Degree-Aware Partitioning for Enhanced GridGraph Performance

Junyoung Kim
Seoul National University
Seoul, South Korea
kjuny00@capp.snu.ac.kr

***Abstract*—Single-machine graph processing systems like GridGraph offer a powerful alternative to distributed clusters by efficiently utilizing the memory hierarchy. GridGraph's core mechanism is a 2D grid partitioning scheme that divides a graph into vertex chunks and edge blocks to process through optimized data locality. However, its default range-based partitioning, which uniformly divides vertices by their ID, struggles with real-world graphs that exhibit highly skewed degree distributions, leading to significant workload imbalance among the partitioned blocks. This imbalance results in edge blocks of vastly different sizes, undermining the cache optimization objectives.**

This paper proposes a degree-aware partitioning method that considers vertex degree distributions during the partitioning process. Our approach addresses this limitation by pre-computing a partition map that groups vertices to ensure the cumulative sum of degrees is balanced across all partitions, creating more uniform edge block sizes that better align with intended cache-optimized dimensions. We implement this method within GridGraph and conduct a performance evaluation against the baseline system using the PageRank algorithm on the LiveJournal and Twitter datasets under various memory constraints.

Our experimental results demonstrate the effectiveness of the proposed scheme. On the highly skewed Twitter dataset, our method achieves a speedup of more than 2x in memory-sufficient configurations. Detailed analysis reveals that improvements stem from enhanced workload load balancing and improved CPU cache efficiency, with benefits extending across both preprocessing and execution phases. We conclude that degree-aware partitioning is a highly effective strategy that significantly enhances GridGraph's performance by creating a more balanced data layout, highlighting the importance of incorporating structural graph properties into partitioning for single-machine, out-of-core systems.

***Index Terms*—GridGraph, Graph processing, Partitioning, Degree distribution, Cache optimization**

I. Introduction

The efficient processing of large-scale graphs is a critical challenge in modern computing, with applications ranging from social network analysis to web search and biological network studies. However, processing massive graphs that exceed available memory capacity presents significant computational challenges. GridGraph [1] addresses this challenge by introducing an efficient out-of-core graph processing system that leverages a grid-based partitioning strategy to optimize memory hierarchy utilization.

The core innovation of GridGraph lies in its systematic partitioning approach, which partitions vertices into chunks based on their IDs and organizing edges into a grid of blocks according to their source and destination vertices. This partitioning strategy is designed to align block sizes with cache and memory capacities, enabling efficient processing through optimized data locality. By carefully sizing blocks to fit within different levels of the memory hierarchy, GridGraph achieves substantial performance improvements over traditional graph processing systems.

However, the default partitioning scheme in GridGraph is purely range-based, dividing vertices into chunks of uniform ID ranges. While this method is simple and fast, it does not account for the highly skewed degree distributions commonly found in real-world graphs. Many real-world graphs, such as social and web graphs, exhibit power-law degree distributions, where a small number of vertices have an extremely high number of connections.

This degree imbalance creates a fundamental mismatch between the uniform vertex partitioning and the actual computational workload distribution. When vertices with vastly different degrees are distributed uniformly across partitions, the resulting edge blocks may exhibit significant size imbalances. Consequently, many blocks deviate substantially from their intended sizes as determined by the parameter P, undermining the cache optimization objectives that GridGraph was designed to achieve.

To address this limitation, this paper proposes and evaluates a **degree-aware partitioning** method. Instead of partitioning the vertex ID space into uniform ranges, our approach partitions vertices into chunks such that the *sum of degrees* within each chunk is approximately equal. This approach aims to create more uniform block sizes that better align with the intended cache-optimized dimensions specified by parameter P, thereby enhancing the overall efficiency of grid-based graph processing.

II. Proposed Method

To mitigate the workload imbalance caused by the baseline's range-based partitioning, we introduce a novel **degree-aware partitioning** scheme. Our method replaces the on-the-fly, ID-based partition calculation with a pre-computed mapping that distributes vertices based on their connectivity. This section details the algorithmic modifications and implementation strategy.

