SoK: The Faults in our Graph Benchmarks

Anonymous authors
Paper under double-blind review

Abstract

Many modern applications process data naturally described by graphs, e.g., social networks, financial transactions, brain networks, and protein interactions. There is a need for special-purpose graph processing systems, since conventional data management systems make several assumptions that do not hold for graph-structured data.

Data management systems are usually evaluated along two axes: scalability and performance. Benchmarking graph processing systems is not a standardized practice, and this can lead to poor baselines and misdrawn conclusions. In particular, evaluations frequently ignore datasets' statistical idiosyncrasies, which significantly affect system performance. Moreover, scalability studies in system evaluations often use datasets that easily fit in memory on a modest desktop. Some studies rely on synthetic graph generators to generate large graphs, but these generators produce graphs with unnatural characteristics that also affect performance, producing misleading results. Currently, the community has no consistent and principled manner with which to compare systems and provide guidance to developers who wish to select the system most suited to their application.

We provide three different systematizations of benchmarking practices. First, we present a 12-year literary review of graph processing benchmarking, including a summary of the prevalence of specific datasets and benchmarks used in these papers. Second, we demonstrate the impact of two statistical properties of datasets that drastically affect benchmark performance. We show how different assignments of IDs to vertices, called *vertex orderings*, dramatically alter benchmark performance due to the different caching behavior they induce. We also show the impact of *zero-degree vertices* on the runtime of benchmarks such as breadth-first search and single-source shortest path. We show that these issues can cause a performance to change by as much as 38% on several popular graph processing systems. Finally, we suggest best practices to account for these issues when evaluating graph systems.

1 Introduction

Graph structured data are used to express complex relationships in social networks [1], protein interactions [2,3], transportation networks [4], and many other domains. Graphs with billions of nodes and edges are common today [5], and large-scale industrial applications often require running analytics on trillion-edge datasets [6]. The research community has produced a constant stream of novel graph processing systems over the past decade to address these challenges.

These systems can be designed to distribute their data across a compute cluster [7–9] or to run on a single compute node by optimizing the data access patterns - be it from memory [10, 11] or disk-resident shards [12–14]. Together these decisions produce a vast design space with different graph processing systems representing just a single point in this design space. Application developers thus face the problem of selecting a graph processing system appropriate for their task.

This challenging task of selecting the appropriate graph system is further exacerbated by the fact that comparing graph processing systems is difficult for several reasons.

First, there is a lack of robust standardization of benchmarking practices such as which metrics are important, how to measure and report them, and how to do so in a reproducible manner. There have been several attempts at standardization in the High-Performance Computation (HPC) community through the Graph500 [15] and GraphChallenge [16] benchmark definitions. However, these benchmarks are not representative of the diversity of use cases of graph processing and neither do they take into account the immense diversity of datasets that exist in the real world. The LDBC Graphalytics [17] benchmark provides a richer set of datasets and benchmark kernels but remains underutilized in practice.

Second, most graph processing systems provide their own implementation of popular benchmark kernels. These different implementations mean that when comparing the performance of two systems, it is difficult, if not impossible, to attribute performance differences to the underlying system itself, the algorithm used for a particular kernel, or the quality of the kernel's implementation. For example, Galois [18] implements four different variations of PageRank [19], each with different runtime performance. Additionally, there are many time-based metrics to compare PageRank: time per iteration, time to run a fixed number of iterations, or time to convergence; different systems use different metrics in their published results. We found that there is no single reference implementation of triangle counting on *directed* graphs resulting in different graph processing systems reporting different numbers of triangles for the same dataset (§3.1.2) .

Third, there are few large representative graph datasets available for benchmarking, which limits the quality of results being reported. For example, a distributed graph processing system that claims to be scalable should be evaluated using datasets that are large enough to warrant distribution. However, we show that this is not true in practice; most systems demonstrate scalability by using popular datasets from the SNAP and KONECT [20, 21] dataset libraries. Twitter2010 [22,23] is one of the largest, most frequently used datasets, yet in its *uncompressed* state is only 26GB - smaller than the available RAM on a high-end laptop or a modest server. The dearth of publicly available large datasets poses a significant problem that is frequently addressed via synthetic graph generators. The RMAT [24] and Kronecker graph generators [25, 26] remain the most popular. However, these generators do not produce graphs that are representative of real world graph datasets, which distorts performance [27].

Fourth, statistical properties of datasets produce hidden, uncontrolled effects at various steps of the graph processing pipeline that drastically affects the performance on the systems. We demonstrate the effect of two such properties inherent in the graph structure of datasets - vertex ID assignments (§3.1) and the presence of zero-degree vertices (§3.2), each of which can introduce changes in key performance metrics such as runtime and cache behavior. Changing the vertex ID assignment of a dataset can cause a performance difference of as much as 38% for the PageRank benchmark on several popular graph processing systems; the presence of isolated vertices can cause a 10x performance boost for benchmarks such as BFS. *Failure to identify these issues produces results that reflect idiosyncrasies of the experimental setup rather than the fundamental behavior of the systems under evaluation.*

Together, these challenges render system comparisons obscure, at best, and meaningless at worst. In addition to these challenges, different developers might have different goals for their application: minimizing runtime, running within a memory or cost budget, rapidly ingesting a stream of updates to the graph, etc. *How can we, as researchers, help developers make wise choices?*

To help developers make wise choices about using, building, and benchmarking graph processing systems, we present a systemization of knowledge surrounding the benchmarking of graph processing systems. Specifically, we present the three following contributions. First, we provide a study of the published literature from a 12 year period on graph processing systems and algorithms that establishes the current status quo for benchmarking of graph processing systems. Second, We present empirical data that illustrates the significance the properties of datasets can have on runtime, using four graph processing systems, the most popular benchmarks, and the most widely used datasets identified in the literature survey. Finally, we provide a set of principles (§4) to guide evaluation of graph processing systems.

2 12 Years of Graph System Benchmarking

Each time we set out to conduct a performance evaluation including widely used graph processing systems, we encountered contradictory and confusing results. Ultimately, we decided to catalog a decade of research in such systems, which uncovered the host of problems discussed in the previous section.

2.1 Methodology

The volume of graph processing research and its generality raises questions as to what criteria define a graph systems paper. Our choice of venues in which we search for papers is similarly ambiguous.

We resolve this ambiguity by clearly defining our ideal corpus and then methodically constructing such a corpus as best we could. Ideally, *we want to include every paper that describes a graph processing system and was published after 2010's Pregel, which is generally regarded as the first "vertex programming" system and a major motivator for recent research.* A "graph processing" system is an implementation that explicitly operates on graph-structured data and produces analysis or query results. Therefore, any relevant paper should explicitly discuss graphs. We impose no generality requirements on such a system; therefore, an implementation of a single-purpose graph algorithm is a graph processing system; any graph mining paper describes a graph processing system. Conversely, a general-purpose processing framework that

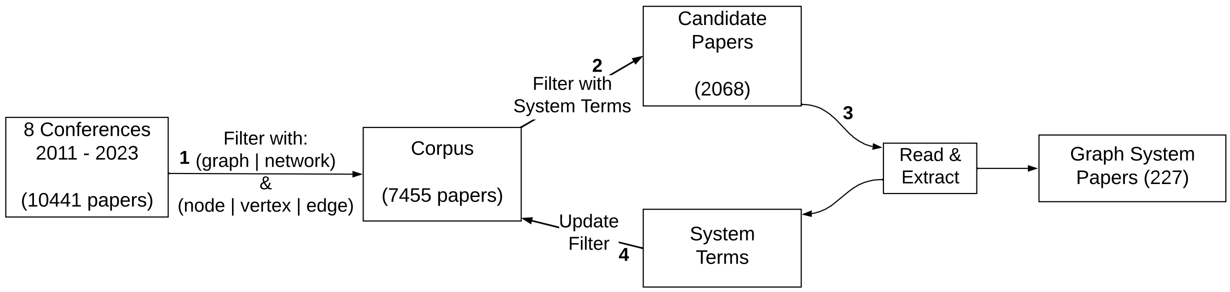


Figure 1: Our procedure to identify contemporary graph system papers. 1.) Our corpus consists of papers published at 10 conferences held between 2011 to 2023 and are pruned using common-sense graph terms. 2.) We filter our corpus with major graph system names to obtain candidate papers. 3.) We read the papers and extract more graph system terms from them. 4.) We update the filter with the new terms and repeat the process.

Venue	Years	Total	Graph Related	Refers to System	**Presents System**
GRADES	2013-22	113	110	71	**22**
KDD	2011-22	3316	2236	477	**10**
OSDI	2012-23	368	298	106	**9**
PODS	2011-22	359	258	57	**0**
SIGMOD	2011-22	1976	1350	456	**63**
SOSP	2011-21	217	177	56	**8**
VLDB	2011-22	2574	1729	505	**86**
NSDI	2011-23	638	541	233	**8**
EuroSys	2011-23	485	467	99	**16**
FAST	2011-23	395	289	8	**5**
Total	**2011-23**	**10441**	**7455**	**2068**	**227**

Table 1: Venues used in the study and the results of the paper selection pipeline.

"can" implement graph algorithms is not within our scope unless some paper explicitly discusses those graph algorithms. For example, we include a paper in our corpus that presents new relational algebra operators for processing graph-structured data [28], while we discard a paper that presents an optimization on Gradient Descent [29].

We did our best to adopt a consistent method in our paper search, but ultimately our judgment and expertise do play important roles in the search process. Inevitably, some distance will exist between our experience and the aggregated experiences of our readers. This could be construed as a sampling error or noise; the nature of such "error" changes with the expectations of the reader. Rather, we encourage the reader to read this paper as an autopsy and diagnosis of some limited, but relatively unbiased, subset of the graph processing community.

