Clique Counting in MapReduce: Algorithms and Experiments

IRENE FINOCCHI, MARCO FINOCCHI, and EMANUELE G. FUSCO,
Sapienza University of Rome

We tackle the problem of counting the number q_k of k-cliques in large-scale graphs, for any constant $k \geq 3$. Clique counting is essential in a variety of applications, including social network analysis. Our algorithms make it possible to compute q_k for several real-world graphs and shed light on its growth rate as a function of k. Even for small values of k, the number q_k of k-cliques can be in the order of tens or hundreds of trillions. As k increases, different graph instances show different behaviors: while on some graphs $q_{k+1} < q_k$, on other benchmarks $q_{k+1} \gg q_k$, up to two orders of magnitude in our observations. Graphs with steep clique growth rates represent particularly tough instances in practice.

Due to the computationally intensive nature of the clique counting problem, we settle for parallel solutions in the MapReduce framework, which has become in the last few years a de facto standard for batch processing of massive datasets. We give both theoretical and experimental contributions.

On the theory side, we design the first exact scalable algorithm for counting (and listing) k-cliques in MapReduce. Our algorithm uses $O(m^{3/2})$ total space and $O(m^{k/2})$ work, where m is the number of graph edges. This matches the best-known bounds for triangle listing when $k = 3$ and is work optimal in the worst case for any k, while keeping the communication cost independent of k. We also design sampling-based estimators that can dramatically reduce the running time and space requirements of the exact approach, while providing very accurate solutions with high probability.

We then assess the effectiveness of different clique counting approaches through an extensive experimental analysis over the Amazon EC2 platform, considering both our algorithms and their state-of-the-art competitors. The experimental results clearly highlight the algorithm of choice in different scenarios and prove our exact approach to be the most effective when the number of k-cliques is large, gracefully scaling to nontrivial values of k even on clusters of small/medium size. Our approximation algorithms achieve extremely accurate estimates and large speedups, especially on the toughest instances for the exact algorithms.

Categories and Subject Descriptors: G.2.2 [**Discrete Mathematics**]: Graph Algorithms; G.4 [**Mathematical Software**]: Algorithm Design and Analysis; F.2.2 [**Theory of Computation**]: Nonnumerical Algorithms and Problems

General Terms: Algorithms, Experimentation, Performance

Additional Key Words and Phrases: Clique listing, graph algorithms, MapReduce, parallel algorithms, experimental algorithmics

ACM Reference Format:
Irene Finocchi, Marco Finocchi, and Emanuele G. Fusco. 2015. Clique counting in MapReduce: Algorithms and experiments. ACM J. Exp. Algor. 20, 1, Article 1.7 (October 2015), 20 pages.
DOI: http://dx.doi.org/10.1145/2794080

This work is supported by Amazon Web Services through an AWS in Education Grant Award received by the first author.
Authors' addresses: I. Finocchi and M. Finocchi, Computer Science Department, Sapienza University of Rome, Via Salaria 113 - 00198, Rome, Italy; E. G. Fusco, Department of Computer, Control, and Management Engineering "Antonio Ruberti", Sapienza University of Rome, Via Ariosto 25 - 00185, Rome, Italy.

DOI: http://dx.doi.org/10.1145/2794080

1. INTRODUCTION

The problem of counting small subgraphs with specific structural properties in large-scale networks has gathered a lot of interest from the research community during the last few years. Counting—and possibly listing—all instances of triangles, cycles, cliques, and other different structures is indeed a fundamental tool for uncovering the properties of complex networks [Milo et al. 2002], with wide-ranging applications that include spam and anomaly detection [Becchetti et al. 2006; Gibson et al. 2005], social network analysis [Hanneman and Riddle 2005; Rajaraman and Ullman 2012], and the discovery of patterns in biological networks [Saha et al. 2010].

Since many interesting graphs have by themselves really large sizes, developing analysis algorithms that scale gracefully on such instances is rather challenging. Designing efficient sequential algorithms is often not enough: even assuming that the input graphs could fit into the memory of commodity hardware, subgraph counting is a computationally intensive problem, and the running times of sequential algorithms can easily become unacceptable in practice. To overcome these issues, many recent works have focused on speeding up the computation by exploiting parallelism (e.g., using MapReduce [Dean and Ghemawat 2004]), by working in external memory models [Vitter 2008], or by settling for approximate—instead of exact—answers (as done, e.g., in data streaming [Muthukrishnan 2005; Demetrescu and Finocchi 2007]).

In this article, we tackle the problem of counting the number of k-cliques in large-scale graphs, for any constant $k \geq 3$. This is a fundamental problem in social network analysis (see, e.g., Hanneman and Riddle [2005], Chapter 11), and algorithms that produce a census of all cliques are also included in widely used software packages, such as UCINET [Borgatti et al. 2002]. We present simple and scalable algorithms suitable to be implemented in the MapReduce framework [Dean and Ghemawat 2004]. MapReduce, together with its open-source implementation Hadoop [Apache Software Foundation 2014], has become a de facto standard for programming massively distributed systems both in industry and academia: it offers programmers the possibility to easily run their code on large clusters while neglecting any issues related to scheduling, synchronization, communication, and error detection (these are automatically handled by the system). Computational models for analyzing MapReduce algorithms are described in Karloff et al. [2010], Goodrich et al. [2011], and Pietracaprina et al. [2012].

Related work. k-clique counting is a natural generalization of triangle counting, where $k = 3$: this is the simplest, most nontrivial version of the problem and has a variety of applications. For instance, it is closely related to the computation of the clustering coefficient of a graph, which is in turn a widely used measure in the analysis of social networks [Kolountzakis et al. 2012]. The detection of large near-cliques, which is exploited by a variety of graph mining applications, also benefits from clique counting algorithms (see, e.g., Tsourakakis [2014]). Many exact and approximate algorithms tailored to triangles have been developed in the literature in different computational models. The best-known sequential counting algorithm is based on fast matrix multiplication [Alon et al. 1997] and has running time $O(m^{\frac{2\omega}{\omega+1}})$, where ω is the matrix multiplication exponent: this makes it infeasible even for medium-size graphs. Practical approaches match the $O(m^{3/2})$ bound first achieved in Itai and Rodeh [1978] and Chiba and Nishizeki [1985], which is optimal for the listing problem. Output-sensitive listing algorithms, which run asymptotically faster when the number of triangles is small, have been recently described in Björklund et al. [2014]. As shown in Ortmann and Brandes [2014], many listing algorithms hinge upon a common abstraction that also yields the state-of-the-art sequential implementations for the enumeration of triangles. The input/output complexity of the triangle listing problem is addressed in Pagh

and Silvestri [2014]. Approximate counting algorithms that operate in the data stream model, where the input graph is streamed as a list of edges and the algorithm must compute a solution using small space, are presented in Bar-Yossef et al. [2002], Jowhari and Ghodsi [2005], Buriol et al. [2006], Pavan et al. [2013], and Buriol et al. [2007], and a randomization technique to speed up any triangle counting algorithm while keeping a good accuracy is proposed in Tsourakakis et al. [2009].

The problem of counting subgraphs different from (and more difficult than) triangles has also been addressed in the literature. In particular, the triangle listing algorithm in Chiba and Nishizeki [1985] can be extended to compute all k-cliques of a graph G in time $O(k\,\alpha(G)^{k-2}m)$, where $\alpha(G)$ is the arboricity of G: for constant values of k, this is $O(m^{k/2})$ in the worst case, since it can be proved that $\alpha(G) = O(\sqrt{m})$. Many works focus on graph streams: for instance, Pavan et al. [2013] show how to approximate the number of small cliques, the algorithm from Buriol et al. [2006] can be extended to any subgraph with three or four nodes [Bordino et al. 2008], while Manjunath et al. [2011] tackle the problem of counting cycles. All these works return estimates, typically concentrated around the true number with high probability. The more general problem of enumerating arbitrary small subgraphs has also been studied in Kane et al. [2012], and the I/O complexity of the enumeration of general pattern graphs is discussed in Silvestri [2014].

Focusing on MapReduce, two different exact algorithms for listing triangles have been proposed in Suri and Vassilvitskii [2011] and validated on real-world datasets. In particular, the `Node Iterator ++` algorithm works in two rounds, generating all possible length 2 paths and checking which of these paths can be closed to form a triangle. This algorithm uses $O(m^{3/2})$ global space. The other algorithm from Suri and Vassilvitskii [2011], called `Partition`, has recently been extended to count arbitrary subgraphs in Afrati et al. [2013], casting it into a general framework based on the computation of multiway joins. These algorithms partition the input graph into subgraphs and then use sequential counting algorithms as a black box on each subgraph. A different generalization of `Partition` to multiple rounds has been described in Park et al. [2014].

A sampling-based randomized approach whose output estimate is strongly concentrated around the true number of triangles (under mild conditions) has been described in Pagh and Tsourakakis [2012]. We remark that subgraph counting becomes more and more computationally demanding as the number k of nodes in the counted subgraph gets larger, due to the combinatorial explosion of the number of candidates. This is especially true for k-cliques, as we will also show in this article, since certain real-world graphs (such as social networks) are characterized by high clustering coefficients and very large numbers of small cliques.

Our results. The contribution in this article is twofold and includes both theoretical and experimental results. On the theory side, we design and analyze the first scalable exact algorithm for counting (and listing) k-cliques as well as sampling-based approximate solutions. In more detail:

—Our exact k-clique counting algorithm uses $O(m^{3/2})$ total space and $O(m^{k/2})$ work, where m is the number of graph edges. The local space and the local running time of mappers and reducers are $O(m)$ and $O(m^{(k-1)/2})$, respectively. For $k = 3$, total space and work match the bounds for triangle counting achieved in Suri and Vassilvitskii [2011]. Similarly to the multiway join algorithm from Afrati et al. [2013], our algorithm is work optimal in the worst case for any k: its work matches the running time of fast sequential algorithms, which is proportional to the arboricity of the graph to the power $k - 2$ [Chiba and Nishizeki 1985]. However, our total space is proportional

to $m^{3/2}$ regardless of k: this means that, differently from Afrati et al. [2013], the communication cost does not grow when k gets larger.
—Our sampling-based estimators reduce dramatically the running time and local space requirements of the exact approach, while providing very accurate solutions with high probability. These algorithms can be proved to belong to the class $\mathcal{MRC}$ [Karloff et al. 2010] for a suitable choice of the probability parameters. For $k = 3$, our concentration results require weaker conditions than the triangle counting algorithm from Pagh and Tsourakakis [2012].

