A Sampling-Based Tool for Scaling Graph Datasets

Ahmed Musaafir, Alexandru Uta
Vrije Universiteit Amsterdam
{a.a.m.musaafir,a.uta}@vu.nl

Henk Dreuning, Ana-Lucia Varbanescu
University of Amsterdam
{h.h.h.dreuning,a.l.varbanescu}@uva.nl

ABSTRACT

Graph processing has become a topic of interest in many domains. However, we still observe a lack of representative datasets for in-depth performance and scalability analysis. Neither data collections, nor graph generators provide enough diversity and control for thorough analysis. To address this problem, we propose *a heuristic method for scaling existing graphs*. Our approach, based on *sampling* and *interconnection*, can provide a scaled "version" of a given graph. Moreover, we provide analytical models to predict the topological properties of the scaled graphs (such as the diameter, degree distribution, density, or the clustering coefficient), and further enable the user to tweak these properties. Property control is achieved through a portfolio of graph interconnection methods (e.g., star, ring, chain, fully connected) applied for combining the graph samples.

We further implement our method as an open-source tool which can be used to quickly provide families of datasets for in-depth benchmarking of graph processing algorithms. Our empirical evaluation demonstrates our tool provides scaled graphs of a wide range of sizes, whose properties match well with model predictions and/or user requirements.

Finally, we also illustrate, through a case-study, how scaled graphs can be used for in-depth performance analysis of graph processing algorithms.

KEYWORDS

graph datasets scaling, heuristic methods, graph sampling, graph scaling tool

ACM Reference Format:
Ahmed Musaafir, Alexandru Uta and Henk Dreuning, Ana-Lucia Varbanescu. 2020. A Sampling-Based Tool for Scaling Graph Datasets. In *Proceedings of the 2020 ACM/SPEC International Conference on Performance Engineering (ICPE '20), April 20–24, 2020, Edmonton, AB, Canada.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3358960.3379144

1 INTRODUCTION

Graphs are popular due to their inherent simplicity in describing entities and interconnections. Thus, graph processing has become a topic of interest in many domains, such as information retrieval, biology, social networks, logistics, and infrastructure networks in real or virtual worlds. The rapid increase in data sizes and analyses complexity qualifies graph processing as a big-data challenge. Therefore, a lot of research effort is invested in devising efficient algorithms for graph processing in all these domains.

Benchmarking [12] shows that graph processing performance depends on many variables, such as *platform*, *algorithm*, *dataset*, and even the underlying *hardware* [28]. Thus, for the same graph processing platform and algorithm, variations in the dataset result in variations in performance, implying that both the platform and algorithm (implementations) are sensitive to graph properties like size and order, degree distribution, diameter, or clustering coefficient. The nature of these dependencies remains unknown: no generic models exist to capture these correlations. Thus, to empirically assess which graph properties impact an algorithm's performance the most, designers need many carefully crafted datasets which meet specific requirements. In most benchmarking methodologies or performance engineering for graph processing systems [6, 7, 9, 12], the issue of representative datasets is only marginally covered.

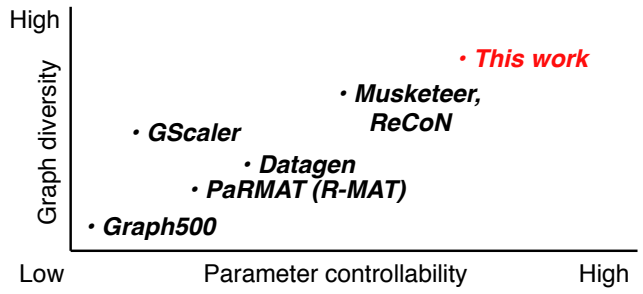

Figure 1: Depiction of our work along existing graph generators/scaling tools. Our work provides more controllability and outputs a wide range of (families of) diverse graphs than existing ones. The positions were determined by the features of each tool (described in Section 2).

Furthermore, in contrast to the many different existing platforms that can be used for graph processing, there is little option for dataset generation. We emphasize that there is need for:

(1) *Generating datasets of scale in a fast manner.*
(2) *Generating diverse datasets that could help uncover performance properties of graph processing systems.*
(3) *Achieving fine-grained parameter control over graph datasets.*

Most researchers use existing data collections [14, 16] or existing synthetic graph generators [5, 18] to obtain *some* datasets. Data collections provide only a few types of graphs. Generators are designed for specific types of networks, and tuned for specific graph properties, lacking generality. Neither option provides enough diversity and/or control to fulfill the requirements of thorough evaluation.

In this work, *we tackle the need for more and better datasets for graph analytics benchmarking* by generating *families of graphs*, where a given graph G is used as *seed* to build similar graphs of different sizes. To do so, we propose a heuristic method, based on *graph sampling*, where *siblings of* G with user-defined properties are obtained by controlled combinations of (multiple) samples of G.

ICPE '20, April 20–24, 2020, Edmonton, AB, Canada
2020. ACM ISBN 978-1-4503-6991-6/20/04…$15.00
https://doi.org/10.1145/3358960.3379144

We provide an open-source prototype implementation of this method as *a tool for graph scaling*[1]. With this tool, users (i.e., algorithm and systems designers) can *scale-down* an existing graph that may be too large for a given system, or *scale-up* an existing graph that is too small for scalability analysis[2] [22].

We validated our method by using our tool with various parameter configurations on a set of eight graphs. Our results demonstrate the feasibility of our approach for scaling graphs both up and down: for both operations, we do obtain families of similar graphs with user-controllable properties that can diversify the output graph. Our tool provides more controllability and graph diversity than existing graph generators or scalers (Fig 1). Furthermore, our tool is fast and runs efficiently on both single- and multi-node computing systems, thus avoiding scale limitations due to shear graph size.

We highlight the contributions of this work as follows:

(1) We propose a novel, sampling-based solution for parameterized, controllable scaling of graphs. By using existing sampling algorithms, our tool enables scaled graphs to preserve local and topological properties as guaranteed by the chosen sampling algorithm.
(2) We devise a set of models to predict the properties of scaled graphs and, based on these models, we provide a set of intuitive guidelines for configuring the parameters to control some of the scaled graph properties.
(3) We implement an open-source, easy-to-use prototype tool, which is fast in generating diverse *families of graphs* based on scaling existing graphs. The graphs we generate can be up to arbitrary large numbers of vertices/edges. Our tool works on both single- and multi-node systems, effectively accommodating graphs of any size.

2 MOTIVATION AND RELATED WORK

Many studies have demonstrated that graph processing performance varies significantly (i.e., orders of magnitude) depending (in non-trivial ways) on platform, algorithm, and dataset [8, 12, 29]. In turn, this means users must carefully select the processing algorithm and platform to be used for their specific problem.

2.1 Motivation: the case for families of graphs

Ideally, we should model the performance of a graph processing algorithm on a given platform as a function of algorithm and graph properties (e.g., size, order, or diameter): such models would provide a reasonable ranking of different algorithms, and thus allow users to select the most promising ones. *However, such models do not (yet) exist: their development has been so far prohibited by the large number of parameters and their diversity.*

An alternative way to determine correlations between graph properties and graph processing performance (leading, eventually, to building models to capture them) is an empirical approach: given an algorithm A and a platform P, provide *controlled variations* of the input datasets, G_i, and measure performance. By correlating the measured performance with the graph variation, the performance impact of different properties on algorithms can be quantified.

[1]Tool repository: https://github.com/amusaafir/graph-scaling
[2]Along this paper, we use "scale-up" and "expanding", as well as "scale-down" and "shrinking" as pairs of equivalent terms.

