Motif Counting Beyond Five Nodes

MARCO BRESSAN and FLAVIO CHIERICHETTI, Sapienza University of Rome
RAVI KUMAR, Google Research
STEFANO LEUCCI, ETH Zürich
ALESSANDRO PANCONESI, Sapienza University of Rome

Counting graphlets is a well-studied problem in graph mining and social network analysis. Recently, several papers explored very simple and natural algorithms based on Monte Carlo sampling of Markov Chains (MC), and reported encouraging results. We show, perhaps surprisingly, that such algorithms are outperformed by color coding (CC) [2], a sophisticated algorithmic technique that we extend to the case of graphlet sampling and for which we prove strong statistical guarantees. Our computational experiments on graphs with millions of nodes show CC to be more accurate than MC; furthermore, we formally show that the mixing time of the MC approach is too high in general, even when the input graph has high conductance. All this comes at a price however. While MC is very efficient in terms of space, CC's memory requirements become demanding when the size of the input graph and that of the graphlets grow. And yet, our experiments show that CC can push the limits of the state-of-the-art, both in terms of the size of the input graph and of that of the graphlets.

CCS Concepts: • **Mathematics of computing** → *Graph enumeration*; *Graph algorithms*; • **Theory of computation** → *Random walks and Markov chains*; • **Information systems** → *Data mining; Web mining*;

Additional Key Words and Phrases: Motif counting, subgraph counting, color coding, graph mining

ACM Reference format:
Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2018. Motif Counting Beyond Five Nodes. *ACM Trans. Knowl. Discov. Data.* 12, 4, Article 48 (April 2018), 25 pages.
https://doi.org/10.1145/3186586

1 INTRODUCTION

Counting graphlets is a well-studied problem in graph mining and social-networks analysis [1, 3, 7, 8, 11, 14, 18, 20, 27–29, 32]. Given an input graph, the problem asks to count the frequencies of all induced connected subgraphs (called *graphlets*), up to isomorphism, of a certain size. This problem

A preliminary version of this article appeared in the *Proceedings of ACM WSDM'17* [6].
Part of the work was done while S. Leucci was at Sapienza University of Rome.
M. Bressan is supported in part by the ERC Starting Grant DMAP 680153 and by the SIR Grant RBSI14Q743. F. Chierichetti is supported in part by the ERC Starting Grant DMAP 680153, by a Google Focused Research Award, and by the SIR Grant RBSI14Q743. A. Panconesi is supported in part by a Google Focused Research Award and by the Sapienza inter-disciplinary research grant C26M15ALKP.
Authors' addresses: M. Bressan (corresponding author), F. Chierichetti, and A. Panconesi, Department of Computer Science, Sapienza University of Rome, via Salaria 113, 00198 Roma, Italy; emails: {bressan, flavio, ale}@di.uniroma1.it; R. Kumar, Google Research, 1600 Amphitheater Parkway, Mountain View, CA 94043, USA; email: ravi.k53@gmail.com; S. Leucci, Department of Computer Science, ETH Zürich, Universitätstrasse 6, 8092 Zürich, Switzerland; email: stefano.leucci@inf.ethz.ch.

ACM 1556-4681/2018/04-ART48 $15.00
https://doi.org/10.1145/3186586

is highly motivated in the context of studying behavioral and biological networks. Understanding the distribution of graphlets allows us to make key inferences about the structural properties of the underlying graph and the interaction of the nodes in the graph (e.g., [22]). It sheds light on the type of local structures that are present in the graph, which can be used for a myriad of analysis [3, 8, 16, 27–29]. For example, an extreme case of graphlets are three-node graphlets: counting triangles is a fundamental problem that has been repeatedly studied for the insights it can yield about the health of a network and also for pushing the boundaries of computation that is possible with large networks [13, 21, 23]. How the graphlets form in the first place and how they temporally evolve are semantically more actionable than the interpretation yielded by the mere evolution of nodes and edges.

Since the exact counting of graphlets can be computationally demanding, one usually settles for less ambitious goals. One such goal, and the one we pursue in this article, is frequency estimation: for each subgraph we want to estimate, as accurately as possible, its relative frequency among all subgraphs of the same size. Less ambitiously still, since the number of subgraphs of a given size grows exponentially, we will be interested in the problem of estimating the relative frequency of only the most frequent ones, say, those that appear at least a certain fraction of the time.

There have been two popular approaches to obtaining such estimates. The first is to use Markov Chain Monte Carlo (henceforth, MC). Given an input graph, consider a MC whose nodes (states) are the graphlets, and where two graphlets are connected if they differ by a node. After adding an appropriate number of self-loops to make the chain regular, it follows from standard facts that a random walk of length equal to (or greater than) the mixing time will stop at a uniform node (i.e., a graphlet). This gives a very simple and space-efficient way to sample the population of graphlets. Just repeat the walk independently a large number of times. Recently, this approach has been tried by several authors with encouraging results [3, 11, 20, 27]. However, for this type of approach to be statistically reliable the crucial question is: how long is the mixing time of this natural walk? To the best of our knowledge, this question has not been addressed in a principled manner.

A second approach to count graphlets (especially, trees) is to use color coding (CC), an elegant randomized algorithmic technique introduced in [2]. CC provides strong, provable statistical guarantees for the problem of approximating exact graphlet counts, from which the frequencies can be easily derived. From a computational point of view, perhaps its main drawback is its space requirement, that can quickly become insurmountable as the graphlet size grows. Furthermore, for its nice statistical guarantees to hold in the case of graphlets, one needs to run CC an exponential (in the subgraph size) number of times, which can be prohibitive. This heavy price must be paid if one needs precise estimates of exact counts. But what if one is just interested, as we are in this article, in estimating the frequency of the most common graphlets, can a linear upper bound be attained?

1.1 Our Contributions

In this work, we study MC and CC as the most viable methods for counting reasonably sized graphlets in massive graphs. Our goal is to understand and compare these methods from a practical point of view and en route show provable guarantees. Let n be the size of the input graph and let k be the size of graphlet. We are interested in input sizes that are currently considered challenging, i.e., in the range $n \geq 10^6$ and $k \geq 5$. As of today, efficient algorithms for counting the frequencies of *all* k-graphlets are known only for $k \leq 5$, and if one wants to scale to $k > 5$, then only results about special classes of graphlets are available (see Section 2).

Our first contribution is to study the mixing time of MC. We show that even if the input graph is well-mixing (as most social networks are) and even if there is one graphlet that appears more that 99% of the time, it is possible that MC will take $\Omega(n^{k-1})$ steps before sampling the most frequent graphlet just a single time! Note that this is not far from the naive $O(n^k)$ bound needed to

perform exact counting by an exhaustive enumeration. In particular, this shows that the mixing time of MC can be huge even when that of the input graph is very small, a somewhat counterintuitive statement. For large graphs and even modest values of k this readily implies that, unless one makes specific assumptions about the input graph and exploits them in the analysis, several MC approaches that have been used in the past effectively give no statistical guarantees. On the positive side, we show that the mixing time of MC is $O(n^2\Delta^{2k})$, a bound that can be useful for graphs of moderate size and small maximum degree Δ.

Our next contribution is to study the effectiveness of CC for graphlet counting. We show that the classic CC technique can be easily extended to sample induced graphlets almost uniformly at random in the graph, which enables us to estimate their frequency. This CC extension has two phases: a building phase and a sampling phase. The building phase, which is basically a dynamic program, is a time and space consuming process and is, in a sense, inevitable. The sampling phase, instead, is rather efficient and in practice much faster than MC. We then show that even a single run of CC, whose output can be seen as a large sample of the population of graphlets, gives reasonably good statistical guarantees. We remark that these bounds are still too weak to provide strong confidence in realistic situations. But, at the very least, they offer some evidence that by compounding the estimates obtained with a very few runs of CC one can get good, perhaps even strong, statistical guarantees. We view our result as an encouraging step along this direction, a line of research that we believe is an interesting one. We also describe an alternative extension of CC that requires less space in the building phase at the expense of sampling efficiency; this is the version we employ in our experiments.