To assess the effectiveness of different clique counting approaches, we conducted a thorough experimental analysis over the Amazon EC2 platform, considering both our algorithms and those described in Afrati et al. [2013] and Suri and Vassilvitskii [2011]. We used publicly available real datasets taken from the SNAP graph library [Leskovec 2014] and synthetic graphs generated according to the preferential attachment model [Barabási and Albert 1999]. As a side effect, our experimental study also sheds light on the number of k-cliques of several SNAP datasets and on its growth rate as a function of k. The outcome of our experiments can be summarized as follows:

—Even for small values of k, some of the input graphs contain a number q_k of k-cliques that can be in the order of tens or hundreds of trillions (recall that a trillion is 10^{12}): in these cases, the output of the k-clique listing problem could easily require terabytes or even petabytes of storage (assuming no compression). As k increases, we witnessed different growth rates of q_k for different graph instances: while on some graphs, $q_{k+1} < q_k$ (even for small values of k), in other instances, $q_{k+1} \gg q_k$, up to two orders of magnitude in our observations. Graphs with a quick growth of the number of k-cliques represent particularly tough instances in practice.
—Among the exact algorithms considered in our analysis, our approach proves to be the most effective when the number of k-cliques is large. There are cases where it is outperformed by the triangle counting algorithm of Suri and Vassilvitskii [2011] and by the multiway join algorithm of Afrati et al. [2013]: this happens either for $k = 3, 4$ or on "easy" instances where parallelization does not pay off (we observed that in these cases a simple sequential algorithm would be even faster). On the tough instances, however, our algorithm can gracefully scale as k gets larger, differently from the multiway join approach [Afrati et al. 2013]. We provide a theoretical justification of these experimental findings.
—Our approximate algorithms exhibit rather stable running times on all graphs for the considered values of k and make it possible to solve in a few minutes instances that were impossible to be solved exactly. The quality of the approximation is extremely good: the error is around 0.08% on average, with more accurate estimates on datasets that are more challenging for the exact algorithms. The variance across different executions, even on different clusters, also appears to be negligible.

Overall, the experiments show the practical effectiveness of our algorithms even on clusters of small/medium size and suggest their scalability to larger clusters. The full experimental package is available at https://github.com/CliqueCounter/QkCount/ for the purpose of repeatability.

Organization of the article. Section 2 gives theoretical preliminaries and briefly reviews MapReduce and the computational model proposed by Karloff et al. [2010]. Section 3 presents our exact algorithm, analyzing its communication and computation costs. Section 4 describes the experimental framework, including the evaluated algorithms and the Amazon cluster setup. Section 5 summarizes our main experimental findings for the exact algorithms. The sampling-based estimators are described and

analyzed—both theoretically and experimentally—in Section 6, and conclusions are addressed in Section 7.

2. PRELIMINARIES

Throughout this article, we denote by q_k the number of cliques on k nodes. For a given graph G and any node u in G, $\Gamma(u)$ is the set of neighbors of node u (u is not included); moreover, $d(u) = |\Gamma(u)|$. We define a total order $\prec$ over the nodes of G as follows: $\forall x, y \in V(G)$, $x \prec y$ if and only if $d(x) < d(y)$ or $d(x) = d(y)$ and $x < y$ (we assume nodes to have comparable and unique labels). Denote by $\Gamma^+(u) \subseteq \Gamma(u)$ the *high-neighborhood* of node u, that is, the set of neighbors x of u such that $u \prec x$; symmetrically, $\Gamma^-(u) = \Gamma(u) \setminus \Gamma^+(u)$ is the set of neighbors x of node u such that $x \prec u$. Given two graphs $G(V, E)$ and $G_1(V_1, E_1)$, G_1 is a subgraph of G if $V_1 \subseteq V$ and $E_1 \subseteq E$. G_1 is an *induced subgraph* of G if, in addition to the previous conditions, for each $u, v \in V_1$ it also holds that $(u, v) \in E_1$ if and only if $(u, v) \in E$. We denote the subgraph induced by the high-neighborhood $\Gamma^+(u)$ of a node u as $G^+(u)$. Our algorithms do not require graphs to be connected. However, since their input is given as a set of edges and isolated nodes are irrelevant for clique counting, we can assume the number n of nodes to be at most twice the number m of edges, that is, $n \leq 2m$. Moreover, we assume the endpoints of each edge to be labeled with their degree (the degree of each node can be precomputed in MapReduce very efficiently [Karloff et al. 2010]).

The following property, which is folklore in the triangle counting literature [Schank and Wagner 2005], will be crucial in the design and analysis of our clique estimators (we report its short proof for completeness):

LEMMA 2.1. *In a graph with m edges, $|\Gamma^+(u)| \leq 2\sqrt{m}$ for each node u.*

PROOF. Let h be the number of nodes with degree larger than $\sqrt{m}$: since there are m edges, it must be that $h \leq 2\sqrt{m}$. If $d(u) > \sqrt{m}$, then nodes in $\Gamma^+(u)$ must also have degree larger than $\sqrt{m}$ and their number is upper bounded by $h \leq 2\sqrt{m}$. If $d(u) \leq \sqrt{m}$, the claim trivially holds since $\Gamma^+(u) \subseteq \Gamma(u)$. □

In the analysis of our approximate estimators, we make use of the following (weaker) version of the Chernoff concentration inequality [Chernoff 1981]:

THEOREM 2.2. *Let $X_1, \ldots, X_h$ be independent identically distributed Bernoulli random variables, with probability of success p. Let $X = \sum_{i=1}^{h} X_i$ be a random variable with expectation $\mu = p \cdot h$. Then, for any $\varepsilon \in (0, 1)$, $Pr\{|X - \mu| > \varepsilon\mu\} \leq 2e^{-\varepsilon^2\mu/3}$.*

We will also exploit the following conjecture of Paul Erdös, proved in Hajnal and Szemerédi [1970]:

THEOREM 2.3. *Every n-node graph with maximum degree Δ is $(\Delta + 1)$-colorable with all color classes of size at least n/Δ.*

We say that an event has high probability when it happens, for a graph G of n nodes, with probability at least $1 - 1/n$, for large enough n.

MapReduce. A MapReduce program is composed of (usually a small number of) rounds. Each round is conceptually divided into three consecutive phases: *map*, *shuffle*, and *reduce*. Input/output values of a round, as well as intermediate data exchanged between mappers and reducers, are stored as $\langle key; value \rangle$ pairs. In the map phase, pairs are arbitrarily distributed among mappers and a programmer-defined map function is applied to each pair. Mappers are stateless and process each input pair independently from the others. The shuffle phase is transparent to the programmer: during this phase, the intermediate output pairs emitted by the mappers are grouped by key. All pairs

ALGORITHM 1: $\texttt{FFF}_k$

Map 1: input $\langle (u, v); \emptyset \rangle$
  if $u \prec v$ **then** emit $\langle u; v \rangle$

Reduce 1: input $\langle u; \Gamma^+(u) \rangle$
  if $|\Gamma^+(u)| \geq k - 1$ **then** emit $\langle u; \Gamma^+(u) \rangle$

Map 2: input $\langle u; \Gamma^+(u) \rangle$ or $\langle (u, v); \emptyset \rangle$
  if input of type $\langle (u, v); \emptyset \rangle$ and $u \prec v$ **then**
    emit $\langle (u, v); \$ \rangle$
  if input of type $\langle u; \Gamma^+(u) \rangle$ **then**
    for each $x_i, x_j \in \Gamma^+(u)$ s.t. $x_i \prec x_j$ **do**
      emit $\langle (x_i, x_j); u \rangle$

Reduce 2: input $\langle (x_i, x_j); \{u_1, \ldots, u_k\} \cup \$ \rangle$
  if input contains \$ **then**
    emit $\langle (x_i, x_j); \{u_1, \ldots, u_k\} \rangle$

Map 3: input $\langle (x_i, x_j); \{u_1, \ldots, u_k\} \rangle$
  for $h \in [1, k]$ **do**
    emit $\langle u_h; (x_i, x_j) \rangle$

Reduce 3: input $\langle u; G^+(u) \rangle$
  let $q_{u,k-1}$ = number of $(k-1)$-cliques in $G^+(u)$
  emit $\langle u; q_{u,k-1} \rangle$

with the same key are then sent to the same reducer, which will process each of them by executing a programmer-defined reduce function. Parallelism comes from concurrent execution of mappers and reducers. Karloff et al. [2010] made an effort to pinpoint the critical aspects of efficient MapReduce algorithms. In particular:

Memory. The memory used by a single mapper/reducer should be sublinear with respect to the total input size (this allows one to exclude trivial algorithms that simply map the whole input to a single reducer, which then solves the problem via a sequential algorithm).
Machines. The total number of machines available should be sublinear in the data size.
Time. Both the map and the reduce functions should run in polynomial time with respect to the original input length.

The model also requires programs to be composed of at most a polylogarithmic number of rounds, since shuffling is a time-consuming operation, and to have a total memory usage (which coincides with the communication cost from Afrati et al. [2013] for the purpose of this article) that grows substantially less than quadratically with respect to the input size. Algorithms respecting these conditions are said to belong to the class $\mathcal{MRC}$.