For example, Figure 2 illustrates the results of such an experiment: we measured the throughput of BFS graph traversal, in edges traversed per second, running on 18 datasets: 10 *correlated datasets* (colored black) and 8 *uncorrelated datasets* (colored gray). All results are presented in the increased order of the graph sizes. The uncorrelated datasets are obtained from existing data repositories, while the correlated datasets are a family of Graph500 graphs of scale 12 to 21 (i.e., graphs have the same structure, and graphs of consecutive scales are roughly doubling in size).

The results for the uncorrelated datasets do not indicate any performance trends: the observed performance is not directly correlated to the size of these graphs (i.e., larger graphs do not necessarily perform worse than smaller graphs - see *Com-livejournal* and *12month1*), because other graph structural properties also impact performance. However, the *correlated* datasets results demonstrate that performance can actually correlate with the properties of the graph, *iff* the graphs form a family for which multiple properties can be controlled. This experiment can be seen as a first step towards a sensitivity analysis.

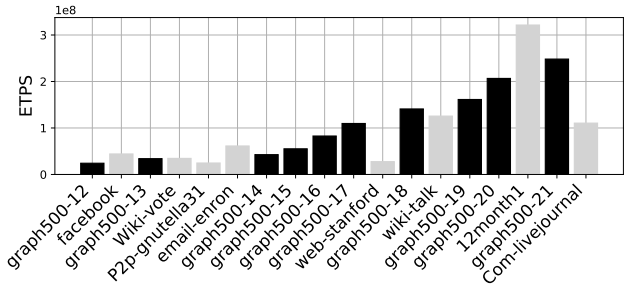

Figure 2: A performance comparison of a BFS traversal in terms of the number of edges traversed per second (ETPS) on different datasets ordered by graph size, from small to large. We distinguish between *uncorrelated datasets* (colored gray), and *correlated* datasets (colored black).

We augment this simple example in Section 7, by presenting a detailed analysis of two case-studies - i.e., BFS and PageRank - of empirical analyses which use families of scaled graphs to further increase our understanding of the impact graph properties have on graph processing performance.

2.2 Related work: obtaining graph families

While a sensitivity-analysis approach can be promising for understanding the interplay between performance and graph properties, setting up such experiments remains challenging: we lack mechanisms for fine-grain control over the properties of *diverse* (generated) input graphs. The Graph500 graphs are generated to be of controlled size and have similar properties (e.g., diameter and average degree). However, the diversity of graphs that generators can provide is very limited. Moreover, for real graphs, such families are not available. Our approach tackles these limitations: we propose a sampling-based method to generate families around real-life graphs.

For example, the Graph500 generator uses the Kronecker model to generate a graph; R-MAT [5] and Datagen [7] are other popular

examples of graph generators. In the Kronecker graph generative model, a graph is generated recursively using the Kronecker product. Each multiplication exponentially increases the size of the graph. The Kronecker graph generative model makes a given network denser over time, while the diameter shrinks [17]. R-MAT provides a solution to quickly generate realistic *graphs following the power-law degree distribution* by operating on the graph's adjacency matrix. R-MAT's model can generate directed, undirected, and bipartite graphs [21]. PaRMAT [13] can be used to generate large R-MAT graphs and provides controllability over certain properties (e.g., directed/undirected, loops). LDBC Datagen [25] is a synthetic graph data generation tool for the LDBC Social Network Benchmark [7]. Datagen graphs *mimic real-world social media graphs* in terms of degree distribution, following a discretized power law distribution. Datagen produces evolving, time-correlated graphs, modeling various types of connections between entities (e.g., friendship, message posting). It also includes several different degree distributions.

All generators, like R-MAT, Kronecker, or Datagen focus on *generating graphs from scratch*. Instead, our approach scales an *existing graph*, preserving some of its potentially unique properties; using multiple scale factors, one can obtain a family of correlated graphs.

Scaling methods can produce more diverse graphs than tools that generate graphs from scratch, because they allow users to provide an arbitrary input graph. For example, GScaler [31] creates a larger graph by decomposing an initial graph into small pieces that are subsequently scaled, based on the indegree/outdegree correlation of nodes and edges. GScaler aims to preserve properties of the original graph, but, in contrast to our approach, it does not provide any control over them. Furthermore, it lacks support in preserving undirected graphs. Musketeer [10] is a newer graph generation method, which takes an original graph as input, along with parameters (such as node growth rate and node edit rate), and aims to reproduce all of its features [21] by applying coarsening and uncoarsening operations to obtain the desired scale of the graph. While this algorithm shows promising results, it is noted by the authors that the existing implementation is not fast enough for large networks. ReCoN [26] is the follow-up algorithm of Musketeer, aiming to preserve its qualities while improving its performance. ReCoN scales a graph by creating disjoint copies of it up until the preferred size, and then applying some randomization on the edges inside and between the communities of the graph. This randomization step of the edges should make the copies become connected with each other, leading to a scaled output graph. Both Musketeer and ReCoN have somewhat similar goals with our expansion idea, but use different methods; moreover, there is little user control on the expanded graph properties. Instead, our approach is more flexible and enables the generation of larger families of graphs, with more diverse siblings.

In summary, our work proposes an heuristic method for building graph families, with controlled properties, starting from a given, real-life network. Our method provides several ways to control graph properties, and thus it can diversify the generated families. We further embed this method into a graph scaling tool, which provides users with a constructive, parameterized approach to generate graph families, and models to understand the graph properties of the scaled datasets. We depict our work along the existing tools based on the diversity of graphs and the parameter controllability in Figure 1. Our solution is superior to related work in both dimensions, by design.

Table 1: Graph property preservation quality per sampling algorithm, represented as likelihood from low (−−) to high (++).

	Random Node	Random Edge	TIES	Random Walk	Forest Fire
Scale (fraction based on)	V	E	V	V/E	V/E
Degree distribution	+	+	++	+	+
Diameter	+	+	−	+	+
Connectivity	−−	−	−	−−	−
Avg. Clustering Coefficient (C.C.)	+	−	++	+	+
Avg. Shortest Path Length (S.P.L.)	+	−	++	+	++

3 SAMPLING METHODS

In this section we briefly discuss graph sampling, which is at the core of our heuristic method, and explain the selection process for a suitable graph sampling algorithm.

Graph sampling is the process of selecting a sub-graph $G' = (V', E')$ of graph $G = (V, E)$, where $V' \subseteq V$ and $E' \subseteq E$. There are many methods and algorithms aiming to obtain a reduced, representative sample from an original graph [11].

To be suitable for our method, which combines samples of the seed graph to generate siblings, a sampling algorithm must *preserve* as many properties as possible from the original graph. We have analyzed five of the most popular sampling algorithms and their claims for graph property preservation [1, 2, 4, 11, 15, 19, 30]. The results of the analysis are shown in Table 1; they indicate that Total Induced Edge Sampling (TIES) [1] has the highest likelihood to preserve most local and topological properties of the original graph. We note that property preservation for each sampling algorithm is not guaranteed, but *more* or *less* likely, as indicated by + or −, respectively; the original graph topology and size, as well as the sample size, can also affect the analysis.

All the sampling algorithms listed in Table 1 have been implemented in our prototype tool. Given the superiority of TIES, all the experiments in this paper (§6) are using this sampling algorithm.

