Towards Fast and Scalable Graph Pattern Mining

Anand Padmanabha Iyer[⋆][*] Zaoxing Liu[†][*] Xin Jin[†]
Shivaram Venkataraman[•] Vladimir Braverman[†] Ion Stoica[⋆]
[⋆]*University of California, Berkeley* [†]*Johns Hopkins University* [•]*Microsoft Research*

Abstract

While there has been a tremendous interest in processing graph-structured data, existing distributed graph processing systems take several minutes or even hours to mine simple patterns on graphs. In this paper, we try to answer the question of whether it is possible to build a graph pattern mining engine that is both *fast* and *scalable*. Leveraging the observation that in several pattern mining tasks, providing an approximate answer is good enough, we propose the use of approximation for graph pattern mining. However, we find that existing approximation techniques do not work for this purpose. Based on this, we present a new approach for approximate graph pattern mining that leverages recent advancements in graph approximation theory. Our preliminary evaluations show encouraging results: compared to state-of-the-art, finding 3-motifs in Twitter graph is 165× faster while incurring only 5% error. We conclude by discussing several systems challenges to make our proposal practical.

1 Introduction

The past few years has seen a resurgence in enterprises storing and processing massive amounts of graph-structured data [1, 2]. Algorithms for graph processing can broadly be classified into two categories. The first, *graph analysis* algorithms, consists of those which compute properties of a graph, typically using neighborhood information. Examples of such algorithms include page rank [41], community detection [26] and label propagation [57]. The second, *graph pattern mining* algorithms, focuses on discovering structural patterns in a graph. Examples of this include motif finding [38], frequent sub-graph mining (FSM) [55] and clique mining [16]. Both categories have been thoroughly explored in academic literature, with researchers proposing several algorithms.

Today, a deluge of graph processing frameworks exist, developed both in academia and open-source [17, 20, 21, 29–31, 35–37, 39, 45–47, 53]. These frameworks typically provide high-level abstractions that make it easy for developers to implement many graph algorithms. While both categories of graph algorithms are equally important, a vast majority of the existing graph processing frameworks have focused on graph analysis algorithms. These frameworks are fast and can scale out to accommodate really large graphs: for instance, GraM [54] can run one iteration of page rank on a trillion-edge graph in 140 seconds in a cluster. In contrast, graph pattern mining systems fail to scale to even moderately sized graphs, and are slow, taking several hours to mine simple patterns [25, 50].

The main culprit that hinders the scalability of pattern mining is the complexity of these algorithms—mining patterns requires complex computations and storing exponentially large intermediate candidate sets. For example, a graph with a million vertices may possibly contain 10^{17} triangles. While distributed graph-processing solutions are good candidates for processing such massive intermediate data, the need to do expensive joins to create candidates severely degrades performance. As a result, state-of-the-art systems like Arabesque [50] propose new abstractions for storing candidates in a distributed setting. However, even with optimized methods to store candidates, Arabesque takes over 10 hours to *count* motifs in a graph with less than 1 billion edges.

In this paper, we ask the question *"Is it possible to build a graph mining system that is both fast and scalable?"* A key observation that we leverage in answering this question is: *in many pattern mining tasks, it is often not necessary to output the exact answer*. For instance, in FSM the task is to find the *frequency* of subgraphs with an end-goal of ordering them by occurrences. Similarly, motif counting determines the number of occurrences of a given motif. In these scenarios, it is sufficient to provide an *approximately* correct answer. Indeed, our conversations with a social networking firm revealed that one of their most time-consuming production jobs is counting graphlets [44] to determine social graph similarity. Another company's core business depends on classifying fraudulent patterns in graphs and this is done by counting the frequency of pattern occurrences. In both cases, an approximate count is good enough. Further more, it is not necessary to materialize *all* possible occurrences of a pattern[1]. Thus, we propose using *approximate* methods to build a fast and scalable graph mining system.

[*]Equal contribution.

[1]In large graphs, it may even be infeasible to output all embeddings.

Approximate analytics is an area that has garnered attention for big data analytics [5, 12, 27], where the goal is to let the user trade-off accuracy for much faster results. The basic idea in approximation systems is to execute the *exact* algorithm on small portions of the data, referred to as *samples*, and then rely on the statistical properties of these samples to compose partial results and/or error characteristics. The fundamental assumption underlying these systems is that there exists a relationship between the input size and the accuracy of the results. However, this assumption falls apart when applied to graph pattern mining. In particular, running the exact algorithm on a sampled graph may not result in reducing the runtime or provide a good estimation of the result (§2.2).