3. EXACT COUNTING

Our algorithms use the total order $\prec$ to decide which node of a given clique Q is responsible for counting Q. In particular, Q is counted by its *smallest* node, that is, the node $u \in Q$ such that $u \prec x$, for all $x \in Q \setminus \{u\}$. At a high level, the strategy is to split the whole graph into many subgraphs, namely, the subgraphs $G^+(u)$ induced by $\Gamma^+(u)$, for each $u \in V(G)$, and count the cliques in each subgraph independently (both nodes and edges of G can appear in more than one subgraph). Our counting algorithm, called $\texttt{FFF}_k$, works in three rounds (see Algorithm 1 for the pseudocode):

Round 1. High-neighborhood computation. The computation of $\Gamma^+(u)$, for all nodes u in G, exploits the degree information attached to the edges. Since the input graph is undirected, for simplicity we assume that each edge appears twice. Mappers emit the pair $\langle x; y \rangle$ for each edge (x, y) such that $x \prec y$, thus allowing the reduce instance with key u to aggregate all nodes $x \in \Gamma^+(u)$.
Round 2. Small-neighborhoods intersection. The aim of the round is to associate each edge (x, y) with $\Gamma^-(x) \cap \Gamma^-(y)$, that is, with the set of nodes u such that $G^+(u)$ contains (x, y). This is done as follows. The map instance with input $\langle u; \Gamma^+(u) \rangle$ emits a pair $\langle (x, y); u \rangle$ for each pair $(x, y) \in \Gamma^+(u) \times \Gamma^+(u)$ such that $x \prec y$. Besides

the output of round 1, similarly to Suri and Vassilvitskii [2011], mappers are fed with the original set of edges and emit a pair $\langle (x, y); \$ \rangle$ for each edge (x, y) with $x \prec y$. This allows the reduce instance with key (x, y) to check whether (x, y) is an edge by looking for symbol \$ among its input values. At the same time, this instance would receive the set $\Gamma^-(x) \cap \Gamma^-(y)$, which is exactly the set of nodes u needing edge (x, y) to construct $G^+(u)$.

Round 3. $(k-1)$*-clique counting in subgraphs induced by high-neighborhoods.* For each node u, count the number of k-cliques for which u is responsible. This is done as follows. Map instances correspond to graph edges. The map instance with key (x, y) emits a pair $\langle u; (x, y) \rangle$ for each node $u \in \Gamma^-(x) \cap \Gamma^-(y)$. After shuffling, the reduce instance with key u receives as input the whole list of edges between nodes in $\Gamma^+(u)$. Hence, it can reconstruct the subgraph $G^+(u)$ induced by the high-neighbors of u and, by counting the $(k-1)$-cliques in this graph, can compute locally the number of k-cliques for which u is responsible.

Notice that the round 3 reducers could be easily modified to output, for each node v, the number of cliques in which v is contained, so that the overall number of cliques containing v could be obtained by summing up the contributions from each subgraph $G^+(u)$. The following theorem analyzes work and space usage of FFF_k:

THEOREM 3.1. *Let G be a graph and let m be the number of its edges. Algorithm FFF_k counts the number of k-cliques of G using $O(m^{3/2})$ total space and $O(m^{k/2})$ work. The local space and the local running time of mappers and reducers are $O(m)$ and $O(m^{(k-1)/2})$, respectively.*

PROOF. The total space usage in round 1 is $O(m)$. Map 2 instances produce key-value pairs of constant size, whose total number is upper bounded by $\sum_{u \in V} \binom{|\Gamma^+(u)|}{2}$, which is at most $2\sqrt{m} \cdot \sum_{u \in V} |\Gamma^+(u)| = O(m^{3/2})$ by Lemma 2.1. The data volume can only decrease after the execution of reduce 2 instances and is not affected by round 3. Hence, the total space usage is $O(m^{3/2})$.

We now consider local space. Map 1 instances use constant memory. By Lemma 2.1, the input to any reduce 1 instance has size $O(\sqrt{m})$. Similarly, any map 2 instance receives $O(\sqrt{m})$ input edges and produces $O(m)$ key-value pairs. Consider a reduce 2 instance and let (x, y) be its key. The input of this instance is $\Gamma^-(x) \cap \Gamma^-(y) \subseteq V$ (without any repetition), and its size is thus $O(n)$. In round 3, map and reduce instances use memory $O(n)$ and $O(m)$, respectively, which concludes the proof of the local space claim (recall that $n \leq 2m$).

By similar arguments, the running time of map instances is $O(1)$, $O(m)$, and $O(n)$, respectively, in the three rounds. Reduce instances require time $O(\sqrt{m})$ and $O(n)$ in rounds 1 and 2, while reduce 3 instances run on graphs of at most $\sqrt{m}$ nodes and require $O(m^{(k-1)/2})$ time. The total work of the algorithm is dominated by the costs of the reducers of the last phase, which is upper bounded by $O(\sum_{u \in V} |\Gamma^+(u)|^{k-1})$. By Lemma 2.1, this is $O(m^{(k-2)/2} \sum_{u \in V} |\Gamma^+(u)| = O(m^{k/2})$.

Each clique is counted exactly once by the reducer associated to its minimum node (according to $\prec$), proving the correctness of the algorithm. □

Discussion. With respect to the requirements defined in Karloff et al. [2010], algorithm FFF_k does not fit in the class $\mathcal{MRC}$ due to the local space requirements of reduce 2, map 3, and reduce 3 instances. These are linear in n or m, whereas the $\mathcal{MRC}$ class requires them to be in $O(m^{1-\varepsilon})$, for some small constant $\varepsilon > 0$. However, as we will see from the experimental evaluation we performed, local memory did not show any criticality in practice, whereas local complexity (which is almost completely neglected

in the class $\mathcal{MRC}$) and global work proved to be the real challenge, since they can grow significantly as k gets larger. Our approximate algorithms presented in Section 6 overcome these issues.

Similarly to the triangle counting algorithms from Suri and Vassilvitskii [2011] and to the multiway join subgraph counting algorithm from Afrati et al. [2013], cast to the specific case of cliques (in short, $\texttt{AFU}_k$), the work of $\texttt{FFF}_k$ is optimal with respect to the clique listing problem. Moreover, the total space of $\texttt{FFF}_k$ is $\Theta(m^{3/2})$ regardless of k, matching the triangle counting bound of the `Node Iterator ++` algorithm from Suri and Vassilvitskii [2011] when $k = 3$. The generalization of algorithm `Partition` to k-cliques, as described in Suri and Vassilvitskii [2011], and $\texttt{AFU}_k$ have instead a communication cost that depends both on a number b of *buckets*, chosen as a parameter, and on the number k of clique nodes (see Afrati et al. [2013] and Suri and Vassilvitskii [2011] for details). In $\texttt{AFU}_k$, reducers are identified by k-tuples of buckets $\langle b_i \leq b_2 \leq \cdots \leq b_k \rangle$. Each edge corresponds to a pair $\langle i, j \rangle$ of buckets (those to which its endpoints are hashed) and is sent to all the reducers whose k-tuple contains both i and j. Each edge is thus replicated $\Omega(b^{k-2})$ times, for an overall communication cost $\Theta(m \cdot b^{k-2})$. Similar arguments apply to the generalization of `Partition`. We will see the implications in Section 5. We also notice that, differently from $\texttt{AFU}_k$ and `Partition`, algorithm $\texttt{FFF}_k$ needs no parameter tuning.

4. EXPERIMENTAL SETUP

4.1. Algorithms and Implementation Details

Besides algorithm $\texttt{FFF}_k$, we included in our test suite the `Node Iterator ++` triangle counting algorithm from Suri and Vassilvitskii [2011] (called `SV`) and the one-round subgraph counting algorithm $\texttt{AFU}_k$ based on multiway joins [Afrati et al. 2013], cast to k-cliques. We did not consider the `Partition` algorithm from Suri and Vassilvitskii [2011] because $\texttt{AFU}_3$ is its optimized version. Both the reduce 3 instances of $\texttt{FFF}_k$ and the reducers of $\texttt{AFU}_k$ use as a subroutine an efficient clique counting algorithm based on neighborhood intersection, inspired by the fastest (optimal) triangle listing algorithm described in Ortmann and Brandes [2014]. In the case of $\texttt{AFU}_k$, we took care of exploiting bucket orderings to discard as soon as possible k-cliques that do not fit in the k-tuple of a given reducer. According to our tests, this results in slightly faster running times w.r.t. to the plain version. All the implementations have been realized in Java using Hadoop 2.2.0. The code is instrumented so as to collect detailed statistics of map/reduce instances at each round, including sizes of the subgraphs involved in a computation (e.g., $|\Gamma^+(u)|$, $|\Gamma^-(x) \cap \Gamma^-(y)|$, and $|G^+(u)|$) and detailed running times. Due to the extremely large size of the log files and in order to prevent Heisenberg effects,[1] all the results reported in this article were obtained with instrumentation disabled, unless otherwise noted. The algorithms have been tested in a variety of settings, using different parameter choices, instance families, and cluster configurations, as described later. The full package is available at `https://github.com/CliqueCounter/QkCount/` for the sake of repeatability.

4.2. Datasets

We used several real-world graphs from the SNAP graph library [Leskovec 2014] as well as synthetic graphs generated according to the preferential attachment model [Barabási and Albert 1999]. We preprocessed all graphs so that they are undirected and each edge endpoint is associated with its degree. Degree computation is a common step to both `SV` and $\texttt{FFF}_k$ and can be done very easily and quickly in MapReduce

[1]Term derived from Heisenberg's uncertainty principle: the very act of observing a phenomenon alters it.

