Improving Graph Compression for Efficient Resource-Constrained Graph Analytics

Algorithm Engineering

Jinha Kim

Outline

- Overview of Compression and Graph Analytics
- Encoding Based / Rule Based Compression
- Historical Background
- Laconic System Design
- Empirical Results
- Strengths and Weaknesses

Improving Graph Compression for Efficient Resource-Constrained Graph Analytics

Qian Xu
Renmin University of China
xuqian@ruc.edu.cn

Juan Yang
Beijing HaiZhi XingTu Technology Co., Ltd
yangjuan@stargraph.cn

Feng Zhang*
Renmin University of China
fengzhang@ruc.edu.cn

Zheng Chen
Renmin University of China
chenzheng123@ruc.edu.cn

Jiawei Guan
Renmin University of China
guanjw@ruc.edu.cn

Kang Chen
Tsinghua University
chenkang@tsinghua.edu.cn

Ju Fan
Renmin University of China
fanj@ruc.edu.cn

Youren Shen
Beijing HaiZhi XingTu Technology Co., Ltd
shenyouren@stargraph.cn

Ke Yang
Beijing HaiZhi XingTu Technology Co., Ltd
yangke@stargraph.cn

Yu Zhang
Renmin University of China
yu-zhang21@ruc.edu.cn

Xiaoyong Du
Renmin University of China
duyong@ruc.edu.cn

ABSTRACT

Recent studies have shown the promise of directly processing compressed graphs. However, its benefits have been limited by high peak-memory usage and unbearably long compression time. In this paper, we introduce Laconic, a novel rule-based graph processing solution that overcomes the challenges of restricted memory and impractical compression time faced by existing approaches. Laconic, for the first time, ensures minimal memory overhead during compression and significantly reduces graph sizes, thus reducing peak memory demand during computations. By employing an efficient parallel compression algorithm, Laconic achieves a remarkable reduction in compression time. In our experiments, we compare Laconic with state-of-the-art solutions. The results demonstrate that Laconic outperforms other methods, reducing peak memory consumption by an average of 70% during compression and 66% during computation. Additionally, Laconic reduces rule compression time by an average of 93% compared to traditional rule-based compression, achieving a 2.47× higher compression ratio, and providing a 2.12× performance speedup.

PVLDB Reference Format:
Qian Xu, Juan Yang, Feng Zhang, Zheng Chen, Jiawei Guan, Kang Chen, Ju Fan, Youren Shen, Ke Yang, Yu Zhang, and Xiaoyong Du. Improving Graph Compression for Efficient Resource-Constrained Graph Analytics. PVLDB, 17(9): 2212 - 2226, 2024.
doi:10.14778/3665844.3665852

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/xuqianmamba/Laconic.

*Feng Zhang is the corresponding author.

Proceedings of the VLDB Endowment, Vol. 17, No. 9 ISSN 2150-8097.
doi:10.14778/3665844.3665852

1 INTRODUCTION

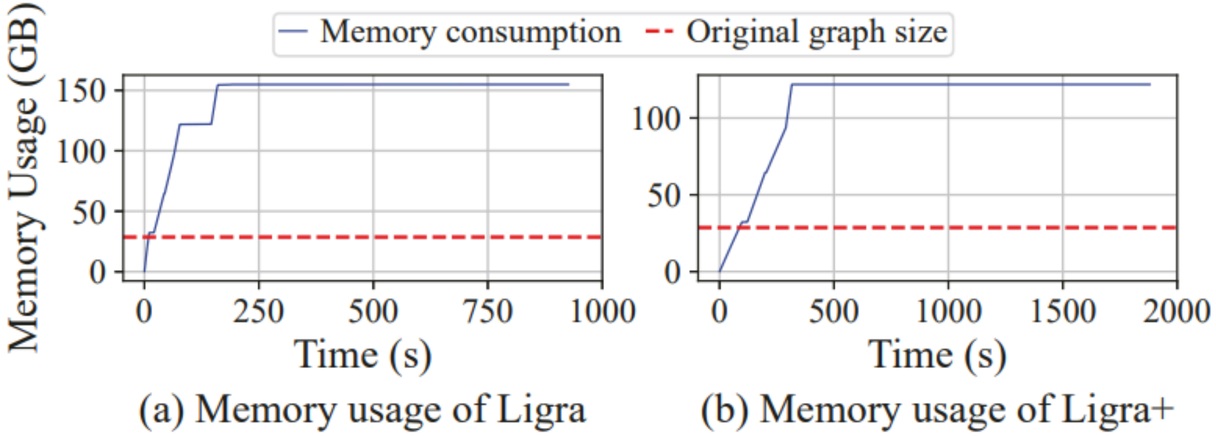
The ever-increasing graph sizes have fueled a surge in graph compression research [5, 9, 10, 12, 22, 25, 31, 49, 69, 70, 90, 91, 94, 96]. Recent studies have highlighted the potential of direct processing on compressed graphs [13, 25], yielding promising results in terms of enhancing graph analytics efficiency. However, despite the efficiency advantages inherent in these methods, a notable oversight persists concerning the consideration of memory consumption and compression time. Typically, these methods demand several times the original graph's size in memory resources and necessitate extended compression time. For example, Figure 1 illustrates the memory consumption of the state-of-the-art in-memory graph computing system Ligra [91] and its implementation with a compression module, Ligra+ [92], during the execution of the PageRank computation and compression tasks on the 28.65 GB *uk-2007-05* graph, respectively. Although Ligra+ effectively reduces the size of the graph, there is no significant decrease in peak memory usage, which is 121.85GB, 4.2 times the size of the original graph, during the transition from compression to computation. Moreover, Ligra+ needs 605 seconds to compress *uk-2007-05* graph, which is unbearable in practice. Similar limitations also exist in CompressGraph [20], the state-of-the-art rule-based graph compression framework, whose peak memory usage is 65.38 GB and compression time is 700 seconds handling *uk-2007-05*. This resource overhead hinders their broad adoption. In order to fully harness the potential of direct computation techniques, there is an urgent need for an effective solution that concurrently minimizes memory utilization and compression time.

