedings of the 19th International Conference on World Wide Web*, ser. WWW '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 591–600. [Online]. Available: https://doi.org/10.1145/1772690.1772751

[21] L. Backstrom, D. Huttenlocher, J. Kleinberg, and X. Lan, "Group formation in large social networks: Membership, growth, and evolution," in *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '06. New York, NY, USA: Association for Computing Machinery, 2006, p. 44–54. [Online]. Available: https://doi.org/10.1145/1150402.1150412

[22] J. Callan, M. Hoy, C. Yoo, and L. Zhao, "Clueweb09 data set," 2009. [Online]. Available: http://www.lemurproject.org/clueweb09/index.php

[23] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters," 2008. [Online]. Available: https://arxiv.org/abs/0810.1355

[24] P. Boldi and S. Vigna, "The WebGraph framework I: Compression techniques," in *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*. Manhattan, USA: ACM Press, 2004, pp. 595–601.

[25] I. K. Tezaur, M. Perego, A. G. Salinger, R. S. Tuminaro, and S. F. Price, "Albany/felix: a parallel, scalable and robust, finite element, first-order stokes approximation ice sheet solver built for advanced analysis," *Geoscientific Model Development*, vol. 8, no. 4, pp. 1197–1220, 2015. [Online]. Available: https://gmd.copernicus.org/articles/8/1197/2015/

[26] N. Schlömer, "pygalmesh: Python interface for cgal's meshing tools," 2019. [Online]. Available: https://github.com/nschloe/pygalmesh

[27] D. Schultes, "The shortest path problem," 2005. [Online]. Available: http://www.diag.uniroma1.it//challenge9/data/tiger/

[28] G. M. Slota, S. Rajamanickam, and K. Madduri, "Bfs and coloring-based parallel algorithms for strongly connected components and related problems," in *2014 IEEE 28th International Parallel and Distributed Processing Symposium*, 2014, pp. 550–559.

[29] R. R. McCune, T. Weninger, and G. Madey, "Thinking like a vertex: A survey of vertex-centric frameworks for large-scale distributed graph processing," *ACM Comput. Surv.*, vol. 48, no. 2, oct 2015. [Online]. Available: https://doi.org/10.1145/2818185