Table I. Benchmark Statistics

	$n (= q_1)$	$m (= q_2)$	q_3	q_4	q_5	q_6	q_7
citPat	3.8×10^6	1.6×10^7	7.5×10^6	3.5×10^6	3.0×10^6	3.1×10^6	1.9×10^6
youTube	1.1×10^5	3.0×10^6	3.0×10^6	5.0×10^6	7.2×10^6	8.4×10^6	8.0×10^6
locGowalla	2.0×10^5	9.5×10^5	2.7×10^6	6.1×10^6	1.5×10^7	2.9×10^7	4.8×10^7
socPokec	1.6×10^6	2.2×10^7	3.3×10^7	4.3×10^7	5.3×10^7	6.5×10^7	8.4×10^7
webGoogle	8.7×10^5	4.3×10^6	1.3×10^7	3.4×10^7	1.0×10^8	2.5×10^8	6.0×10^8
webStan	2.8×10^5	2.0×10^6	1.1×10^7	7.9×10^7	6.2×10^8	4.9×10^9	3.5×10^{10}
asSkit	1.7×10^6	1.1×10^7	2.9×10^7	1.5×10^8	1.2×10^9	9.8×10^9	7.3×10^{10}
orkut	3.1×10^6	1.2×10^8	6.3×10^8	3.2×10^9	1.6×10^{10}	7.5×10^{10}	3.5×10^{11}
webBerkStan	6.8×10^5	6.6×10^6	6.5×10^7	1.1×10^9	2.2×10^{10}	4.6×10^{11}	9.4×10^{12}
comLiveJ	4.0×10^6	3.5×10^7	1.8×10^8	5.2×10^9	2.5×10^{11}	1.1×10^{13}	*4.4 × 10^14*
socLiveJ1	4.8×10^6	4.3×10^7	2.9×10^8	9.9×10^9	4.7×10^{11}	2.1×10^{13}	*8.6 × 10^14*
egoGplus	1.1×10^5	1.2×10^7	1.1×10^9	7.8×10^{10}	4.7×10^{12}	*2.4 × 10^14*	*1.1 × 10^16*

Order of magnitude of n, m, and q_k (numbers of nodes, edges, and k-cliques, respectively) for $k \in [3, 7]$. Clique numbers in *italic* are estimates obtained by the algorithm described in Section 6. The table in Appendix A shows the exact values of n, m, and q_k; the storage in MB; and the ratio q_{k+1}/q_k that gives the clique growth rate as a function of k. The horizontal line separates "easy" from "hard" instances, where the latter are characterized by a large number of cliques and a steep clique growth rate.

[Karloff et al. 2010]. Throughout the article, we report on the results obtained for a variety of online social networks (called `orkut`, `socPokec`, `youTube`, `locGowalla`, `socLiveJ1`, `comLiveJ`, `egoGplus`), web graphs (`webBerkStan`, `webGoogle`, `webStan`), an Internet topology graph (`asSkit`), and a citation network among US patents (`citPat`). The main characteristics of these datasets are summarized in Table I. Notice that only the number q_3 of triangles was available from Leskovec [2014] before our study. With respect to q_3, the number of k-cliques for larger values of k can grow considerably, up to the order of tens or even hundreds of trillions. The horizontal line in Table I separates "easy" and "hard" instances: the latter ones, at the bottom of the table, are characterized by large numbers of cliques and/or steep clique growth rates. We refer to Appendix A for the exact figures.

4.3. Platform

The experiments have been carried out on three different Amazon EC2 clusters, running Hadoop 2.2.0. Besides the master node, the three clusters included four, eight, and 16 worker nodes, respectively, devoted to both Hadoop tasks and the HDFS. We used Amazon EC2 `m3.xlarge` instances, each providing four virtual cores (based on Intel Xeon E5-2670 v2 Sandy Bridge processors), 7.5GiB of main memory, and a 32GB solid-state disk. We set the number of reduce tasks to match the number of virtual cores in each cluster and disabled speculative execution. We also modified the memory requirements of the containers in order to improve load balancing on the cluster. The precise Hadoop configuration used on the Amazon clusters is provided with our experimental package available on `github`.

5. COMPUTATIONAL EXPERIMENTS

In this section, we summarize our main experimental findings. We first present results obtained on a 16-node Amazon cluster, analyzing the effects of k on the performance of the algorithms, and we then address scalability issues on different cluster sizes. Our experiments account for more than 60 hours of computation over the EC2 platform. Table II is the main outcome of this study, showing the running times of all the evaluated algorithms for $k \leq 7$ on the SNAP graphs ordered by increasing q_7 (see Appendix A). Results for synthetic instances were consistent with real datasets and are not reported.

Table II. Running Time (minutes:seconds) of the Algorithms on a 16-node Cluster with 64 Total Cores

	SV	FFF_3	AFU_3	FFF_4	AFU_4	FFF_5	AFU_5	FFF_6	AFU_6	FFF_7	AFU_7
citPat	2:44	3:22	2:23	3:11	3:11	3:13	2:18	3:13	2:19	3:09	2:24
youTube	2:06	2:04	1:25	2:39	1:41	2:34	1:33	2:36	1:39	2:38	1:49
locGowalla	2:36	3:08	1:18	3:04	1:21	3:02	1:30	3:04	1:24	3:03	1:30
socPokec	4:03	4:15	2:18	4:02	2:29	4:13	2:39	4:15	2:51	4:09	3:02
webGoogle	2:13	2:44	1:23	2:43	1:27	2:43	1:32	2:40	1:40	2:40	1:52
webStan	2:02	2:39	1:15	2:29	1:27	2:37	2:06	2:36	4:00	2:05	14:12
asSkit	2:44	3:14	1:43	3:17	2:59	3:18	5:34	3:14	25:30	4:12	>40
orkut	30:07	24:00	8:21	23:08	20:17	23:10	>50	23:22	>50	28:08	-
webBerkStan	2:28	3:00	1:37	3:01	2:53	3:08	8:24	4:56	>30	50:17	-
comLiveJ	5:31	5:31	2:53	5:24	4:06	6:13	14:02	41:22	>170	-	-
socLiveJ1	6:36	6:33	3:14	6:43	5:10	7:51	23:35	86:34	>180	-	-
egoGplus	22:54	17:19	2:06	17:54	16:55	39:01	>90	-	-	-	-

To minimize the costs, we killed some executions of AFU_k that took more than twice the time of FFF_k. For the sake of comparison, the running times of Algorithm SV reported in Suri and Vassilvitskii [2011] are 1.90, 1.77, and 5.33 minutes on asSkit, webBerkStan, and socLiveJ1, respectively, on a 1,636-node cluster.

Triangle counting: the costs of rounds. Since the overhead of setting up a round—including shuffling—is nonnegligible in MapReduce, the one-round AFU_3 algorithm is always much faster than SV and FFF_3, which respectively require two and three rounds. FFF_3 is slower than SV on most datasets but faster on orkut and egoGplus, which have the largest number of triangles (see also Table I). We conjectured that this may be due to the early computation of length 2 paths performed in SV by the round 1 reducers (see Suri and Vassilvitskii [2011] for details), which uselessly increases the communication cost of round 1. To test our hypothesis, we engineered a variant of SV that delays 2-path computation to the map phase of round 2. The variant showed largely improved running times, solving orkut and egoGplus in 22 and 12 minutes (instead of 30 and 23, respectively) and being faster than FFF_3 on all benchmarks.

Running time analysis for $k \geq 4$. Algorithm FFF_4 can compute the number of 4-cliques within roughly the same time required to count triangles. AFU_4 remains faster than FFF_4 but is always slower than AFU_3: see, in particular, orkut and egoGplus. In general, when $k \geq 5$, FFF_k proves to be more and more effective than AFU_k and its running times scale gracefully with k, especially on the most difficult instances characterized by a steep growth of the number of k-cliques (asSkit, the LiveJournal networks, orkut, webBerkStan, and egoGplus). To explain this behavior, recall that AFU_k requires one to choose the number b of buckets that, together with k, determines the number of reducers. The communication cost, as observed in Section 3, grows as $\Theta(m \cdot b^{k-2})$, which in practice calls for small values of b: if b is too large, even a small input graph could quickly grow to terabytes of disk usage for moderate values of k. On the other hand, since the local running times of reducers are inversely proportional to b, if b is too small w.r.t. k, the worst-case reducers incur high running times (if, e.g., $k > b/2$, some k-tuple contains more than half of the buckets and the corresponding reducer will receive more than half of the total number of edges). The running times of such reducers (as well as their memory requirements) remain comparable to that of the sequential algorithm. The practical implication of this communication/runtime tradeoff is that the best performances of AFU_k have been obtained within our experimental setup using rather small values of b. We nevertheless performed experiments with different choices of b: as suggested in Afrati et al. [2013], we first set b so that $\binom{b+k-1}{k}$ is as close as possible to the available reducers, and we then considered a variety of different values. Table II

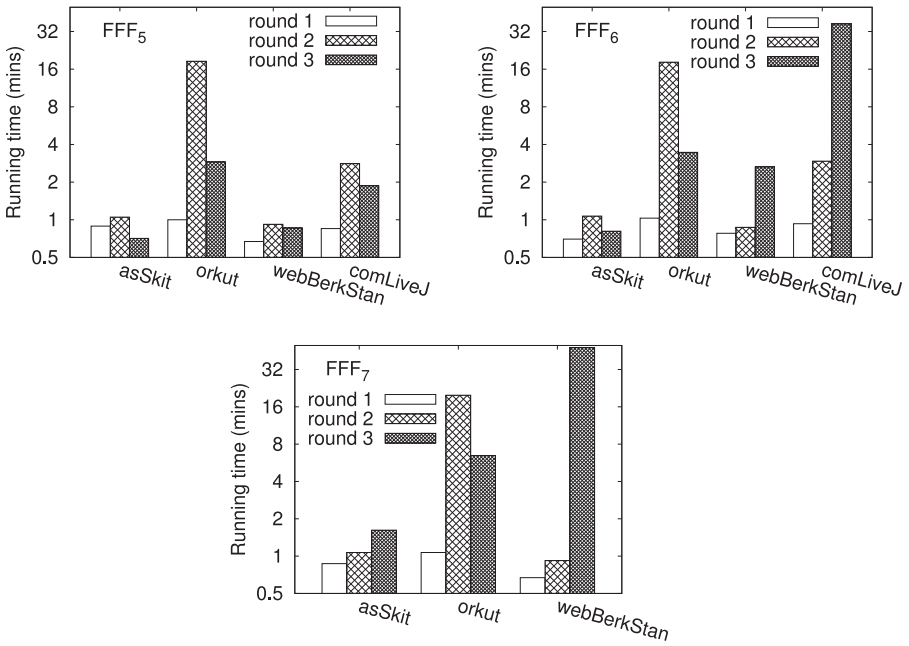

Fig. 1. Round-by-round running times of $\texttt{FFF}_k$ on four representative datasets for $k \in [5, 7]$.

always reports the fastest running time that we could obtain by separately tuning b for each instance and k.