2.1.1 Selecting Papers

We began by downloading the full proceedings from the venues indicated in Table 1 (10441 papers). We followed this step with a recursive filtering process, which uses an expanding set of graph systems (our search phrases), that continues until it produces no new systems.

In the first step, we filtered the corpus to identify papers that might be related to graph systems. To do that, we devised the following simple rule as a first-level filter: a paper might be graph-related if it contains the terms *graph* or *network and* any one of *node*, *vertex*, or *edge*. This first-level filtering produces an initial set of papers that are likely related to graph processing. At the end of this stage, we were left with 7455 papers.

In the second level filtering, we use the reduced corpus from the first stage, and search for *Giraph, GraphLab, GraphX, PowerGraph, and Pregel* - which we consider to be the first few important graph systems. We then manually inspect all the papers containing these seed phrases to find new graph systems. We add each newly found graph system to the set of seed phrases and iterate until we stop finding new systems.

This methodology suffers from several flaws - not all systems are named, many papers discuss the same system or name systems that are outside our initial corpus, and the selection of papers in the filtering process suffers from the writers' bias. Therefore, there is no, one-to-one mapping between papers and search terms. We present this metastudy as a review of a limited subset of the graph processing community. However, this process, illustrated in Figure 1, identified 2068 papers, from which we manually identified 227 papers that describe systems or algorithms that process graph data.

2.2 Findings

For each of the 227 papers, we manually determined all the benchmarks and datasets used in its evaluation. This process is necessarily manual because there is a significant entity linkage concern regarding benchmarks and datasets. For example, a paper might reference a "KONECT Wikipedia dataset", but in fact there are numerous Wikipedia datasets on KONECT and the specific dataset can be identified only by its vertex and edge counts. This is further complicated by variations in the reported vertex and edge counts of identical datasets - we encounter papers that reference names that refer to non-specific datasets as well as specific datasets with varying features.

2.2.1 Datasets

We identified 331 different datasets used in our corpus. However, fewer than 5% of these datasets have been used in more than 10 papers. The top 10 datasets account for over 37% of all datasets used. Figure 2 shows the usage of the various datasets across research papers. On average, a dataset is used in only three different papers.

The most widely used dataset, appearing in more than one-third of the research papers in our corpus, is Twitter2010 [23]. This is problematic in a couple of different ways. First, papers report different numbers of edges for Twitter2010 [30, 31]. Second, there are (at least) two different Twitter datasets and some papers in our corpus simply refer to "the Twitter dataset." There is a second Twitter dataset from MPI [32], which has 10 million more vertices and 200 million more edges than Twitter2010 [23]. While this inconsistency might not cause problems if all comparisons are done with the same dataset, it can be

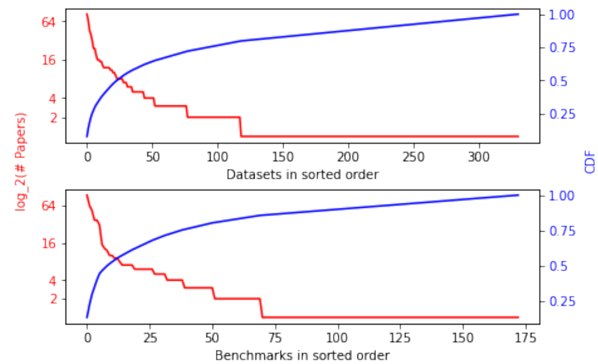


Figure 2: Frequency of usage of datasets and benchmarks that appear in our 227 paper corpus. The red axis shows the number of papers that use that particular dataset or benchmark. The blue axis shows the CDF of the percentage of the usage of a particular dataset or benchmark out of the total usage of the total usage of all datasets or benchmarks across our paper corpus.

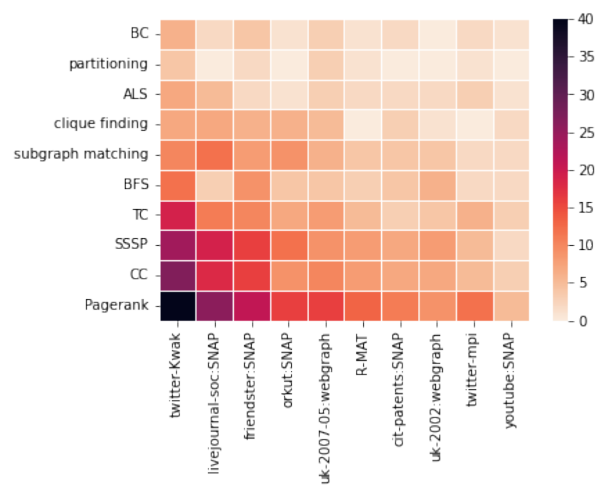


Figure 3: The frequency of co-usage of the top-10 benchmarks and top-10 datasets

problematic to compare systems that do not make their artifacts publicly available.

As can be seen in Figure 3, social network datasets and synthetic graphs (that produce a large social network-like graph) are the dominant graph type used for comparative benchmarking. The large skew in-degree distribution of such graphs is a useful tool to test the partition quality of a distributed graph processing system and examine the load-balancing scheme. However, not all graphs possess similar degree skew, and not all graph processing systems are designed to be distributed. There are other statistical properties of a dataset that can determine its utility in stressing different graph processing systems - large diameter, sparsity (number of edges per vertex), average triangle count per vertex, and maximum in- and out-degrees to name a few. These are often ignored in favor of selecting the largest graph researchers can find. For instance, Besta et al. [33] identified that Flickr [34] and Livemocha [35] have similar node and edge counts and diameters but vary greatly in the performance characteristics for 4-clique mining due to the large difference in the number of triangles (indicative of clustering property). A social network such as Livemocha should have a few 4-cliques of friendships, but in a network such as Flickr, where related photos share some metadata (such as location), 4-cliques should be common. Graph Mining Suite [33] presents a better approach to selecting appropriate datasets for benchmarking: emphasizing a rich diversity of dataset origins and selecting them in a way that enhances the represented statistical properties.

Most disturbingly, we found that the Netflix dataset is one of the top 10 most widely used datasets, even though *it is illegal to distribute this dataset and arguably unethical to experiment with it*. The dataset contains recommendations by users who were later deanonymized by researchers [36]. Both an FTC inquiry and a class action lawsuit that Netflix settled in 2010 [37] caused Netflix to cancel further competitions and withdraw the dataset, whose license clearly states that it cannot be redistributed without permission. Nevertheless, between 2010 and 2023, nine papers in our corpus had used this dataset. *This practice must stop.*

Less legally and ethically troubling, but more technically troubling, is the fact that we found inconsistencies in the number of vertices and edges reported for the datasets across the papers. *This makes interpreting results tricky as one can't be sure if two papers use the same dataset.*

This issue of inconsistently sized datasets is exacerbated by the practice of simply naming datasets without reference. In our corpus we found twelve different variants of "the Wikipedia dataset"; we were unable to locate five of them even after extensive web searching. Four of the remaining Wikipedia datasets simply cited KONECT as the source of the dataset but KONECT has 37 different "Wikipedia datasets". Similar issues arose with datasets named uk-20XX; some papers cite them only as "the uk dataset". In our corpus, we found at least four different uk datasets being used. On further investigation, we found that Laboratory for Web Algorithms [38–40] is the source of the uk datasets and has 14 different uk datasets.

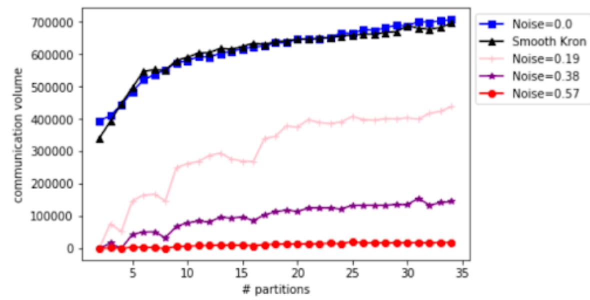


Figure 4: Partitionability of graphs generated by Smooth Kronecker compared to graphs generated by Noisy Kronecker generators

2.2.2 Synthetic Graph Generators

Real graph datasets are usually small in size and often fit in memory. In the absence of readily-available truly large graphs researchers rely on synthetic graph generators to such an extent that graph datasets generated from RMAT or Kronecker graph generators [10, 24–26, 41, 42] are the third most commonly used source for datasets.

Despite their popularity, default Kronecker generators are not well-suited for benchmarking graph processing systems as these generators produce graphs with unrealistic node degree distributions inducing a "combing effect" [27]. To rectify this, researchers developed Noisy Kronecker models that add uniform random noise to the graph generation process to fix the degree distribution problem [43]. But graphs generated by Noisy Kronecker models generate graphs that are more easily partitionable than the original graph the generator models. Smooth Kronecker [27] is a synthetic graph generator that fixes the "combing effect" without adding any noise in the generation process. Additionally, Figure 4 shows that there is no difference in the partitionability of the graphs generated by the Smooth Kronecker model and the original graphs the generator models. The Smooth Kronecker graph generator produces graph datasets that are better suited for benchmarking graph processing systems.

However, none of these graph generators were designed to generate graph datasets that evolve over time. With modern cloud applications increasingly operating on real-time evolving data, dynamic graph algorithms have been developed that operate on dynamic graphs. To the best of our knowledge, there exists only one graph generator, DyGraph [44][1] that produces graph datasets for benchmarking dynamic graph algorithms.

2.2.3 Benchmarks

We identified 173 unique benchmarks in our corpus; only ∼5% (4.6%) of the benchmarks have been used in more than 10 papers and the 10 most frequently used benchmarks account for 49% of all benchmark usage. Figure 2 shows the distribution of the various benchmarks used by the papers in our corpus. On average, a benchmark is used in only three different papers; only a select few are widely used. This use of bespoke benchmarks makes it difficult to understand precisely what is being measured or gain any intuition about the system under study.