In this paper, we propose leveraging graph approximation theory to build approximate pattern mining systems. The key idea we exploit is that approximate pattern mining can be viewed as equivalent to *probabilistically sampling random instances of the pattern*. This observation lets us run sampling methods with very high parallelism and provides drastic reduction in run-time while sacrificing a small amount of accuracy. For example, our preliminary evaluation shows that our approach is 165× faster compared to the state-of-the-art for mining 3-motifs while incurring only 5% error.

There are a number of systems challenges in realizing a practical approximate pattern mining system. While we use the theory as a foundation, we need to extend the state-of-the-art approximation techniques not only to general patterns, but also to distributed settings. Further, an important problem in any approximation system is in allowing users to navigate the tradeoff between the result accuracy and latency. While existing approximate processing systems have proposed a number of approaches for this task, we find that they do not fit our needs. In the rest of this paper, we discuss our approach, and our initial directions in tackling each of these challenges.

2 Background & Motivation

We begin by motivating the need for a new approach to approximate pattern mining.

2.1 Graph Pattern Mining

Mining patterns in a graph represents an important class of graph processing problems. The objective here is to find instances of a given pattern in a graph where a *pattern* is any arbitrary subgraph. Thus, pattern mining algorithms aim to output all subgraphs, commonly referred to as *embeddings*, that match the input pattern. Matching is done via sub-graph isomorphism, which is well known to be NP-complete. Several varieties of graph pattern mining problems exist, ranging from finding cliques to mining frequent subgraphs. We refer the reader to [6, 50] for an excellent, in-depth overview of graph mining algorithms.

A common approach to implement pattern mining algorithms is to iterate over all possible embeddings in the graph starting with the simplest pattern (e.g., a vertex or an edge). The system checks all such *candidate* embeddings, and prunes those which cannot be part of the final answer. The resulting candidates are then expanded by adding one vertex or edge, and the process repeated until it is not possible to expand the exploration further. The obvious challenge in graph pattern mining, as opposed to graph analysis, is the exponentially large candidate set that need to be checked.

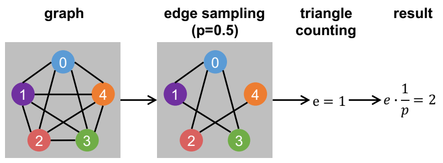
Distributed graph processing frameworks are built to support massive amounts of data, and thus may seem like an ideal candidate for this situation. Unfortunately when applied to graph mining problems, they face several challenges. Arabesque [50], a recently proposed distributed graph mining system, discusses these challenges in detail, and proposes solutions to tackle several of them. However, even Arabesque is unable to scale to large graphs due to the need to materialize candidates and exchange them between machines. As an example, Arabesque takes over 10 hours to count motifs of size 3 in a graph with less than a billion edges in a cluster of 20 machines, each having 32 cores and 256GB of memory.

2.2 Approximate Processing on Graphs

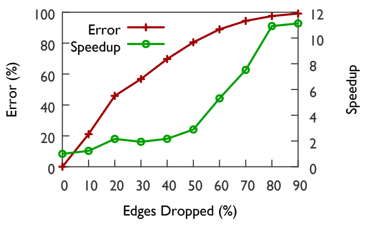
Approximate processing is an approach that has been used with tremendous success in solving similar problems in both big data analytics [5, 27] and databases [19, 22, 23]. Thus it is natural to explore similar techniques for pattern mining in graphs given our earlier description of enterprise use cases. However, simply extending existing approaches to graphs is insufficient.

The common underlying idea in approximate processing systems is to *sample the input* that a query or an algorithm works on. Several techniques for sampling the input exists, for instance, BlinkDB [5] leverages stratified sampling. To estimate the error, approximation systems rely on the assumption that the sample size relates to the error in the output (e.g., if we sample K items from the original input, then the error in aggregate queries, such as `SUM`, is inversely proportional to $\sqrt{K}$). It is straightforward to envision extending this approach to graph pattern mining—given a graph and a pattern to mine in the graph, we first sample the graph, and run the pattern mining algorithm on the sampled graph.

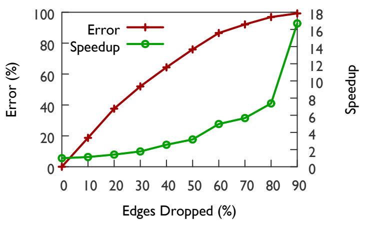
Figure 1a depicts the idea as applied to triangle counting. In this example, the input graph contains 10 triangles. Using uniform sampling on the edges we obtain a graph with 50% of the edges. Applying triangle counting on this sample yieldings an answer of 1. There are a number of approaches to scale this number to the actual graph. One naive approach is to double it, since we reduced the input by half. To verify the feasibility of the approach, we eval-



(a) Uniform edge sampling.