The $\Theta(m^{3/2})$ communication cost of $\texttt{FFF}_k$ proved to be a bottleneck only in a few datasets, while the actual clique enumeration remains the most time-consuming phase. However, the running times of reducers are much shorter, not only in theory but also in practice, than the application of a sequential algorithm to the whole graph: hence, using more rounds results in poor performance only when the overall task has a short duration, but quickly outperforms both $\texttt{AFU}_k$ and sequential algorithms on the most demanding graphs and as k increases. Insights on round analysis are given next.

Round-by-round analysis of $\texttt{FFF}_k$. In Figure 1, we compare the running times of each round of $\texttt{FFF}_k$ on a selection of benchmarks. Round 1 is typically negligible, regardless of the benchmark and of the value of k. Round 2, which computes 2-paths and small-neighborhoods intersections, is the most expensive step for $k \leq 5$, but its running time can only decrease when k grows (due to the test $|\Gamma^+(u)| \geq k - 1$ performed by reduce 1 instances; see Algorithm 1). Round 3 becomes more and more expensive as k gets larger, and dominates the running time on `webBerkStan` and `comLiveJ` already for $k = 6$. This confirms the intuition supported by our theoretical analysis: computing $(k-1)$-cliques on the subgraphs $G^+(u)$ induced by high-neighborhoods can be rather time-consuming and becomes the dominant operation as k gets larger. Figure 2(a) shows the cumulative distribution of $|G^+(u)|$, focusing on reduce 3 instances that required more than 100ms: notice that a constant fraction of nodes has rather large induced subgraphs (e.g., in `egoGplus`, about 5,000 nodes have high-neighborhoods with a number of edges in between 2^{16} and 2^{18}, which is the largest $|G^+(u)|$).

Scalability on different clusters. Since MapReduce algorithms are inherently parallel, a natural question is how their running times are affected by the cluster size (and, ultimately, by the available number of cores). Figure 3 exemplifies the running times

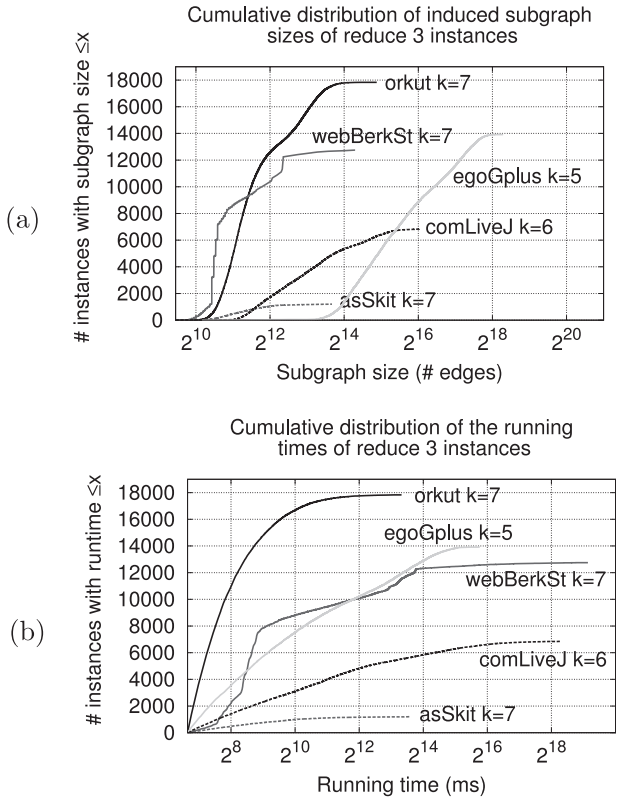

Fig. 2. Analysis of reduce instances of round 3 of algorithm $\texttt{FFF}_k$ on a selection of benchmarks.

on three clusters of four, eight, and 16 nodes. We focus on $\texttt{FFF}_k$, which proved to be the algorithm of choice for $k \geq 5$. As an example, the average speedups of $\texttt{FFF}_6$ when doubling the cluster size from four to eight nodes and from eight to 16 nodes are 1.44 and 1.53, respectively (the average is taken over the three graph instances). We remark that the maximum theoretical speedup is 2. Similar values can be obtained for the other values of k.

While we could not experiment on considerably larger clusters (Amazon AWS limits the number of on-demand instances that can be requested), Figure 2 provides some insights. We focus on round 3, which proved to be the most expensive step when q_k is large (see, e.g., `webBerkStan` and `comLiveJ` in Figure 1). Figure 2(b) shows the cumulative distribution of the running times of reduce 3 instances (for executions longer than 100ms). We focused on the least favorable scenarios, corresponding to large values of k for which the problem is more computationally intensive. The rightmost point on each curve gives the runtime of the slowest reduce instance, which reaches 9 minutes when computing q_7 on `webBerkStan`. Although most curves are steeper for short durations, in all cases there are hundreds or even thousands of reduce instances with running times comparable to the slowest one: for example, in `egoGplus`, more than 2,000 reducers are within a factor $8\times$ of the slowest one, and even in the q_7 computation for `webBerkStan`, 169 instances require more than 1 minute. This suggests that $\texttt{FFF}_k$ is amenable to further parallelization: we expect that, on a larger cluster, the abundant time-demanding instances could be effectively scheduled to different nodes, yielding

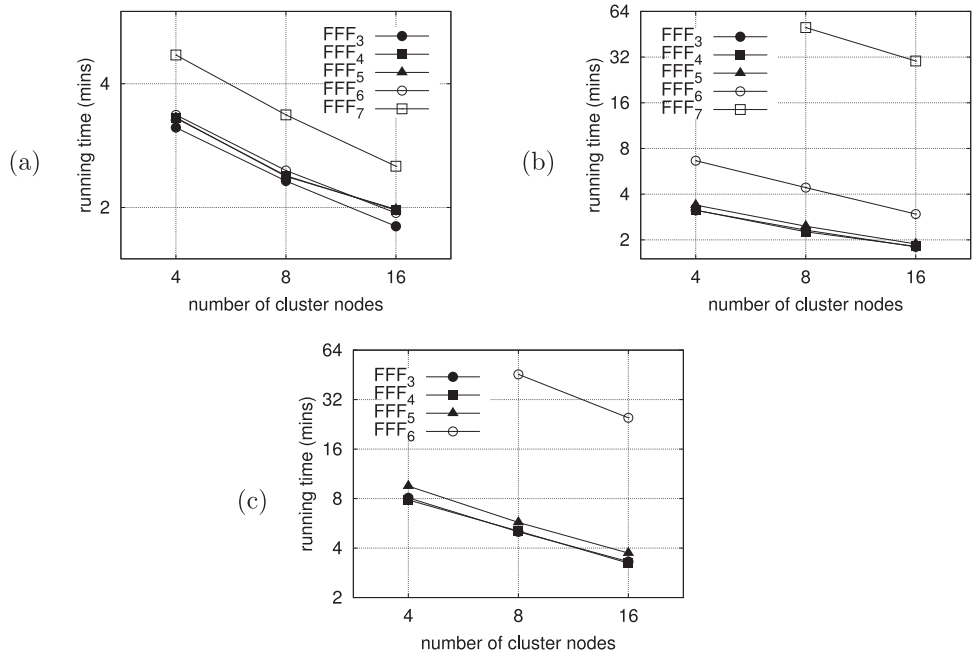

Fig. 3. Scalability on different cluster sizes: (a) `asSkit`; (b) `webBerkStan`; (c) `comLiveJ`.

globally shorter running times. This analysis is in line with the distribution of induced subgraph sizes observed in Figure 2(a), supporting the conclusion that the harmful "curse of the last reducer" phenomenon [Suri and Vassilvitskii 2011]—where typically 99% of the map/reduce instances terminate quickly but a very long time could be needed for the last task to succeed—can be significantly mitigated even when k is increased.

6. APPROXIMATE COUNTING

In this section, we analyze two variants of a sampling strategy that allows us to decrease the overall space usage, starting from the output of map 2 instances. The space saving in map 2 instances propagates to the following phases, reducing the space used by reduce 2 as well as map and reduce 3 instances, and also results in an improved running time (due to reduced local complexities and global work). Instead of performing the sampling directly on the list of edges of the graph, we work by sampling pairs of high-neighbors that are emitted by map 2 instances. If each pair of high-neighbors of a given node u is emitted with probability p, then each edge of $G^+(u)$ will be included with probability p in the subgraph built by the reduce 3 instance with key u; however, the same edge e in two distinct subgraphs $G^+(u)$ and $G^+(u')$ is sampled independently, which results in improved concentration around the mean.

6.1. Plain Pair Sampling

We first address the case when pairs of high-neighbors are sampled uniformly at random. In detail, we modify algorithm $\texttt{FFF}_k$ as follows:

—Map 2 instances emit the key-value pair $\langle (x_i, x_j); u \rangle$, for all pairs (x_i, x_j) with $x_i \prec x_j$ in $\Gamma^+(u)$, *with probability* p.
—Reduce 3 instances emit the pair $\langle u; q_u / p^{(k-1)(k-2)/2} \rangle$.