PageRank, used in over 54% of the papers, was the most popular benchmark. However, there are two fundamentally different ways to compute page rank. Some systems compute page rank at each vertex for a fixed number of iterations, while others compute page rank until the PageRank values converge. In our corpus, over 55% of the PageRank usages report the elapsed time for either a single iteration or for a small, fixed number (usually 10 or 20) of iterations. About 20% of the papers using PageRank as a benchmark reported the time it took for PageRank to converge, while the remaining 25% just report "runtime" without any further specification about whether the benchmark runs for some fixed iteration count or until convergence.

Even comparing results that all measure time for a specified number of iterations is challenging. Usually, each iteration updates each vertex exactly once; however, some newer systems use asynchronous methods that run on every vertex as often as new data becomes available, or they prioritize vertices that are far from convergence. In contrast to the fixed iteration setup, an asynchronous implementation typically runs until the page ranks for vertices are within a target tolerance of convergence. Synchronous PageRank implementations are so common in contemporary graph systems evaluation that it discourages comparisons to asynchronous implementations of PageRank. For example, PowerGraph (2012) [8] was one of the first vertex programming systems to support asynchronous execution. However, in their comparative evaluation they only benchmark synchronous PageRank, because prior works measured and published "per-iteration" runtimes that are defined only for synchronous implementations.

The authors of GraphMat (2015) [45], an iterative Sparse Matrix Vector Multiplication (SPMV) system, use Synchronous PageRank to compare to PowerGraph and Galois, concluding that GraphMat is 2.6X faster per-iteration than Galois. This factor is comparable to the difference between the best synchronous and asynchronous implementations in Galois. Galois outperforms GraphMat on every end-to-end runtime measure but underperforms on every per-iteration measure, from which GraphMat's authors conclude that, on average, they outperform Galois; this claim is almost entirely due to synchronous PageRank results. To be clear, GraphMat's evaluation followed conventional best practices and was consistent with prior work. Our concern is that *consistency with prior evaluations has brought the field to a state where the most popular comparative benchmark discriminates against alternative and arguably superior solutions.*

Such inconsistencies are also visible in recent papers on dynamic graph processing systems. These systems support fast analytic performance in the presence of modifications to the graph [46]. For example, LLAMA [47], GraphOne [48], and Teseo [49] are three such systems, published in 2015, 202, and 2021 respectively. The GraphOne paper claimed to be 4x faster than LLAMA in PageRank workloads, however, the Teseo paper (published after the other two) showed that LLAMA was consistently faster than GraphOne in PageRank (by up to 3x). Such inconsistencies demonstrate that various experimental features such as the degree of parallelism, hyperparameter settings, dataset characteristics (e.g., the size and topology of the graph), or implementation of algorithms can significantly affect performance.

Without being fully aware of what workload characteristics are important and how systems perform with respect to them, it is impossible for practitioners to identify the right system. Hence, there is a need for standard benchmarking specifications that, at a minimum, expelicitly state all the relevant features, and better yet, evaluate systems under a range of them.

2.2.4 Co-occurence of datasets and benchmarks

Figure 3 shows the co-occurrence frequency of the most commonly used top eleven datasets and benchmarks. Using PageRank on the Twitter2010 [23] dataset is, by far, the most popular choice in our corpus. This substantiates

[1] Available at `https://adacenter.org/dygraph`

the claim made by the Naiad authors in 2013 that "several systems for iterative graph computation have adopted the computation of PageRank on a Twitter follower graph as a standard benchmark" [50]. Unfortunately, the publishers of the Twitter2010 dataset have reported that "Twitter2010's follower-following topology is substantially different from other social networks in terms of its non power-law degree distribution, short-effective diameter, and lack of reciprocity" [22]. This makes the results obtained for running various benchmarks on PageRank and Connected Components not generalizable to other social network datasets.

The popularity of the Single-Source Shortest Path (SSSP) benchmark with unweighted datasets such as Twitter2010 [23] and soc-LiveJournal [51] is equally surprising as SSSP requires edge-weights. When using this benchmark with these data sets, researchers produce weighted datasets from unweighted datasets *by assigning random weights to the edges*. Details of the randomization and the weights are never documented. This make it to impossible to reproduce the results for SSSP benchmarks.

Triangle Counting is another popular benchmark used in 16% of our papers. Triangle Counting is well-defined on undirected graphs but is ambiguous for directed graphs. In a directed graph, a triangle counting kernel can either count cycle triangles or trust triangles. Given that the top 10 most popular real graph datasets are directed, using a triangle counting kernel intended for undirected graphs on directed graph datasets is an example of a mismatched dataset-benchmark combination.

2.2.5 Conference Conformance Bias

Figure 5a and Figure 5b show dataset and benchmark use across conferences. Each box represents the percentage of graph papers at that conference that used a particular dataset/benchmark. A co-occurrence correlation exists between the papers accepted to a conference and the benchmarks used in that work. As can be seen in Figure 5b, for example, more than 50% of the graph papers at SOSP, FAST, and EuroSys used PageRank as a benchmark.

Similarly, there is also a preference exhibited by the conference venue towards some datasets as can be seen in Figure 5a. Papers accepted at KDD tend to overuse the Twitter2010 [23] and yahoo-webscope datasets. 55% of the papers at OSDI use the uk-2007 [52] dataset, which is at least 40% more than the usage of this dataset in any other conference.

These results indicate the overuse of certain datasets and benchmarks and the use of these together in preferred pairs.

2.2.6 Artifact evaluation of graph processing systems

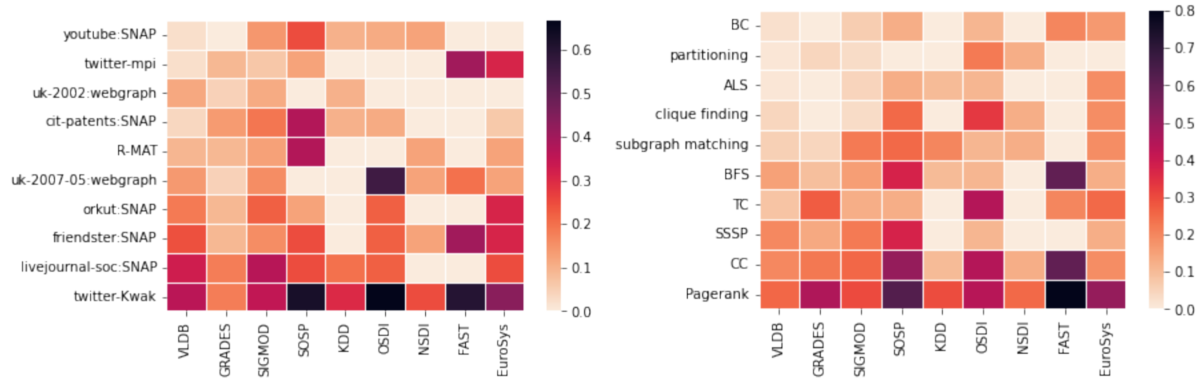
There has been a strong push for artifact availability and result reproduction in the database and systems community, and most conferences we surveyed have an artifact evaluation effort. There are however shortcomings to this approach - not all artifacts can be made publicly available for legal reasons, and the evaluated artifacts and results are not a citable entity. This problem is pervasive to all science, and the data science community has taken the lead in fixing the problem of dataset and code availability by creating repositories such as Zenodo [53] and Dataverse [54]. The Data Management and Systems communities can follow their lead and recommend that published papers come with citable and accessible datasets and software, detailed usability instructions, and the environmental setup needed to reproduce results.

Table 2 shows the artifact evaluation badges awarded to the papers in our corpus by various conferences. We consider only those time periods in which a conference ran an artifact evaluation procedure. Out of the 55 papers that could have obtained the "Artifact Available" badge, less than one-fourth (12) papers were awarded the badge. Shockingly, out of the 99 papers that could have obtained the "Results Reproduced" badge, only three papers were awarded the badge. Note that the badges are only awarded if the authors of a paper explicitly sign up for the artifact evaluation process. The absence of a badge does not necessarily mean that the paper's artifact might not be publicly available or that its results might not hold.

3 Quantitative Study

The previous section described many of the problems we encountered in the empirical work on graph processing systems. This section provides a quantitative demonstration of the severity of these problems. Specifically, we focus on performance discrepancies caused due to vertex isomorphisms (i.e., different vertex ID assignment) and the presence of zero degree vertices.

For our quantitative study, we chose four popular graph processing systems - GraphChi [12], Ligra [10], Galois [18], and Gemini [11]. We chose these systems as they are popularly used and cited and have well-documented open-source implementations available for a



(a) (b)

Figure 5: (a) shows the use of the top-10 datasets across conferences; (b) shows the use of the top-10 benchmarks across conferences. We omit PODS as there is no paper from PODS in our corpus.

Conference	Period	Graph Papers	Artifact Available	Artifact Evaluated	Result Reproduced
SOSP	2019-21	3 (92)	1 (59)	2 (46)	1 (39)
OSDI	2020-23	1 (205)	1 (135)	1 (131)	1 (113)
EuroSys	2021-23	6 (137)	2 (85)	2 (69)	0 (42)
SIGMOD	2016-17	17 (341)	0 (14)	0 (8)	0 (22)
SIGMOD	2018-19	11 (385)	NA	NA	1 (17)
SIGMOD	2020-22	22	0 (54)	0 (55)	0 (54)
VLDB	2018-19	16 (483)	NA	NA	0 (3)
VLDB	2020-22	23 (676)	8 (302)	NA	0 (6)

Table 2: Artifact evaluation badges awarded to graph processing systems at various conferences. NA indicates that the conference did not hand out that specific badge during the specified time period. Numbers in parentheses show the total numbers for the conference in the specified period. (Artifact Evaluated column indicates the Reusable badge for SIGMOD and the Functional badge for other conferences).