TIES consists of two steps: (1) edge-based node sampling and (2) induction [1, 3]. During step (1), a random set of edges from the original graph G are selected, along with their end-vertices, until the desired fraction of vertices is reached. So far, TIES is similar to Random Edge Sampling. However, in order to counter the sparseness of the Random Edge Sampling algorithm, step (2) adds all other edges that exist between the vertices selected in step (1).

Furthermore, we must emphasize that the *scale* of the sampled graph depends on either the fraction of desired vertices or edges. For example, when using TIES or Random Node, the desired scale is based on the number of vertices. For Edge Sampling, this is typically (although not strictly enforced by) the fraction of edges. Topological-based sampling algorithms like Random Walk or Forest

Fire can use *either* vertices or edges. In our case-studies (§7), we have used the number of edges when using Forest Fire as sampling algorithm.

4 SAMPLING-BASED GRAPH SCALING

In this section we introduce our heuristic, sampling-based method for graph scaling. We further detail the method and its parameters, and analyse the impact of its parameters on the properties of the scaled graphs.

4.1 Overview of the method

Intuitively, a scaled version of a graph G is a larger or smaller graph, similar to G, where certain topological and local properties are preserved. Thus, the main challenge when scaling graphs is to provide a method that defines and/or controls *similarity* and *property preservation* between the original and the scaled graphs. For example, when *doubling* a graph, no domain-agnostic definition exists for how the diameter of the new graph should compare to that of the original graph. Instead, our method enables users to control the scaling according to problem-specific requirements. In other words, we allow the users to decide whether the diameter should double or stay (roughly) the same.

The core idea of our method is to build a scaled graph G' by interconnecting several samples of the original graph, G. By varying the type and size of these samples, and the way they are interconnected, we provide a large family of similar graphs with controlled properties. Inside this family, *similarity* is ensured by only using representative samples of the original graph as building blocks, while *property preservation* is controlled by the supported interconnection templates.

We note that this approach fundamentally differentiates our method from graph-evolution methods: our goal is not to generate a later version of the same graph, but to provide several, smaller/larger versions of the same graph, which could have followed different evolution paths. Moreover, we do not concern ourselves with the feasibility of the scaled graphs for the domain of origin (i.e., when doubling a road network, our method does not enforce the result to also be a road network). Such feasibility properties can be further enforced by filtering the generated graphs according to the rules of the domain, in the rare cases when these are available.

4.2 The method and its parameters

The easiest method for scaling-down graphs is simply sampling the original graph to the desired size, thus ensuring G' has the properties guaranteed by the sampling algorithm.

Scaling-up, on the other hand, requires the generation of new vertices and edges for the larger graph. In our method, we add edges and vertices by combining several (partial) graph replicas, i.e., samples of the original graph G of sizes (0.0–1.0]. Figure 3 illustrates our approach for graph scaling with an example. In this case, we want to expand graph G 4.5 times (this includes the original graph itself). First, we generate three full replicas of G, denoted G_i, $i = 1..3$. We link each G_i to the original graph by adding one edge. Next, we generate a partial replica of G, half its size, by sampling. The sample G_s, with $E_s \subset E$ and $V_s \subset V$, is finally linked back to the original graph in a similar fashion. The result $G_e =< E_e, V_e >$ can be seen as a superset of G, i.e., $E_e \supseteq E$ and $V_e \supseteq V$. Note further that we expect $|V_e| \approx |V| \times s$ and $|E_e| \approx |E| \times s + \lceil (s-1) \rceil$.

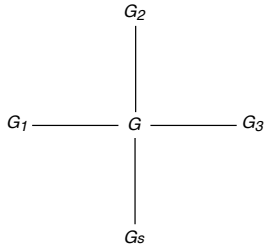

Figure 3: Using graph copies.

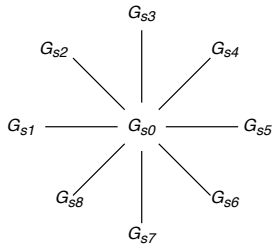

Figure 4: Using sampling.

4.2.1 Generalized scaling. Instead of interconnecting full copies of the graph, we can also use samples. For example, to expand the graph $s = 4.5$ times, we can use $n = 9$ samples, each representing 0.5 of the original graph (see Figure 4). If our sampling algorithm is random (as in the case of TIES), we can generate a different sampled graph, G_{s_i}, $i = 0..8$, for each sampling operation we conduct. As a result, the graph is more diverse, but the entire operation is significantly more compute-intensive: we expect a 9x slowdown because we need to run the sampling algorithm nine times. Note that for this particular variation, there is no guarantee that every node from the input graph G exists in the final result, depending on the used sampling algorithm.

Of course, the generalization can continue further: the samples need not be equal and the same strategy can be applied for scaling-down graphs, too. Thus, we formalize our generalized graph scaling as follows.

Definition 1. *Given a graph $G =< E, V >$ and a scaling factor $s > 0$, the scaled version of G, noted $G' =< E', V' >$ is defined as the interconnection of $n > 0$ samples of G, denoted $G_{s_i} =< E_{s_i}, V_{s_i} >$, with $0 < s_i \leq 1$ the sampling factor of each sample, such that $s = \sum_{i=0}^{n-1} s_i$. Thus, $E' = \bigcup_{i=0}^{n-1} E_{s_i} \cup C$, where C is the set of connecting edges between the samples ($C = \{\forall c = (u, v) | u \in V_{s_i}, v \in V_{s_j}, i \neq j\}$) and $V' = \bigcup_{i=0}^{n-1} V_{s_i}$.*

To *control* the properties of the scaled graphs, two classes of parameters are provided: *sampling parameters* (e.g., the sampling algorithm, sample sizes, and number of samples to use), and *interconnection parameters* (e.g., interconnection topology, density, and bridge-vertex selection). All these parameters impact the properties of the scaled graph, G'. In the following paragraphs, we describe these parameters in more detail and analyze their impact on the properties of G'; in Section 6 we will empirically assess their impact on the properties of G'.

4.2.2 Sampling parameters. The sampling algorithm plays an important role in our scaling approach: the properties it preserves are key to understanding and controlling the properties of the scaled graph. We support 5 algorithms, and the selection is currently left to the user based on hints similar to those in Table 1; a semi-automated selection is envisioned for a next version of our tool. As expected, the number of samples, their sizes, and the choice between homogeneous and non-homogeneous samples are also parameters of

interest for the properties of G'. For example, in the previous case where the scaling factor is 4.5, we can use 9 samples of 50%, or 5 samples of 90%, or, furthermore, 5 samples of 70% plus the full graph. As expected, larger samples better preserve the properties of the original graph [15]. In the tool, we warn against using samples lower than 0.4.

4.2.3 Interconnection topologies. The sampled graphs can be interconnected using a variety of topologies. The current implementation supports various network topologies, such as star, ring, chain and fully-connected (Figure 5). The topology can be also defined by the user. The interconnection topology will impact some of the properties of the resulting graph. For example, the diameter of G_e will be smaller for a star interconnection than for a chain interconnection. A similar behavior is expected for the average path length property.