Direct processing compressed graphs in resource-constrained environment yields three compelling advantages. First, performing computations directly on compressed data has been demonstrated to significantly enhance graph analytics performance [3, 34, 53, 74, 75]. On one hand, analytical algorithms can leverage the precomputed results from redundant data, thereby reducing computational overhead. On the other hand, compressed graphs can be stored

Graph Analytics

- Analyzing data stored in the form of a graph (nodes as entities and edges as relationships)
- Examples are:
  - Path finding or reachability
  - Compute Centrality measures
- Many graph analytic frameworks exist such as Ligra (in-memory and multicore) and Ligra+ (additional compression module for graphs larger than in-memory)

Compression for Graph Analytics



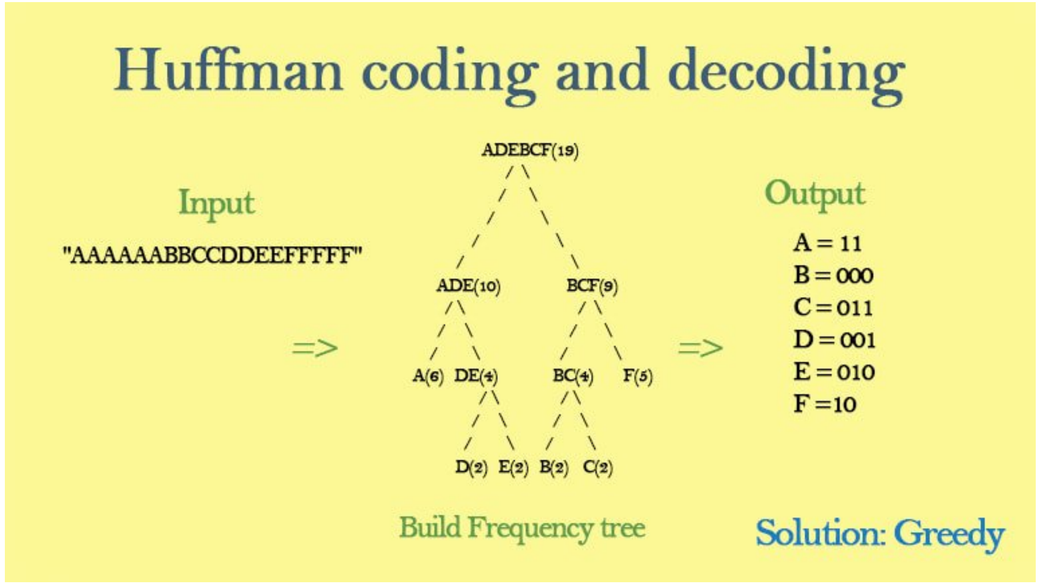
(a) Memory usage of Ligra (b) Memory usage of Ligra+

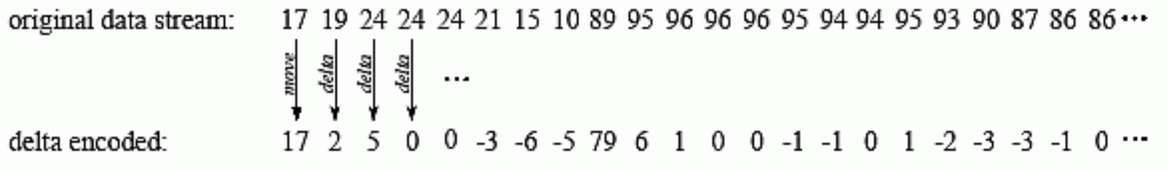
Figure 1: Memory consumption on *uk-2007-05* during graph processing.