A. Baseline Range-based Partitioning

The original GridGraph system [1] employs a range-based partitioning strategy, as detailed in Algorithm 1. For each edge, the partition indices *i* (row) and *j* (column) are determined on-the-fly by applying the `GetPartitionId` function to the source and destination vertex IDs, respectively. This function divides the entire vertex ID space into *P* contiguous chunks, which are of equal size in terms of the number of vertex IDs they contain.

Algorithm 1 Baseline Range-based Partitioning

```
1:  function GetPartitionId($V, P$, vertex_id)    ▷ V: total vertices, P: partitions, vertex_id: target vertex
2:      if $V \pmod P = 0$ then
3:          $partition\_size \leftarrow V/P$
4:          return vertex_id/partition_size
5:      else
6:          $partition\_size \leftarrow \lfloor V/P \rfloor + 1$
7:          $split\_point \leftarrow (V \pmod P) \times$ partition_size
8:          if vertex_id $<$ split_point then
9:              return vertex_id/partition_size
10:         else
11:             return (vertex_id $-$ split_point)$/$(partition_size $- 1) + (V \pmod P)$
12:         end if
13:     end if
14: end function
```

B. Degree-Aware Partitioning

Our proposed method fundamentally changes how partitions are defined. Instead of uniform ID ranges, we create partitions that contain a uniform cumulative degree sum.

First, as a preliminary step, we perform a single pass over the input edge file to compute the precise **in-degree** and **out-degree** for every vertex in the graph. These two degree arrays are then saved to separate files for later use.

Next, we use these degree arrays to generate partition maps. As shown in Algorithm 2, the `CreateDegreePartitionMap` function generates a mapping from each vertex ID to a partition index. It processes vertices in ascending order of their IDs, greedily assigning them to the current partition until a target degree sum (i.e., total degree / *P*) is reached. At each partition boundary, it makes a greedy choice to either include or exclude the next vertex based on which option results in a cumulative sum closer to the target. This ensures that partitions containing high-degree vertices will span a smaller range of vertex IDs, while partitions of low-degree vertices will span a larger range.

To construct the 2D grid, we generate two separate partition maps. The map for determining the source partition (row index *i*) is created using the **out-degree** array. The map for the destination partition (column index *j*) is created using the **in-degree** array.

Finally, during the grid generation process, the partition indices are then found via a simple and fast array lookup (e.g., `i = source_partition_map[source_id]`) instead of the arithmetic calculation used in the baseline. Furthermore, our modified PageRank application can directly load the pre-computed degree files, eliminating the redundant degree calculation step required in the baseline PageRank implementation.

Algorithm 2 Proposed Degree-aware Partition Map Generation

```
1:  function CreateDegreePartitionMap(degrees, P, V, total_degree)    ▷ degrees: per-vertex degree array
2:      $partition\_map \leftarrow$ new Array of size $V$
3:      $target\_sum \leftarrow$ total_degree$/P$
4:      $current\_pid \leftarrow 0$
5:      $current\_sum \leftarrow 0$
6:      for $v\_id \leftarrow 0$ to $V - 1$ do
7:          if $current\_pid < P - 1$ and $current\_sum +$ degrees$[v\_id] > target\_sum$ then
8:              $diff_{after} \leftarrow |(current\_sum +$ degrees$[v_{id}]) - target\_sum|$
9:              $diff_{before} \leftarrow |current\_sum - target\_sum|$
10:             if $diff_{before} \leq diff_{after}$ then
11:                 $current\_pid \leftarrow current\_pid + 1$
12:                 $current\_sum \leftarrow 0$
13:             end if
14:         end if
15:         $partition\_map[v\_id] \leftarrow current\_pid$
16:         $current\_sum \leftarrow current\_sum +$ degrees$[v\_id]$
17:     end for
18:     return $partition\_map$
19: end function
```

III. Experiments

A. Experimental Methodology

This section details the experimental setup and the methodology used to evaluate the performance of our proposed partitioning scheme.