We call the resulting approximation algorithm pFFF$_k$. Let $\mathcal{Q} = \{Q_1, \ldots, Q_{q_k}\}$ be the set of k-cliques of G. Let $Q_i \in \mathcal{Q}$ be a clique and let u be its smallest node (according to $\prec$). We say that Q_i is *sampled* if all pairs in $Q_i \setminus \{u\}$ were emitted by the map 2 instance with key u. We now prove the results about space usage, correctness, and concentration of the plain pair sampling algorithm.

LEMMA 6.1. *If p is a sampling probability such that $1 \geq p \geq 1/m^\alpha$ for some suitable constant $\alpha \in [0, 1)$, then algorithm pFFF$_k$ uses local space $O(mp)$ with high probability.*

PROOF. We will prove the claim for reduce 3 instances; similar arguments can be used to prove the bounds for reduce 2 and map 3 instances, recalling that $n \leq 2m$.

By Lemma 2.1, each subgraph $G^+(u)$ has at most $2\sqrt{m}$ nodes. Since the reduce 3 instance with key u receives an edge $(x, y) \in G^+(u)$ if and only if the map 2 instance with key u sampled the pair (x, y) (event that has probability p), we have that the expected input size of any reduce 2 instance is at most $2pm$.

Being that each pair of high-neighbors of a node u was sampled independently by all the others, a simple application of the Chernoff bound allows us to prove that the probability of one reduce 3 instance to receive more than $4pm$ values is less than $e^{-2pm/3}$, and a union bound gives that the probability of any of the reduce 3 instances to receive more than $4pm$ values is bounded by $n/e^{2pm/3}$. Since $p \geq 1/m^\alpha$, for large enough m this probability is smaller than $1/m$, which concludes the proof. □

LEMMA 6.2. *Algorithm FFF$_k$ with edge sampling returns an estimate $\widetilde{q_k}$ with expected value $E[\widetilde{q_k}] = q_k$, where q_k is the number of k-cliques in the input graph G.*

PROOF. Let $Q_i \in \mathcal{Q}$ be a clique in G and let u be its smallest node (according to $\prec$). Clique Q_i contributes to the estimate of the number of cliques in G if the map 2 instance handling input $\langle u; \Gamma^+(u)\rangle$ emits all pairs of nodes in $Q_i \setminus u$, that is, if it is sampled. Let X_i be the random variable indicating the event "the clique Q_i is sampled." Since each pair is sampled by the algorithm independently with probability p and there are $\binom{k-1}{2}$ distinct (unordered) pairs in a set of $k-1$ elements, $Pr\{X_i = 1\} = p^{(k-1)(k-2)/2}$, and hence $E[X_i] = p^{(k-1)(k-2)/2}$. Now we can define $X = \sum_{i=1}^{q_k} X_i$, and by linearity of expectation, we have that $E[X] = q_k p^{(k-1)(k-2)/2}$. Since $\widetilde{q_k} = X/p^{(k-1)(k-2)/2}$, the claim follows. □

THEOREM 6.3. *Let G be a graph with m edges and q_k k-cliques. Let $\widetilde{q_k}$ be the estimate returned by algorithm FFF$_k$ with edge sampling probability p. For any constant $\varepsilon > 0$, there exists a constant $h > 0$ such that $|\widetilde{q_k} - q_k| \leq \varepsilon q_k$ with high probability if $p^{(k-1)(k-2)/2} > \frac{hm^{(k-3)/2} \ln m}{\varepsilon^2 q_k}$.*

PROOF. We proved that the expected value of $\widetilde{q_k}$ is q_k in Lemma 6.2; we will now deal with the concentration of $\widetilde{q_k}$ around its mean. Let H be the graph defined as follows:

—The node set of H is the set of k-cliques $\mathcal{Q}$ of G.
—Let Q_i and Q_j be two k-cliques with the same smallest node u: there is an edge between Q_i and Q_j in H if and only if $Q_i \setminus \{u\}$ and $Q_j \setminus \{u\}$ have at least one common edge.

Cliques that are adjacent in H must thus share at least three of their k nodes, including the smallest node. Considering that graphs $G^+(u)$ have at most $\sqrt{m}$ nodes, we have that the maximum degree of a node in H is therefore $O(m^{(k-3)/2})$.

Theorem 2.3 by Hajnal and Szemerédi implies that there exists a node coloring of H, using $C \in O(m^{(k-3)/2})$ colors, such that each monochromatic set of nodes has size $\Theta(q_k/m^{(k-3)/2})$, since H has q_k nodes.

For each $j \in [1, q_k]$, let X_j be the indicator variable that is 1 when Q_j is sampled. Let $S_1, \ldots, S_C$ be the sets of monochromatic nodes in H and, for each $i \in [1, C]$, let

$$X_{S_i} = \sum_{j:Q_j \in S_i} X_j. \tag{1}$$

The terms of X_{S_i} are independent. Hence, we can apply to X_{S_i} the Chernoff bound as in Theorem 2.2, obtaining:

$$Pr\{|X_{S_i} - \mu_i| > \varepsilon\mu_i\} \leq 2e^{-\mu_i\varepsilon^2/3},$$

where $\mu_i = E[X_{S_i}]$. By Equation (1), for each set S_i we have

$$\mu_i \in \Theta\left(\frac{p^{(k-1)(k-2)/2}q_k}{m^{(k-3)/2}}\right)$$

since $E[X_j] = p^{(k-1)(k-2)/2}$, as shown in the proof of Lemma 6.2. By the same proof, $\mu = \sum_{i=1}^{C} \mu_i = p^{(k-1)(k-2)/2}q_k$.

If we define $X = \sum_{i=1}^{C} X_{S_i}$ and we apply the union bound, we can conclude that

$$Pr\{|X - \mu| > \varepsilon\mu\} \leq c_1 m^{\frac{k-3}{2}} e^{-\frac{c_2\varepsilon^2 q_k p^{(k-1)(k-2)/2}}{m^{(k-3)/2}}}$$

by appropriately choosing constants c_1 and c_2. By imposing

$$c_1 m^{\frac{k-3}{2}} e^{-\frac{c_2\varepsilon^2 q_k p^{(k-1)(k-2)/2}}{m^{(k-3)/2}}} \leq \frac{1}{m},$$

the claim follows with standard algebraic calculations. □

6.2. Color-Based Sampling

Pagh and Tsourakakis [2012] proposed a sampling technique that yields an increase in the expected number of sampled cliques without increasing the number of sampled edges. This is achieved by coloring the nodes of the graph and sampling all monochromatic edges. The same idea can be applied in our setting to sample the emitted pairs in map 2 instances by coloring all nodes in each $\Gamma^+(u)$ with c colors and emitting all monochromatic pairs. This has the following implications. Each edge in $G^+(u)$ is sampled with probability $1/c$. A k-clique Q_i with smallest node u is sampled with probability $1/c^{k-2}$, given by the probability of assigning all nodes in $Q_i \setminus \{u\}$ with the same color. Hence, reduce 3 instances have to be modified in order to return $\langle u; q_u c^{k-2}\rangle$ as partial estimates of the number of k-cliques of G. Let us call the resulting approximation algorithm $\texttt{cFFF}_k$.

The analysis given in Section 6.1 can be naturally extended to algorithm $\texttt{cFFF}_k$. Lemma 6.1 holds for algorithm $\texttt{cFFF}_k$ just by using $p = 1/c$. The estimate returned by algorithm $\texttt{cFFF}_k$ has expected value q_k, and this can be proved using arguments similar to those in the proof of Lemma 6.2. The arguments of the proof of Theorem 6.3 can also be used to prove concentration around the mean for algorithm $\texttt{cFFF}_k$, considering that correlation of sampled k-cliques arises as soon as the cliques share a node besides the minimum node, instead of an edge. Hence, concentration is achieved with high probability under the conditions expressed by the following theorem:

THEOREM 6.4. *Let G be a graph of m edges and q_k k-cliques. Let $\widetilde{q_k}$ be the estimate returned by algorithm* cFFF$_k$ *with c colors. For any constant $\varepsilon > 0$ there exists a constant $h > 0$, such that $|\widetilde{q_k} - q_k| \leq \varepsilon q_k$ with high probability if $1/c^{k-2} > \frac{hm^{k-2} \ln m}{\varepsilon^2 q_k}$.*

For the case $k = 3$, Theorem 6.4 guarantees (with high probability) concentration around the mean when $1/c \geq (hm \ln m)/(\varepsilon^2 q_3)$, which improves the bound in Pagh and Tsourakakis [2012], whose worst-case analysis imposes $1/c^2 \geq (h'n^2 \ln n)/(\varepsilon^2 q_3)$ to guarantee concentration on an n-node graph. This is due to the fact that we color the same node $x \in \Gamma^+(u) \cap \Gamma^+(v)$ independently for u and v, which allows us to reduce the maximum degree of the interference graph H in the application of the Hajnal-Szemeredi theorem.

Discussion. Our estimators fit in the class $\mathcal{MRC}$ as long as $p \leq 1/m^\alpha$ (and equivalently $c \geq m^\alpha$), for a small constant α, as shown by Lemma 6.1. Moreover, the space reduction translates in improved bounds for the local complexities (in particular for reduce 3 instances that are the most computationally intensive) and work. Notice that the concentration result in Theorem 6.4 is weaker than that in Theorem 6.3. This is due to the fact that the interference graph used in the proof of Theorem 6.3 has an even smaller maximum degree than that resulting from the color-based sampling. As observed earlier, this is beneficial in the application of the Hajnal-Szemerédi theorem. However, the color-based sampling strategy increases the expected number of sampled cliques, using sampling probability $p = 1/c$, with respect to plain pair sampling. The expected number of sampled cliques shrinks by a factor $p^{(k-1)(k-2)/2}$ for plain sampling, and only by a factor p^{k-2} for color-based sampling. In practice, this boosts the accuracy of the color-based algorithm, which becomes better and better than plain sampling when k grows. We clearly observed this phenomenon, which was also discussed in Pagh and Tsourakakis [2012] for triangles, in our experiments.