- At the time there were many attempts to first compress large graphs for graph analytics (still issues with high peak-memory usage and long compression time)
- Benchmark memory consumption of Ligra+ (with compression module) and Ligra on execution of PageRank on 28.65 GB uk-2007-05 graph
- No significant decrease in peak memory (Ligra+ reaches 121.85 GB which is 4 times the original graph) Also need 605 seconds for compression time (too long)

Encoding Based Compression

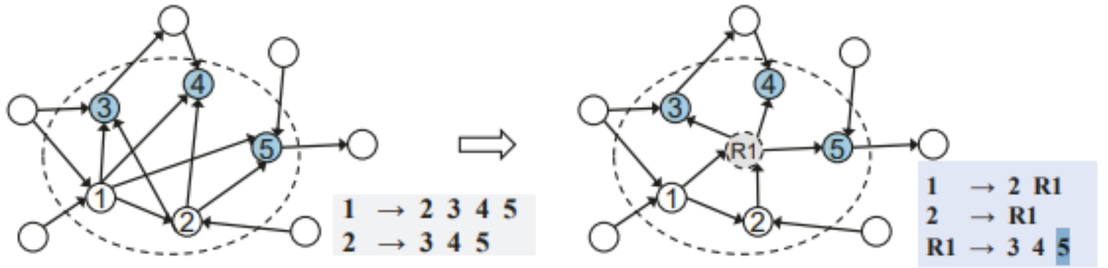
- Encode each vertex with few codes which allows memory saving
- Variable length encoding like Huffman encoding change the fixed size representations of vertices in adjacency lists (based on different vertex's occurrence frequency)
- Delta encoding works by taking sorted adjacency list of a vertex and taking consecutive differences
- In practice the two can be combined





Rule Based Compression

- Compression of data that identifies recurring patterns and encode them as rules
- Compress graph is an example of rule based compression (leverages rule vertices to remove repeated adjacent edges)



(a) Input graph in adjacency list format (b) Compressed graph in rule-based representation

Figure 2: An example of rule and rule-based compression.

Benefits & Challenges of Graph Compression for Analytics

Benefits

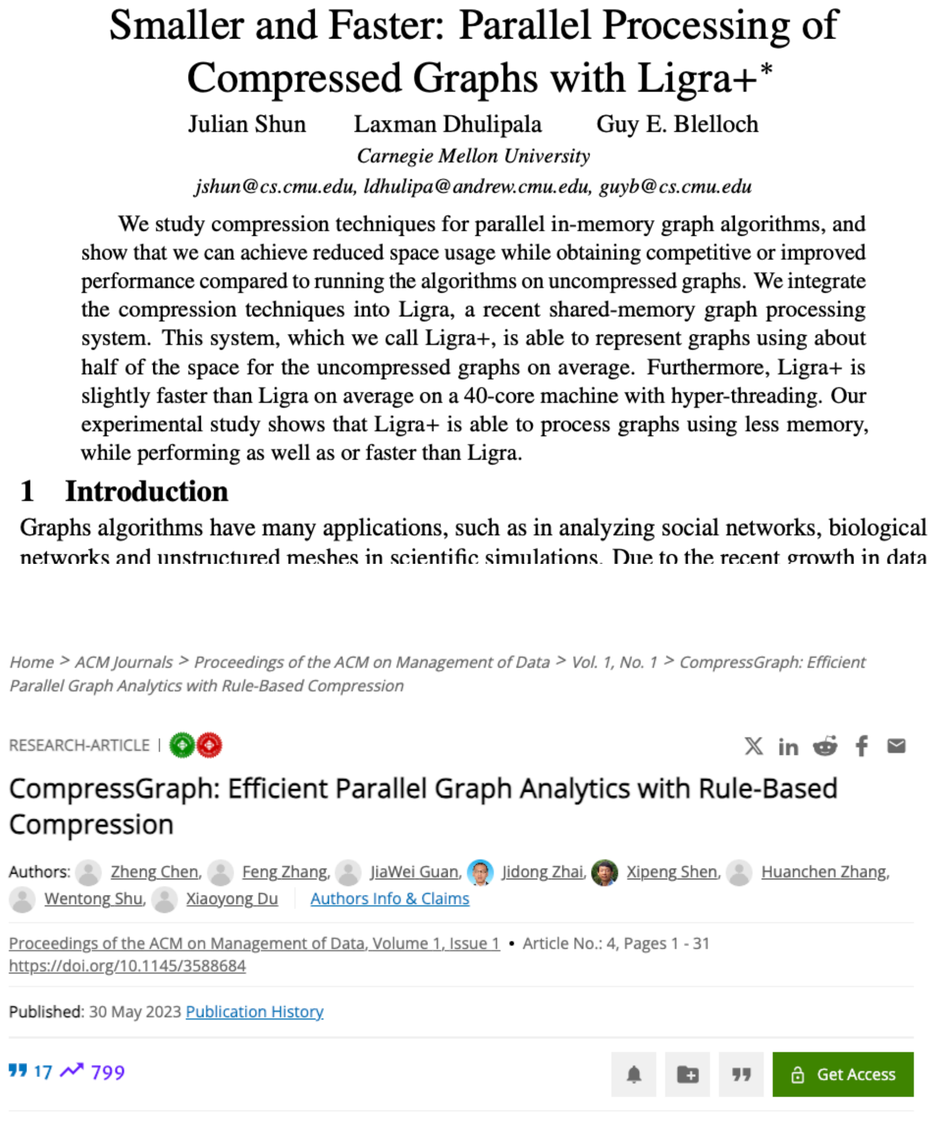
- Allows leveraging precomputed results on redundant data (via compression)
- Compressed graphs can be stored more efficiently in CPU cache (thus improving data locality)
- Well-designed graph compression allows reduction of peak memory consumption (as done by Laconic)
- Enable analysis of larger graphs in memory constrained systems