TABLE I: Experimental Environment Specifications

Category	Specification
CPU	2 x Intel(R) Xeon(R) Gold 5218 @ 2.30GHz (Total 32 Cores, 64 Threads)
Memory	DDR4
Storage	HDD
L3 Cache	22MB (22528KB)
OS	Ubuntu 16.04.7 LTS
Compiler	g++ (Ubuntu) 5.4.0

1) Experimental Setup: All experiments are conducted on a single machine with the specifications detailed in Table I. The objective of our evaluation is to compare the performance of our proposed **degree-based** partitioning method against the original range-based partitioning (**Baseline**) of GridGraph. We use the PageRank algorithm (20 iterations) as our benchmark workload. Two real-world datasets are used for this evaluation: **LiveJournal** [2], a moderately sized social network, and **Twitter** [3], a large-scale, highly skewed follower network.

2) Evaluation Procedure: Our evaluation follows a two-phase procedure to ensure a fair and comprehensive comparison.

First, since the partition parameter P significantly influences partitioning performance, we determine its optimal value for each dataset-method combination to isolate its effect. According to the GridGraph paper, P is chosen to optimize the use of the CPU's last-level cache. Therefore, this phase was conducted with a fixed, large memory budget (M = 128GB) to create a quasi-in-memory environment, thereby minimizing interference from disk I/O overhead. Based on these tests, the optimal P values were determined as follows:

- **LiveJournal**: P=64 for both Baseline and Degree-based methods.
- **Twitter**: P=128 for the Baseline method, and P=16 for the Degree-based method.

Second, using the determined optimal P for each configuration, we evaluate the performance scalability by varying the available memory budget M across three levels: 8GB, 32GB, and 128GB. This multi-memory evaluation allows assessment of the proposed method's effectiveness under different resource constraints.

To ensure the accuracy and reliability of our results, each experimental configuration was executed three times, and we report the average execution time. Given that partitioning serves as a cache optimization technique and cache effects are critical to the evaluation, the system's file system cache was cleared before each experimental run to eliminate potential interference from previous executions.

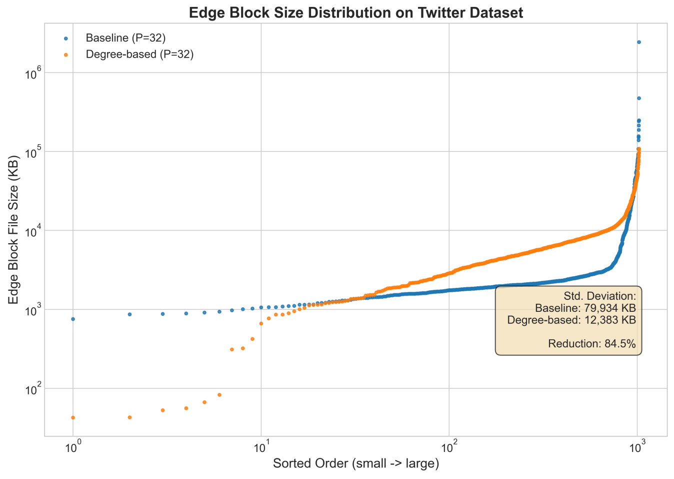
B. Experimental Results

This section presents the performance evaluation of our proposed degree-based partitioning method against the baseline range-based partitioning method of GridGraph. We first analyze the resulting partition balance quantitatively and then demonstrate how this improvement translates into significant end-to-end performance gains for the PageRank algorithm.

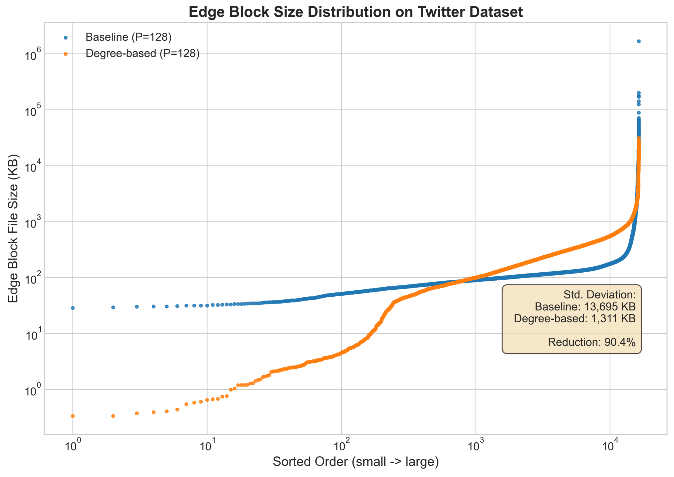
TABLE II: End-to-End Execution Time (seconds) and Performance Improvement.