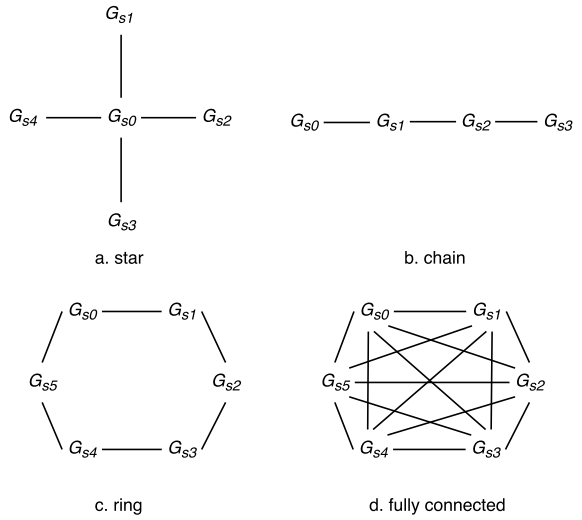

Figure 5: Examples of topology structure.

4.2.4 Selecting bridge-vertices. We define the generated edges that interconnect the samples as *bridge-edges* (or, shorter, *bridges*), and the vertices they connect (i.e., the source and destination of a bridge-edge) as *bridge-vertices*. The selection of bridge-vertices can be done by the user. In our current implementation and experiments, we use either *random* or *high-degree* bridge-vertices.

4.2.5 Multi-edge interconnections. When interconnecting samples with single bridges, both the average degree and the density of the expanded graph are, with high probability, lower than those of the original graph. To alleviate this problem, we can enable multiple bridges between the G_e components. This imposes additional parameter tuning, as we need to decide how many bridges and how to select the bridge-vertices, and, at the same time, enable additional control over the properties of the expanded graph.

4.3 Parameters and Property-prediction models

All parameters described in the previous section (the sampling combinations and the interconnection topology, density, and bridge selection) have an impact on the quality of the scaled graph.

They are also inputs for our graph scaling tool, where a user must select parameters' values depending on the required properties for G'. The correlation between the parameters' values and the resulting graph properties is not trivial. Thus, we have devised analytical property-prediction models to capture these correlations and, therefore, help users make informed choices. A small subset of these models, built for a chain topology with different scaling parameters, is shown in Table 2. The complete set of property-prediction models, for all supported topologies, is provided and constantly updated in the public repository of our tool [22].

For each model, several user guidelines for parameter tuning have been extracted[3]. For example, based on the models presented in Table 2, a guideline can be: *"Scale-up: to scale-up the diameter, too, choose a chain topology with a single random bridge"* or *"Scale-up: to preserve density, choose a chain topology with multiple bridges."*. Currently, users need to follow these guidelines to search for the right parameters' values. Unfortunately, defining a complete set of such rules is currently impossible because not all scenarios and/or user requirements are known. In turn, this means that users might require several iterations until generating the right family of graphs. We introduce the notion of an *auto-tuner*, that helps users find the right parameters to hit a specific graph property in the next section.

5 GRAPH SCALING TOOL

In this section we describe our graph scaling tool and its current implementation.

5.1 User interaction

We have designed a tool that implements both up- and down-scaling, using our methodology. This tool allows users to experiment with different combinations of parameters and quickly assess the results.

For scaling-down, the user must specify an input graph G and the desired shrinking factor as a fraction of the original graph G. Based on this input, the tool will output the scaled-down graph G', as well as a brief analysis of its properties.

For scaling-up, the user must specify an input graph G and a scaling factor that indicates how much larger G' should be, in terms of numbers of vertices or edges (depending on the selected sampling algorithm). Furthermore, the sampling, topology, bridges, and multi-edge interconnections parameters must also be specified. Using our property-prediction models (see Table 2 for examples), a prediction of the G' properties is immediately provided, for the user to assess whether the result would be acceptable.

5.2 Auto-tuner

Using our models, we provide an user-in-the-loop auto-tuner that iteratively searches for a graph with a certain (user) preferred property value. In the current implementation of the tool, we specifically provide this mechanism to control the diameter of the scaled-up

[3]The full list of models is presented and updated in our repository [23].

Table 2: Quantifying the impact of different parameters choices for constructing G_e using the *chain* topology. Note that the values obtained by using these models are upper-bound (e.g., for the diameter property, it is the *maximum* diameter that can be obtained). Models marked with * have been determined empirically.

Bridge Vertices	Random	Random	High degree	High degree
Bridges	1	b	1	b
$\lvert V_e \rvert$	$\sum_1^n \lvert V_{s_i} \rvert$	$\sum_1^n \lvert V_{s_i} \rvert$	$\sum_1^n \lvert V_{s_i} \rvert$	$\sum_1^n \lvert V_{s_i} \rvert$
$\lvert E_e \rvert$	$\sum_1^n \lvert E_{s_i} \rvert + (n-1)$	$\sum_1^n \lvert E_{s_i} \rvert + (n-1) \times b$	$\sum_1^n \lvert E_{s_i} \rvert + (n-1)$	$\sum_1^n \lvert E_{s_i} \rvert + (n-1) \times b$
#Components	$n \times G_{s_i}$	$n \times {G_{s_i}}^*$	$n \times G_{s_i}$	$n \times {G_{s_i}}^*$
Diameter	$\sum_1^n \max D(S_i) + (n-1)$	$\sum_1^n \max D(S_i) + (n-1)$	$\sum_1^n \max D(S_i) + (n-1)$	$\sum_1^n \max D(S_i) + (n-1)$
Avg. C.C.	Similar	Decreases*	Similar*	Decreases*
Avg. S.P.L.	$\approx (n-1) \times AvgP(S_i)$	Decreases with b	$\approx (n-1) \times AvgP(S_i)^*$	Decreases with b*
Avg. degree	Similar	Increases with b	Similar*	Increases with b*
Density	Decreases	Increases with b	Decreases	Increases with b

graph. The auto-tuner uses a binary search tree, where the (maximum) values of the diameter are initially calculated based on the models of the topology and are subsequently inserted into the tree. The values of the diameter for each topology can be decreased by adding more/different interconnections. A larger diameter value can be obtained by using a different topology or by using smaller sample sizes (in case, for example, the chain topology was not able to reach the desired diameter). For N iterations, the auto-tuner adjusts the scale-up parameters accordingly based on the existing parameters and values in the tree, and analyses the value of the diameter from the output graph. The parameters (and the resulting graph) that lead to a diameter best matching the request are provided to the user, who can also decide whether the search continues or not.

A main challenge that comes into play is when combining multiple preferred property values where the correlation between the properties is not directly observable through the models. For example, hitting a preferred diameter whilst also satisfying a specific clustering coefficient value is a challenge that still needs to be addressed.

5.3 Implementation

Scaling a graph has two stages: sampling and combining. In case of scaling-down, for robustness, our current prototype tool defaults to sampling, and skips combination. For scaling-up, both phases are required, with the sampling dominating the overall execution time of the scaling operation (e.g., the complexity of TIES is $O(|V|+|E|)$).

Our tool is available as a single node version, able to processes graphs as large as the local system's memory allows it. Moreover, where GPUs are available, acceleration (up to 70×) in the induction phase is provided. Finally, for large graphs, we employ distributed sampling, based on our own MPI-based prototype, available in the repository. This distributed version uses local sampling, and has a deterministic policy (i.e., duplicate removal) to mitigate potential conflicts between local samples[4].

[4]More details on the implementation and effects of this policy are available in the repository.