Challenges

- Balance minimizing memory usage and maximizing computational efficiency
- Certain graph compression may achieve high compression rates but may add notable compression time overhead
- Rule-based and reference compression have good tradeoffs between compression ratio and graph processing efficiency but results in prolonged compression duration & increased memory usage (not suitable for memory constrained scenarios)

Historical Background / Relevant Works

- Ligra+ supports is a framework that explores high-speed processing on compressed graphs expanding Ligra
- Compress graphs was state-of-the-art rule-based compression
- Relevant works for large graph computation tasks involve in-memory compression, disk-based computation (suffer from high I/O costs) and distributed computation (higher development and maintenance costs)
- WebGraph and K^2 tree are existing heavy compression algorithms with high compression ratios (but increase computation burden)



Smaller and Faster: Parallel Processing of Compressed Graphs with Ligra+*

Julian Shun Laxman Dhulipala Guy E. Blelloch

Carnegie Mellon University

jshun@cs.cmu.edu, ldhulipa@andrew.cmu.edu, guyb@cs.cmu.edu

We study compression techniques for parallel in-memory graph algorithms, and show that we can achieve reduced space usage while obtaining competitive or improved performance compared to running the algorithms on uncompressed graphs. We integrate the compression techniques into Ligra, a recent shared-memory graph processing system. This system, which we call Ligra+, is able to represent graphs using about half of the space for the uncompressed graphs on average. Furthermore, Ligra+ is slightly faster than Ligra on average on a 40-core machine with hyper-threading. Our experimental study shows that Ligra+ is able to process graphs using less memory, while performing as well as or faster than Ligra.

1 Introduction

Graphs algorithms have many applications, such as in analyzing social networks, biological networks and unstructured meshes in scientific simulations. Due to the recent growth in data

Home > ACM Journals > Proceedings of the ACM on Management of Data > Vol. 1, No. 1 > CompressGraph: Efficient Parallel Graph Analytics with Rule-Based Compression

RESEARCH-ARTICLE |

CompressGraph: Efficient Parallel Graph Analytics with Rule-Based Compression

Authors: Zheng Chen, Feng Zhang, JiaWei Guan, Jidong Zhai, Xipeng Shen, Huanchen Zhang, Wentong Shu, Xiaoyong Du | Authors Info & Claims

Proceedings of the ACM on Management of Data, Volume 1, Issue 1 • Article No.: 4, Pages 1 - 31
https://doi.org/10.1145/3588684

Published: 30 May 2023 Publication History

17 799

Get Access

Laconic Compression Modules

1. Memory aware adaptive graph slicing module (adaptively partitions the original graph into subgraphs taking into account available memory and ensure memory use stays under budget)

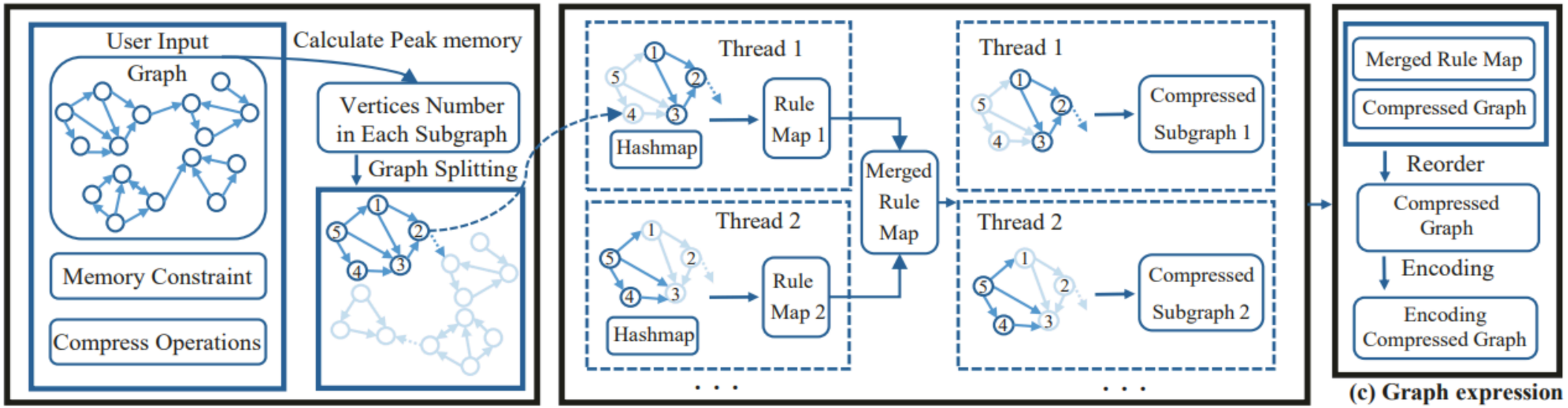
- Supports customizable configuration for precise control of peak memory usage of compression process

2. Parallel rule-based subgraph compression module (compresses each sliced subgraph and removes redundancies

3. Graph expression recovery module (compressed subgraph merging, vertex reordering, parallel encoding compression)

- The last module involves hybrid that combines rule-based compression with other compression schemes (higher compression ratios)

Laconic Compression Modules



(a) Memory-aware adaptive graph slicing module

(b) Parallel rule-based subgraph compression module

(c) Graph expression recovery module

Figure 3: Laconic overview.