Dataset	**Memory (GB)**	**Baseline (s)**	**Degree-based (s)**	**Improvement (%)**
LiveJournal (P=64 vs P=64)	8	10.96	9.12	16.8%
	32	10.44	8.74	16.3%
	128	10.15	9.02	11.1%
Twitter (P=128 vs P=16)	8	1174.82	1116.49	5.0%[1]
	32	377.30	185.19	50.9%
	128	378.89	189.31	50.0%

[1]In the heavily I/O-bound 8GB scenario, the performance of both methods is dominated by disk access overhead, which diminishes the benefits of the degree-based method's balancing.



(a) Partitioning with P = 32

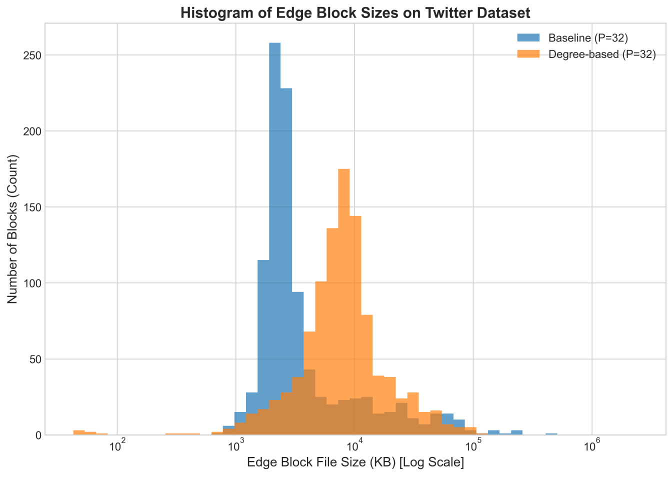


(b) Partitioning with P = 128

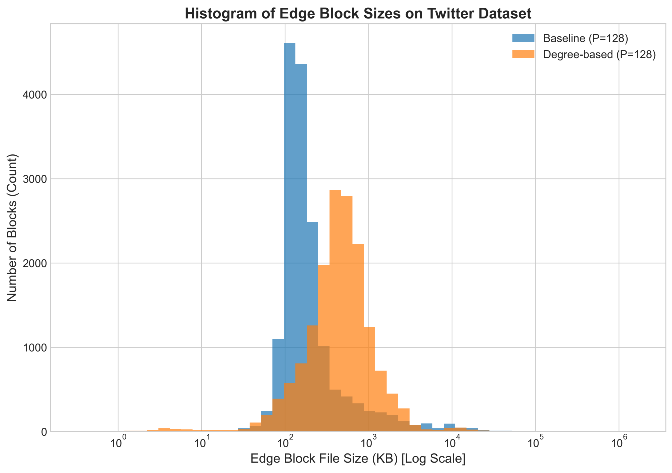
Fig. 1: Comparison of Edge Block Size Distributions on the Twitter Dataset. For both P=32 and P=128 configurations, the degree-based method results in a more uniform distribution.

1) Analysis of Partition Balance: To evaluate the effectiveness of each partitioning strategy, we first analyzed the distribution of the resulting edge block sizes. Figure 1 shows the sorted block size distribution for both methods on the Twitter dataset for P=32 and P=128 configurations. While these log-log plots reveal the wide range of block sizes, a clear advantage is not immediately obvious, as both methods produce curves characteristic of a power-law distribution. This highlights the difficulty of assessing the balance of highly skewed distributions from this perspective alone.