6 EXPERIMENTAL EVALUATION

In this section, we present our experimental setup and analyze a subset of our results[5]. The goal of all the experiments is to show the capabilities and limitations of our method. Therefore, we measure the quality of scaling up and down on nine different graph datasets (shown in Table 3) obtained from the SNAP [20], KONECT [14] and NetworkRepository [24] archives. Our main findings are:

(1) Scaled-down graphs, based on sampling using TIES, respect the property preservation guarantees provided by the original algorithm. We specifically address degree distribution, diameter, average clustering coefficient and average shortest path length distribution. Additional properties, like the number of connected components (connectivity), vary per graph (Section 6.1). An in-depth statistical analysis (facilitated by our tool) is required to determine more systematic rules.
(2) Scaled-up graphs, using TIES sampling and our interconnection rules, comply by our models for the properties guaranteed by TIES. The degree distribution, diameter, average clustering coefficient, and average shortest path length distribution all comply (Section 6.2). In contrast, for properties that are not guaranteed by TIES, scaled-up graphs do not show uniform behavior.
(3) Our tool is fast and scales well across multiple compute nodes for the sampling phase (Section 6.3).
(4) Parameter setting for our tool can be cumbersome for a novice user. While our models are accurate in terms of choices, there are cases when they are not precise enough to narrow down parameters to specific values. In these cases, some (auto-)tuning is necessary. The current auto-tuner for the diameter obtains a graph that matches the preferred diameter of the user after a number of iterations.

In the following paragraphs, we present and analyze in more detail a subset of our experimental results. We include detailed results only for *Com-Orkut* for graph down-scaling, and for *Facebook* for up-scaling. The results for all the graphs in Table 3 are available online [22].

[5]Results for all datasets are provided in the repository of the tool [22].

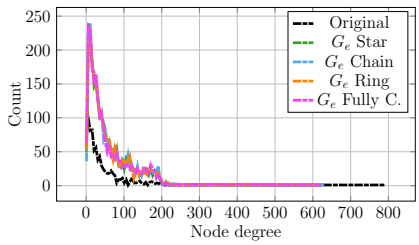

Figure 6: Degree distribution of the expanded Facebook graphs (G_e random) using different topologies.

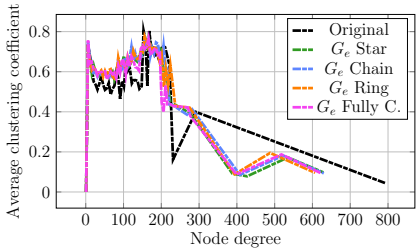

Figure 7: Average clustering coefficient distribution of the expanded Facebook graph (G_e random). Note that a small number of high degree nodes exist (shown by the straight lines).

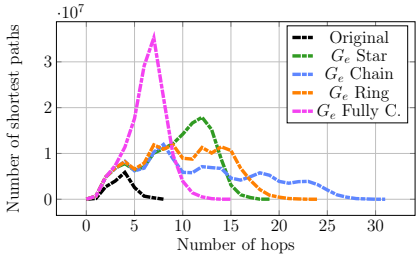

Figure 8: Shortest path length distribution of the expanded Facebook graph using different topologies with a single interconnection and random bridge selection (G_e random).

Table 3: List of the graph datasets used for scaling-up and scaling-down experiments. Datasets marked with * are also used in the case-studies (7).

Dataset	Nodes	Edges	Results in
Undirected			
Com-Orkut	3,072,441	117,185,083	§6.1
Com-LiveJournal*	3,997,962	34,681,189	Repository, 7
12Month1*	872,622	22,501,700	7
Enron-email	36,692	183,831	Repository
Facebook	4,039	88,234	§6.2,6.3
Directed			
Wiki-talk	2,394,385	4,659,565	Repository
Web-Stanford	281,903	1,992,636	Repository
Gnutella	62,586	147,892	Repository
Wiki-vote	7,115	100,762	Repository

Table 4: Properties of the Com-Orkut graph along different samples. *Obtained from a subset of starting test nodes.

	G	G_s 0.8	G_s 0.5	G_s 0.3
Nodes	3,072,441	2,457,952	1,536,220	921,733
Edges	117,185,083	108,686,099	73,626,482	42,194,208
Avg. deg	76.28	88.44	95.85	91.55
Diameter*	9	9	10	8
Density	2.48e-05	3.59e-05	6.24e-05	9.93e-05
#Components	1	7	17	36
Connected	Yes	No	No	No
Avg. C.C.	0.16	0.15	0.15	0.14
Avg. S.P.L.*	4.19	4.05	3.97	3.95
Runtime (s)	-	71.01	66.00	61.96

6.1 Down-scaling graphs

Using our tool and TIES as sampling algorithm, we have shrunk all datasets using 3 factors: 0.3, 0.5, and 0.8. Table 4 provides an overview of the properties of the scaled-down *Com-Orkut* graphs.

Our scaling-down results across all datasets demonstrate similar property preservation as observed by the authors of TIES, and reflect the qualities of TIES as shown in Table 1.

However, the number of connected components shows erratic behavior for different graphs and sample sizes. As this is a property for which TIES offers no guarantees, further analysis is required to understand whether behavioral patterns can be observed and further used for predicting sample properties.

6.2 Up-scaling graphs

This section presents our graph expansion analysis, using the *Facebook* dataset. All the models we use to guide our expectations (similar to those presented in Table 2) and more dataset-specific results are described in our online repository [22].

6.2.1 Topology, bridges, and multi-edge interconnections. Using our tool, each dataset has been expanded 3 times using a sample size of 0.5 (resulting in 6 different samples), along different topologies, multi-edge interconnections and bridge-vertices selection. Table 5 shows the results of the expansion of the Facebook graph using these different types of parameters. Note that the first row represents the properties of the original graph G. The second row contains an example of a single sampled version G_s using a sample size of 0.5.

We observe that the expanded graphs closely match our models, with the exception of the *connected components* property, as this is not being preserved by the sampling algorithm in general. Furthermore, the *average clustering coefficient* drops significantly across all topologies as we add more interconnections. An interesting observation is that the use of high-degree bridge selection reduces this drop in comparison to random bridge selection.

By observing the distributions of properties from several expansions, we note that:

(1) The *degree distribution* (Figure 6) of the expanded graphs have similar degree numbers, but a larger amount, because the different samples are connected using a single interconnection, thus allowing multiple nodes of the same degree to exist; at the same time, local node degrees are less affected and the average degree of the graph itself remains similar to a single sample. This behavior also holds for other datasets [22].
(2) The *average clustering coefficient distribution* for the G_e *random* expansion (Figure 7) is close to the original graph, because of the use of a single interconnection. However, the distribution of G_e *high-degree, denser* expansions does not resemble the original graph any longer and becomes lower as we add more

Table 5: Properties of the Facebook graph obtained from various expansions using different topologies (3 times expansion using a sample size of 0.5.)