(b) 3-chains in Twitter graph



(c) Triangles in UK graph

Figure 1: Simply extending approximate processing techniques to graph pattern mining does not work.

uated it on the Twitter graph [34] for finding 3-chains and the UK webgraph [14] graph for triangle counting. The relation between the sample size, error and the speedup compared to running on the original graph ($\frac{T_{orig}}{T_{sample}}$) is shown in figs. 1b and 1c respectively.

These results show fundamental limitations of the approach. We see that there is no relation between the size of the graph (sample) and the error or the speedup achieved. Even very small samples do not provide noticeable speedups, and conversely, even very large samples end up with significant errors. Thus, we conclude that the existing approximation approach of *running the exact algorithm on one or more samples of the input is incompatible with graph pattern mining*.

3 Our Approach

3.1 Approximate Pattern Mining

The key idea we leverage is to *sample instances of a given pattern* from the graph, and based on the sampling probability and how many instances we find, we estimate the total number of instances of that pattern in the graph. Since only sampling once would yield large *variance* on the results, we independently sample multiple times and take the average to reduce the variance. We call each sampling process an *estimator*. By using r estimators and making r sufficient large, we are able to get accurate results with bounded errors. Since an estimator takes computation and memory resource to sample a pattern, picking the number of estimators r provides a trade-off between result accuracy and resource consumption.

While the intuition of using such sampling to approximate pattern counts is straightforward, the approximation bound analysis is quite subtle. In fact there is a large body of theoretical work on various algorithms to sample patterns and analysis to prove their bounds [7, 10, 18, 32, 42, 43, 51]. Consider triangle counting as an example. Naively, one would design an estimator that uniformly samples three edges from the graph without replacement. Since the probability of sampling one edge is $1/m$ in a graph of m edges, the probability of sampling three edges is $1/m^3$. If the three sampled edges form a triangle, the estimator outputs triangle count to be m^3; otherwise, it outputs 0. While such an estimator is

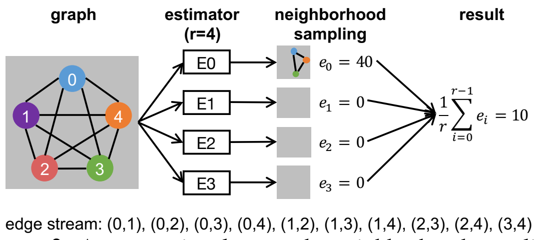

graph
estimator (r=4)
neighborhood sampling
result
E0
E1
E2
E3
$e_0 = 40$
$e_1 = 0$
$e_2 = 0$
$e_3 = 0$
$\frac{1}{r}\sum_{i=0}^{r-1} e_i = 10$
edge stream: (0,1), (0,2), (0,3), (0,4), (1,2), (1,3), (1,4), (2,3), (2,4), (3,4)

Figure 2: Approx. triangle count by neighborhood sampling

unbiased, since m is large, the probability that the estimator would find a triangle is very low and the variance of the result is very large. To reduce the variance, we would require a large number of estimators, and this increases time and memory consumption.

We use *neighborhood sampling* [43] in our framework, which is an efficient technique to sample graph patterns. Intuitively, compared to the naive sampling approach, neighborhood sampling increases the probability that an estimator would actually find an instance of the given pattern, and thus requires fewer estimators to achieve the same accuracy. Neighborhood sampling models the graph as a stream of edges and starting from a random edge, we gradually add more edges until the edges form the pattern or it is impossible to form the pattern. Let E denote the event that a pattern is formed, $E_1, E_2, \ldots, E_k$ are the events that edges $m_1, m_2, \ldots, m_k$ are sampled and stored. Thus the probability that a pattern is actually sampled can be calculated as $Pr(E) = Pr(E_1 \cap E_2 \cdots \cap E_k) = Pr(E_1) \times Pr(E_2|E_1) \cdots \times Pr(E_k|E_1, \ldots, E_{k-1})$.