Our last contribution is an experimental analysis with real-world graphs to compare the performance of MC and CC when the goal is estimating the frequencies of the most common graphlets. Our experiments are performed on the largest graphs used in recent work, whose sizes vary from small to several millions of nodes, on a commodity machine. The values of k we use range from 5 to 8. In a nutshell, it turns out that CC provides much better estimates, both on a sample-size budget and on a running-time budget. The drawback of CC is mainly its space complexity, which limits the size of the largest instances on which it can run. It is often the case that theoretical bounds are too coarse, while in actuality algorithms are much better behaved. Indeed, this seems to be the case with CC, for which we consistently observe that the estimate given by just a single run of CC is comparable to that obtained by averaging many runs, an outcome that is in line with our bounds. All in all, CC allowed us to reach beyond the current limits of graphlet counting. Notably, we were able to estimate the distribution of graphlets of size 6, 7, and 8 in graphs for which $k = 5$ was the state-of-the-art. As a rule of thumb, in our opinion CC remains preferable where accuracy guarantees are critical, while MC becomes competitive in the remaining cases.

1.2 Outline of the Article

The rest of the article is organised as follows. Section 2 discusses related work. Section 3 pins down the notation and definitions used throughout the article. Our results on MC are given in Section 4. The CC extension is described and analyzed in Section 5. Experimental results are presented in Section 6. The final remarks of Section 7 conclude the article.

2 RELATED WORK

The naive algorithm for counting the exact number of occurrences of all graphlets of size k in an n-node graph by enumeration takes $O(k^2n^k)$ time. Faster exact algorithms are known [10, 30], but their complexity remains $n^{\Theta(k)}$ and are infeasible in practice already for moderate values of n and k. Indeed, the problem is #W[1]-hard and thus unlikely to admit an $f(k)n^{O(1)}$-time algorithm [12]. For $k = 4$, a major improvement in exact counting is the combinatorial method of [1], which was

shown to scale to $n = 148$M. For $k = 5$, [18] managed graphs on up to $n = 4.8$M nodes by exploiting a decomposition of graphlets together with a degree-based orientation of the host graph. Note that these methods are tailored to $k = 4$ and 5 and it is not known how to extend them to $k > 5$. In this work, instead, we focus on techniques that can scale to $k > 5$, at least in principle, even if by only approximating the graphlet count.

In practice, counting (large) graphlets is often performed using approximate algorithms or heuristics. One heuristic approach is *path sampling*, a technique introduced in [14], which consists in sampling a path uniformly at random from G and checking the graphlet it induces; in this way one can sample graphlets having paths as spanning trees. In practice this has been shown to be effective for 4-graphlets [14], and can be adapted to $k = 5$ by using a set of sampling strategies for trees on five nodes [28, 29]. However, for $k > 5$ this method becomes overly intricate and can significantly suffer from rejection (the path is rejected if its k nodes are not distinct). Again, here we aim at counting graphlets of size $k > 5$ through approaches that give provable guarantees.

The first random walk-based algorithm, GUISE, was introduced in [3] and allowed the authors to collect, in just a few minutes, samples of graphlets of size $k = 4$ and 5 in graphs with up to four million nodes—a significant advancement of the state-of-the-art at the time. Further generalizations and refinement of this technique followed [8, 11, 19, 27], confirming its prowess at least for sampling graphlets of size $k \leq 5$ in graphs of a few million nodes (and $k = 6$ on one small graph). Unfortunately, for the two of these algorithms that are currently faster [8, 11] it is unclear how to extend the techniques to the case $k > 5$; in fact, the results available are for $k \leq 5$. The two techniques developed earlier [19, 27], although reportedly slower, are less sophisticated and can be easily employed for any value of k. However, no non-trivial bounds on the running time or number of samples needed to achieve a given accuracy are known for these methods. Obtaining such a result is part of the goals of our work.

The attractive bounds offered by CC has made it possible to push the task of estimating subgraph counts in the realm of graphs with millions of nodes. A first distributed algorithm based on CC, ParSE [32], was used to count seven different subgraphs of size k ranging from 4 to 10 in graphs with up to 20 million nodes. A subsequent distributed scalable implementation of CC, SCALA [20], allowed the authors to count on graphs with 1—2M nodes the number of non-induced paths and trees. Another recent effort to scale CC is [7]: using a distributed algorithm, the authors estimate the occurrences of 10 different subgraphs of treewidth 2 and size up to $k = 10$ nodes, in graphs of up to 2M nodes. While these encouraging results make clear that CC is a promising approach, they leave wide open the important question of estimating the distribution of *induced* subgraphs, aka graphlets. In this article, we show how CC fits the purpose—with almost no overhead.

Finally, a preliminary version of the present work [6] provided a first theoretical and experimental comparison between random walks and CC, suggesting the two are both viable, with CC winning on some large instances. In this article, we complement those by extensively showing that CC is the most promising technique for scaling graphlet counting to $k > 5$ in graphs with millions of nodes. We also improve the lower bound of Lemma 14 in [6] from $\Omega(n^{k-2})$ to $\Omega(n^{k-1})$, see Theorem 4.4.

3 PRELIMINARIES

A graph $G = (V, E)$ is composed of a set V of nodes and a set $E \subseteq \binom{V}{2}$ of edges.[1] In this article, we will assume the graph is connected and undirected. The *degree* of a node $v \in V$ is the number of nodes w such that $\{v, w\}$ is an edge of G: $\deg_G(v) = |\{w \mid \{v, w\} \in E\}|$. We use $\Delta(G)$ to denote the

[1]For a finite set S and an integer $0 \leq k \leq |S|$, we use $\binom{S}{k}$ to denote the set of k-subsets of S, i.e., $\binom{S}{k} = \{T \mid T \subseteq S, |T| = k\}$.

maximum degree of G: $\Delta(G) = \max_{v \in V} \deg_G(v)$. When G is obvious from the context, we simply use $\deg(\cdot)$ and Δ.

Graphlets. Given a graph $G = (V, E)$, and a subset $W \subseteq V$ of its nodes, we let the subgraph of G induced by W, or $G|W$, be the graph composed of the set of nodes W and the set of edges $E \cap \binom{W}{2}$. If $|W| = k$ and if $G|W$ is connected, then $G|W$ is a *k-graphlet* of G. We denote by $\mathcal{V}_k(G)$ the set of k-graphlets of G. Finally, if H is a connected undirected graph on k nodes, we say "the number of occurrences of H in G" to refer to the number of elements in $\mathcal{V}_k(G)$ that are isomorphic to H.

4 GRAPHLETS VIA RANDOM WALKS

In this section, we analyze the performance of graphlet-sampling algorithms based on random walks. Such algorithms are appealing for their simplicity and their encouraging empirical performance; on the other hand, however, they often come with weak theoretical guarantees in terms of n and k. Here, we aim at obtaining bounds on those guarantees. We focus on the two algorithms that are known to work for $k \geq 5$ [19, 27]; faster ones are known for $k \leq 5$ [8, 11] and we shortly discuss them at the end of the section. Both algorithms are based on a very natural approach—the simple random walk on the space of k-graphlet occurrences.

Recall that $\mathcal{V}_k(G)$ is the set of k-graphlet occurrences of G. Consider then a new graph whose nodes are $\mathcal{V}_k(G)$. There is an edge between two graphlets if and only if the node set of one can be obtained by the node set of the other by removing one node and adding another. More precisely,

$$\mathcal{E}_k(G) = \{\{X, Y\} \mid X, Y \in \mathcal{V}_k(G) \text{ and } |X \cap Y| = k - 1\}.$$

We let $\mathcal{G}_k(G)$ be the graph $\mathcal{G}_k(G) = (\mathcal{V}_k(G), \mathcal{E}_k(G))$. When G is clear from the context, we use the notation $\mathcal{G}_k = (\mathcal{V}_k, \mathcal{E}_k)$. We are interested in studying the simple random walk on $\mathcal{G}_k$. First of all, note that the connectedness of G implies the connectedness of $\mathcal{G}_k$. Furthermore, the MC associated to walk is aperiodic under mild conditions: it is sufficient that G is non-bipartite or there is $v \in G$ with $d_v \geq 3$, see e.g., Theorem 3.3 in [27]. We thus assume the chain is aperiodic. If we then let $d(H)$ denote the degree in $\mathcal{G}_k$ of a k-graphlet occurrence $H \in \mathcal{G}_k$, by standard MC theory, we have:

Observation 4.1. *The simple random walk on $\mathcal{G}_k$ converges toward the distribution where H has probability $p(H) = \frac{d(H)}{2|\mathcal{E}_k|}$.*

This observation can be used to build an unbiased estimator for the graphlet frequencies. To this end, one must ensure that each graphlet occurrence visited by the walk has the same weight in the final count. This can be achieved via rejection sampling (accepting H with probability $\frac{1}{d(H)}$) or via reweighting (counting H as a "fraction" $\frac{1}{d(H)}$ of a sample). These rejection and reweighting techniques are at the heart of all state-of-the-art graphlet-sampling algorithms based on random walks [8, 11, 19, 27]. The fundamental question now is how many steps are needed to reach stationarity, or more formally, what is the *mixing time* [17] of the random walk—the number of steps required to reach (within an ϵ-statistical error from) the stationary distribution. The remainder of this section is devoted to developing bounds on such a mixing time.