To better visualize the density and balance of the distributions, we present the same data as histograms in Figure 2. Here, the difference becomes apparent. The baseline method consistently exhibits a highly skewed distribution, characterized by a tall, narrow peak of very small blocks on the left and a long tail of a few extremely large blocks (hotspots) on the right. In stark contrast, with the degree-based method, the prominent peak of small blocks is eliminated. Instead,



(a) Partitioning with P = 32



(b) Partitioning with P = 128

Fig. 2: The figure compares the block size distributions of the Baseline and Degree-based methods at two different partitioning granularities: (a) P = 32 and (b) P = 128. The x-axis represents the block file size on a logarithmic scale, while the y-axis shows the number of blocks within each size bin. In both scenarios, the Degree-based method transforms the baseline's highly skewed distribution, which has a large peak of small blocks, into a more centered distribution, visually demonstrating a more balanced partitioning.

the block sizes form a more centered distribution, visually confirming that a more balanced partition has been achieved.

To quantify this visually observed improvement, we calculated the standard deviation of the block sizes. For the P=32 case, our proposed degree-based method reduced the standard deviation by a remarkable 84.5% (from 79,933 KB to 12,382 KB) compared to the baseline. The improvement is even more pronounced for the P=128 configuration, where the standard deviation was reduced by an exceptional 90.4% (from 13,695 KB to 1,311 KB).

This significant, numerically-verified reduction in variance, supported by the histogram visualization, proves that our method substantially mitigates workload imbalance. This enhanced balance is the direct cause of the end-to-end performance gains detailed in the following section.

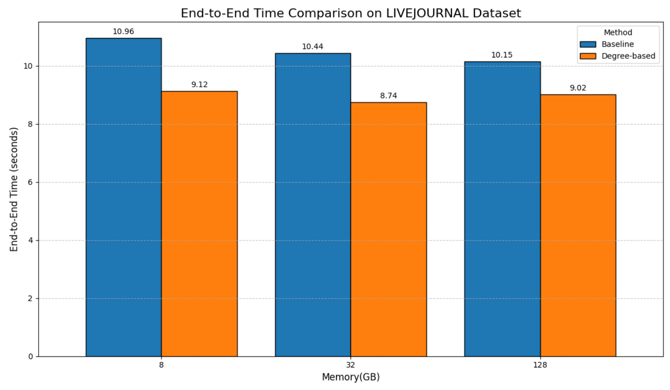


Fig. 3: End-to-End Time Comparison on the LiveJournal Dataset. The total execution time, including both preprocessing and 20 iterations of PageRank, for the Baseline and Degree-based methods under three memory configurations (8GB, 32GB, and 128GB).

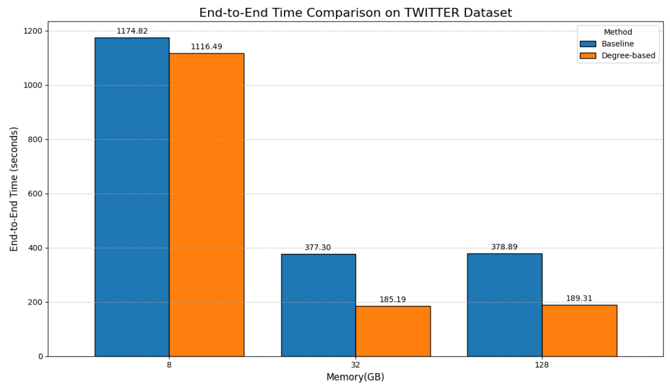


Fig. 4: End-to-End Time Comparison on the Twitter Dataset. This figure compares the total end-to-end execution time of the two partitioning methods on the large-scale, skewed Twitter dataset under three different memory configurations.

2) Overall Performance Comparison: This dramatic improvement in partition balance translates directly into significant end-to-end performance gains, as measured by the total time for preprocessing and 20 iterations of PageRank.