We leverage neighborhood sampling to build several estimators to sample patterns. For triangle counting, if an estimator successfully samples a triangle, converting probability to expectation, $e_i = mc$ will be the estimate of triangles in the graph, where m is the number of edges in the graph and c is the number of neighbors of the first sampled edge that appearing after it. For a total of r estimators, we will output $\frac{1}{r}\sum_r e_i$ as the approximate value. Figure 2 presents an example graph with five nodes. Estimator E_0 finds the triangle formed by edges $(0,3)$, $(0,4)$ and $(3,4)$. The probability of finding $(0,3)$ is $1/m = 1/10$. Since $(0,3)$ has four adjacent edges that appear after it in the order (i.e., $(0,4)$, $(1,3)$, $(2,3)$, and $(3,4)$), the probability that finds $(0,4)$ is $1/c = 1/4$. Therefore, the probability of finding this specific triangle $(0,3,4)$

3-Motif Count	System	|V|	|E|	Runtime
Ours (5% error)	16 x 8	4.8M	68.9M	11.5s
Arabesque [50]	16 x 8	4.8M	68.9M	299.2s
Ours (5% error)	16 x 8	41.7M	1.47B	4m
Arabesque [50]	20x32	**180M**	**0.9B**	10h45m

Table 1: Using approximation, we are able to not only reduce run time, but also process *larger* graphs on *smaller* clusters.

is $1/(mc) = 1/40$, and thus E_0 estimates the number of triangles to be 40, which is a biased result. With more independent estimators E_1, E_2, and E_3, the estimated count becomes more accurate as the final result takes the average of the four estimators.

3.2 Evaluation of Potential

One of the first questions that we need to answer before exploring the practicality and challenges in our proposal is to understand how much benefit we can obtain by leveraging approximation. To do so, we implemented a simple pattern mining task, counting 3-motifs, using the approximation technique described earlier in Apache Spark [56]. We chose two datasets: LiveJournal (68.9M edges) [3] and Twitter (1.47B edges) [34], and use 16 machines on Amazon EC2 (8 cores each) to run an experiment which tries to find the count of 3-motifs, and compare against Arabesque [50]. We set the number of estimators to achieve an error rate of 5%. Table 1 shows the results.

We were unable to get Arabesque to handle the Twitter graph in our cluster, so we use the numbers in [50] for the larger graph. We see that our approach significantly outperforms Arabesque, and that the performance gap increases in the larger graph. Our approach is able to achieve *more than 2 orders of magnitude (*165×*)* reduction in computation time while using less resources and incurring only a small (5%) loss in accuracy.

4 Challenges

Several challenges lie ahead of us before we can achieve our goal of a general purpose approximate graph mining system. We describe some of them in this section.

4.1 Challenge 1: General Patterns

Neighborhood sampling was proposed in the context of triangle counting, so we need to extend it to handle general patterns. We plan to explore this using one simple observation: the sampling process in each estimator can be seen as comprising of *two phases*. In the first, *sampling* phase, edges are sampled either randomly, or using adjacency information of already sampled edges. The phase ends when the sampled edges have fixed the vertices for a given pattern. Then we wait for the edges that complete the pattern, hence the process enters *closing* phase.

The amount of time an estimator process spends in each of these phases, and the number of edges sampled in them depend on the pattern. In triangle counting, there is only one way to form the triangle: the sampling phase finds two adjacent edges, and the closing phase awaits the third edge to form a triangle with the two sampled edges. For a general graph pattern with multiple nodes, there can be multiple ways to form the pattern. One approach to generalize the sampling is to restrict the implementation of mining tasks using the two phases (e.g., using a simple API). Then, the challenge is to compute the probability of finding the pattern automatically given a mining task written using this restricted model.

4.2 Challenge 2: Distributed Settings

Neighborhood sampling viewed as comprising of two phases is massively parallel, since the sampling and closing phases remains the same for each estimator and can be captured using a simple data-parallel *map* phase, and the results can be aggregated using a *reduce* phase. Unfortunately, we cannot simply scale up this process horizontally by locally running the process in each machine and aggregating results, since the probability analysis assumes that each estimator sees the entire graph. Partitioning the graph into multiple machines results in missing patterns that span partitions, and significantly underestimates results, the magnitude of which depends on the partitioning strategy. One possible solution for this problem is to account for the error due to this underestimation by scaling the result by a factor $f(w)$, which is related to the number of partitions w. For this to work, we must not only precisely compute $f(w)$, but also do it for any pattern.

4.3 Challenge 3: Error-Latency Profile

A key feature in any approximate processing system is the ability for users to trade-off accuracy for result latency. To allow users to navigate this trade-off, our solution needs to understand the relation between latency and error. In our approach, the only configurable parameter is the number of estimators used for mining. Setting a specific number of estimators, N_e, results in a fixed runtime and an error within a bound. Thus, by varying the number of estimators, we can vary the accuracy achieved and the computation time accordingly. To enable picking the right number of estimators, we would need two profiles, estimators vs. latency and estimators vs. error.