Processing System	Version
Galois [18]	Commit 7f20a9131667
GraphChi [12]	Commit 6461c89f217f
Gemini [11]	Commit 170e7d36794f
Ligra [10]	Commit 64d7ef450a22

Table 3: Versions of Graph Processing Systems used in experiments

wide variety of benchmarks. Table 3 shows the versions of the graph processing systems used in our experiments.

All of the results reported for GraphChi, Galois, and Gemini were collected on an Intel i7-core 3.1GHz processor machine with 32GB of RAM. All of the results reported for Ligra were collected on an Intel(R) Xeon(R) 8-core CPU E5-2407 v2@2.4 GHz processor with 98GB of RAM, due to Ligra's higher memory requirements.

3.1 Effect of Vertex Orderings

Many graph processing systems, quite reasonably, process vertices in order of their unique IDs. However, for any graph, there exist many isomorphic graphs that can be obtained simply by renumbering the vertices. We define a **Vertex Ordering** of a graph dataset as one particular assignment of vertex IDs to the vertices in the graph. We demonstrate that this is primarily due to the fact that different Vertex Orderings produce different cache behavior, which in turn, leads to sometimes significant performance differences.

Reordering graphs is not limited to renumbering vertices as different orderings of edges can also impact the performance. In prior work, McSherry et al [55] show that "Hilbert Ordering" of the edges produces better performance results than the default order of most datasets. The effects of reordering graphs are so profound and surprising that there is active research into developing lightweight and efficient re-ordering schemes to speed-up graph processing [56–60].

In this work, we limit ourselves to Vertex Orderings. We describe a small subset of the various Vertex Orderings that can be used in benchmarking graph processing systems. We choose these Vertex Orderings as they are either popular or have well-defined semantics that provide some robustness guarantees. The Vertex Orderings are as follows:

- **Default**: Ordering that the dataset "ships" with. We find that this *default* ordering tends to be either overly optimistic or overly pessimistic in terms of cache behavior.
- **Degree**: Ordering where the nodes are assigned IDs in ascending order of their degree.
- **Revdegree**: Ordering where the nodes are assigned IDs in descending order of their degree.

3.1.1 Effect on Benchmark Performance

We demonstrate the effect of vertex order on runtime and cache behavior using the most widely used benchmarks (Pagerank until convergence) on the most widely used dataset (Twitter2010 [23]), using three different Vertex Orderings (default, degree, and revdegree) across different graph processing systems. We do not present any results for Ligra on Twitter2010 [23], as none of the runs completed after one full week.

To ensure a fair comparison between single-threaded and multi-threaded graph-processing frameworks, we report runtime as the total "user" time across all threads. We run our experiments on a hot cache, to produce accurate cache measurements. We warm the cache by completing one run of the benchmark on the dataset. We eliminate the effect of random noise by running each benchmark on each isomorphism 25 times and report the mean and standard deviation for the runtime and the cache miss rate.

Figure 6 shows the results of running different implementations of PageRank algorithm with Galois, Gemini, and GraphChi on the Twitter2010 [23], citPatents [61], and soc-LiveJournal [51] datasets. We found that Degree Ordering seems to be the best ordering across the graph processing systems for the Twitter2010 [23] dataset. However, it seems that the best ordering for other datasets varies across systems. These results also illustrate that benchmark performance is correlated with the cache behavior of the vertex ordering. In our experiment, Degree Ordering for Twitter2010 [23] yields a speedup of nearly 40% compared to Default Ordering. This suggests that researchers must be careful when comparing results across systems for any dataset, as the choice of different orderings for the same dataset can produce significant, yet misleading, results.

The difference in performance due to Vertex Orderings is not just present across systems. Even in the same system, various implementations of the same algorithm can produce drastically different results on the same dataset. This issue is even more complicated, because different orderings produce best results for different algorithms. Figure 7 shows the runtime for different vertex orderings for all the different implementations of PageRank and ConnectedComponents in Galois. We show the results for Galois as it has the greatest number of different implementations for each benchmark. In a Bulk Synchronous Parallel (BSP) system, a thread computing RageRank, for example, can "Pull" updates from it's neighbors, or "Push" updates to it's neighbors [62]. According to the Galois manual, "the pull variants perform better than the push. The residual version performs and scales the best since it is optimized for improved locality and use of memory bandwidth". The official Galois documentation and papers [63] provide more information about these algorithms and scheduling strategies.

Vertex Orderings can induce significant runtime differences. In particular, default orderings are frequently overly optimistic or overly pessimistic, so we should understand the default ordering as well as the impact that it can have on the system. *In particular, it is essential that researchers avoid inadvertently using different orderings on different systems.*

3.1.2 Effect on Benchmark Correctness

Theoretically, the various vertex orderings should have no impact on the correctness of the algorithm. However, we found that for certain systems, the various Vertex Orderings can produce different *incorrect results* for triangle counting. Triangle Counting on Ligra produces different numbers of triangles for the same directed graph with different Vertex Orderings. On further inspection, we found that this is a symptom of a larger problem: there is no standard approach to counting triangle counting in directed graphs.

We found at least two definitions of triangles on directed graphs: cycle triangles and trust triangles (when two connected vertices both have directed edges to a third common vertex) [64]. Figure 8 shows two SQL queries that compute the number of trust and cycle triangles on

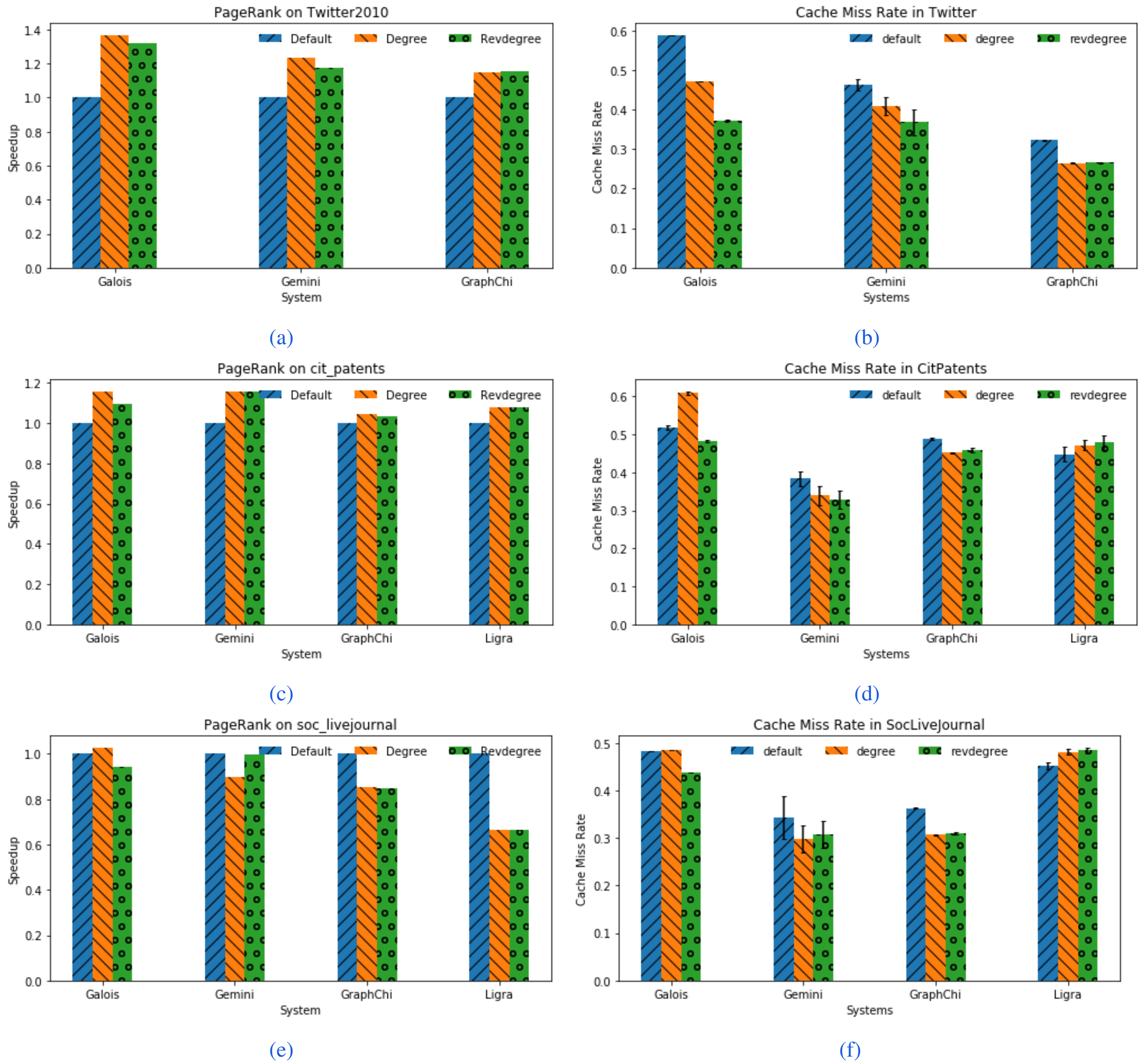
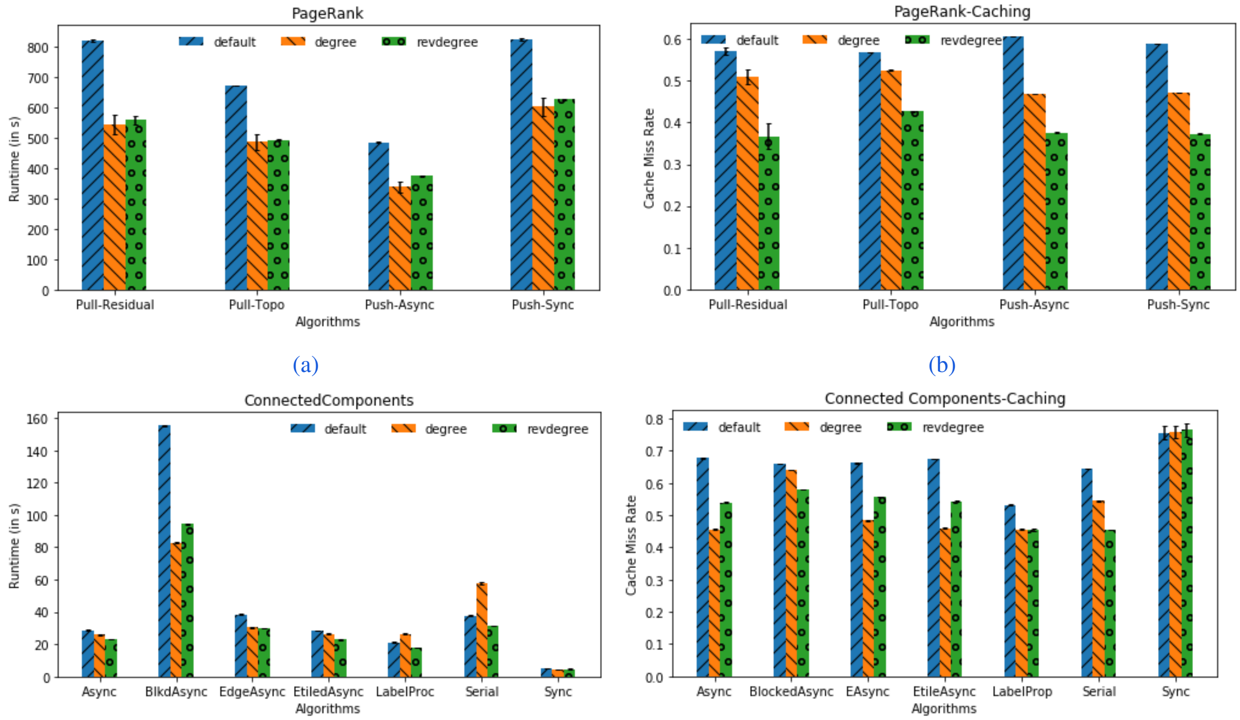