Let us start by recalling some standard notion. Given a set $W \subseteq V$ of nodes, the *volume* of W is $\text{vol}(W) = \sum_{v \in W} \deg(w)$. The *cut* induced by $W \subseteq V$ is equal to the number of edges that have exactly one endpoint in W, that is, $cut(W) = |\{e : |e \cap W| = 1\}|$. The conductance of a set of nodes $W \subseteq V$ is defined as $\phi(W) = \frac{\text{cut}(W)}{\text{vol}(W)}$. The *conductance* of G is defined as

$$\phi(G) = \min_{\substack{W \subseteq V \\ \text{vol}(W) \leq |E|}} \phi(W).$$

Consider the uniform random walk on G, where at each node $v \in V$, the next node to visit is chosen uniformly at random from among the neighbors of v. Cheeger's inequality [9] implies that the mixing time of the walk is between $\Omega(\phi(G)^{-1})$ and $O(\phi(G)^{-2} \log \frac{1}{\epsilon})$. A large conductance thus implies a small mixing time and vice versa.

Social graphs have been empirically observed to have small mixing times [16]. A natural question then is: can we give small upper bounds on the mixing time of $\mathcal{G}_k$ by using the fact that G has a small mixing time? We will show in Section 4.2 that, unfortunately, the answer to this question is negative in general: there are graphs G with very large (constant) conductance for which the corresponding $\mathcal{G}_k$ has a tiny conductance. Before addressing these lower bounds, in the next section, we give an *upper bound* on the mixing time of $\mathcal{G}_k$ that may be of use in low-degree graphs.

4.1 An Upper Bound on the Mixing Time

Theorem 4.2. *For any given G, we have $\phi(\mathcal{G}_k) \geq \Omega(n^{-1}\Delta^{-k})$, and thus the mixing time of the simple random walk on $\mathcal{G}_k$ is upper bounded by $O(n^2\Delta^{2k})$.*

Proof. Observe that, since G is connected, the degrees of nodes in $\mathcal{G}_k$ range from 1 to $k \cdot \Delta(G) = k\Delta$. We first upper bound the number of k-graphlets of G, i.e., $|\mathcal{V}|$. Note that any element $H \in \mathcal{V}$ with node set $\{v_1, \ldots, v_k\}$ can be mapped to a unique tuple $(v_1, \ldots, v_k)$, where for each $i = 2, \ldots, k$, node v_i neighbors some of $v_1, \ldots, v_{i-1}$. By bounding the number of such tuples, we can thus bound $|\mathcal{V}|$. Any such tuple can be constructed by choosing $v_1, v_2, \ldots, v_k$ in turn. We have n different choices for v_1. Once v_1 is chosen, we have at most Δ possible choices for v_2, since it must be a neighbor of v_1. Similarly, we have at most 2Δ choices for v_3, which must be a neighbor of v_1 or v_2, and so on till v_k for which we have at most $(k-1)\Delta$ choices. Therefore, $|\mathcal{V}| \leq n(k-1)! \cdot \Delta^{k-1}$.

Now, since for any $H \in \mathcal{V}_k$ the degree of H in $\mathcal{G}_k$ is at most $k\Delta$, the volume of any subset of nodes of $\mathcal{V}_k$ can be upper bounded by $k\Delta \cdot |\mathcal{V}_k| \leq k!n\Delta^k$. Furthermore, since $\mathcal{G}_k$ is connected, any non-empty and proper subset of nodes of $\mathcal{V}_k$ will have at least one edge in the cut. It follows that $\phi(\mathcal{G}_k) \geq \frac{1}{k!}n^{-1}\Delta^{-k}$. The upper bound on the mixing time of $\mathcal{G}_k$ then follows. □

4.2 Lower Bounds on the Mixing Time

We next show a mixing time lower bound by exhibiting a graph G with large conductance such that $\mathcal{G}_k$ has tiny conductance.

Definition 4.3. Let $k \in \mathbb{Z}^+$ be given. Let $\ell \in \mathbb{Z}^+$ be sufficiently large. Take ℓ disjoint paths of $2k$ nodes each; create two additional nodes a and b; for each path, add an edge between one of its endpoints and a, and an edge between its other endpoint and b. Let $G = (V, E)$ be the resulting graph.

Note that $|V| = 2\ell k + 2$; let $n = |V|$. One can easily see that the conductance of G is a constant, $\phi(G) = \Theta(1/k) = \Theta(1)$. We next prove that the conductance of $\mathcal{G}_k$ is tiny, which by Cheeger's inequality implies that its mixing time is huge, thus obtaining:

Theorem 4.4. *Let G be the graph of Definition 4.3. Then, the mixing time of G is $\Theta(1)$, and yet the mixing time of $\mathcal{G}_k$ is at least $\Omega(n^{k-1})$.*

Proof. Consider the closed ball S of radius k centered at a. Observe that it is (i) disjoint and (ii) isomorphic to the closed ball T of radius k centered at b. Now, consider the set X of k-graphlets that contain only nodes in S, and the set Y of k-graphlets that contain only nodes in T. By (ii), the subgraph that X induces on $\mathcal{G}_k$ will be isomorphic to the subgraph that Y induces on $\mathcal{G}_k$. Moreover, by (i), those two subgraphs will be disjoint. Thus, $\text{vol}(X) \leq \text{vol}(\mathcal{G}_k)/2 = |\mathcal{E}_k|$ and hence $\phi(\mathcal{G}'_k) \leq \phi(X)$. Now, $|X| = \Omega(n^{k-1})$ since there are $\Omega(n^{k-1})$ graphlets isomorphic to the star on k

nodes centered at a. Also, each such graphlet H satisfies $\deg_{\mathcal{G}_k}(H) = \Omega(n)$, since it is a neighbor of the $\Omega(k\ell)$ graphlets H' that share a and $k-2$ neighbors of a with H itself. Therefore, $\text{vol}(X) \geq |X| \deg_{\mathcal{G}_k}(H) = \Omega(n^k)$. Also, $\text{cut}(X) = \frac{n-2}{2k} \leq O(n)$. Therefore, we have $\phi(\mathcal{G}_k) \leq \phi(X) = \frac{\text{cut}(X)}{\text{vol}(X)} = O(n^{1-k})$. □

We observe that in time $O(n^k)$, one can just enumerate all the k-subsets of nodes of a graph of n nodes, check whether they form a k-graphlet and, if so, which graphlet do they form. As shown above, the random walk on $\mathcal{G}'_k$ has to run for at least $\Omega(n^{k-1})$ steps to guarantee any statistical significance of the sampled graphlet.

Next, we show that not only the random walk does not converge in $o(n^{k-1})$ steps for some graphs with constant conductance, but in fact there are constant-conductance graphs such that $o(n^{k-1})$ steps of the random walk are not even enough to see any copy of a graphlet that occurs an overwhelming fraction (i.e., $1 - o(1)$) of the times. This means we need $\Omega(n^{k-1})$ steps to see some occurrence of a graphlet appearing more than 99% of the times in the graph. We will consider a graph similar to the one in Definition 4.3.

Definition 4.5. Let $k \in \mathbb{Z}^+$ be given. Let $\ell \in \mathbb{Z}^+$ be sufficiently large. Take ℓ disjoint paths of $2k$ nodes each; create an additional node a and, for each path, add an edge between one of its endpoints and a. Construct a clique out of the ℓ other endpoints of the ℓ paths. Let $G = (V, E)$ be the resulting graph.