4.3.1 Estimators vs. Latency

The time complexity of our approximation algorithm is linear in terms of the number of edges in the graph and the number of estimators. Given a graph and a particular pattern, the computation time is dominated by the number of estimators when the number of estimators is large enough. As an example, Figure 3 shows the relationship between the computation time and the number of estimators for triangle counting in the Twitter graph [34]. We can see

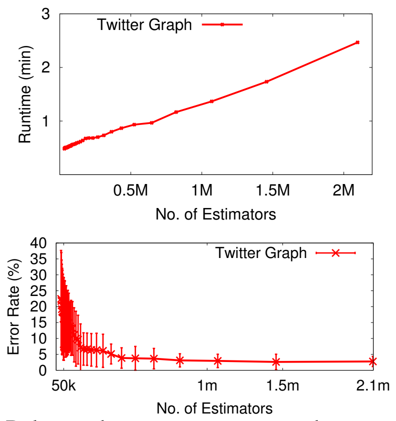


Figure 3: Relations between estimators and run-time/error rate.

that the curve is close to linear when the number of estimators is larger than 0.5M. When the number of estimators is small, the computation time is also affected by other items and thus the curve is not strictly linear. However, for these regions, it is not computationally expensive to profile more data points, and as these regions have high error, they are less likely to be of interest to users. Thus, we plan to study classical regression-based techniques to build this profile and using the profile can aid in picking the number of estimators to use within a profiling cost.

4.3.2 Estimators vs. Error

As seen from Figure 3, the estimator vs. error profile is non-linear. Building this profile is challenging due to several reasons. Exhaustive profiling is out of the question due to its prohibitive time requirements. Further, the actual errors vary within some range for the same number of estimators due to the randomness in our approach. This makes theoretical closed-bound solutions difficult. Finally, to estimate the error, we need to know the ground-truth. However, computing the ground-truth undermines the usefulness of approximate processing.

We plan to investigate a number of ways to build this curve in an efficient manner. This includes using a piecewise modeling of the curve and leveraging experiment design [52] or bayesian optimization [11] to fit the model. Further, we plan to explore well known statistical techniques like bootstrap [33]. Finally, we are also planning to look at probabilistic techniques that can use a small portion of the graph to build this profile *without* the need to know the ground-truth.

4.4 Challenge 4: Handling Updates

While existing graph processing systems assume graphs to be static, real-world graphs are dynamic. Some previous works have looked at supporting incremental computations on evolving graphs [31, 39, 40], but they do not extend to pattern mining. The challenge here is to incorporate incremental profile building techniques to support graph evolution, i.e., can we avoid rebuilding the profiles? For instance, it may be possible to use stale profile without much degradation if we can predict when the profile is not good enough to constitute a rebuilding. Further, it may be possible to cache estimator states and reuse them later during rebuilding. Finally, an interesting direction to look at is the possibility of precomputing some base patterns that could be building blocks for other patterns.

5 Related Work

A number of systems have been proposed in the literature for **graph processing** [17, 29, 30, 35, 36, 45–47, 53] and **graph mining** [4, 48, 50]. Processing systems typically only focus on *graph analysis*, and do not support efficient pattern mining. Mining systems on the other hand require significant time to process even moderately sized graphs. [49] discusses an approximate motif counting algorithms in HPC clusters. However, its focus is on optimizing MPI communication techniques for one specific algorithm, and hence does not extend to general graph patterns.

Approximate analytics systems [5, 12, 27] have recently gained popularity due to the time required to process large datasets. These systems reduce the amount of data that is used in the analysis in the hope that this reduces processing time. However, as we show in this work, this technique does not extend to graph pattern mining.

Theory community has invested significant effort in **approximate graph algorithms** for several graph analysis tasks [8, 9, 13, 15, 24, 28]. None of these are aimed at distributed processing, nor do they propose ways to understand the performance profile of the algorithms when deployed in the real-world. We leverage this rich theoretical foundation by proposing the use of these techniques to mine general patterns in distributed settings.

6 Conclusion

In this paper, we proposed our approach towards building a distributed graph pattern mining system that is both *fast* and *scalable* to large graphs. Our proposal leverages approximation to achieve this goal, by building on advancements in graph approximation theory. We discussed several challenges in realizing our proposal, which we are actively pursuing. Our preliminary evaluations show promise in our proposed techniques.