Figure 6: Speedup attained on PageRank on Twitter2010, citPatents, and soc-LiveJournal with Galois, Gemini, GraphChi, and Ligra for Degree and Revdegree orderings as compared to the Default Ordering. We do not present results for running Ligra's PageRank on Twitter2010 as it did not finish within a 7-day time period.



(a) (b) (c) (d)

Figure 7: Effect of Vertex Orderings on performance of various algorithms for running PageRank and ConnectedComponents with Galois on Twitter2s010 dataset.

a graph represented by an edge table, where each tuple in the edge table represents a directed edge. We compute the "ground truth" number of triangles in a given dataset using these queries and compare the results to those obtained from a brute force implementation that runs over a text-based, edge-list representation, common to many published graph datasets. The bottom two rows of Table 4 show the number of cycle and trust triangles obtained from running these queries on citPatents [61], soc-LiveJournal [51], and Twitter2010 [23].

However, we found that popular graph processing systems do not count cyclical or trust triangles correctly on directed graphs. We found only one paper [31] that specifies counting trust triangles for their triangle counting benchmark on directed graphs. Disturbingly, triangle counting implementations of these systems differ sufficiently that running them on identical datasets produces different numbers of triangles, as shown in Table 4. Galois's triangle counting benchmark assumes that the graph is undirected, and when run on directed graphs, the benchmark's behavior is essentially undefined. This undefined behavior is elegantly depicted by the fact that Galois' two implementations of triangle counting, Node Iterator and Edge Iterator, report different numbers of triangles. Ligra enforces no such restriction and measures trust triangles. Unfortunately, their implementation depends on vertex ID order. (A trust triangle is counted only if the directed edges go from a higher numbered vertex ID to a lower numbered vertex ID.) Thus, two graphs that are identical, except for vertex relabeling (that is, two isomorphic graphs), will yield different triangle counts. Note that for undirected graphs, the different systems report the same number of triangles.

Additionally, one paper in the corpus uses an approximate triangle counting algorithm based on top-n eigenvalues called EigenTriangle [65]. As this is an approximate algorithm, it will not yield 100% accurate results all the time. We have no insight as to how many other papers in our corpus might have used this or another approximate algorithm as the use of approximate algorithms is not documented in these papers.

Source	citPatents [61]	soc-LiveJournal [51]	Twitter2010 [23]
GT: Cycle Triangles	0	234455819	44738118599
GT: Trust Triangles	7515027	946400853	143011093363
GT: Cycle + Trust Triangles	7515027	1180856672	187749211962
Galois (NI)	264897	132859748	22849431223
Galois (EI)	188649	122558230	23027408359
Ligra	7514962	187551052	DNF
GraphChi	7515023	285730264	34824916864
Snap Website	7515023	285730264	NA

Table 4: Number of Triangles reported by various graph systems for popular datasets. Rows marked with GT indicate the ground truth we calculated using the queries defined in Figure 8. DNF indicates that the benchmark did not finish in time. NA indicates that number of triangles was not available for a particular dataset.

Dataset	Total Vertices	Zero-Degree Vertices	Non Zero-Degree Vertices	% of Zero-Degree Vertices
Twitter2010 [23]	61578414	19926184	41652230	32
citPatents [61]	6009554	2234786	3774768	37
soc-LiveJournal [51]	4847570	0	4847570	0
friendster [66]	124836179	59227813	65608366	47
uk-2007-05 [52]	105896434	677865	105218569	6

Table 5: Total Number of Vertices and Zero-Degree Vertices in popular Datasets

```
--Trust Triangle Count
select count(*)
from edges e1
join edges e2 on e1.dst = e2.src
join edges e3 on e2.dst = e3.dst
             and e3.src = e1.src

--Cycle Triangle Count
select count(*)
from edges e1
join edges e2 on e1.dst = e2.src
              and e1.src < e2.src
join edges e3 on e2.dst = e3.src
              and e3.dst = e1.src
              and e2.src < e3.src;
```

Figure 8: SQL queries to compute two different types of triangles for directed graphs

3.2 Effect of Zero-Degree Vertices

Zero-Degree vertices, also known as isolated vertices, are vertices that have neither incoming nor outgoing edges. In graph datasets, these vertices do not appear in edge lists, but their IDs lie within the range of the minimum and maximum vertex IDs. Table 5 shows the total number of vertices and number of zero-degree vertices of popular datasets. The large number of zero-degree vertices in the Friendster dataset [66] explains why some describe the dataset as a 124 million node graph [67], while others describe it as a 65 million node graph [68]. Zero-Degree vertices affect benchmark results in myriad ways.

3.2.1 BFS and Zero-Degree Vertices

As mentioned in §3.1, the BFS ordering is ill-defined, because many real world graphs do not have a natural root node. In the absence of any natural root, many graph processing systems [10, 11, 18] use the vertex with ID 0 as the default root and vertex with ID 1 as the target node. However, chaos ensues in some benchmarks if that first vertex happens to be zero-degree node, as is the case in three of the five datasets in Table 5.

?? illustrates the effect of choosing a zero-degree vertex versus a non-zero-degree vertex as the starting node for the BFS benchmark on Galois and Gemini. We randomly select 20 different non-zero-degree vertices from the Twitter2010 [23] dataset and measure BFS runtime starting from that node on Galois's serial-mode and parallel-mode BFS and on Gemini. For Gemini, we use the authors' in-built measurement mechanism, which measures the end to end runtime and not the time spent by all threads. As **??** shows, choosing a zero-degree vertex on Twitter2010 [23] produces a 6x-7x speedup on

Galois and nearly 10x speedup on Gemini.

As most reported results in the literature fail to report either their Vertex Ordering or their BFS start node, it is virtually impossible to extract meaning from such results. Based on our experiments, we hypothesize that the BFS results shown in the comparison of Galois and Ligra in the original Galois paper [18] are using zero-degree BFS time as both Ligra and Galois have the same runtime on Twitter2010 [23] which, in our experience, happens only if they are using the vertex with ID 0 as the root node, which the implementations of both these systems indeed do by default.

3.2.2 Effect on Triangle Counting

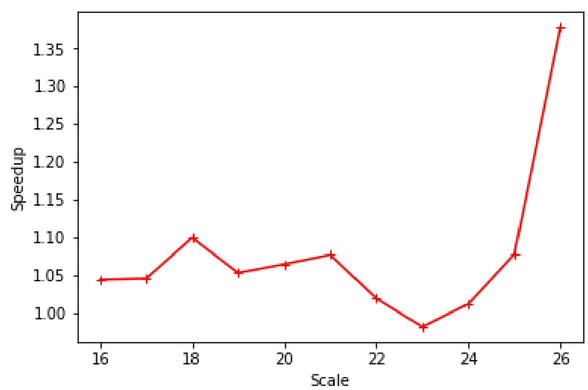


Figure 9: Speedup obtained on Triangle Counting with GraphChi after removing zero-degree vertices from RMAT generated graphs with scale ranging from 16 to 26.

While zero-degree vertices are most problematic for benchmarks that use a start node, they also have an effect on other benchmarks. To better understand the effect of removing zero-degree vertices, we generated 11 RMAT graphs with "scale" ranging from 16 to 26 with an "edgefactor" of 26. In an RMAT graph, $|V| = 2^{scale}$ and $|E| = |V| \times \texttt{edgefactor}$. However, these vertices also include zero-degree vertices, and increasing scale increases both the absolute number and proportion of zero-degree vertices. Figure 9 shows the effect of removing zero-degree vertices when running Triangle Counting on GraphChi. Increasing the scale factor (X-axis) increases both the "size of the graph" and the fraction of zero-degree vertices. Figure 9 shows that the speedup obtained from removing these zero-degree vertices seems to grow exponentially once the graph becomes sufficiently large (i.e., at scale=23). This is a concern, considering that in the absence of real data for large graphs, researchers increasingly rely on synthetic generators. If a particular system were to remove the zero-degree vertices during preprocessing, the system could potentially produce speedups of more than 30%.