As before, let $n = |V| = 2\ell k + 1$ and one can show that $\phi(G) = \Theta(1/k) = \Theta(1)$. We prove:

Theorem 4.6. *Let G be the graph of Definition 4.5. Then, G has mixing time $\Theta(1)$, and the k-cliques are a $1 - o(1)$ fraction of the k-graphlets of G. However, if the random walk on $\mathcal{G}_k$ starts from any graphlet containing node a, with high probability it will require $\Omega(n^{k-1})$ steps to reach any k-clique.*

Proof. The number of k-cliques inside the clique of cardinality ℓ is equal to $\binom{\ell}{k} = \Theta(\ell^k) = \Theta(n^k)$. The number of graphlets that contain some node of the clique, and some node outside the clique is $\Theta(n^{k-1})$. The number of k-graphlets that contain a is no more than $O(n^{k-1})$. There are $\Theta(n)$ graphlets that do not contain nodes of the clique and a. Therefore, the number of k-clique graphlets is a $1 - O(1/n) = 1 - o(1)$ fraction of the total number of graphlets.

We now show that, if we start the random walk on $\mathcal{G}_k$ from any graphlet containing a, with high probability we will require $\Omega(n^{k-1})$ steps to reach any graphlet that does not contain a, and thus to reach any k-clique. Let us partition the set of graphlets that contain a into k zones $P_1, \ldots, P_k$, where a graphlet belongs to zone P_i if the maximum distance between one of its nodes and a is i. The starting graphlet is therefore in P_1. We aim to show that it takes $\Omega(n^{k-1})$ steps to reach P_k. Clearly, reaching some graphlet in P_k is necessary if we are to reach some k-clique. Consider now the walk between the P_i's. Observe that, if we are in P_i we can either remain there, or move to P_{i+1} (if $i < k$), or move back to P_{i-1} (if $i > 1$). However, the probability of moving to P_{i+1} is no more than $\frac{k^2}{\ell} = O(n^{-1})$. Moreover, if $i > 1$, then the probability of reaching, in $O(k)$ steps, P_{i-1} from P_i is at least $1 - O(1/n)$. The time required to reach P_k, then, is $\Omega(n^{k-1})$. □

4.3 Other Random Walk Techniques

We conclude by briefly discussing the random-walk algorithms of [8] and [11]. The algorithm of [8] performs a simple random walk on G, and then samples (with appropriate weights) all k-graphlets containing the last $k-1$ nodes visited by the walk if those are distinct. The idea is that the random walk on G mixes faster than that on $\mathcal{G}_k$, but the drawback is that we can only observe k-graphlets that contain a simple path of length $k-1$: for example, we can count cliques or cycles, but not stars. For $k \leq 5$, this is not a problem, as the stars are the only unobservable graphlets and their

frequency can be estimated from the frequencies of other graphlets [8]. However, the number of unobservable graphlets grows to 4 for $k = 6$ and to 33 for $k = 7$; moreover, those graphlets are among the top frequent ones in practice (see Section 6). The algorithm in [11] aims at overcoming both the high mixing time and the unobservability of graphlets. This is done by allowing the simple random walk on G to "waddle" so that it can visit the spanning tree of a graphlet H even if that spanning tree is not a path. Such an approach, however, requires to devise a "waddling strategy" for each graphlet that does not contain a spanning path and only for $k = 4$ such a strategy has been shown. Furthermore, the formal guarantees of such an approach, and in particular a bound on the variance of the estimator, seem difficult to pin down.

5 GRAPHLETS VIA COLOR CODING

In this section, we present two algorithms for graphlet counting that are based on CC, a powerful algorithmic technique introduced by Alon et al. [2]. Given the input graph $G = (V, E)$ and k, coloring coding first assigns uniformly and independently to each node of G a random label in $[k] := \{1, \ldots, k\}$, referred to as a *color*. The goal now is to count the number of *non-induced* trees of k nodes in G—called *treelets*—that are *colorful*, i.e., whose labels have no repetitions. This can be done efficiently by dynamic programming, thanks to the fact that treelets with disjoint set of labels must lie on disjoint set of nodes. Since a treelet is colorful with relatively low probability, one needs to repeat the coloring sufficiently many times in order to "hit" any given treelet.

In its original version, CC requires time $O(c^k \cdot |E|)$ and space $O(c^k \cdot |V|)$ for some $c > 1$, which has made it possible to push the task of estimating subgraph counts in the realm of graphs with millions of nodes. However, the existing algorithms only count subgraphs occurrences that are not induced, and that are either trees or "tree-like" in the sense of having small treewidth. We show that treelet counting can be extended to graphlet counting based on the observation that by counting treelets, we have counted (with high probability if repeated many times) all the spanning trees of every graphlet. To summarize, a good estimate of treelets can be translated into a good estimate of graphlets.

In this section, we first describe the two algorithms that are based on CC. The first algorithm is an extension of the algorithm of Alon et al. [2], and the second is a modification that uses less space at the expense of sampling speed. The latter algorithm is the one we use in our experiments, where we are limited by the amount of available memory. We then prove concentration on the number of colorful treelets produced by one run of (either of these) algorithms. We will use this to prove concentration on the number of colorful graphlets.

5.1 Algorithms

Here, we describe two algorithms based on CC that can count and sample colorful non-induced treelets uniformly at random. We then show how that suffices to sample colorful induced graphlets, as well. Both algorithms consist of a building phase and a sampling phase, and start with a coloring phase where each node $v \in V$ of G is assigned a color $c(v)$ chosen independently and uniform at random from $[k]$.

5.1.1 The First Algorithm (CC1). Our first algorithm, CC1, is as follows. In the building phase (see Algorithm 1), which is essentially the algorithm of Alon et al. [2], we start by creating for each node $v \in G$ a counter $C(T, S, v) = 1$, where T is the trivial graphlet on one node and S is the set of colors $\{c(v)\}$ containing just the color of v. This is the counter of the number of (non-induced) treelets isomorphic to T with color labels spanning S and rooted at v. Then, we perform a dynamic programming to count treelets of size $h = 2, \ldots, k$. For each h in turn, we consider each possible rooted tree T on $h \leq k$ nodes and each possible set $S \subseteq [k]$ with $|S| = h$. Then, for each node $v \in V$,

we can compute the number $C(T, S, v)$ of occurrences of (non-induced) treelets rooted at v that are isomorphic to T and whose colors span the set S, as follows. Split ideally T into two rooted subtrees T_1 and T_2 by removing any edge e incident to the root of T (the endpoints of the edge become the roots of the subtrees); it is easy to see that $C(T, S, v)$ satisfies the following relationship:

$$C(T, S, v) = \frac{1}{d} \sum_{(v,u)\in E} \sum_{\substack{S_1,\, S_2 \subset S \\ S_1 \cap S_2 = \emptyset}} C(T_1, S_1, v) \cdot C(T_2, S_2, u), \tag{1}$$

where d is a normalization constant that is equal to the number of rooted trees isomorphic to T_2 among the subtrees rooted in the children of the root T. To compute $C(T, S, v)$ one then just sweeps over all edges uv of G, combining the counters of u and v. The correctness and complexity of this construction are proved in [2].