Acknowledgments We thank the reviewers for their valuable feedback. In addition to NSF CISE Expeditions Award CCF-1730628, this research is supported in part by DHS Award HSHQDC-16-3-00083, and gifts from Alibaba, Amazon Web Services, Ant Financial, CapitalOne, Ericsson, Facebook, Google, Huawei, Intel, Microsoft, Scotiabank, Splunk and VMware. Liu, Braverman, and Jin are supported in part by NSF grants No. 1447639, 1650041, 1652257, and CRII-NeTS-1755646, Cisco faculty award, ONR Award N00014-18-1-2364, and a Facebook Communications & Networking Research Award.

References

[1] Enterprise DBMS, Q1 2014. https://www.forrester.com/report/TechRadar+Enterprise+DBMS+Q1+2014/-/E-RES106801.

[2] Graph DBMS increased their popularity by 500% within the last 2 years. http://db-engines.com/en/blog_post//43.

[3] Stanford Large Network Dataset Collection. https://snap.stanford.edu/.

[4] Aboulnaga, A., Xiang, J., and Guo, C. Scalable maximum clique computation using mapreduce. In *Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013)* (Washington, DC, USA, 2013), ICDE '13, IEEE Computer Society, pp. 74–85.

[5] Agarwal, S., Mozafari, B., Panda, A., Milner, H., Madden, S., and Stoica, I. Blinkdb: Queries with bounded errors and bounded response times on very large data. In *Proceedings of the 8th ACM European Conference on Computer Systems* (New York, NY, USA, 2013), EuroSys '13, ACM, pp. 29–42.

[6] Aggarwal, C. C., and Wang, H., Eds. *Managing and Mining Graph Data*, vol. 40 of *Advances in Database Systems*. Springer, 2010.

[7] Ahmed, N. K., Duffield, N., Neville, J., and Kompella, R. Graph sample and hold: A framework for big-graph analytics. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2014), KDD '14, ACM, pp. 1446–1455.

[8] Ahn, K. J., Guha, S., and McGregor, A. Analyzing graph structure via linear measurements. In *Proceedings of the Twenty-third Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2012), SODA '12, Society for Industrial and Applied Mathematics, pp. 459–467.

[9] Ahn, K. J., Guha, S., and McGregor, A. Graph sketches: Sparsification, spanners, and subgraphs. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (New York, NY, USA, 2012), PODS '12, ACM, pp. 5–14.

[10] Al Hasan, M., and Zaki, M. J. Output space sampling for graph patterns. *Proc. VLDB Endow. 2*, 1 (Aug. 2009), 730–741.

[11] Alipourfard, O., Liu, H. H., Chen, J., Venkataraman, S., Yu, M., and Zhang, M. Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)* (Boston, MA, 2017), USENIX Association, pp. 469–482.

[12] Ananthanarayanan, G., Hung, M. C.-C., Ren, X., Stoica, I., Wierman, A., and Yu, M. Grass: Trimming stragglers in approximation analytics. In *NSDI* (2014), pp. 289–302.

[13] Assadi, S., Khanna, S., and Li, Y. On estimating maximum matching size in graph streams. In *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms* (Philadelphia, PA, USA, 2017), SODA '17, Society for Industrial and Applied Mathematics, pp. 1723–1742.

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

[15] Braverman, V., Ostrovsky, R., and Vilenchik, D. *How Hard Is Counting Triangles in the Streaming Model?* Springer Berlin Heidelberg, Berlin, Heidelberg, 2013, pp. 244–254.

[16] Bron, C., and Kerbosch, J. Algorithm 457: finding all cliques of an undirected graph. *Communications of the ACM 16*, 9 (1973), 575–577.

[17] Buluç, A., and Gilbert, J. R. The combinatorial BLAS: design, implementation, and applications. *IJHPCA 25*, 4 (2011), 496–509.

[18] Buriol, L. S., Frahling, G., Leonardi, S., Marchetti-Spaccamela, A., and Sohler, C. Counting triangles in data streams. In *Proceedings of the Twenty-fifth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2006), PODS '06, ACM, pp. 253–262.

[19] Chaudhuri, S., Das, G., and Narasayya, V. Optimized stratified sampling for approximate query processing. *ACM Trans. Database Syst. 32*, 2 (June 2007).

[20] Cheng, R., Hong, J., Kyrola, A., Miao, Y., Weng, X., Wu, M., Yang, F., Zhou, L., Zhao, F., and Chen, E. Kineograph: Taking the pulse of a fast-changing and connected world. In *Proceedings of the 7th ACM European Conference on Computer Systems* (New York, NY, USA, 2012), EuroSys '12, ACM, pp. 85–98.