3.2.3 Effect on PageRank and ConnectedComponents

Removal of zero-degree vertices can also produce speedup for PageRank and Connected Components. Figure 10 shows the speedup gained on PageRank and Connected Components by removing zero-degree vertices from cit-Patents [61] in Galois, GraphChi, and Gemini. We use the Push-Sync algorithm for PageRank, and we measure the time until convergence. Notice that for both benchmarks, removing zero-degree vertices can produce a speedup of up to 13%. Thus, we would advise that researchers be wary of the impact of zero-degree vertices when evaluating systems on different benchmarks.

4 Best Practices

Based on our findings from the literature study in §2 and impact of dataset properties on benchmark performance in §3, we propose a set of best practices for benchmarking graph processing systems.

Standardization. To overcome the lack of diversity in the datasets and benchmarks used, the community should develop a rigorously-specified set of standardized benchmarks and should publish more well-described, large datasets. A rigorous benchmarking approach would be to use a mix of real world and (well constructed) synthetic datasets of varying topologies. As identified by Besta et. al. [33], it is critical to use datasets of different origins (e.g., road networks, social networks, biological networks, etc.) to evaluate graph processing systems. The current practice of poorly matched dataset-benchmark combinations will undoubtedly propagate to future research through citation networks, unless we promote constructive diversity in benchmarking practices.

Synthetic Graph Generators. For generating large graphs for evaluating the scalability of graph processing systems, researchers should use the Smooth Kronecker graph generator [27] instead of other RMAT or Kronecker graph generators.

Dataset Hashes. Dataset providers should publish hashes of their datasets, and authors should report them. In addition to publishing and citing hashes, the community needs to use standard names and DOIs.

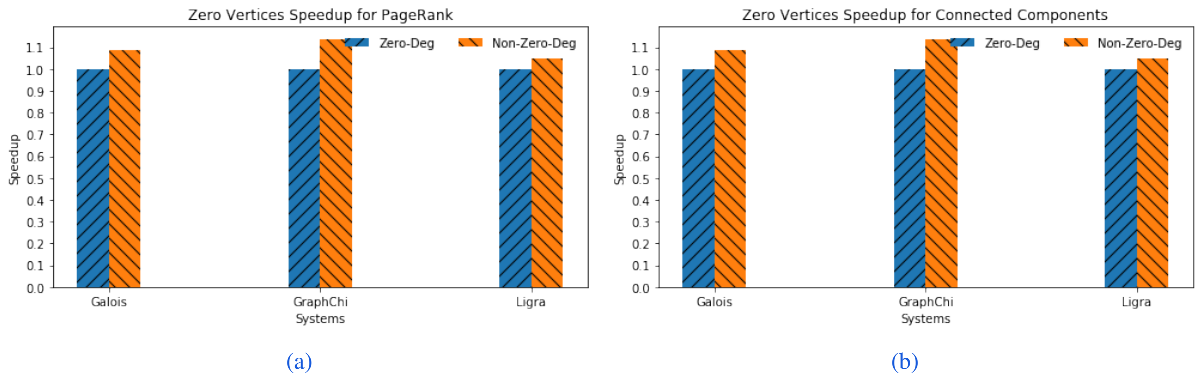


Figure 10: Effect of removing zero-degree Vertices from citPatents Dataset; for running PageRank benchmark using Galois, GraphChi, and Ligra

Detailed Metrics. As recommended in GAIL [69], benchmarking efforts and reporting should be expanded to include more detailed metrics about kernel execution time, the amount of algorithmic work (number of edges traversed per kernel), cache effectiveness (the number of memory requests made per edge traversed), and the memory bandwidth utilization (time taken per memory request).

Preprocessing. Preprocessing the datasets constitutes a significant part of the overall runtime of graph processing systems. Often, the preprocessing time can be amortized if the same data format is used for several tasks and if the graph doesn't evolve too rapidly. However, it is critical to measure and report the preprocessing overheads so that users can make an informed choice about how well-suited the system is to their workload and dataset.

PageRank Usage. When using the *PageRank* benchmark, we advise using tolerance convergence measurement instead of iteration time.

Benchmark & Dataset Selection. Benchmark selection and Dataset selection are not separable; researchers should use datasets appropriate for the benchmark.

Triangle Counting. Researchers who use triangle counting as a benchmark must specify A) whether the graph is directed or undirected, and B) if directed, whether they are counting cyclical triangles, trust triangles, or both. In no case, should a triangle counting implementation return different counts depending on the IDs assigned to vertices. We encourage the community to adopt trust triangles as the official definition of a triangle in directed graphs as the definition of a trust triangle is entirely independent of the vertex ordering of the dataset.

Vertex Orderings. To nullify the impact of different vertex orderings, researchers must report the ordering used. We recommend that researchers show results for at least three different orderings: Default, Degree, and RevDegree. Under no circumstances, should researchers use different orderings of the same dataset for different systems.

Correct Start Nodes . Any benchmark whose results depend on a start node should ensure that their start nodes are not zero-degree and should report the start node(s) used unambiguously.

Handling Zero-Degree Vertices. To deal with the hidden effects of zero-degree vertices, researchers should use packed representations that remove zero-degree vertices from datasets. Additionally, researchers should explicitly state the total number of vertices and the number of zero-degree vertices in their datasets.

Artifact Availability. Authors of graph processing papers should at the very least make their artifact publicly available via the Artifact Evaluation procedure. As part of the artifact, they must include their system, all preprocessing steps, as well as DOIs to datasets that they used as part of their evaluation.

5 Related Work

One of the earliest warnings about improper use of datasets and benchmarks comes from Sebastiano Vigna [70] when he discussed how using PageRank as a benchmark is "interesting" only on *"large"* datasets. Since then, it has become common knowledge that benchmarking large graph processing systems is difficult. This is best captured in the LDBC vision paper [71]: The authors describe the challenges that arise from the diversity in datasets, algorithms, and platforms used in graph benchmarking. Owing to this difficulty in gaining clear insights into the system and its performance, there have also been several attempts at comparing different graph processing frameworks in a more standardized setup. Experimental studies, such as the one by Nadathur et. al. [72], try to compare graph processing systems precisely and also shed much-needed light on the performance characteristics of these systems. The authors first establish benchmark baselines using hand-optimized code and then compare the performance of graph frameworks on these benchmarks. They use it to "delineate bottlenecks arising from algorithms themselves vs. programming model abstractions vs. the framework implementations."

In a much more vocal and direct attack on the field of "scalable" processing systems, Frank McSherry et al [55] provided a new, unbiased metric for measuring the scalability of distributed graph processing systems. The metric provided, called COST (Configuration that Outperforms a Single Thread) measures the hardware configuration required by systems to outperform a single-threaded implementation of the same task. The paper presented the real scalability of distributed graph processing systems without rewarding systems that exhibit substantial, but parallelizable, overheads. The authors additionally show that single-threaded implementations can significantly outperform "scalable" distributed frameworks and that they should be used as benchmarks for measuring scalability. Since 2015, only a handful of research papers have followed the advice of the COST paper and demonstrated scaling results against a single-thread configuration. Gemini [11], Automine [73], GRAM [74], G-POP [75] are few notable exceptions. The goal of our paper is a generalization of the COST paper in the sense that instead of just focusing on the scalability issues, we are focused on identifying and rectifying the most pervasive forms of incorrect and inaccurate benchmarking practices that are prevalent in the community.

GAIL [69] allows performance engineers to "weigh the trade-offs between the three most important graph algorithm performance factors: algorithm, implementation, and hardware platform".

Thankfully, there has been a strong push towards standardization to address the broken benchmarking practices in the community. There has been a lot of active research in search of good benchmarks *and* datasets that can indicate the realistic performance of the systems being profiled. Starting with Graph500 [15], there have been many efforts in this direction such as GAP-BS [76], Graphalytics [77], Waterloo Universal Benchmark [78], and LDBC social network benchmark [79]. Despite these standardization efforts and the existence of performant reference code such as GAPBS [76] and GBBS [80], graph processing systems continue to use their own implementations of graph kernels. However, none of these efforts address the issues raised in our work:

1. They do not consider or actively track the preprocessing cost of generating a data format and layout that the graph system expects to work well
2. While they ship with synthetic graph generators, these benchmark suites do not control the presence of zero-degree vertices or the effects of having skewed degree distributions.
3. Most (with the notable exception of Graphalytics) do not provide any interesting metrics to understand the performance of the system beyond kernel completion time. There are plenty of useful metrics such as Traversed Vertices/Edges per Second, peak memory consumption, the total cost of ownership, etc.

Additionally, standard benchmarking suites fall short of providing a collection of interesting and varying datasets that cover a representative sample of real-world graphs. GAPBS provides 5 datasets in total, while Graphalytics provides 51 out of which only 6 are real-world graphs. There is no guidance or recommendation provided around which datasets might provide insights about system performance as is recommended by GraphMineSuite [33]. There is a real dearth of *large* graph datasets, and while there have been a few trillion-scale generators released such as Trillion-G [41], the onus is upon the companies that have graph-structured datasets to release more large benchmark datasets.

This work shares their fundamental vision of better benchmarking practices. We focus on illustrating the significant consequences of subtle modifications to the dataset. Vertex ordering is an important factor in graph analysis and should be explicitly controlled for as in the study by Abbas et. al. [81]. To the best of our knowledge, vertex re-orderings are not explicitly controlled for in any comparative study we have encountered in our research corpus. We are, to the best of our understanding, the first to demonstrate the impact of isolated vertices in synthetic datasets on benchmark performance.