ALGORITHM 1: CC1-Build

input: graph G, graphlet size k

1 **for** v *in* G **do**
2 $\quad c(v)$ = color drawn u.a.r. from $[k]$
3 $\quad C((\{v\}, \emptyset), \{c(v)\}, v) = 1$
4 **for** $h = 2$ *to* k **do**
5 $\quad$ **for** v *in* G **do**
6 $\quad\quad$ **for** *each* $T : |T| = h$ **do**
7 $\quad\quad\quad$ **for** *each* $S \in \binom{[k]}{h}$ **do**
8 $\quad\quad\quad\quad C(T, S, v) = d^{-1} \sum_{(v,u)\in E} \sum_{\substack{S_1, S_2 \subset S \\ S_1 \cap S_2 = \emptyset}} C(T_1, S_1, v) \cdot C(T_2, S_2, u)$
9 **for** v *in* G **do**
10 $\quad$ build rng() with $\Pr[T, v] \propto C(T, [k], v)$ **for** *each* $T : |T| = h$ **do**
11 $\quad\quad$ **for** *each* $S \in \binom{[k]}{h}$ **do**
12 $\quad\quad\quad$ **for** *each* $S_1 \subset S$ **do**
13 $\quad\quad\quad\quad$ build rng(T_1, T_2, S, v) with $\Pr[S_1, S_2] \propto C(T_1, S_1, v) \sum_{u:uv\in E} C(T_2, S_2, u)$
14 $\quad\quad\quad\quad$ build rng(T_2, S_2, v) with $\Pr[u] \propto C(T_2, S_2, u)$

In the sampling phase (see Algorithm 2), we use the counters $C(T, S, v)$ to sample a colorful treelet uniformly at random. First, we randomly choose a node v of G and a treelet T on k nodes with probability proportional to the overall number of occurrences of treelets isomorphic to T rooted at v, i.e., to $C(T, [k], v)$. We then choose one of the $C(T, S, v)$ treelets rooted at v that are isomorphic to T and are colored with the colors in S (in this first step, $S = [k]$). To this aim, we split T into T_1 and T_2 as described above. Then, we select a pair of color subsets S_1 and $S_2 = S \setminus S_1$, with size $|S_1| = |T_1|$ and $|S_2| = |T_2|$, with probability proportional the number of T's rooted at v that are formed by T_1 and T_2 colored with S_1 and S_2; that is, proportional to $C(T_1, S_1, v) \cdot \sum_{u:(u,v)\in E} C(T_2, S_2, u)$. Next, we choose the neighbor u where to root T_2 with probability proportional to $C(T_2, S_2, u)$. Then, we recursively sample one of the $C(T_1, S_1, v)$ (resp. $C(T_2, S_2, v)$) treelets isomorphic to T_1 (resp. T_2) and colored with the colors in S_1 (resp. S_2) from v (resp. u). It is immediate to verify that this procedure yields a colorful treelet occurrence chosen u.a.r. among all those in G.

To reduce the complexity of the sampling phase, in CC2 we pre-build the random number generators (lines 9–14), as follows. First, for any possible treelet on k nodes and any $v \in G$, we build a generator returning the pair (T, v) with probability proportional to $C(T, [k], v)$. Then, for each node $v \in G$, we build $O(c^k)$ generators to draw the pairs of label sets and to draw a neighbor of

u. Such generators can be built in time and space linear in the size of the alphabet, and produce a sample in $O(1)$ time [26]. Thus, in our case they take overall $O(c^k|E|)$ time and space.

ALGORITHM 2: CC1-Sample

input: (T, S, v) or Null
output: colorful k-treelet chosen u.a.r. from those isomorphic to T with labels S rooted at v

```
if input = Null then
    (T, v) = rng()
    S = [k]
if |T| = 1 then
    return ({v}, ∅), {c(v)}
decompose T into T1 + e + T2
(S1, S2) = rng(T1, T2, S, v)
u = rng(T2, S2, v)
H1 = CC1-Sample(T1, S1, v)
H2 = CC1-Sample(T2, S2, u)
return H1 + vu + H2
```

From the algorithms, one can immediately obtain the following complexity bounds:

Theorem 5.1. *CC1-Build takes time and space $O(c^k|E|)$ for some $c > 0$. CC1-Sample takes time $O(k)$.*

5.1.2 The Second Algorithm (CC2). Our second algorithm, CC2 is a simple variant of CC1 that saves memory at the expense of sampling speed. In CC2, we do not precompute the random number generators that allow to (recursively) select the subtrees T_1, T_2 and the subsets of labels S_1, S_2 in time $O(1)$. Instead, we create the distributions of T_1, T_2 and S_1, S_2 at sampling time starting from the counters $C(\cdots)$. This requires to sum the counters $C(\cdots)$ over all neighbors of v, but since the degrees of the graphs we deal with are not too large, the impact on the sampling time is hopefully small. The advantage is that the space complexity of CC2 becomes $O(c^k|V|)$. This allows us to reduce the overall memory footprint of the algorithm—a determining factor in practice, since CC is typically limited by memory. Formally, one can prove:

Theorem 5.2. *CC2-Build takes time $O(c^k|E|)$ and space $O(c^k|V|)$. CC2-Sample generates a treelet sample T in time $O(c^k \sum_{v \in T} d_v)$.*

Finally, in CC2 we store only the positive counters $C(\cdots)$. In this way, the amount of memory required by CC2 is actually dictated by the number of different colored treelets that are in the graph. This can be much smaller than $c^k|V|$. For this reason, in practice the memory footprint of CC2 is not strictly proportional to the size of the graph, and on some larger graphs, we could reach larger values of k (see Section 6).

5.1.3 From Treelets to Graphlets. Once we can sample an occurrence of a colorful treelet on k nodes uniformly at random from the set of all colorful treelet occurrences in G, it is possible to also sample a colorful graphlet H by noticing that a graphlet contributes $k\sigma(H)$ to the treelet count, where $\sigma(H)$ is the number of spanning trees of H (the factor k comes from the fact that a single colorful tree contributes k to the count). Hence, to sample a colorful graphlet uniformly at random one can proceed as follows:

(i) sample an occurrence T of a treelet on k nodes from G
(ii) consider the graphlet H induced by the nodes of T, and
(iii) reject H with probability $1 - \frac{1}{\sigma(H)}$.

If the goal is to obtain an unbiased estimate of the graphlets frequency, at step (iii) instead of rejection sampling, we can simply add to the graphlet count the quantity $\frac{1}{\sigma(H)}$. This never increases the variance of the resulting estimator and, depending on the distribution, can significantly decrease it. We use this approach in our implementation. The value $\sigma(H)$ can be (pre)computed for any given H in time $O(k^\omega)$, where ω is the matrix multiplication exponent, e.g., via Kirchhoff's Matrix-Tree Theorem [24]. In our case, we compute $\sigma(H)$ the first time H is sampled, caching it for later reuse.

5.2 Concentration of Colored Graphlets

In this section, we prove concentration bounds on the number of colorful graphlets produced by CC1 and CC2 and, in fact, by any random uniform coloring of the nodes of G. We assume $k \geq 3$ and we denote by $g = |\mathcal{V}_k(G)|$ the total number of k-graphlets in G. Our goal is to show that, with high probability, the coloring preserves the distribution of graphlets. More formally, consider a subset $\mathcal{S} \subseteq \mathcal{V}_k(G)$ of the k-graphlets of G; for instance, the set of all k-cliques of G. The expected number of such graphlets that are colorful is $\sum_{H \in \mathcal{S}} \Pr[H \text{ is colorful}] = |\mathcal{S}| \frac{k!}{k^k}$; so we could recover $|\mathcal{S}|$ from an accurate estimate of such a number. Unfortunately, the actual number of colorful graphlets can fall far from $|\mathcal{S}| \frac{k!}{k^k}$. Indeed, there may be strong correlations between the colorings of different graphlets in $\mathcal{S}$; for instance, there can be $\Theta(n^{k-2})$ graphlets sharing two nodes v, v' of G, and if v and v' have the same color, then all those $\Theta(n^{k-2})$ will be simultaneously uncolorful. However, we can show that, if $\mathcal{S}$ is large enough, then there is concentration. Our main result is the following:

Theorem 5.3. *Consider any $\mathcal{S} \subseteq \mathcal{V}_k(G)$, and let $Z_\mathcal{S}$ be the random variable counting the number of $H \in \mathcal{S}$ that are made colorful by a coloring of G. Let $s = |\mathcal{S}|$ and $\mu_\mathcal{S} = \mathbb{E}[Z_\mathcal{S}]$. Then, for any $\epsilon > 0$:*

$$\Pr\Big[\, |Z_\mathcal{S} - \mu_\mathcal{S}| > \epsilon\, \mu_\mathcal{S} \,\Big] \leq e^{-\Omega(\epsilon^2 s^{1-1/k}/g^{1-2/k})}, \tag{2}$$

where the $\Omega(\cdot)$ notation hides factors that depend on k but not on g and s.

The exponent of the bound is in $\Omega(1)$ as long as $s \in \Omega(g^{1-\frac{1}{k-1}})$, i.e., as long as $\mathcal{S}$ is relatively large w.r.t. the total number of graphlets of G. For instance, when counting the most frequent graphlets (say, those appearing at least a 1% of the times) we are looking at $s \in \Omega(g)$, which is in that range. The proof of Theorem 5.3 is rather technical and can be skipped without impairing the understanding of the rest of the article; the interested reader can find it in Section 5.3.