[21] Ching, A., Edunov, S., Kabiljo, M., Logothetis, D., and Muthukrishnan, S. One trillion edges: Graph processing at facebook-scale. *Proc. VLDB Endow. 8*, 12 (Aug. 2015), 1804–1815.

[22] Condie, T., Conway, N., Alvaro, P., Hellerstein, J. M., Gerth, J., Talbot, J., Elmeleegy, K., and Sears, R. Online aggregation and continuous query support in mapreduce. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2010), SIGMOD '10, ACM, pp. 1115–1118.

[23] Cormode, G., Garofalakis, M. N., Haas, P. J., and Jermaine, C. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends in Databases 4*, 1-3 (2012), 1–294.

[24] Das Sarma, A., Gollapudi, S., and Panigrahy, R. Estimating pagerank on graph streams. In *Proceedings of the Twenty-seventh ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (New York, NY, USA, 2008), PODS '08, ACM, pp. 69–78.

[25] Elseidy, M., Abdelhamid, E., Skiadopoulos, S., and Kalnis, P. Grami: Frequent subgraph and pattern mining in a single large graph. *Proc. VLDB Endow. 7*, 7 (Mar. 2014), 517–528.

[26] Fortunato, S. Community detection in graphs. *Physics reports 486*, 3 (2010), 75–174.

[27] Goiri, I., Bianchini, R., Nagarakatte, S., and Nguyen, T. D. Approxhadoop: Bringing approximations to mapreduce frameworks. In *Proceedings of the Twentieth International Conference on Architectural Support for Programming Languages and Operating Systems* (New York, NY, USA, 2015), ASPLOS '15, ACM, pp. 383–397.

[28] Gong, N. Z., Xu, W., Huang, L., Mittal, P., Stefanov, E., Sekar, V., and Song, D. Evolution of social-attribute networks: Measurements, modeling, and implications using google+. In *Proceedings of the 2012 Internet Measurement Conference* (New York, NY, USA, 2012), IMC '12, ACM, pp. 131–144.

[29] Gonzalez, J., Xin, R., Dave, A., Crankshaw, D., and Franklin, Stoica, I. Graphx: Graph processing in a distributed dataflow framework. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)* (Broomfield, CO, Oct. 2014), USENIX Association.

[30] Gonzalez, J. E., Low, Y., Gu, H., Bickson, D., and Guestrin, C. Powergraph: Distributed graph-parallel computation on natural graphs. In *Proceedings of the 10th USENIX Conference on Operating Systems Design and Implementation* (Berkeley, CA, USA, 2012), OSDI'12, USENIX Association, pp. 17–30.

[31] Han, W., Miao, Y., Li, K., Wu, M., Yang, F., Zhou, L., Prabhakaran, V., Chen, W., and Chen, E. Chronos: A graph engine for temporal graph analysis. In *Proceedings of the Ninth European Conference on Computer Systems* (New York, NY, USA, 2014), EuroSys '14, ACM, pp. 1:1–1:14.

[32] Jha, M., Seshadhri, C., and Pinar, A. A space-efficient streaming algorithm for estimating transitivity and triangle counts using the birthday paradox. *ACM Trans. Knowl. Discov. Data 9*, 3 (Feb. 2015), 15:1–15:21.

[33] Kleiner, A., Talwalkar, A., Sarkar, P., and Jordan, M. I. A scalable bootstrap for massive data. *Journal of the Royal Statistical Society: Series B (Statistical Methodology) 76*, 4 (2014), 795–816.

[34] Kwak, H., Lee, C., Park, H., and Moon, S. What is Twitter, a social network or a news media? In *WWW '10: Proceedings of the 19th international conference on World wide web* (New York, NY, USA, 2010), ACM, pp. 591–600.

[35] Kyrola, A., Blelloch, G., and Guestrin, C. Graphchi: Large-scale graph computation on just a pc. In *Presented as part of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI 12)* (Hollywood, CA, 2012), USENIX, pp. 31–46.

[36] Low, Y., Gonzalez, J., Kyrola, A., Bickson, D., Guestrin, C., and Hellerstein, J. M. Graphlab: A new framework for parallel machine learning. In *UAI* (2010), P. Grünwald and P. Spirtes, Eds., AUAI Press, pp. 340–349.