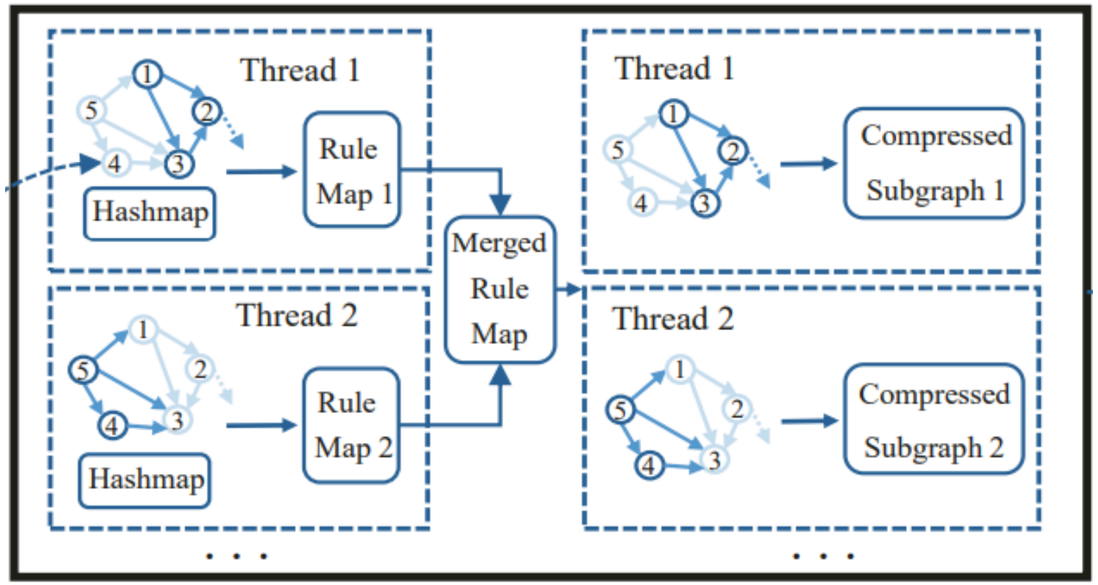
Memory-Aware Adaptive Graph Slicing Module

- Computes number of subgraphs and size of each subgraph to complete graph analytics under constrained memory limits
- Partition graph into subgraphs that avoids introducing edges that connect subgraphs (partition vertex and neighbors into different subgraphs)
- Controlling size of subgraph is important (if too small, lower redundancy and lower compression ratios and if too large, then excessive high peak memory)

Lemma 1. *By adjusting the size of the divided subgraph* $G_{sub}(|V_{sub}|, |E_{sub}|)$ *to satisfy* $O(\sqrt{|V_{sub}|} \cdot |V| + |E_{sub}|) \leq M$, *where* M *represents the available memory, we can complete the graph compression under memory constrained environment.*

Parallel Rule-Based Subgraph Compression

- Laconic's Rule-Based compression only captures rules of size 2 in each iteration (differs from traditional rule-based compression)
- Laconic also further subdivides subgraphs and parallelize over parts of it followed by merging their compressed subgraph outputs



(b) Parallel rule-based subgraph compression module

Parallel Rule-Based Subgraph Compression

- Traditional rule traversal finds most frequent (not necessary binary) rule sequentially (affects parallelism) while Laconic can traverse all frequent binary rules in one round and select those above freq count

- Laconic allows parallelization and instead of having to rewrite after each rule (multiple rounds can create non-binary rules in the end)

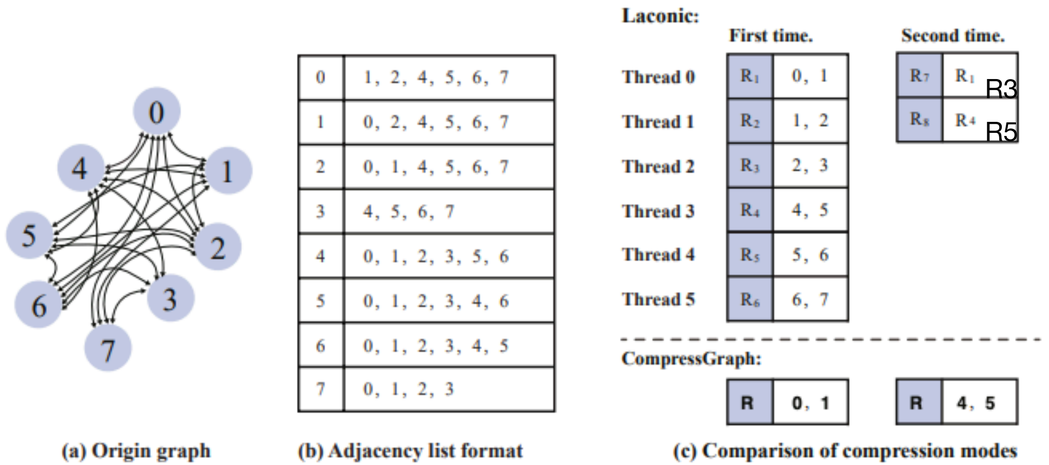


Figure 4: Rule generation comparison between Laconic and traditional rule-based compression.