We next prove that the bound of Theorem 5.3 is tight. Formally:

Theorem 5.4. *There exist arbitrarily large graphs G with a subset of k-graphlets $\mathcal{S}$ such that $|\mathcal{S}| = \Omega(g^{1-\frac{1}{k-1}})$ and $\Pr[Z_\mathcal{S} = 0] \geq \frac{1}{k} = \Omega(1)$.*

Proof. Consider G formed by a star on $n-1$ nodes and an additional node u' attached to a leaf node u of the star. There are $\binom{n-1}{k-1} = \Omega(n^{k-1})$ graphlets isomorphic to stars, thus $g = \Omega(n^{k-1})$. The set $\mathcal{S}$ of graphlets not spanned by stars contains all and only those graphlets containing both u and u', which are $\binom{n-2}{k-2} = \Omega(n^{k-2}) = \Omega(g^{1-\frac{1}{k-1}})$. The probability that *all* such graphlets are not colorful, and thus that $Z_\mathcal{S} = 0$, is at least $\Pr[u \text{ and } u' \text{ have the same color}] \geq \frac{1}{k}$. □

By Theorem 5.3, the distribution of colorful graphlets (which can be sampled via our algorithms CC1 and CC2) closely matches the overall distribution of graphlets, at least for graphlets that occur often enough. A formal statement is the following:

Corollary 5.5. *Let $\mu_H \in [0, 1]$ be the fraction of graphlet occurrences of G that are isomorphic to H, and let S be a graphlet drawn uniformly at random from the set of colorful graphlets of G. Then,*

for any $\epsilon > 0$, *with probability* $1 - e^{-\Omega(\epsilon^2 g^{1/k})}$:

$$\Pr\left[S \text{ is an occurrence of } H\right] \in \left[\mu_H - \frac{2\epsilon}{1+\epsilon}, \mu_H + \frac{2\epsilon}{1-\epsilon}\right]. \tag{3}$$

Proof. Let $Z_{\mathcal{S}}$ be the number of colorful elements of $\mathcal{S} = \mathcal{S}_H$, the set of graphlets isomorphic to H. Let Z be the total number of colorful graphlets in G. What we are bounding is the probability that $Z_{\mathcal{S}}/Z$ is too far from $\mathbb{E}[Z_{\mathcal{S}}]/\mathbb{E}[Z] = \mu_H$. Suppose that $\mathbb{E}[Z_{\mathcal{S}}] \geq \frac{1}{2}\mathbb{E}[Z]$. Then, $\mathbb{E}[Z_{\mathcal{S}}], \mathbb{E}[Z] \in \Theta(g)$ and by the bounds of Theorem 5.3 with probability $1 - e^{-\Omega(\epsilon^2 g^{1/k})}$, we have $Z_{\mathcal{S}} \in \mathbb{E}[Z_{\mathcal{S}}](1 \pm \epsilon)$ and $Z \in \mathbb{E}[Z_{\mathcal{S}}](1 \pm \epsilon)$. By standard calculations, it follows that $\mathbb{E}[Z_{\mathcal{S}}]/\mathbb{E}[Z] \in \left[\mu_H - \frac{2\epsilon}{1+\epsilon}, \mu_H + \frac{2\epsilon}{1-\epsilon}\right]$. If $\mathbb{E}[Z_{\mathcal{S}}] < \frac{1}{2}\mathbb{E}[Z]$, then one can obtain the bound by applying the argument above to $Z - Z_{\mathcal{S}}$. □

Finally, if one creates $\lambda \geq 1$ random colorings of G and takes the average counts of each graphlet, one gets the following improved concentration bound.

Corollary 5.6. *If we consider* λ *independent colorings of* G *and let* $Z_{\mathcal{S}} = \lambda^{-1} \sum_{i=1}^{\lambda} Z^i_{\mathcal{S}}$ *where* $Z^i_{\mathcal{S}}$ *counts the number of colorful graphlets of* $\mathcal{S}$ *in the* i*th coloring, then the bound of Theorem 5.3 becomes*:

$$\Pr\left[\, |Z_{\mathcal{S}} - \mu_{\mathcal{S}}| > \epsilon \cdot \mu_{\mathcal{S}} \,\right] \leq e^{-\Omega(\lambda \epsilon^2 s^{1-1/k}/g^{1-2/k})}.$$

Proof. It is sufficient to modify slightly the proof of Theorem 5.3 (see below). Consider λ martingale sequences, each one like the one used in the proof, associated to λ independent colorings of G. Juxtapose them to obtain a single martingale sequence of length λn, whose expectation is $\mu'_{\mathcal{S}} = \lambda \mu_{\mathcal{S}}$. We can then apply the Azuma–Hoeffding inequality to this martingale sequence. The denominator of the bound's exponent becomes $2\lambda \sum_{i=1}^{n} c_i^2$, where the c_i's are the same of the proof of Theorem 5.3. Into the numerator of the exponent, now we plug $t = \epsilon \mu'_{\mathcal{S}} = \epsilon \lambda \mu_{\mathcal{S}}$. Since t is squared at the numerator, one can check that the bound gains a factor λ at the exponent. □

5.3 Proof of Theorem 5.3

The key steps are as follows. We assume the nodes of G are colored in non-increasing order of number of graphlets they appear in (clearly any order is equivalent). We then consider the (Doob) martingale that counts the expected number of colorful graphlets in $\mathcal{S}$ given the colors assigned to the first i nodes, for each $i = 1, \ldots, n$. By applying the method of bounded differences, we get a concentration inequality whose exponent's denominator has one term for each node of G, telling how much the martingale can oscillate when we color that node. Thanks to the ordering of the nodes, bounding the vast majority of terms due to nodes that appear in a few graphlets is relatively easy; less so for the other terms, that also depend on how many graphlets can be shared by two nodes. This requires us to prove that two nodes cannot simultaneously appear in too many graphlets (one of the two must appear in asymptotically more).

Let us start with some notation. For $i = 1, \ldots, n$, let $X_i \in [k]$ be the random variable denoting the color of node i. For $j = 1, \ldots, s$ let $Y_j \in \{0, 1\}$ be the indicator random variable of the event that the jth graphlet of $\mathcal{S}$ is colorful, and let $Z = Z_{\mathcal{S}} = \sum_{j=1}^{s} Y_j$ be the total number of colorful graphlets of $\mathcal{S}$. Finally, for $i = 1, \ldots, n$ let $Z_i = \mathbb{E}[Z|X_1, \ldots, X_i]$ be the expectation of Z as a function of the colors assigned to the first i nodes, and let $Z_0 = \mathbb{E}[Z]$. The sequence $Z_0, \ldots, Z_n$ is a Doob martingale with respect to $X_1, \ldots, X_n$, and Azuma's inequality implies

$$\Pr\left[\, |Z - \mathbb{E}[Z]| > t \,\right] < 2e^{-\frac{t^2}{2\sum_{i=1}^{n} c_i^2}}, \tag{4}$$

whenever $|Z_i - Z_{i-1}| \leq c_i$.