There is community recognition that there are sometimes mismatches between data sets and benchmarks. The algorithms used in graph-processing benchmarks are either simplistic, such as the BFS traversal algorithm used in Graph500, or are tailored for operations that are specific to (distributed) graph databases. They do not match the diverse operations and algorithms seen in distributed graph processing platforms. This discrepancy is explicitly addressed in Graph Mine Suite [33], and they provide a rich list of datasets and note the statistical properties that make them well-suited to benchmarking a graph mining system. We advocate for similar reasoning behind benchmark and dataset selection.

6 Conclusion

Graph-structured data is increasingly important, and we see no indication that further work in producing efficient and scalable graph processing systems will cease. However, ensuring that next-generation graph processing systems are truly advancing the state of the art requires that we use sound methodology in assessing performance. We have shown that current practice falls far short of this idea. It might be tempting to interpret the lack of diversity in graph data sets and benchmarks as a kind of standardization. However, the majority of these data sets are too small to demand the scale of modern processing systems; they fit in the main memory of most laptop and desktop machines.

State-of-the-art evaluations suffer from poor evaluation practices, such as imprecisely describing datasets, failing to both describe and measure preprocessing steps, omitting crucial experimental setup parameters, and ignoring zero–degree vertices and vertex orderings. This last omission is particularly troubling. Different vertex orderings lead to drastically different results for the same graph on the same graph processing system, yet most papers are silent on such orderings. Zero degree vertices can render BFS benchmark results meaningless. Every graph processing system seems to pick a unique definition of a "triangle" when it comes to the Triangle Counting on directed graphs benchmark. Good science requires that we do better.

References

[1] Alan Mislove, Massimiliano Marcon, Krishna P Gummadi, Peter Druschel, and Bobby Bhattacharjee. Measurement and analysis of online social networks. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, pages 29–42. ACM, 2007.

[2] Alexander Roth, Andrea Franceschini, Damian Szklarczyk, Jean Muller, Manuel Stark, Michael Kuhn, Milan Simonovic, Pablo Minguez, Tobias Doerks, Christian von Mering, Lars J. Jensen, and Peer Bork. The string database in 2011: functional interaction networks of proteins, globally integrated and scored. *Nucleic Acids Research*, 39:D561–D568, 11 2010.

[3] Oliver Mason and Mark Verwoerd. Graph theory and networks in biology. *IET systems biology*, 1(2):89–119, 2007.

[4] Hannah Bast, Daniel Delling, Andrew Goldberg, Matthias Müller-Hannemann, Thomas Pajor, Peter Sanders, Dorothea Wagner, and Renato F Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.

[5] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin, and M. Tamer Özsu. The ubiquity of large graphs and surprising challenges of graph processing. *Proc. VLDB Endow.*, 11(4):420–431, oct 2018.

[6] Avery Ching, Sergey Edunov, Maja Kabiljo, Dionysios Logothetis, and Sambavi Muthukrishnan. One trillion edges: Graph processing at facebook-scale. *Proceedings of the VLDB Endowment*, 8(12):1804–1815, 2015.

[7] Joseph E Gonzalez, Reynold S Xin, Ankur Dave, Daniel Crankshaw, Michael J Franklin, and Ion Stoica. Graphx: Graph processing in a distributed dataflow framework. In *OSDI*, volume 14, pages 599–613, 2014.

[8] Joseph E Gonzalez, Yucheng Low, Haijie Gu, Danny Bickson, and Carlos Guestrin. Powergraph: Distributed graph-parallel computation on natural graphs. In *Presented as part of the 10th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 12)*, pages 17–30, 2012.

[9] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.

[10] Julian Shun and Guy E Blelloch. Ligra: a lightweight graph processing framework for shared memory. In *ACM Sigplan Notices*, volume 48, pages 135–146. ACM, 2013.

[11] Xiaowei Zhu, Wenguang Chen, Weimin Zheng, and Xiaosong Ma. Gemini: A computation-centric distributed graph processing system. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 301–316, 2016.

[12] Aapo Kyrola, Guy E Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a pc. USENIX, 2012.

[13] Juno Kim and Steven Swanson. Blaze: fast graph processing on fast ssds. In *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*, pages 617–631. IEEE Computer Society, 2022.

[14] Xiaowei Zhu, Wentao Han, and Wenguang Chen. {GridGraph}:{Large-Scale} graph processing on a single machine using 2-level hierarchical partitioning. In *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, pages 375–386, 2015.

[15] Graph500 benchmarks. `https://graph500.org/`.

[16] Siddharth Samsi, Vijay Gadepally, Michael Hurley, Michael Jones, Edward Kao, Sanjeev Mohindra, Paul Monticciolo, Albert Reuther, Steven Smith, William Song, Diane Staheli, and Jeremy Kepner. Graphchallenge.org: Raising the bar on graph analytic performance. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*, pages 1–7, 2018.

[17] Mihai Capotă, Tim Hegeman, Alexandru Iosup, Arnau Prat-Pérez, Orri Erling, and Peter Boncz. Graphalytics: A big data benchmark for graph-processing platforms. In *Proceedings of the GRADES'15*, page 7. ACM, 2015.

[18] Donald Nguyen, Andrew Lenharth, and Keshav Pingali. A lightweight infrastructure for graph analytics. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 456–471. ACM, 2013.

[19] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The pagerank citation ranking: Bringing order to the web. Technical report, Stanford InfoLab, 1999.

[20] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[21] Jérôme Kunegis. Konect: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, pages 1343–1350. ACM, 2013.

[22] Haewoon Kwak, Changhyun Lee, Hosung Park, and Sue Moon. What is twitter, a social network or a news media? In *Proceedings of the 19th international conference on World wide web*, pages 591–600. AcM, 2010.

[23] Twitter Dataset from SNAP, SHA256: a8ea26e01be2d7d3bff29e4b80e3fac11deb83bb7c6 91552896e48e9f13098b1. 41.65M Vertices, 1.46B Edges. `https://snap.stanford.edu/data/twitter-2010.html`.

[24] Deepayan Chakrabarti, Yiping Zhan, and Christos Faloutsos. R-mat: A recursive model for graph mining. In *Proceedings of the 2004 SIAM International Conference on Data Mining*, pages 442–446. SIAM, 2004.

[25] Jurij Leskovec, Deepayan Chakrabarti, Jon Kleinberg, and Christos Faloutsos. Realistic, mathematically tractable graph generation and evolution, using kronecker multiplication. In *European conference on principles of data mining and knowledge discovery*, pages 133–145. Springer, 2005.

[26] Jure Leskovec, Deepayan Chakrabarti, Jon Kleinberg, Christos Faloutsos, and Zoubin Ghahramani. Kronecker graphs: An approach to modeling networks. *Journal of Machine Learning Research*, 11(Feb):985–1042, 2010.

[27] Vaastav Anand, Puneet Mehrotra, Daniel Margo, and Margo Seltzer. Smooth kronecker: Solving the combing problem in kronecker graphs. In *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pages 1–10, 2020.

[28] Kangfei Zhao and Jeffrey Xu Yu. All-in-one: Graph processing in rdbmss revisited. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 1165–1180, New York, NY, USA, 2017. Association for Computing Machinery.

[29] Zoi Kaoudi, Jorge-Arnulfo Quiane-Ruiz, Saravanan Thirumuruganathan, Sanjay Chawla, and Divy Agrawal. A cost-based optimizer for gradient descent optimization. In *Proceedings of the 2017 ACM International Conference on Management of Data*, SIGMOD '17, page 977–992, New York, NY, USA, 2017. Association for Computing Machinery.

[30] Yingxia Shao, Bin Cui, Lei Chen, Lin Ma, Junjie Yao, and Ning Xu. Parallel subgraph listing in a large-scale graph. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data*, pages 625–636, 2014.

[31] Sungpack Hong, Jan Van Der Lugt, Adam Welc, Raghavan Raman, and Hassan Chafi. Early experiences in using a domain-specific language for large-scale graph analysis. In *First International Workshop on Graph Data Management Experiences and Systems*, pages 1–6, 2013.

[32] Meeyoung Cha, Hamed Haddadi, Fabricio Benevenuto, and Krishna P. Gummadi. Measuring User Influence in Twitter: The Million Follower Fallacy. In *In Proceedings of the 4th International AAAI Conference on Weblogs and Social Media (ICWSM)*.

[33] Maciej Besta, Zur Vonarburg-Shmaria, Yannick Schaffner, Leonardo Schwarz, Grzegorz Kwasniewski, Lukas Gianinazzi, Jakub Beranek, Kacper Janda, Tobias Holenstein, Sebastian Leisinger, Peter Tatkowski, Esref Ozdemir, Adrian Balla, Marcin Copik, Philipp Lindenberger, Marek Konieczny, Onur Mutlu, and Torsten Hoefler. Graphminesuite: Enabling high-performance and programmable graph mining algorithms with set algebra. *Proc. VLDB Endow.*, 14(11):1922–1935, jul 2021.

[34] Flickr Image Relationships from SNAP. 106k Vertices, 2.31M Edges. http://snap.stanford.edu/data/web-flickr.html.

[35] Livemocha from SNAP. 104k Vertices, 2.2M Edges. http://snap.stanford.edu/data/web-flickr.html.

[36] Arvind Narayanan and Vitaly Shmatikov. Robust de-anonymization of large sparse datasets. In *2008 IEEE Symposium on Security and Privacy (sp 2008)*, pages 111–125. IEEE, 2008.

[37] Ryan Singel. Netflix cancels recommendation contest after privacy lawsuit, Jun 2017.