Parallel Rule-Based Subgraph Compression

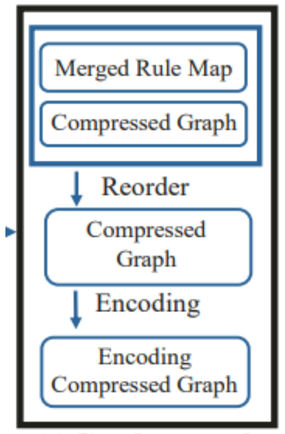
- First parallel stage: Laconic assigns subgraphs to threads and each thread hashes and inserts every consecutive binary pair to some bucket.
- Second parallel stage: Each thread are assigned buckets to count occurrences of pairs within and filter out pairs below frequent threshold. After that assign RuleIds for each
- Third parallel stage: Laconic re-traverses subgraph and replaces queued rules in parallel with non-overlap policy

Algorithm 1: Parallel rule-based subgraph compression

```
1  Function lock_free_rule_based_compression(G = (V, E),
     iterations_num γ, frequent_threshold δ, thread_num λ):
2      subgraphs[] ← split_graph(G)
3      buckets ← create a mapping from a pair to 0..λ
4      for i ← 0 to γ do
5          init_hash_buckets(buckets)
6          for sg in subgraphs do
7              hash_pair_to_buckets(sg, buckets)
8              filter_and_assign_rule_ids(buckets, δ)
9              sg ← find_and_replace_rules(sg, buckets)
10     filter_infreq_rules(subgraphs)
11     CG ← merge_all_subgraphs(subgraphs)
12     return CG
13 Function hash_pair_to_buckets(subgraph, buckets):
14     for v in subgraph do
15         D ← degree[v]
16         adj ← adjacency list of v
17         for i ← 0 to D − 1 do
18             buckets.insert(pair_hash(adj[i], adj[i + 1]))
19 Function filter_and_assign_rule_ids(buckets, δ):
20     for bucket in Buckets do
21         count_freq_and_remove_infreq(bucket, δ)
22         for pair in bucket do
23             assign_id_to_rules(pair)
```

Graph Expression Recovery Module

- Filter infrequent rules and remove them and merge the compressed subgraphs together
- Reordering process (renumbering nodes' ID) to restore original vertex arrangement disrupted by rule compression. (that might result in greater "gaps")
- Includes parallel encoding compression sub-module (Delta encoding) on the reordered compressed graph enhance compression rate



(c) Graph expression recovery module

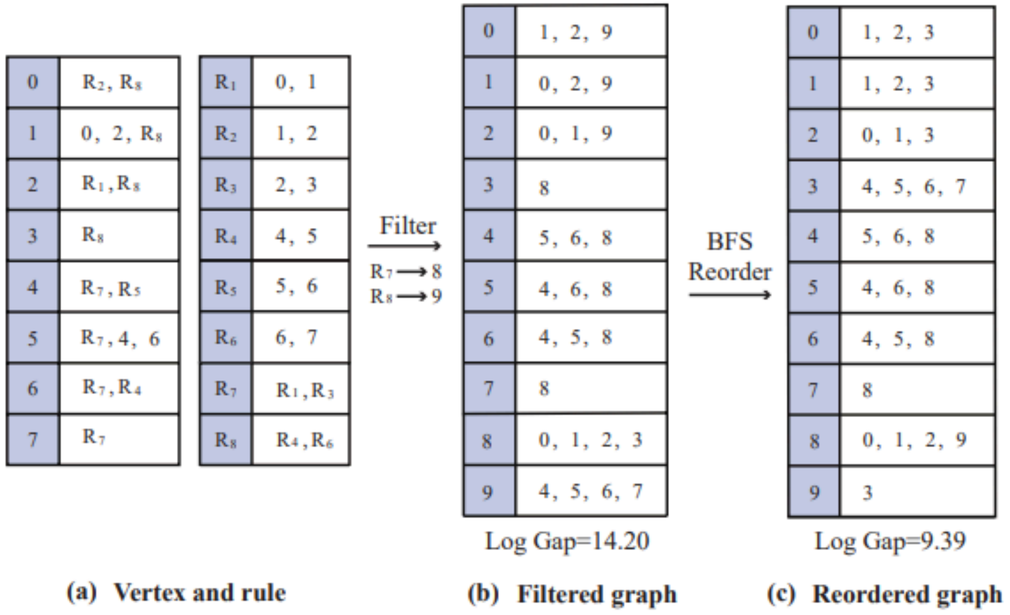


Figure 5: Example on locality recovery sub-module.