- **LiveJournal Dataset Performance:** As shown in Figure 3, the degree-based method achieves consistent performance improvements across all memory configurations. The degree-based approach delivers 10-17% faster PageRank execution compared to the baseline method.
- **Twitter Dataset Performance:** The performance gains are even more dramatic on the Twitter dataset, which exhibits severe degree imbalance and larger scale. As illustrated in Figure 4, the degree-based method demonstrates overwhelming superiority in memory-sufficient environments. With 32GB and 128GB memory, the

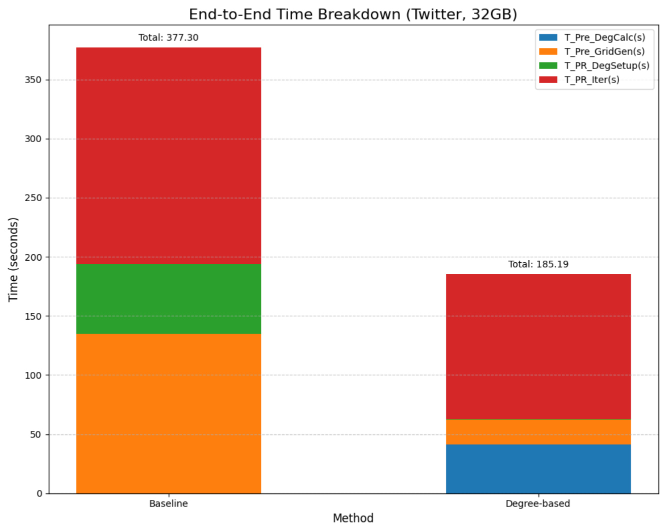


Fig. 5: Breakdown of End-to-End Execution Time on the Twitter Dataset with a 32GB Memory Budget. The total time for both Baseline and Degree-based methods is decomposed into four constituent phases: degree calculation during preprocessing (T_Pre_DegCalc), grid generation (T_Pre_GridGen), PageRank setup (T_PR_DegSetup), and PageRank iteration (T_PR_Iter).

degree-based approach achieves more than 2× speedup over the baseline method reducing the execution time by a remarkable 50.9% (377.30s vs. 185.19s) and 50.0% (378.89s vs. 189.31s), respectively. This result highlights the proposed method's strength in handling graphs with severe degree imbalance by creating a more balanced workload.

3) In-depth Analysis of Performance Improvement: To understand the source of this significant performance gain, we conducted a detailed breakdown analysis of execution time components using the Twitter dataset with 32GB memory configuration. (Figure 5). The difference between T_Pre_DegCalc (in the Degree-based preprocess) and T_PR_DegSetup (in the baseline PageRank) is primarily a shift in when the degree calculation is performed, not a fundamental change in the work itself. The most significant and insightful performance improvements are found in the grid generation (T_Pre_GridGen) and PageRank iteration (T_PR_Iter) phases.

- **Analysis of T_Pre_GridGen Speedup:** To isolate the impact of different optimal P values (P=128 for baseline vs P=16 for degree-based), we conducted additional experiments with identical P=32 configuration for both methods. Even with the same P value, the preprocessing time difference remained substantial (74.57s vs 22.61s), confirming that the performance improvement stems from factors beyond block management overhead. We attribute this improvement to the same source for T_PR_Iter Speedup.
- **Analysis of T_PR_Iter Speedup:** The reduction in the core PageRank iteration time is the intended and most critical result of our method. As seen in Figure 5, the iteration time (T_PR_Iter) for the Degree-based method is significantly shorter than that of the baseline. This is due to two primary factors:
  1) Workload Balancing : The baseline method's tendency to create "hotspot" blocks leads to poor parallel efficiency, where some threads are overloaded while others sit idle. Our degree-based approach distributes the computational workload (represented by the sum of degrees) much more evenly across all partitions. This ensures all threads are kept consistently busy, maximizing parallel hardware utilization.
  2) Improved CPU Cache Efficiency : Better-balanced partitions lead to improved cache utilization patterns, since originally partitioning with p values are intended to fit the blocks in the cache size.

These results validate our hypothesis that degree-aware partitioning addresses the fundamental limitations of index-based partitioning in real-world graphs with irregular degree distributions.

IV. Conclusion

This paper presents a **degree-aware partitioning** approach for GridGraph that addresses the fundamental limitation of index-based partitioning in real-world graph processing. The proposed method considers vertex degree distributions during partitioning to create more balanced edge blocks, thereby improving cache efficiency and overall system performance.