[38] Paolo Boldi and Sebastiano Vigna. The WebGraph framework I: Compression techniques. In *Proc. of the Thirteenth International World Wide Web Conference (WWW 2004)*, pages 595–601, Manhattan, USA, 2004. ACM Press.

[39] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In Sadagopan Srinivasan, Krithi Ramamritham, Arun Kumar, M. P. Ravindra, Elisa Bertino, and Ravi Kumar, editors, *Proceedings of the 20th international conference on World Wide Web*, pages 587–596. ACM Press, 2011.

[40] Laboratory for web algorithmics. http://law.di.unimi.it/datasets.php. Accessed: 2020-02-22.

[41] Himchan Park and Min-Soo Kim. Trilliong: A trillion-scale synthetic graph generator using a recursive vector model. In *Proceedings of the 2017 ACM International Conference on Management of Data*, pages 913–928, 2017.

[42] Farzad Khorasani, Rajiv Gupta, and Laxmi N. Bhuyan. Scalable simd-efficient graph processing on gpus. In *Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques*, PACT '15, pages 39–50, 2015.

[43] Comandur Seshadhri, Ali Pinar, and Tamara G Kolda. An in-depth analysis of stochastic kronecker graphs. *Journal of the ACM (JACM)*, 60(2):13, 2013.

[44] Andrew McCrabb, Hellina Nigatu, Absalat Getachew, and Valeria Bertacco. Dygraph: a dynamic graph generator and benchmark suite. In *Proceedings of the 5th ACM SIGMOD Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pages 1–8, 2022.

[45] Narayanan Sundaram, Nadathur Satish, Md Mostofa Ali Patwary, Subramanya R. Dulloor, Michael J. Anderson, Satya Gautam Vadlamudi, Dipankar Das, and Pradeep Dubey. Graphmat: High performance graph analytics made productive. *Proc. VLDB Endow.*, 8(11):1214–1225, July 2015.

[46] Maciej Besta, Marc Fischer, Vasiliki Kalavri, Michael Kapralov, and Torsten Hoefler. Practice of streaming processing of dynamic graphs: Concepts, models, and systems. *IEEE Transactions on Parallel and Distributed Systems*, 2021.

[47] Peter Macko, Virendra J Marathe, Daniel W Margo, and Margo I Seltzer. Llama: Efficient graph analytics using large multiversioned arrays. In *2015 IEEE 31st International Conference on Data Engineering*, pages 363–374. IEEE, 2015.

[48] Pradeep Kumar and H Howie Huang. Graphone: A data store for real-time analytics on evolving graphs. *ACM Transactions on Storage (TOS)*, 15(4):1–40, 2020.

[49] Dean De Leo and Peter Boncz. Teseo and the analysis of structural dynamic graphs. *Proceedings of the VLDB Endowment*, 14(6):1053–1066, 2021.

[50] Derek G Murray, Frank McSherry, Rebecca Isaacs, Michael Isard, Paul Barham, and Martín Abadi. Naiad: a timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 439–455. ACM, 2013.

[51] SocLiveJournal Dataset from SNAP, SHA256: b554569a74a2bfb9c56a3bdc68e55eeb54f68320dc1a3e7230718b03e6fb6e20. 4.84M Vertices, 68.99M Edges. https://snap.stanford.edu/data/soc-LiveJournal1.html.

[52] Uk-2007-05 dataset from Snap, MD5: 2ee454bf05cf0943abcaec2a9d4d33b5. 105.89M Vertices, 3.74B Edges. http://law.di.unimi.it/webdata/uk-2007-05/.

[53] Zenodo: research. shared. https://zenodo.org/. Accessed: 2023-07-26.

[54] The Dataverse Project: open source research data repository software. https://dataverse.org/. Accessed: 2023-07-26.

[55] Frank McSherry, Michael Isard, and Derek Gordon Murray. Scalability! but at what cost? In *HotOS*, volume 15, pages 14–14. Citeseer, 2015.

[56] Vignesh Balaji and Brandon Lucia. When is graph reordering an optimization? studying the effect of lightweight graph reordering across applications and input graphs. In *2018 IEEE International Symposium on Workload Characterization (IISWC)*, pages 203–214, 2018.

[57] Hao Wei, Jeffrey Xu Yu, Can Lu, and Xuemin Lin. Speedup graph processing by graph ordering. In *Proceedings of the 2016 International Conference on Management of Data*, SIGMOD '16, page 1813–1828, New York, NY, USA, 2016. Association for Computing Machinery.

[58] Junya Arai, Hiroaki Shiokawa, Takeshi Yamamuro, Makoto Onizuka, and Sotetsu Iwamura. Rabbit order: Just-in-time parallel reordering for fast graph analysis. In *2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 22–31, 2016.

[59] Yunming Zhang, Vladimir Kiriansky, Charith Mendis, Matei Zaharia, and Saman Amarasinghe. Optimizing cache performance for graph analytics. *arXiv preprint arXiv:1608.01362*, 2016.

[60] Manuel Hotz, Theodoros Chondrogiannis, Leonard Wörteler, and Michael Grossniklaus. Experiences with implementing landmark embedding in neo4j. In *Proceedings of the 2nd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pages 1–9, 2019.

[61] Patent Citation Dataset from SNAP, SHA256: a8ea26e01be2d7d3bff29e4b80e3fac11deb83bb7c691552896e48e9f13098b1. 3.77M Vertices, 16.52M Edges. https://snap.stanford.edu/data/cit-Patents.html.

[62] Joyce Jiyoung Whang, Andrew Lenharth, Inderjit S Dhillon, and Keshav Pingali. Scalable data-driven pagerank: Algorithms, system issues, and lessons learned. In *European Conference on Parallel Processing*, pages 438–450. Springer, 2015.

[63] Galois: Tutorial.

[64] Yudi Santoso. *Triangle counting and listing in directed and undirected graphs using single machines*. PhD thesis, 2018.

[65] Charalampos E Tsourakakis. Fast counting of triangles in large real networks without counting: Algorithms and laws. In *2008 Eighth IEEE International Conference on Data Mining*, pages 608–617. IEEE, 2008.

[66] Friendster dataset from SNAP, SHA256: bfbdc65a4af73e8732c8e1e2f55cc963a10894b12d3143544d5e9932f3feea95. 65.60M Vertices, 1.80B Edges. https://snap.stanford.edu/data/com-Friendster.html.

[67] Jinho Lee, Heesu Kim, Sungjoo Yoo, Kiyoung Choi, H Peter Hofstee, Gi-Joon Nam, Mark R Nutter, and Damir Jamsek. Extrav: Boosting graph processing near storage with a coherent accelerator. *Proceedings of the VLDB Endowment*, 10(12):1706–1717, 2017.

[68] Amitabha Roy, Ivo Mihailovic, and Willy Zwaenepoel. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, pages 472–488. ACM, 2013.

[69] Scott Beamer, Krste Asanović, and David Patterson. Gail: The graph algorithm iron law. In *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*, IA³ ’15, New York, NY, USA, 2015. Association for Computing Machinery.

[70] Sebastiano Vigna. Stanford matrix considered harmful. *arXiv preprint arXiv:0710.1962*, 2007.

[71] Yong Guo, Ana Lucia Varbanescu, Alexandru Iosup, Claudio Martella, and Theodore L Willke. Benchmarking graph-processing platforms: a vision. In *Proceedings of the 5th ACM/SPEC international conference on Performance engineering*, pages 289–292, 2014.

[72] Nadathur Satish, Narayanan Sundaram, Md Mostofa Ali Patwary, Jiwon Seo, Jongsoo Park, M Amber Hassaan, Shubho Sengupta, Zhaoming Yin, and Pradeep Dubey. Navigating the maze of graph analytics frameworks using massive graph datasets. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pages 979–990. ACM, 2014.

[73] Daniel Mawhirter and Bo Wu. Automine: Harmonizing high-level abstraction and high performance for graph mining. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, SOSP ’19, page 509–523, New York, NY, USA, 2019. Association for Computing Machinery.

[74] Ming Wu, Fan Yang, Jilong Xue, Wencong Xiao, Youshan Miao, Lan Wei, Haoxiang Lin, Yafei Dai, and Lidong Zhou. Gram: Scaling graph computation to the trillions. In *Proceedings of the Sixth ACM Symposium on Cloud Computing*, SoCC ’15, page 408–421, New York, NY, USA, 2015. Association for Computing Machinery.

[75] Kartik Lakhotia, Rajgopal Kannan, Sourav Pati, and Viktor Prasanna. Gpop: A scalable cache- and memory-efficient framework for graph processing over parts. *ACM Trans. Parallel Comput.*, 7(1), mar 2020.

[76] Scott Beamer, Krste Asanović, and David Patterson. The gap benchmark suite. *arXiv preprint arXiv:1508.03619*, 2015.

[77] Ldbc graphalytics. https://graphalytics.org/.

[78] Khaled Ammar and M. Özsu. Wgb: Towards a universal graph benchmark. In *WBDB*, page 58–72, 2013.

[79] Orri Erling, Alex Averbuch, Josep Larriba-Pey, Hassan Chafi, Andrey Gubichev, Arnau Prat, Minh-Duc Pham, and Peter Boncz. The ldbc social network benchmark: Interactive workload. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 619–630, 2015.

[80] Laxman Dhulipala, Jessica Shi, Tom Tseng, Guy E Blelloch, and Julian Shun. The graph based benchmark suite (gbbs). In *Proceedings of the 3rd Joint International Workshop on Graph Data Management Experiences & Systems (GRADES) and Network Data Analytics (NDA)*, pages 1–8, 2020.

[81] Zainab Abbas, Vasiliki Kalavri, Paris Carbone, and Vladimir Vlassov. Streaming graph partitioning: an experimental study. *Proceedings of the VLDB Endowment*, 11(11):1590–1603, 2018.