Let now $g_{\mathcal{S}}(u) = |\{H \in \mathcal{S} : u \in H\}|$ be the number of graphlets of $\mathcal{S}$ in which u appears. The rest of the proof is devoted to showing that, if the nodes of G are sorted in non-increasing order of

$g_{\mathcal{S}}(i)$, then $\sum_{i=1}^{n} c_i^2 = O(s^{1+\frac{1}{k}}/g^{1-\frac{2}{k}})$. Together with Equation (4), and since $\mu_{\mathcal{S}} = \frac{k!}{k^k}s = \Theta(s)$, this implies the theorem's claim for $t = \epsilon\mu_{\mathcal{S}}$. Start by breaking $\sum_{i=1}^{n} c_i^2$ in two parts:

$$\sum_{i=1}^{n} c_i^2 = \sum_{i=1}^{\ell} c_i^2 + \sum_{i=\ell+1}^{n} c_i^2, \tag{5}$$

for some ℓ to be chosen later. Note that $c_i \leq g_{\mathcal{S}}(i)$: the conditioning on X_i can alter, by at most 1, the expectation of only those Y_j associated to graphlets containing i. Also, $g_{\mathcal{S}}(i) \leq \frac{1}{i}\sum_{u=1}^{n} g_{\mathcal{S}}(u)$ by the ordering of the nodes. Finally, $\sum_{u=1}^{n} g_{\mathcal{S}}(u) = ks = O(s)$, and thus $g_{\mathcal{S}}(i) = O(\frac{s}{i})$. Hence, the second term in Equation (5) can be bounded as

$$\sum_{i=\ell+1}^{n} c_i^2 \leq \sum_{i=\ell+1}^{n} g_{\mathcal{S}}(i)^2 = \sum_{i=\ell+1}^{n} O\left(\frac{s}{i}\right)^2 = O\left(\frac{s^2}{\ell}\right). \tag{6}$$

The rest of the proof focuses on bounding $\sum_{i=1}^{\ell} c_i^2$. First of all notice, that if the graphlet associated to Y does not contain i or does not contain one among $1, \ldots, i-1$, then $\mathbb{E}[Y|X_1, \ldots, X_i] = \mathbb{E}[Y|X_1, \ldots, X_{i-1}]$. Therefore, c_i is bounded by the number of graphlets of $\mathcal{S}$ that contain both i and at least one of $1, \ldots, i-1$. Let then $g_{\mathcal{S}}(i,j) = |\{H \in \mathcal{S} : i, j \in H\}|$ and let $g(i,j) = g_{\mathcal{G}_k}(i,j)$. Clearly, $g_{\mathcal{S}}(i,j) \leq g(i,j)$. Thus, we have

$$c_i \leq \sum_{j=1}^{i-1} g_{\mathcal{S}}(i,j) \leq \sum_{j=1}^{i-1} g(i,j). \tag{7}$$

Assume now that $g(i,j) = O((g(i)+g(j))^{\frac{k-2}{k-1}})$, which we indeed prove later. Therefore, the right-hand side of the equation above is $\sum_{j=1}^{i-1} g(i,j) = O(\sum_{j=1}^{i-1}(g(i)+g(j))^{\frac{k-2}{k-1}})$. Now by the ordering of nodes $g(i)+g(j) \leq 2g(j) = O(g/j)$, hence

$$c_i \leq O\left(\sum_{j=1}^{i-1}(g(i)+g(j))^{\frac{k-2}{k-1}}\right) = O\left(\sum_{j=1}^{i-1}(g/j)^{\frac{k-2}{k-1}}\right) = O\left(g^{1-\frac{1}{k-1}} i^{\frac{1}{k-1}}\right), \tag{8}$$

with the last equality following from standard analysis. By using this bound in $\sum_{i=1}^{\ell} c_i^2$, we get

$$\sum_{i=1}^{\ell} c_i^2 = O\left(\sum_{i=1}^{\ell} g^{2-\frac{2}{k-1}} i^{\frac{2}{k-1}}\right) = O\left(g^{2-\frac{2}{k-1}} \ell^{1+\frac{2}{k-1}}\right). \tag{9}$$

Finally, by setting $\ell = s^{1-\frac{1}{k}} g^{\frac{2}{k}-1}$ in both Equations (6) and (9), we obtain $\sum_{i=1}^{n} c_i^2 = O\left(s^{1+\frac{1}{k}} g^{1-\frac{2}{k}}\right)$ as desired.

It only remains to prove:

Lemma 5.7. *For any $u, v \in G$ it holds $g(u,v) = O\left((g(u)+g(v))^{\frac{k-2}{k-1}}\right)$.*

Proof. For any $k \geq 1$ let $\mathcal{H}_k(u)$ denote the set of graphlets of size k of G that contain u (so $\mathcal{H}_1(u)$ contains only the trivial graphlet formed by u alone). For any graphlet occurrence H, let d_H be the sum of the degrees of its nodes in G; i.e., if $u_1, \ldots, u_k$ are the nodes of H then $d_H = \sum_{i=1}^{k} d_{u_i}$. To avoid ambiguities w.r.t. k, we use $g_k(u)$ instead of $g(u)$ to denote $|\mathcal{H}_k(u)|$. Note that $g_k(u) > 0$ for all k since G is connected by hypothesis, so we can safely employ Landau notation.

We start by proving that $g_k(u)$ is proportional to the sum of the degrees of the graphlets of size $k-1$ containing u:

$$\sum_{\mathcal{H}_{k-1}(u)} \frac{1}{k}\left\lceil \frac{1}{k-1}\left(d_H - 2\binom{k-1}{2}\right)\right\rceil \leq g_k(u) \leq \sum_{\mathcal{H}_{k-1}(u)} d_H.$$

The upper bound follows immediately by noting that any element of $\mathcal{H}_k(u)$ can be obtained by adding to some $H \in \mathcal{H}_{k-1}(u)$ one of the neighbors of its nodes, and those neighbors are at most d_H. Consider now any $H \in \mathcal{H}_{k-1}(u)$. Since the arcs within H are at most $\binom{k-1}{2}$, and since G is connected, there must be at least $d_H - 2\binom{k-1}{2} > 0$ arcs between H and $G \setminus H$. These arcs then lead to at least $\lceil \frac{1}{k-1}(d_H - 2\binom{k-1}{2}) \rceil$ distinct nodes, each of which can be added to obtain an element of $\mathcal{H}_k(u)$. Any element of $\mathcal{H}_k(u)$ can be obtained in this way, and from at most k elements of $\mathcal{H}_{k-1}(u)$. The lower bound then follows by summing $\lceil \frac{1}{k-1}(d_H - 2\binom{k-1}{2}) \rceil$ over all $H \in \mathcal{H}_{k-1}(u)$ and dividing by k.

We can now prove the following crucial fact:

$$g_k(u) = \Omega\left(g_{k-1}(u)^{\frac{k-1}{k-2}}\right). \tag{10}$$

The proof is by induction on k. The claim holds trivially for $k = 2$. Let us assume it holds for some $k \geq 2$ and focus on proving it for $k+1$. Since $g_{k+1}(u) = \Omega(\sum_{H' \in \mathcal{H}_k(u)} d_{H'})$, we will show the right-hand side is in $\Omega(g_k(u)^{\frac{k}{k-1}})$. Recall that from any $H \in \mathcal{H}_{k-1}(u)$ and its neighbors in G one can create $\lceil \frac{1}{k-1}(d_H - 2\binom{k-1}{2}) \rceil = \Omega(d_H)$ graphlets of $\mathcal{H}_k(u)$; each such graphlet H' includes all the nodes of H, hence has degree $d_{H'} \geq d_H$, and may be obtained from at most k distinct graphlets H. Therefore,

$$\sum_{\mathcal{H}_k(u)} d_{H'} \geq \frac{1}{k} \sum_{\mathcal{H}_{k-1}(u)} \Omega(d_H) \cdot d_H = \Omega\left(\sum_{\mathcal{H}_{k-1}(u)} d_H^2\right). \tag{11}$$

Now,

$$\sum_{\mathcal{H}_{k-1}(u)} d_H^2 \geq \frac{1}{g_{k-1}(u)} \left(\sum_{\mathcal{H}_{k-1}(u)} d_H\right)^2 = \Omega\left(\frac{g_k(u)^2}{g_{k-1}(u)}\right), \tag{12}$$

where the first inequality follows from convexity and the second from $\sum_{\mathcal{H}_{k-1}(u)} d_H = \Omega(g_k(u))$. Now by the inductive hypothesis $g_{k-1}(u) = O(g_k(u)^{\frac{k-2}{k-1}})$, which used in the denominator of the right-hand side proves Equation (10).

We can now conclude the proof of Lemma 5.7. First of all note that any graphlet of size k containing both u and v is the union of (the sets of nodes of) two smaller graphlets: one of size h containing u (but possibly not v), and one of size $k-h$ containing v (but possibly not u), for some $h \in \{1, \ldots, k-1\}$. It follows that:

$$g(u,v) \leq \sum_{h=1}^{k-1} g_h(u) g_{k-h}(v). \tag{13}$$

Since k is a constant, we can thus choose $h \in \{1, \ldots, k-1\}$ such that $g_h(u)g_{k-h}(v) \geq \Omega(g(u,v))$. If $h = 1$, since $g_1(u) = 1$, then $g_{k-1}(v) = \Omega(g(u,v))$, and $g_k(v) = \Omega((g(u,v))^{\frac{k-1}{k-2}})$ by Equation (10). Similarly, if $h = k-1$, we obtain $g_k(u) = \Omega((g(u,v))^{\frac{k-1}{k-2}})$.