- Log gap is sum of all of the logs of gaps in the adjacency list

Empirical Results

- **Methodologies that are compared:** CompressGraph, Ligra, Ligra+, WebGraph, GridGraph (compared with these as they are state-of-the-art and some such as Ligra+ include compression mechanism and WebGraph is known for exceptional compression ratios)
- **Aspects for evaluation:**
  - Peak memory usage throughout compression and computation
  - Time and space overhead during computation
  - Compression ratio
  - Compression time
- **Benchmarks (Graph Applications)**
  - Connected Components, betweenness centrality, triangle counting, single-source shortest path, PageRank

Empirical Results

Table 1: Datasets.

Dataset	Graph	Abbreviation	$\|V\|$	$\|E\|$	Size
1	web-BerkStan	BERKSTAN	685,231	7,600,581	63MB
2	in-2004	IN	1,382,909	16,917,053	153MB
3	eu-2005	EU@2005	862,665	19,235,139	153MB
4	uk-2007-05@1000000	UK@1000000	1,000,000	41,247,159	322MB
5	indochina-2004	INDOCHINA	7,414,866	194,109,311	1.50GB
6	arabic-2005	ARABIC	22,744,080	639,999,458	4.93GB
7	it-2004	IT	41,291,594	1,150,725,436	8.88GB
8	gsh-2015-host	GSH	68,660,142	1,802,747,600	13.94GB
9	sk-2005	SK	50,636,154	1,949,412,601	14.9GB
10	uk-2007-05	UK	105,896,555	3,738,733,648	28.65GB
11	clueweb12	CLUE	978,408,098	42,574,107,469	324.49GB
12	uk-2014	UK@2014	787,801,471	47,614,527,250	360.63GB

Platform. We conduct a standard performance evaluation of Laconic on a CPU server equipped with an Intel Core i7-12700K CPU, featuring 12 cores and 20 threads. The server has 128GB of global memory, and the operating system used is Ubuntu 20.04.01. To demonstrate Laconic's superiority in handling large graphs such as *uk-2014* and *clueweb12*, we also measure Laconic's performance on another server with an Intel Xeon Gold 6230 CPU @ 2.10GHz, featuring 40 cores and 80 threads. The server has 600GB global memory, and the operating system used is Ubuntu 20.04.6.

Table 2: Peak memory usage during graph compression and processing.

Dataset	Original Size	CompressGraph	Ligra	Ligra+	Laconic
UK@2014	360.63 GB	-	-	-	**261.65 GB**
CLUE	324.49 GB	-	-	-	**288.76 GB**
UK	28.65 GB	65.376 GB	154.90 GB	121.85 GB	**24.86 GB**
SK	14.90 GB	57.40 GB	80.10 GB	62.87 GB	**13.89 GB**
GSH	13.94 GB	84.45 GB	74.80 GB	58.85 GB	**29.48 GB**
IT	8.88 GB	34.77 GB	47.59 GB	37.42 GB	**9.68 GB**
ARABIC	4.93 GB	19.16 GB	26.11 GB	20.46 GB	**5.24 GB**
INDOCHINA	1.50 GB	6.27 GB	7.71 GB	6.00 GB	**2.42 GB**
UK@1000000	322 MB	966.71 MB	1.54 GB	1.18 GB	**375.88 MB**
EU@2005	153 MB	744.83 MB	750.30 MB	577.27 MB	**281.91 MB**
IN	153 MB	834.35 MB	693.31 MB	540.61 MB	**314.00 MB**
BERKSTAN	63MB	393.63MB	250.96MB	230.62MB	**100.31 MB**

Empirical Results

Table 3: Comparison of compression ratios and compressed graph sizes of Ligra+, CompressGraph and Laconic.

Dataset	Original Size	Ligra+ Compressed Size	Ligra+ Compression Ratio	CompressGraph Compressed Size	CompressGraph Compression Ratio	Laconic Compressed Size	Laconic Compression Ratio
UK@2014	360.63GB	-	-	-	-	23.89GB	16.30
CLUE	324.49GB	-	-	-	-	38.99GB	8.32
UK	28.65GB	5.44GB	5.26	4.48GB	6.38	3.48GB	8.23
SK	14.90GB	5.04GB	2.95	2.62GB	5.68	1.85GB	8.05
GSH	13.94GB	5.93GB	2.35	6.47GB	2.15	4.76GB	2.93
IT	8.88GB	3.04GB	2.92	1.72GB	5.16	1.22GB	7.29
ARABIC	4.93GB	1.71GB	2.88	0.96GB	5.12	699MB	7.22
INDOCHINA	1.50GB	525MB	2.92	295MB	5.20	216MB	7.10
UK@1000000	322MB	106MB	3.03	47MB	6.76	32MB	9.88
EU@2005	153MB	61MB	2.50	43MB	3.52	32MB	4.69
IN	153MB	58MB	2.63	48MB	3.16	35MB	4.32
BERKSTAN	63MB	30MB	2.10	26MB	2.43	20MB	3.15

Empirical Results

Table 4: Compression time evaluation. PFT is short for *parallel frequency threshold*.