6.3. Experiments with Approximate Counting

We experimented with both edge-based and color-based sampling (algorithms pFFF$_k$ and cFFF$_k$, respectively), choosing different sampling probabilities and running each algorithm three times on the same instance and platform configuration to increase the statistical confidence of our results. As predicted by the theoretical analysis, both sampling strategies are beneficial for round 2, since they reduce the number of emitted 2-paths. In turn, reducing 2-paths decreases the number of edges in the induced subgraphs constructed at round 3, yielding substantial benefits on the running time of this round: in particular, in all our tests, we observed that the running time of reduce 3 instances of both pFFF$_k$ and cFFF$_k$ remains almost constant as k increases. Table III summarizes the behavior of the algorithms, showing elapsed time, speedup over the exact algorithm, and accuracy.

The experiments were performed on the 16-node cluster using 10 colors, which corresponds to a sampling probability 0.1. The achieved speedups are dramatic (up to 25× in socLiveJ1) in all those cases where the exact algorithm took a long time. We were able to compute in a few minutes the estimated number of q_6 and q_7 of graphs where the exact computation would have required several hours. Algorithm pFFF$_k$ achieves a better speedup than cFFF$_k$, whose implementation is slightly more complex. However, the benefit is hardly noticeable. The accuracy of cFFF$_k$ is very good, especially on the datasets that were most difficult for the exact algorithm: the 24× faster computation on socLiveJ1, for instance, returned an estimate that was only 0.61% away from the exact value of q_6. Conversely, pFFF$_k$ can be much less accurate, with an error as high as 100%. This confirms the observations reported in the discussion at the end of Section 6.2.

Table III. Running Time, Speedup, and Approximation Quality of $\texttt{cFFF}_k$ and $\texttt{pFFF}_k$ for $k \in [3, 7]$; the Speedup Is with Respect to $\texttt{FFF}_k$ (see also Table II)

	$\texttt{cFFF}_3$			$\texttt{cFFF}_4$			$\texttt{cFFF}_5$			$\texttt{cFFF}_6$			$\texttt{cFFF}_7$		
	time	sp.	err (%)	time	sp.	err (%)	time	sp.	err (%)	time	sp.	err (%)	time	sp.	err (%)
citPat	3:07	1.04	0.06	3:06	1.01	0.37	3:07	1.01	9.37	3:09	1.01	1.20	3:50	0.88	38.7
youTube	2:49	0.81	0.02	2:47	0.96	1.14	2:53	0.92	2.75	2:48	0.95	15.81	2:47	0.96	20.74
locGowalla	2:47	1.24	0.32	2:46	1.23	1.63	2:47	1.22	0.58	2:44	1.24	2.04	2:44	1.24	4.92
socPokec	3:14	1.32	0.04	3:14	1.28	0.02	3:14	1.31	2.07	3:15	1.32	8.32	3:11	1.31	12.39
webGoogle	2:56	0.95	0.05	2:52	0.96	0.16	2:54	0.96	0.54	2:53	0.94	3.16	2:52	0.95	0.05
webStan	2:49	0.95	0.03	2:47	0.92	0.04	2:49	0.95	0.76	2:47	0.96	2.65	2:48	0.82	6.5
asSkit	2:53	1.12	0.01	2:51	1.15	0.23	2:49	1.17	0.86	2:49	1.15	4.39	2.44	1.54	1.06
orkut	6:08	3.91	0.02	5:52	3.94	0.05	6:09	3.77	0.08	5:57	3.93	0.34	6:30	4.33	2.39
webBerkSt	2:45	1.09	0.05	2:46	1.09	0.16	2:47	1.13	0.17	2:42	1.83	1.22	2:44	18.4	0.39
comLiveJ	3:24	1.62	0.05	3:27	1.57	0.03	3:29	1.78	0.21	3:25	12.11	0.24	3:22	-	-
socLiveJ1	3:42	1.77	0.02	3:40	1.83	0.03	3:31	2.23	0.03	3:36	24.05	0.61	3:41	-	-
egoGplus	4:18	4.03	0.01	4:18	4.16	0.02	4:17	9.11	0.08	4:36	-	-	5:34	-	-
	$\texttt{pFFF}_3$			$\texttt{pFFF}_4$			$\texttt{pFFF}_5$			$\texttt{pFFF}_6$			$\texttt{pFFF}_7$		
	time	sp.	err (%)	time	sp.	err (%)	time	sp.	err (%)	time	sp.	err (%)	time	sp.	err (%)
citPat	3:09	1.04	0.08	3:05	1.02	0.66	3:06	1.02	64.63	3:04	1.03	99.99	2.59	1.19	100
youTube	2:48	0.82	0.30	2:48	0.96	2.99	2:49	0.93	38.70	2:47	0.95	99.99	2:46	0.97	100
locGowalla	2:49	1.23	0.05	2:45	1.24	0.86	2:47	1.22	31.32	2:45	1.24	99.99	2:45	1.23	100
socPokec	3:17	1.30	0.01	3:16	1.27	0.4	3:13	1.31	14.08	3:16	1.31	99.99	3:15	1.29	100
webGoogle	2:56	0.95	0.08	2:56	0.94	1.19	2:53	0.96	14.22	2:50	0.97	99.99	2:51	0.95	100
webStan	2:50	0.96	0.04	2:50	0.91	0.34	2:54	0.93	7.39	2:49	0.94	99.99	2:46	0.83	100
asSkit	3:07	1.02	0.09	3:03	1.04	0.7	3:04	1.04	0.18	3:01	1.04	99.99	2:59	1.59	100
orkut	6:02	3.99	0.02	6:05	3.81	0.01	6:08	3.79	0.99	6:12	3.79	46.18	6:04	4.64	100
webBerkSt	3:00	1	0.04	2:56	1.17	0.10	2:57	1.19	1.38	2:57	1.77	28.02	2:57	19.52	100
comLiveJ	3:30	1.60	0.01	3:30	1.58	0.04	3:32	1.84	0.15	3:30	12.49	1.99	3:28	-	-
socLiveJ1	3:42	1.85	0.01	3:43	1.87	0.03	3:41	2.20	0.04	3:39	25.46	0.06	3:38	-	-
egoGplus	4:22	4.07	0.01	4:25	4.12	0.03	4:21	9.26	0.07	4:18	-	-	4:21	-	-

7. CONCLUDING REMARKS

We have proposed and analyzed, both theoretically and experimentally, a suite of MapReduce algorithms for counting k-cliques in large-scale undirected graphs, for any constant $k \geq 3$. The experiments, conducted on the Amazon EC2 platform, clearly highlight the algorithm of choice in different scenarios, showing that our algorithms gracefully scale to nontrivial values of k, larger instances, and diverse cluster sizes.

It is worth noting that our approach could be slightly modified in order to trade overall space usage for local running time. The actual count of $(k-1)$-cliques at round 3 could be indeed postponed for all nodes u such that $G^+(u)$ is too large. In an additional round, map instances would replicate each "uncounted" subgraph $G^+(u)$ once per high-neighbor v of u, distributing the workload to many reducers. The reduce instance with key (u, v) would thus count the number of $(k-2)$-cliques in its copy of $G^+(u)$. This process could be repeated up to $k-4$ times before copying $\sqrt{m}$ times $G^+(u)$ becomes more expensive than counting: each iteration would increase by a factor $\sqrt{m}$ the global space usage and reduce by the same factor the local running times of the reducers, without affecting the total work. We expect this tradeoff to be rather effective on very large clusters, especially for skewed distributions of the high-neighborhood sizes and large values of k, and we regard assessing its practicality as an interesting direction.

With respect to approximate solutions, our experiments clearly show that color sampling largely outperforms edge sampling in terms of accuracy. This is in contrast with

the theoretical concentration bounds proved in Section 6.1 and in Section 6.2, respectively. Obtaining milder conditions for the concentration bound of the color sampling algorithm is an interesting open problem.

As a side effect, our experiments uncovered a dichotomy in the SNAP graphs that were used as benchmarks: while in some graphs the number of k-cliques remains relatively small even for $k = 4, 5, 6$, in other graphs the number of larger cliques skyrockets. This suggests a different inherent structure in these benchmarks: detecting and understanding these differences represents a very interesting open problem.

APPENDIX A: DETAILED BENCHMARK STATISTICS AND NUMBER OF CLIQUES

Benchmark Statistics: Number n of Nodes, Number m of Edges, and Numbers of Cliques on 3, 4, 5, 6, and 7 Nodes