	Topology	Bridge	#Intercon.	Nodes	Edges	Avg. deg	Diameter	Density	#Components	Connected	Avg. C.C.	Avg. S.P.L.
Facebook G	-	-	-	4,039	88,234	43.69	8	1.10e-2	1	Yes	0.62	3.69
G_s (0.5)	-	-	-	2,020	57,602	57.03	7	2.82e-2	2	No	0.63	3.55
G_e (random)	Star	Random	1	12,117	339,497	56.04	19	4.62e-3	7	No	0.63	9.26
G_e (random, denser)	Star	Random	45,000	12,114	559,798	92.42	6	7.62e-3	2	No	0.31	2.65
G_e (high-degree)	Star	High-degree	1	12,115	341,636	56.40	20	4.65e-3	7	No	0.64	9.19
G_e (high-degree, denser)	Star	High-degree	45,000	12,115	560,168	92.48	6	7.63e-3	10	No	0.46	2.92
G_e (random)	Chain	Random	1	12,114	340,091	56.15	31	4.63e-3	9	No	0.63	11.79
G_e (random, denser)	Chain	Random	45,000	12,115	561,560	92.70	7	7.65e-3	1	Yes	0.27	2.99
G_e (high-degree)	Chain	High-degree	1	12,114	340,588	56.23	33	4.64e-3	7	No	0.64	11.85
G_e (high-degree, denser)	Chain	High-degree	45,000	12,117	561,841	92.74	10	7.65e-3	9	No	0.44	3.46
G_e (random)	Ring	Random	1	12,117	339,013	55.96	24	4.61e-3	7	No	0.64	9.91
G_e (random, denser)	Ring	Random	45,000	12,114	605,007	99.89	6	8.24e-3	1	Yes	0.23	2.68
G_e (high-degree)	Ring	High-degree	1	12,114	340,040	56.14	23	4.63e-3	6	No	0.64	9.50
G_e (high-degree, denser)	Ring	High-degree	45,000	12,115	604,145	99.74	8	8.23e-3	8	No	0.42	3.06
G_e (random)	Fully C.	Random	1	12,114	339,777	56.09	15	4.63e-3	7	No	0.63	6.35
G_e (random, denser)	Fully C.	Random	45,000	12,115	1,637,128	270.26	4	2.23e-2	1	Yes	0.09	2.15
G_e (high-degree)	Fully C.	High-degree	1	12,115	341,080	56.30	15	4.64e-3	10	No	0.63	6.15
G_e (high-degree, denser)	Fully C.	High-degree	45,000	12,114	1,631,950	269.43	10	2.22e-2	8	No	0.37	2.64

interconnections. A similar behavior of this property can be observed for other datasets [22].

(3) The fully connected topology in the *average shortest path distribution* (Figure 8) has the highest number of shortest paths, while other topologies have lower numbers of paths and more hops. This corresponds to the point statistics numbers we provide in our results (including Table 5).

6.2.2 Scaling factor and sample size.

In order to observe the impact of the scaling factor, we have expanded the *Facebook* dataset up to ≈1.1 million edges and the *Com-Orkut* dataset up to ≈1.5 billion edges using a factor of 6.5 and 10, and a sample size of 0.5. These experiments result in 13 different samples for the scaling factor of 6.5, and 20 different samples for the scaling factor of 10. The properties for these different expansions are provided in Table 6.

Furthermore, Table 7 shows the result of expanding the *Facebook* dataset 4.5 times using a sample size of 0.5, thus resulting in a number of 9 different samples, and a sample size of 0.9, which results in a number of 5 different samples. Both expansions still adhere to our models and expectations.

6.2.3 Controllability analysis.

Table 8 summarizes our empirical analysis of the controllability of graph up-scaling. The analysis focuses on the correlation between scaling parameters and resulting graph properties.

6.3 Performance

Finally, we have analyzed the execution time and scalability of our tool. The execution of the tool is composed of I/O operations (reading/writing graphs from disk), a sampling phase executed multiple times (as directed by the user), and a combining phase where the bridge-vertices are selected and the new edges are introduced. The most compute-intensive phase is the sampling one, which is repeatedly invoked. We have included the total run-time in Table 4 for the different samples. Note that these performance numbers depend on the (edge) size of the selected input graph.

For scaling-up, our tool outperforms most existing (distributed) platforms whilst running on a single machine. For example, the total execution time of our largest scale-up (*Com-livejournal* graph using a scaling factor of 20 and sample size of 0.5) was under 50 minutes. Existing tools on the other hand, when at all able to complete this operation using the same target graph size, take several hours to accomplish the task. The complete performance analysis results, as well as a detailed description of the software and hardware setup, are presented in our repository.

7 CASE-STUDIES

A prominent use-case for our graph scaling approach is detailed performance analysis of different graph processing algorithms and/or platforms (see Section 2; our goal is to extend and diversify the graph families provided by generators like Graph500 [6] with families seeded by real-life networks.

To demonstrate in detail how these scaled graph families can and should be used for performance analysis, we conduct two case studies: we analyse how the properties of scaled-up graphs impact the performance of two graph processing algorithms. Specifically, we measure the processing time of Breadth-First Search (BFS) and

Table 6: Example of expanding the Facebook and Com-Orkut graph using a scaling factor of 6.5, and 10. *Obtained from 10,000 starting test nodes..

	Topology	Bridge	#Intercon.	Nodes	Edges	Avg. deg	Diameter*	Density	Components	Connected	Avg. C.C.	Avg. S.P.L.*
Facebook (original) G	-	-	-	4,039	88,234	43.69	8	1.10e-2	1	Yes	0.62	3.69
G_s (0.5)	-	-	-	2,020	57,602	57.03	7	2.82e-2	2	No	0.63	3.55
G_e (6.5x, 0.5)	Chain	Random	1	26,252	736,666	56.12	55	2.13e-3	26	No	0.63	20.19
G_e (10x, 0.5)	Chain	Random	1	40,385	1,142,378	56.57	68	1.40e-3	25	No	0.63	23.10
Com-orkut (original) G	-	-	-	3,072,441	117,185,083	76.28	9	2.48e-05	1	Yes	0.16	4.19
Gs (0.5)	-	-	-	1,536,220	73,626,482	95.85	10	6.24e-05	17	No	0.15	3.97
Ge (6.5, 0.5)	Star	Random	1	19,970,861	957,116,088	95.85	22	4.79e-06	216	No	0.14	12.03
Ge (10x, 0.5)	Star	Random	1	30,724,401	1,472,465,199	95.84	21	3.11e-06	352	No	0.14	11.59
Ge (6.5x, 0.5)	Chain	Random	1	19,970,862	957,135,693	95.85	63	4.79e-06	243	No	0.14	23.97
Ge (10x, 0.5)	Chain	Random	1	30,724,401	1,472,597,664	95.85	95	3.11e-06	370	No	0.14	33.38
Ge (6.5x, 0.5)	Ring	Random	1	19,970,861	957,166,336	95.85	37	4.79e-06	215	No	0.14	17.96
Ge (10x, 0.5)	Ring	Random	1	30,724,406	1,472,502,728	95.85	54	3.11e-06	368	No	0.14	26.43
Ge (6.5x, 0.5)	Fully C.	Random	1	19,970,861	957,009,373	95.84	14	4.79e-06	214	No	0.14	7.63
Ge (10x, 0.5)	Fully C.	Random	1	30,724,404	1,472,491,564	95.85	15	3.11e-06	334	No	0.14	7.83

Table 7: Example of expanding the original Facebook graph 4.5 times using 9 different samples of size 0.5 (small sample) and an expansion using 5 different samples of size 0.9 (large sample). *Obtained from 13,000 starting test nodes.

	G	G_s (0.5)	G_e (4.5, 0.5)	G_e (4.5, 0.9)
Topology	-	-	Star	Star
Bridge	-	-	Random	Random
#Intercon.	-	-	1	1
Nodes	4,039	2,020	18,172	18,175
Edges	88,234	57,602	515,213	431,014
Avg. deg	43.69	57.03	56.70	47.42
Diameter*	8	7	20	20
Density	1.10e-2	2.82e-2	3.12e-3	2.60e-3
Components	1	2	12	2
Connected	Yes	No	No	No
Avg. C.C.	0.62	0.63	0.63	0.60
Avg. S.P.L.*	3.69	3.55	9.80	9.76

PageRank (PR) using GraphMat [27], a state-of-the-art platform for distributed graph processing, on a family of scaled-up *com-livejournal* and *12month1* graphs (listed in Table 3). This benchmark has been conducted using the Graphalytics benchmark suite [12]. Each dataset and algorithm benchmark consists of three runs using a single machine and 32 threads. Note that the processing time is the actual algorithm running time and does not include any platform-specific overhead or loading of the graph, as specified in the Graphalytics benchmark suite.