Dataset	Ligra+ (s)	CompressGraph (s)	Laconic (s) - PFT = 5				Laconic (s) - PFT = 4			
			Iteration 1	Iteration 2	Iteration 3	Total	Iteration 1	Iteration 2	Iteration 3	Total
UK	605.03	690.43	81.67	66.82	42.13	190.62	82.51	67.42	44.58	194.51
SK	326.81	334.71	23.61	22.14	22.43	68.19	24.51	23.41	21.94	69.87
GSH	324.03	812.63	61.70	62.88	59.44	183.03	61.17	61.66	56.46	179.31
IT	175.25	198.43	15.56	14.55	13.96	44.08	15.40	13.98	12.76	42.14
ARABIC	53.54	105.34	5.08	3.62	2.88	11.58	5.11	2.80	3.78	11.70
INDOCHINA	28.50	51.66	7.62	7.08	7.02	22.73	7.62	7.79	6.70	22.12
UK@1000000	3.38	4.07	0.37	0.33	0.29	1.00	0.40	0.31	0.32	1.03
EU@2005	1.57	2.39	0.22	0.21	0.21	0.66	0.24	0.22	0.24	0.70
IN	1.25	2.23	0.19	0.18	0.16	0.54	0.20	0.18	0.19	0.58
BERKSTAN	0.97	1.51	0.11	0.10	0.10	0.31	0.12	0.11	0.09	0.32

Empirical Results

Peak memory consumption and speedup when performing analytic tasks on compressed graph:

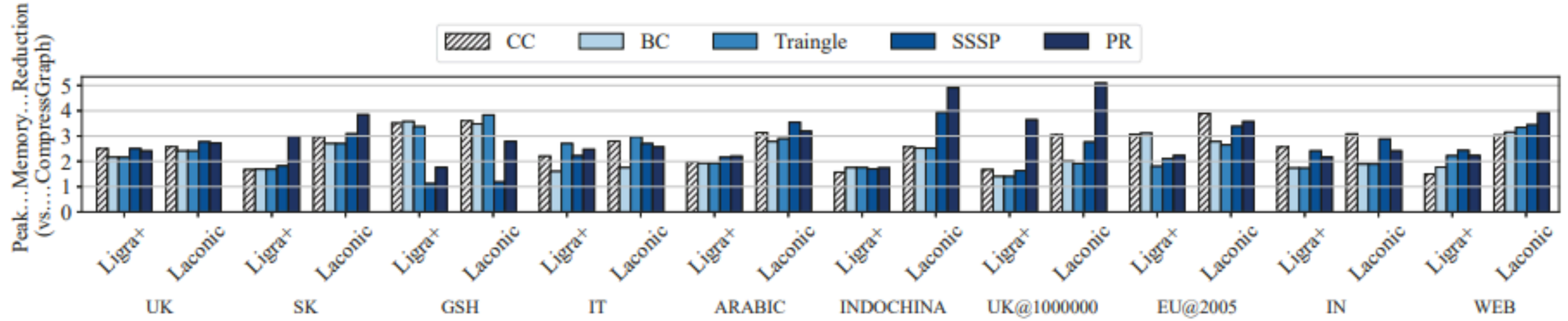


Figure 7: Peak memory reduction.

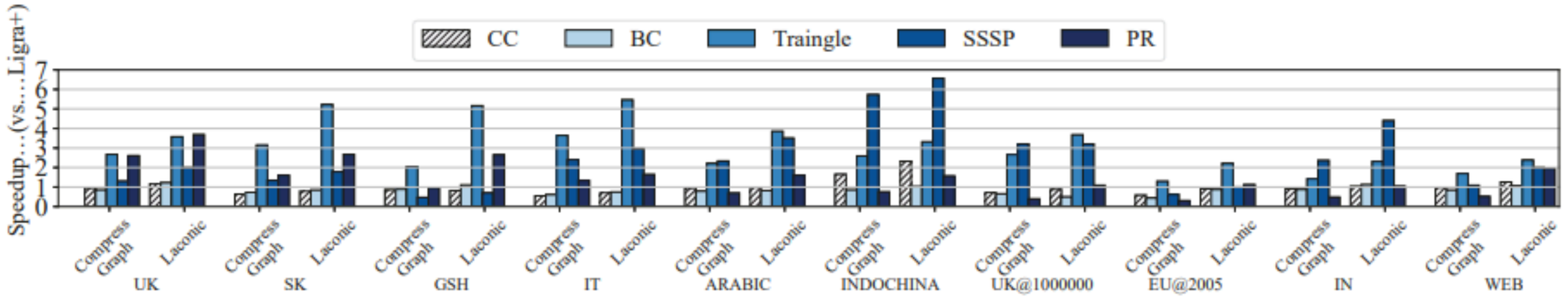


Figure 8: Performance speedup.

Possible Directions, Strengths, Weaknesses

- How does Laconic perform against distributed approaches?
- Could variable length encoding along with delta encoding further improve compression rates?
- Might disk-based systems for graph analytics also integrate a compression step first prior to analytics similar to Laconic and find benefits?
- Proposes a novel memory aware adaptive graph slicing module and parallelization algorithm of rule-based encoding (unlike traditional rule-based encoding)
- (On five graph alg, 12 diverse graphs and stota methods) Reduces on average 70% to 66% memory consumption during compression and computation respectively and reduces 93% compression time on average. 2.47x compression ratio and 2.12x performance speedup