Our experimental evaluation demonstrates substantial performance improvements across different datasets and memory configurations. The degree-based approach achieves consistent 10-17% performance gains on the LiveJournal dataset and delivers more than 2× speedup on the Twitter dataset under memory-sufficient conditions. These improvements stem from better workload balancing and enhanced CPU cache efficiency, validating the effectiveness of considering graph structure characteristics during partitioning.

Future Work. Building on the success of our degree-aware approach, several avenues for future research emerge. While vertex degree serves as an effective proxy for workload, future work could explore more sophisticated partitioning metrics that capture other graph characteristics and structures. Furthermore, the principles of our pre-computation and balanced mapping approach are not limited to GridGraph. Applying and evaluating this degree-aware strategy on other graph processing frameworks could broaden the impact of structure-conscious design principles.

References

[1] Zhu, Xiaowei, Wentao Han, and Wenguang Chen. "GridGraph:Large-Scale graph processing on a single machine using 2-level hierarchical partitioning." 2015 USENIX Annual Technical Conference (USENIX ATC 15). 2015.

[2] BACKSTROM, L., HUTTENLOCHER, D., KLEINBERG, J., AND LAN, X. Groupformation in large social networks: membership, growth, and evolution. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data min ing (2006), ACM, pp. 44–54.

[3] Haewoon Kwak, et al. ”What is Twitter, a social network or a news media?.” Proceedings of the 19th international conference on World wide web. ACM, 2010.

Appendix

TABLE III: Detailed breakdown of average execution times (in seconds) and their ratios (%).

Dataset	Ver.	P	M(GB)	T_prep	T_pr	T_total	T_pre_d	T_pre_g	T_pr_d	T_pr_i	%_pre_d	%_pre_g	%_pr_d	%_pr_i
LiveJournal	baseline	64	8	2.95	8.01	10.96	0.00	2.95	0.73	7.28	0.0	26.9	6.7	66.4
LiveJournal	baseline	64	32	2.95	7.49	10.44	0.00	2.95	0.74	6.75	0.0	28.3	7.1	64.7
LiveJournal	baseline	64	128	2.95	7.20	10.15	0.00	2.95	0.75	6.45	0.0	29.1	7.4	63.6
LiveJournal	Degree-based	64	8	2.89	6.23	9.12	1.58	1.31	0.12	6.11	17.3	14.4	1.3	67.0
LiveJournal	Degree-based	64	32	2.89	5.85	8.74	1.58	1.31	0.09	5.76	18.1	15.0	1.0	65.9
LiveJournal	Degree-based	64	128	2.89	6.13	9.02	1.58	1.31	0.09	6.04	17.5	14.5	1.0	67.0
twitter	baseline	128	8	134.99	1039.83	1174.82	0.00	134.99	49.13	990.70	0.0	11.5	4.2	84.3
twitter	baseline	128	32	134.99	242.31	377.30	0.00	134.99	59.06	183.25	0.0	35.8	15.7	48.6
twitter	baseline	128	128	134.99	243.90	378.89	0.00	134.99	60.54	183.36	0.0	35.6	16.0	48.4
twitter	Degree-based	16	8	62.18	1054.31	1116.49	40.99	21.19	0.90	1053.41	3.7	1.9	0.1	94.3
twitter	Degree-based	16	32	62.18	123.01	185.19	40.99	21.19	0.91	122.10	22.1	11.4	0.5	65.9
twitter	Degree-based	16	128	62.18	127.13	189.31	40.99	21.19	0.90	126.22	21.7	11.2	0.5	66.7

- **Ver.**: Version (baseline or Degree-based)
- **P**: Partitioning factor P
- **M(GB)**: Memory budget in Gigabytes
- **T_prep**: Total Preprocessing Time (s)
- **T_pr**: Total PageRank Time (s)
- **T_total**: Total End-to-End Time (s)
- **T_pre_d**: Preprocessing: Degree Calculation Time (s)
- **T_pre_g**: Preprocessing: Grid Generation Time (s)
- **T_pr_d**: PageRank: Degree Setup Time (calculation or read) (s)
- **T_pr_i**: PageRank: Iteration Time (s)
- **%_...**: Ratio of each phase to the Total End-to-End Time (%)