Figure 9 shows the processing times of the scaled-up graph family. All scale-ups use a sample size of 0.5 and *random bridges* for interconnections. The exact performance numbers (including the properties of each scaled-up graph) are available in our repository [22]. Note that our case-studies use both TIES and Forest Fire as sampling algorithms for the scaling operations. We selected TIES because of its superiority in terms of property preservation. We added Forest Fire to also allow an algorithm that preserves the single component connectivity of the original input graph (which may get disconnected when using TIES) and to ensure the root BFS node is part of the sample (it is possible for TIES not to select this node in its sampling passes). While Forest Fire does preserve the single component connectivity of the original input graph, the sampling quality in terms of property preservation is not as good as TIES', unless larger sample sizes are being used.

Another notable difference is that TIES hits the exact number of vertices depending on the desired scaling size, while the edges are typically oversampled, while in samples generated by Forest Fire, the number of edges can be controlled through the scaling size, and the number of vertices is typically slightly oversampled.

Furthermore, in order to merely observe the effects that the topology and interconnections have on performance regardless of the selected sampling algorithm (and sample size), we also conduct benchmarks using full-sized copies (i.e., sample size equal to 1) of the initial graph. Note that in this case, the topologies for scale 2 are all equivalent (one graph is simply connected to the other graph across all topologies).

Based on the data presented in Figure 9, we make the following observations:

(1) The scale-up parameters affect the performance of both datasets in the same manner: the same performance trend can be observed (per scale-up) between the two datasets.
(2) The processing time for all scale-ups is affected by the diameter and average shortest path length significantly, and follows the same pattern as shown in Figure 8. For example, the chain topology increases the processing time of BFS the most in comparison to a star topology (followed by the ring, star, and fully connected topology) as the diameter and average shortest path lengths are the longest. In contrast, the

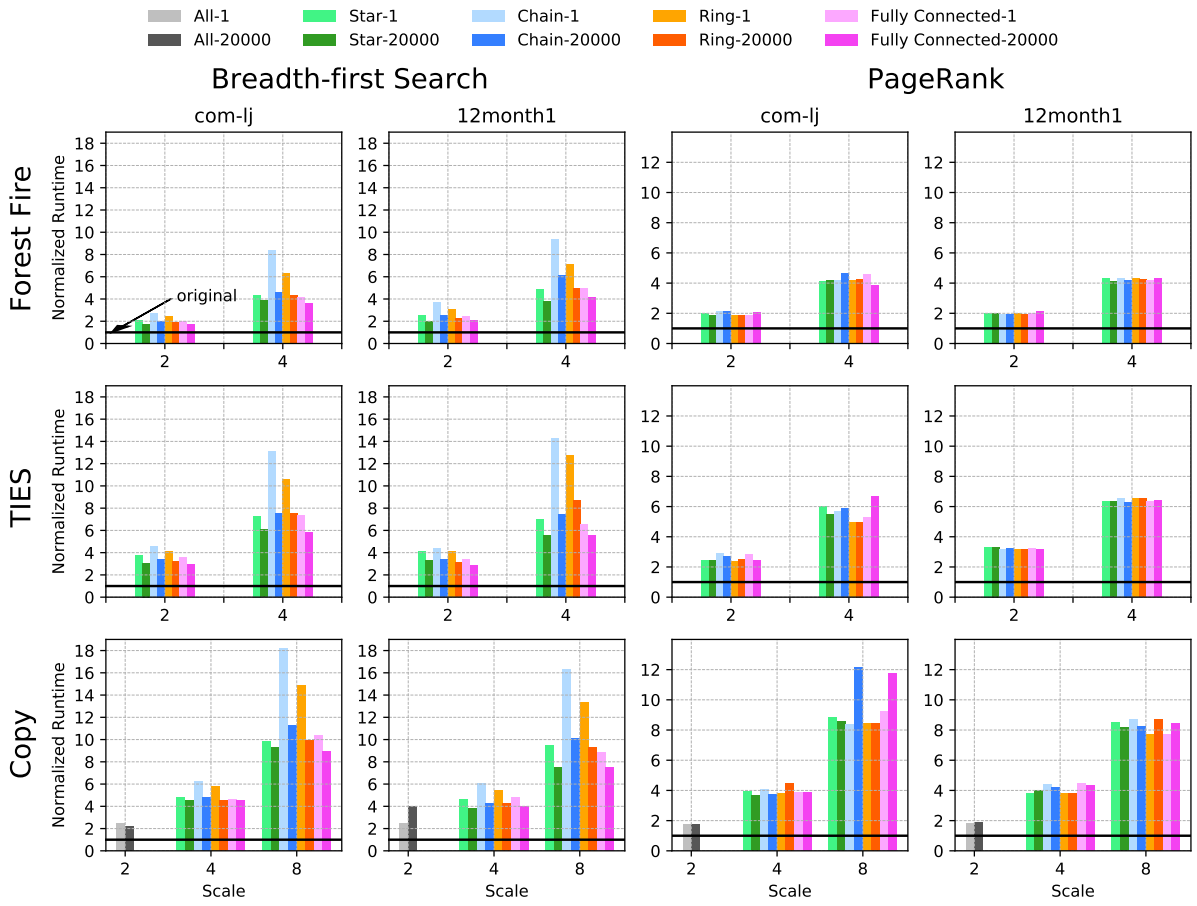

Figure 9: Normalized processing time of Breadth-First Search (BFS) and PageRank (PR) on scaled-up *com-livejournal* and *12month* datasets, using Forest Fire and TIES as sampling algorithm with different topologies and 1 - 20,000 interconnections. All scale-ups use a sample size of 0.5 and *random* interconnections. The last row shows the effects on performance when using full copies (sample size = 1.0) of the graph rather than a sampling algorithm. Note that the scale for Forest Fire is based on the number of edges, while for TIES it is based on the number of nodes. For full copies, this is both the number of vertices and edges.

fully connected topology has, in all but one case, the lowest processing time in comparison to the other topologies of scale-ups using equivalent parameters, as these structural properties are the smallest. The only exception is during the x4 scale-up when using TIES as sampling algorithm with 20,000 interconnections.

(3) Adding more interconnections (thus reducing the diameter and average path length) reduces the processing time of BFS. In contrast to PR, adding more interconnections does not reduce the processing time necessarily.

(4) For all scale-ups using TIES, the BFS processing time is higher than when using Forest Fire (likely due to oversampling). Furthermore, most of our results are robust and show no unexpected behavior in terms of performance, despite the chance of unexpected results when running BFS due to the small deviance in the number of components per scale-up. Unless the user has knowledge of the largest sub-graph and which vertex belongs in there (along with other scale-ups), we recommend using Forest Fire as sampling algorithm when running BFS, as these concerns can be discarded.

(5) For BFS and PR, the fully connected topology with Forest Fire as sampling algorithm and a single interconnection have roughly linear scalability.

(6) Using full-sized copies of the graph (rather than samples) for scale-ups shows that for BFS, the chain topology with a single interconnection has the largest processing time. This number is larger than when using sampling algorithms. The fully connected topology with a high number of interconnections had the lowest processing time. For PR, the chain and fully connected topologies with many interconnections have the largest processing time. In contrast, the star topology with a higher number of interconnections has a significantly lower processing time.