[37] Macko, P., Marathe, V. J., Margo, D. W., and Seltzer, M. I. Llama: Efficient graph analytics using large multiversioned arrays. In *2015 IEEE 31st International Conference on Data Engineering* (April 2015), pp. 363–374.

[38] Milo, R., Shen-Orr, S., Itzkovitz, S., Kashtan, N., Chklovskii, D., and Alon, U. Network motifs: simple building blocks of complex networks. *Science 298*, 5594 (2002), 824–827.

[39] Murray, D. G., McSherry, F., Isaacs, R., Isard, M., Barham, P., and Abadi, M. Naiad: A timely dataflow system. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 439–455.

[40] Padmanabha Iyer, A., Li, L. E., Das, T., and Stoica, I. Time-evolving graph processing at scale. In *Proceedings of the Fourth International Workshop on Graph Data Management Experiences and Systems* (2016), ACM, p. 5.

[41] Page, L., Brin, S., Motwani, R., and Winograd, T. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, November 1999. Previous number = SIDL-WP-1999-0120.

[42] Pagh, R., and Tsourakakis, C. E. Colorful triangle counting and a mapreduce implementation. *CoRR abs/1103.6073* (2011).

[43] Pavan, A., Tangwongsan, K., Tirthapura, S., and Wu, K.-L. Counting and sampling triangles from a graph stream. *Proc. VLDB Endow. 6*, 14 (Sept. 2013), 1870–1881.

[44] Pržulj, N., Corneil, D. G., and Jurisica, I. Modeling interactome: Scale-free or geometric? *Bioinformatics 20*, 18 (Dec. 2004), 3508–3515.

[45] Quamar, A., Deshpande, A., and Lin, J. Nscale: Neighborhood-centric large-scale graph analytics in the cloud. *The VLDB Journal 25*, 2 (Apr. 2016), 125–150.

[46] Roy, A., Bindschaedler, L., Malicevic, J., and Zwaenepoel, W. Chaos: Scale-out graph processing from secondary storage. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, ACM, pp. 410–424.

[47] Roy, A., Mihailovic, I., and Zwaenepoel, W. X-stream: Edge-centric graph processing using streaming partitions. In *Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles* (New York, NY, USA, 2013), SOSP '13, ACM, pp. 472–488.

[48] Shao, Y., Cui, B., Chen, L., Ma, L., Yao, J., and Xu, N. Parallel subgraph listing in a large-scale graph. In *Proceedings of the 2014 ACM SIGMOD International Conference on Management of Data* (New York, NY, USA, 2014), SIGMOD '14, ACM, pp. 625–636.

[49] Slota, G. M., and Madduri, K. Complex network analysis using parallel approximate motif counting. In *2014 IEEE 28th International Parallel and Distributed Processing Symposium* (May 2014), pp. 405–414.

[50] Teixeira, C. H. C., Fonseca, A. J., Serafini, M., Siganos, G., Zaki, M. J., and Aboulnaga, A. Arabesque: A system for distributed graph mining. In *Proceedings of the 25th Symposium on Operating Systems Principles* (New York, NY, USA, 2015), SOSP '15, ACM, pp. 425–440.

[51] Tsourakakis, C. E., Kang, U., Miller, G. L., and Faloutsos, C. Doulion: Counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, NY, USA, 2009), KDD '09, ACM, pp. 837–846.

[52] Venkataraman, S., Yang, Z., Franklin, M., Recht, B., and Stoica, I. Ernest: Efficient performance prediction for large-scale advanced analytics. In *13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16)* (Santa Clara, CA, 2016), USENIX Association, pp. 363–378.

[53] Wang, G., Xie, W., Demers, A. J., and Gehrke, J. Asynchronous large-scale graph processing made easy. In *CIDR* (2013).

[54] Wu, M., Yang, F., Xue, J., Xiao, W., Miao, Y., Wei, L., Lin, H., Dai, Y., and Zhou, L. Gram: Scaling graph computation to the trillions. In *Proceedings of the Sixth ACM Symposium on Cloud Computing* (New York, NY, USA, 2015), SoCC '15, ACM, pp. 408–421.

[55] Yan, X., and Han, J. gspan: Graph-based substructure pattern mining. In *Data Mining, 2002. ICDM 2003. Proceedings. 2002 IEEE International Conference on* (2002), IEEE, pp. 721–724.

[56] Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M. J., Shenker, S., and Stoica, I. Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation* (Berkeley, CA, USA, 2012), NSDI'12, USENIX Association, pp. 2–2.

[57] Zhu, X., and Ghahramani, Z. Learning from labeled and unlabeled data with label propagation.