	$n\,(=q_1)$	$m\,(=q_2)$	q_3	q_4	q_5	q_6	q_7
citPat	3 774 768	16 518 947	7 515 023	3 501 071	3 039 636	3 151 595	1 874 488
(264.0 MB)		(4.38×)	(0.45×)	(0.47×)	(0.87×)	(1.04×)	(0.6×)
youTube	1 134 890	2 987 624	3 056 386	4 986 965	7 211 947	8 443 803	7 959 704
(38.7 MB)		(2.63×)	(1.02×)	(1.63×)	(1.44×)	(1.17×)	(0.94×)
locGowalla	196 591	950 327	2 273 138	6 086 852	14 570 875	28 928 240	47 630 720
(11.1 MB)		(4.83×)	(2.39×)	(2.67×)	(2.39×)	(1.98×)	(1.65×)
socPokec	1 632 803	22 301 964	32 557 458	42 947 031	52 831 618	65 281 896	83 896 509
(309.1 MB)		(13.65×)	(1.46×)	(1.32×)	(1.23×)	(1.18×)	(1.28×)
webGoogle	875 713	4 322 051	13 391 903	39 881 472	105 110 267	252 967 829	605 470 026
(59.5 MB)		(4.93×)	(3.1×)	(2.98×)	(2.63×)	(2.40×)	(2.39×)
webStan	281 903	1 992 636	11 329 473	78 757 781	620 210 972	4 859 571 082	34 690 796 481
(26.4 MB)		(7.07×)	(5.68×)	(6.95×)	(7.87×)	(7.83×)	(7.13×)
asSkit	1 696 415	11 095 298	28 769 868	148 834 439	1 183 885 507	9 759 000 981	73 142 566 591
(149.1 MB)		(6.54×)	(2.59×)	(5.17×)	(7.95×)	(8.24×)	(7.49×)
orkut	3 072 441	117 185 083	627 584 181	3 221 946 137	15 766 607 860	75 249 427 585	353 962 921 685
(1 687.8 MB)		(38.14×)	(5.35×)	(5.13×)	(4.89×)	(4.77×)	(4.70×)
webBerkStan	685 230	6 649 470	64 690 980	1 065 796 916	21 870 178 738	460 155 286 971	9 398 610 960 254
(89.4 MB)		(9.70×)	(9.73×)	(16.48×)	(20.52×)	(21.04×)	(20.42×)
comLiveJ	3 997 962	34 681 189	177 820 130	5 216 918 441	246 378 629 120	10 990 740 312 954	*445 377 238 737 777*
(501.6 MB)		(8.67×)	(5.12×)	(29.34×)	(47.22×)	(44.6×)	*(40.52×)*
socLiveJ1	4 847 571	42 851 237	285 730 264	9 933 532 019	467 429 836 174	20 703 476 954 640	*849 206 163 678 934*
(627.7 MB)		(8.84×)	(6.67×)	(34.7×)	(47.05×)	(44.29×)	*(41.01×)*
egoGplus	107 614	12 238 285	1 073 677 742	78 398 980 887	4 727 009 242 306	*242 781 609 271 577*	*11 381 161 386 691 540*
(538.5 MB)		(113.72×)	(87.73×)	(73.01×)	(60.29×)	*(51.36×)*	*(46.87×)*

Clique numbers in *italic* are approximations obtained by our color-based sampling algorithm, using 10 colors). Benchmarks are sorted by increasing q_7. For each benchmark, we also report in parentheses the storage in MB with no compression and the ratio q_{k+1}/q_k (which is half the average node degree for $k = 1$). Notice the large values of these ratios for benchmarks socLiveJ1, comLiveJ, webBerkStan, and egoGplus.

ACKNOWLEDGMENTS

We are indebted to Emilio Coppa for his suggestions about tuning the Hadoop configuration parameters and to the anonymous reviewers of this article for many useful comments.

REFERENCES

Foto N. Afrati, Dimitris Fotakis, and Jeffrey D. Ullman. 2013. Enumerating subgraph instances using MapReduce. In *Proc. 29th IEEE International Conference on Data Engineering (ICDE'13)*. 62–73.

Noga Alon, Raphael Yuster, and Uri Zwick. 1997. Finding and counting given length cycles. *Algorithmica* 17, 3 (1997), 209–223.

Apache Software Foundation. 2014. Apache Hadoop. Retrieved from http://hadoop.apache.org/.

Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. 2002. Reductions in streaming algorithms, with an application to counting triangles in graphs. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*. Society for Industrial and Applied Mathematics, 623–632.

Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (1999), 509–512.

Luca Becchetti, Carlos Castillo, Debora Donato, Stefano Leonardi, and Ricardo A. Baeza-Yates. 2006. Link-based characterization and detection of web spam. In *Proc. 2nd International Workshop on Adversarial Information Retrieval on the Web (AIRWeb'06)*. 1–8.

Andreas Björklund, Rasmus Pagh, Virginia Vassilevska Williams, and Uri Zwick. 2014. Listing triangles. In *Proc. 41st International Colloquium on Automata, Languages, and Programming (ICALP'14)*. 223–234.

Ilaria Bordino, Debora Donato, Aristides Gionis, and Stefano Leonardi. 2008. Mining large networks with subgraph counting. In *Proc. 8th IEEE International Conference on Data Mining (ICDM'08)*. IEEE, 737–742.

Stephen Borgatti, Martin Everett, and Linton Freeman. 2002. *UCINET for Windows: Software for Social Network Analysis*. Analytic Technologies, Harvard, MA.

Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, Alberto Marchetti-Spaccamela, and Christian Sohler. 2006. Counting triangles in data streams. In *Proc. 25th ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems*. ACM, 253–262.

Luciana S. Buriol, Gereon Frahling, Stefano Leonardi, and Christian Sohler. 2007. Estimating clustering indexes in data streams. In *Proc. 15th Annual European Symposium on Algorithms (ESA'07)*. Springer, 618–632.

Herman Chernoff. 1981. A note on an inequality involving the normal distribution. *Annals of Probability* 9, 3 (1981), 533–535.

Norishige Chiba and Takao Nishizeki. 1985. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing* 14, 1 (1985), 210–223.

Jeffrey Dean and Sanjay Ghemawat. 2004. MapReduce: Simplified data processing on large clusters. In *Proc. 6th Conference on Operating Systems Design & Implementation (OSDI'04)*. 10–10.

Camil Demetrescu and Irene Finocchi. 2007. Algorithms for data streams. In *Handbook of Applied Algorithms: Solving Scientific, Engineering, and Practical Problems*. John Wiley and Sons.

David Gibson, Ravi Kumar, and Andrew Tomkins. 2005. Discovering large dense subgraphs in massive graphs. In *Proc. 31st International Conference on Very Large Data Bases*. VLDB Endowment, 721–732.

Michael T. Goodrich, Nodari Sitchinava, and Qin Zhang. 2011. Sorting, searching, and simulation in the mapreduce framework. In *Proc. 22nd International Conference on Algorithms and Computation (ISAAC'11)*. 374–383.

András Hajnal and Endre Szemerédi. 1970. Proof of a conjecture of Erdös. *Combinatorial Theory and Its Applications* 2 (1970), 601–623.

Robert A. Hanneman and Mark Riddle. 2005. *Introduction to Social Network Methods*. University of California, Riverside, Riverside, CA.

Alon Itai and Michael Rodeh. 1978. Finding a minimum circuit in a graph. *SIAM Journal on Computing* 7, 4 (1978), 413–423.

Hossein Jowhari and Mohammad Ghodsi. 2005. New streaming algorithms for counting triangles in graphs. In *Proc. 11th Annual International Conference on Computing and Combinatorics (COCOON'05)*. 710–716.

Daniel M. Kane, Kurt Mehlhorn, Thomas Sauerwald, and He Sun. 2012. Counting arbitrary subgraphs in data streams. In *Proc. 39th International Colloquium on Automata, Languages and Programming (ICALP'12)*, Vol. 7392, 598–609.

Howard Karloff, Siddharth Suri, and Sergei Vassilvitskii. 2010. A model of computation for MapReduce. In *Proc. 21st Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*. 938–948.

Mihail N. Kolountzakis, Gary L. Miller, Richard Peng, and Charalampos E. Tsourakakis. 2012. Efficient triangle counting in large graphs via degree-based vertex partitioning. *Internet Mathematics* 8, 1–2 (2012), 161–185.

Jure Leskovec. 2014. SNAP graph library. Retrieved from http://snap.stanford.edu/.

Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. 2011. Approximate counting of cycles in streams. In *Proc. 19th Annual European Symposium on Algorithms (ESA'11)*. 677–688.

Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii, and Uri Alon. 2002. Network motifs: Simple building blocks of complex networks. *Science* 298, 5594 (2002), 824–827.

S. Muthukrishnan. 2005. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science* 1, 2 (2005), 117–236.

Mark Ortmann and Ulrik Brandes. 2014. Triangle listing algorithms: Back from the diversion. In *Proc. 16th Workshop on Algorithm Engineering and Experiments (ALENEX'14)*. 1–8.

Rasmus Pagh and Francesco Silvestri. 2014. The Input/Output complexity of triangle enumeration. In *Proc. 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems (PODS'14)*. ACM, 224–233.

Rasmus Pagh and Charalampos E. Tsourakakis. 2012. Colorful triangle counting and a mapreduce implementation. *Information Processing Letters* 112, 7 (2012), 277–281.

Ha-Myung Park, Francesco Silvestri, U. Kang, and Rasmus Pagh. 2014. MapReduce triangle enumeration with guarantees. In *Proc. 23rd ACM International Conference on Information and Knowledge Management (CIKM'14)*. 1739–1748.

Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. 2013. Counting and sampling triangles from a graph stream. *PVLDB* 6, 14 (2013), 1870–1881.

Andrea Pietracaprina, Geppino Pucci, Matteo Riondato, Francesco Silvestri, and Eli Upfal. 2012. Space-round tradeoffs for MapReduce computations. In *Proc. 26th ACM International Conference on Supercomputing (ICS'12)*. 235–244.

Anand Rajaraman and Jeffrey David Ullman. 2012. *Mining of Massive Datasets*. Cambridge University Press.

Barna Saha, Allison Hoch, Samir Khuller, Louiqa Raschid, and Xiao-Ning Zhang. 2010. Dense subgraphs with restrictions and applications to gene annotation graphs. In *Research in Computational Molecular Biology*. Springer, 456–472.

Thomas Schank and Dorothea Wagner. 2005. Approximating clustering coefficient and transitivity. *Journal on Graph Algorithms Applications* 9, 2 (2005), 265–275.

Francesco Silvestri. 2014. Subgraph enumeration in massive graphs. *CoRR* abs/1402.3444 (2014).

Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *Proc. 20th International Conference on World Wide Web (WWW'11)*. 607–614.

Charalampos E. Tsourakakis. 2014. A novel approach to finding near-cliques: The triangle-densest subgraph problem. *CoRR* abs/1405.1477 (2014).

Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. 2009. DOULION: Counting triangles in massive graphs with a coin. In *Proc. 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. 837–846.

Jeffrey Scott Vitter. 2008. Algorithms and data structures for external memory. *Foundations and Trends in Theoretical Computer Science* 2, 4 (2008), 305–474.

Received October 2014; revised April 2015; accepted June 2015