In summary, our case-studies demonstrate (1) a useful degree of control on the properties of scaled graphs, and (2) how different

Property	Observation
Nodes	Directly controllable through the scaling factor.
Edges	Roughly controllable through the scaling factor (in case of TIES additional edges are added due to the total induction step; for an edge-based sampling method this can be directly controlled).
Avg. degree	Controllable through the sample size, amount of interconnections and topology. Degree distribution is controllable through the scaling factor and amount of interconnections.
Diameter	Roughly controllable through the scaling factor, topology and interconnections. Note that the diameter can get lower than the original graph when many interconnections are added.
Graph density	Controllable through interconnections.
Connectivity	Not controllable since it is not being preserved by the sampling algorithm.
Avg. Clustering Coefficient	Downwards controllable by adding multi-interconnections.
Avg. Shortest Path Length	Roughly controllable through the scaling factor, topology, interconnections and bridge-vertices selection. Note that the average path length can get lower than the number of the original graph when many interconnections are added.

Table 8: Summary of the controllability per property.

properties-performance correlation hypotheses can be verified using our graph families. We are confident these case-studies provide a first step towards an empirical approach (based on systematic sensitivity analysis) to build performance models parameterized by graph properties.

8 CONCLUSION

Graphs and graph processing belong to the very active field of data science. With new analysis methods, datasets, and algorithms emerging frequently, the community faces a benchmarking crisis: the lack of representative datasets hinders in-depth performance and scalability analysis for these new methods.

In this work, we have provided a pragmatic solution to this challenge: a heuristic, sampling-based method for scaling existing graphs. Our method starts with a seed graph and generates a family of similar graphs, with properties controlled by user requirements. We have implemented our method in an easy to use, flexible open-source tool that addresses the needs of graph processing system designers and developers. We have validated our tool on a set of nine different graphs. Our in-depth empirical analysis shows that generating diverse graphs, with predictable and controllable properties is feasible. Moreover, our case-studies demonstrated that these diverse graph families - only provided by our tool - can be used for uncovering the behavior of graph processing platforms. Our future work focuses on further improving user-interaction, by enabling more control for experience users, and more automation for novice users.

REFERENCES

[1] Nesreen Ahmed, Jennifer Neville, and Ramana Rao Kompella. 2011. Network sampling via edge-based node selection with graph induction. (2011).

[2] Nesreen K Ahmed, Jennifer Neville, and Ramana Kompella. 2012. Network sampling designs for relational classification. In Sixth International AAAI Conference on Weblogs and Social Media.

[3] Mohammad Al Hasan. 2016. Methods and applications of network sampling. In Optimization Challenges in Complex, Networked and Risky Systems. INFORMS, 115–139.

[4] Neli Blagus, Lovro Šubelj, and Marko Bajec. 2014. Assessing the effectiveness of real-world network simplification. Physica A: Statistical Mechanics and its Applications 413 (2014), 134–146.

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

[6] The Graph 500 Steering Committee. 2010–2016. The Graph 500 List. http://www.graph500.org.

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

[8] Graphalytics.org. 2019. Graphalytics Global Competition. https://graphalytics.org/competition.

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

[10] Alexander Gutfraind, Ilya Safro, and Lauren Ancel Meyers. 2015. Multiscale network generation. In 2015 18th International Conference on Information Fusion (Fusion). IEEE, 158–165.

[11] Pili Hu and Wing Cheong Lau. 2013. A survey and taxonomy of graph sampling. arXiv preprint arXiv:1308.5865 (2013).

[12] Alexandru Iosup, Tim Hegeman, Wing Lung Ngai, Stijn Heldens, Arnau Prat-Pérez, Thomas Manhardto, Hassan Chafio, Mihai Capotă, Narayanan Sundaram, Michael Anderson, et al. 2016. LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. Proceedings of the VLDB Endowment 9, 13 (2016), 1317–1328.

[13] Farzad Khorasani, Rajiv Gupta, and Laxmi N Bhuyan. 2015. Scalable simd-efficient graph processing on gpus. In 2015 International Conference on Parallel Architecture and Compilation (PACT). IEEE, 39–50.

[14] Jérôme Kunegis. 2013. KONECT: The Koblenz Network Collection. In Proceedings of the 22nd International Conference on World Wide Web. ACM, 1343–1350.

[15] Sang Hoon Lee, Pan-Jun Kim, and Hawoong Jeong. 2006. Statistical properties of sampled networks. Physical review E 73, 1 (2006), 016102.

[16] Jure Leskovec. 2006. Stanford Network Analysis Platform (SNAP). Stanford University (2006).

[17] Jure Leskovec. 2008. Kronecker Graphs (presentation). (2008).

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

[19] Jure Leskovec and Christos Faloutsos. 2006. Sampling from large graphs. In Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining. ACM, 631–636.

[20] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[21] Joshua Lothian, Sarah Powers, Blair D Sullivan, Matthew Baker, Jonathan Schrock, and Stephen W Poole. 2013. Synthetic graph generation for data-intensive HPC benchmarking: Background and framework. Oak Ridge National Laboratory, Tech. Rep. ORNL/TM-2013/339 (2013).

[22] Ahmed Musaafir. 2019. Graph scaling tool (repository). https://github.com/amusaafir/graph-scaling.
[23] Ahmed Musaafir and Ana Varbanescu. 2018. Graph scaling models. https://github.com/amusaafir/graph-scaling/blob/master/scaling-guidelines.pdf.
[24] Ryan Rossi and Nesreen Ahmed. 2015. The network data repository with interactive graph analytics and visualization. In Twenty-Ninth AAAI Conference on Artificial Intelligence.
[25] Mirko Spasic, Milos Jovanovik, and Arnau Prat-Pérez. 2016. An RDF Dataset Generator for the Social Network Benchmark with Real-World Coherence.. In BLINK@ ISWC.
[26] Christian L Staudt, Michael Hamann, Ilya Safro, Alexander Gutfraind, and Henning Meyerhenke. 2016. Generating scaled replicas of real-world complex networks. In International Workshop on Complex Networks and their Applications. Springer, 17–28.
[27] Narayanan Sundaram, Nadathur Satish, Md Mostofa Ali Patwary, Subramanya R Dulloor, Michael J Anderson, Satya Gautam Vadlamudi, Dipankar Das, and Pradeep Dubey. 2015. Graphmat: High performance graph analytics made productive. Proceedings of the VLDB Endowment 8, 11 (2015), 1214–1225.
[28] Alexandru Uta, Ana Lucia Varbanescu, Ahmed Musaafir, Chris Lemaire, and Alexandru Iosup. 2018. Exploring hpc and big data convergence: A graph processing study on intel knights landing. In 2018 IEEE International Conference on Cluster Computing (CLUSTER). IEEE, 66–77.
[29] Merijn Verstraaten, Ana Lucia Varbanescu, and Cees de Laat. 2018. Mix-and-Match: A Model-driven Runtime Optimisation Strategy for BFS on GPUs. In 2018 IEEE/ACM 8th Workshop on Irregular Applications: Architectures and Algorithms (IA3). IEEE, 53–60.
[30] Marija Vištica, Ani Grubišic, and Branko Žitko. 2016. Applying graph sampling methods on student model initialization in intelligent tutoring systems. The International Journal of Information and Learning Technology 33, 4 (2016), 202–218.
[31] JW Zhang and YC Tay. 2016. GSCALER: Synthetically Scaling A Given Graph.. In EDBT, Vol. 16. 53–64.