Assume then $h \in \{2, \ldots, k-2\}$. If $g_h(u) = \Omega(g(u,v)^{\frac{h-1}{k-2}})$, by Equation (10) $g_k(u) = \Omega(g_h(u)^{\frac{k-1}{h-1}}) = \Omega(g(u,v)^{\frac{k-1}{k-2}})$. Otherwise $g_{k-h}(v) = \Omega(g(u,v)/g_h(u)) = \Omega(g(u,v)^{\frac{k-h-1}{k-2}})$, but then Equation (10) implies $g_k(v) = \Omega(g_{k-h}(v)^{\frac{k-1}{k-h-1}}) = \Omega(g(u,v)^{\frac{k-1}{k-2}})$. In any case, $g(u) + g(v) = g_k(u) + g_k(v) = \Omega(g(u,v)^{\frac{k-1}{k-2}})$, which concludes the proof. □

6 EXPERIMENTS

In this section, we compare the practical performance of CC2 and of its main competitor, the Pairwise Subgraph Random walk (PSRW) of [27]—as discussed in Section 4.3, this is the only

Table 1. The Graph Datasets Used in Our Experiments (Largest Connected Component Only)

Name	Nodes	Edges	k	Source dataset
WordAssoc	10, 6K	63, 8K	8	LAW, wordassociation-2011
Facebook	63, 4K	0, 8M	8	MPI-SWS, Facebook New Orleans
Amazon	0, 3M	0, 9M	8	SNAP, com-amazon.ungraph
DBLP	0, 3M	1, 0M	7	SNAP, com-dblp.ungraph
Yelp	0, 2M	1, 3M	7	YLP, Yelp
Road-PA	1, 1M	1, 5M	6	SNAP, roadnet-PA
Road-CA	2, 0M	2, 8M	7	SNAP, roadnet-CA
BerkStan	0, 7M	6, 6M	5	SNAP, web-BerkStan
Skitter	1, 7M	11, 1M	5	SNAP, as-skitter
Patents	3, 8M	16, 5M	6	SNAP, cit-Patents
Road-US	24, 9M	28, 9M	see note	NDR, inf-road-usa
LiveJournal	5, 4M	49, 5M	6	LAW, ljournal-2008
Hollywood	1, 9M	114, 3M	6	LAW, hollywood-2009
Twitter	41, 7M	117, 2M	see note	LAW, twitter-2010
Orkut	3, 1M	223, 5M	5	MPI-SWS, orkut-2007

Here, k is the largest graphlet size for which we could successfully run algorithm CC2 within the memory resource limits of our machines. LAW: http://law.di.unimi.it/, [4, 5]. MPI-SWS: http://socialnetworks.mpi-sws.org/data-wosn2009.html, [25]. SNAP: https://snap.stanford.edu/, [15, 31]. YLP: https://www.yelp.com/dataset_challenge/.

random-walk technique available that can be employed for $k > 5$. We note that PSRW (like other similar random walk techniques) has been developed with the primary goal of minimizing the number of nodes of G visited by the walk; in the present article, however, we investigate it in terms of samples taken, running time, and accuracy.

6.1 Setup

We implemented CC2 and PSRW in a multi-threading fashion. For CC2, in the building phase at each level of the dynamic program each thread takes care of merging a subset of counters, while in the sampling phase each thread executes the sampling algorithm independently. For PSRW, each thread runs a single random walk independently. We note that PSRW's running time is dominated by enumerating the possible transitions from the current graphlet occurrence H. We implemented this routine by intersecting, for each $u \in H$, the set of neighbors of the connected components left in H by removing u; this reduced the running time by as much as 100× w.r.t. the naive implementation. Our code is written in Java and based on the WebGraph library.[2] It is publicly accessible at https://github.com/Steven--/graphlets. Our platform was a commodity machine equipped with 64GiB of main memory and 32 Intel Xeon CPU cores at 2.5GHz with 30MB of L3 cache, using Oracle's Java Virtual Machine (JVM) (version 1.8.0).

Experiments were executed on 15 graphs that appeared as largest instances in previous work; Table 1 shows the graphs and the largest k for which CC2 ran successfully, i.e., within the available memory limits. Each graph was made undirected, and only the largest connected component was kept, to ensure the correctness of PSRW (as the walk cannot reach one component from another). For Twitter and Road-US, just loading the graph exceeded the available main memory, and we could not run either PSRW or CC2; as a sanity check, we tried a high-end machine with 240GiB

[2] http://webgraph.di.unimi.it/.

memory, and we could run CC2 on Twitter for $k = 4$ and on Road-US for $k = 7$. Those two graphs are therefore omitted from now on.

Concerning the ground-truth graphlet frequencies, we operated as follows. For $k = 5$, for all graphs except Hollywood we could successfully run the exact algorithm of [18] and obtain the precise count of all graphlets. In each other case, we took the average of 50 independent runs of CC2, for a total of five million samples; the empirical low variance of the estimates (see below) strongly suggests that such an average is indeed very close to the true distribution.

6.2 Color Coding Versus Random Walks

We measured the accuracy of CC2 and PSRW as a function of the number of samples and of the running time. For PSRW, we performed 25 independent runs; each execution picks a random initial node in the graph, then simulates 32 random walks in parallel from that node (each walk using a different number generator seed). For CC2, we took the 50 independent runs used to compute the ground truth. In both the cases, we stopped at $100k$ graphlet samples, and along the way we measured the elapsed wall-clock time returned by the operating system, excluding the time to load the graph. Note that running times shall be taken with caution; our is a specific implementation, and times may change as a result of optimizations or hardware modifications.[3] Moreover, small times are affected by artefacts such as the warm-up of the JVM.

Accuracy versus sample size. Let us start with the accuracy versus the number of samples. For each single execution, we computed the accuracy of the estimates, meant as the 1-norm of the residual between the ground truth and the distribution estimated by the algorithm. Then, for each instance (graph, k, number of samples taken), we computed the average the distance over all the executions. For each k, for each input graph, we computed the mean and standard deviation of this 1-norm accuracy over all runs, for both PSRW and CC2. This procedure was repeated after $1k$ samples, $10k$ samples, and $100k$ samples taken by the algorithms. Figure 1 summarizes the results. CC2 appears always at least as accurate as PSRW, and in fact on many instances its accuracy is better by orders of magnitude. Note that, while for $k > 5$ the ground truth is the average of all CC2 runs, which may be in favor of CC2, for $k = 5$, we are using the exact graphlet count.

Figure 2 shows the accuracy of PSRW and CC2 as a function of the number of samples. We observe that CC2 starts converging immediately, while PSRW often exhibits a "bootstrap" phase where the distance from the ground truth remains virtually unchanged. This makes sense: CC2 immediately starts sampling from the distribution of colorful graphlets (hopefully close to the ground truth), while PSRW needs to reach mixing time first. It also suggests that our worst-case lower bounds on the mixing time (Section 4.2) might be not so far from reality after all.

Single-graphlet accuracy. Next, we measured the accuracy of PSRW and CC2 in estimating the frequencies of single graphlets. Note that any sampling-based method has an intrinsically poor accuracy for graphlets that occur very rarely; if we take s samples, a reasonable threshold is to consider only graphlets with ground-truth frequency at least $1/s$. Furthermore, since the number of k-graphlets is already in the hundreds for $k = 6$, we need an aggregate measure of accuracy. Consistently with past work, we used the normalized root-mean-square error (NRMSE). Denote by $f_H(G)$ the ground-truth relative frequency of H in G. Denote by $\hat{f}_H^i(g)$ the frequency estimated obtained from the ith run of the algorithm. Then, the normalized mean-square error is $NMSE(H, G) = r^{-1} \sum_{i=1}^{\mathrm{r}} (\hat{f}_H^i(g)/f_H(G) - 1)^2$. The NRMSE$(H, G)$ is simply the square root of $NMSE(H, G)$. Table 2 shows the average NRMSE of PSRW and CC2 over all runs, all graphs,

[3]For instance, because CC2 accesses vast memory regions in a random fashion, while PSRW exploits more the CPU and its data cache.

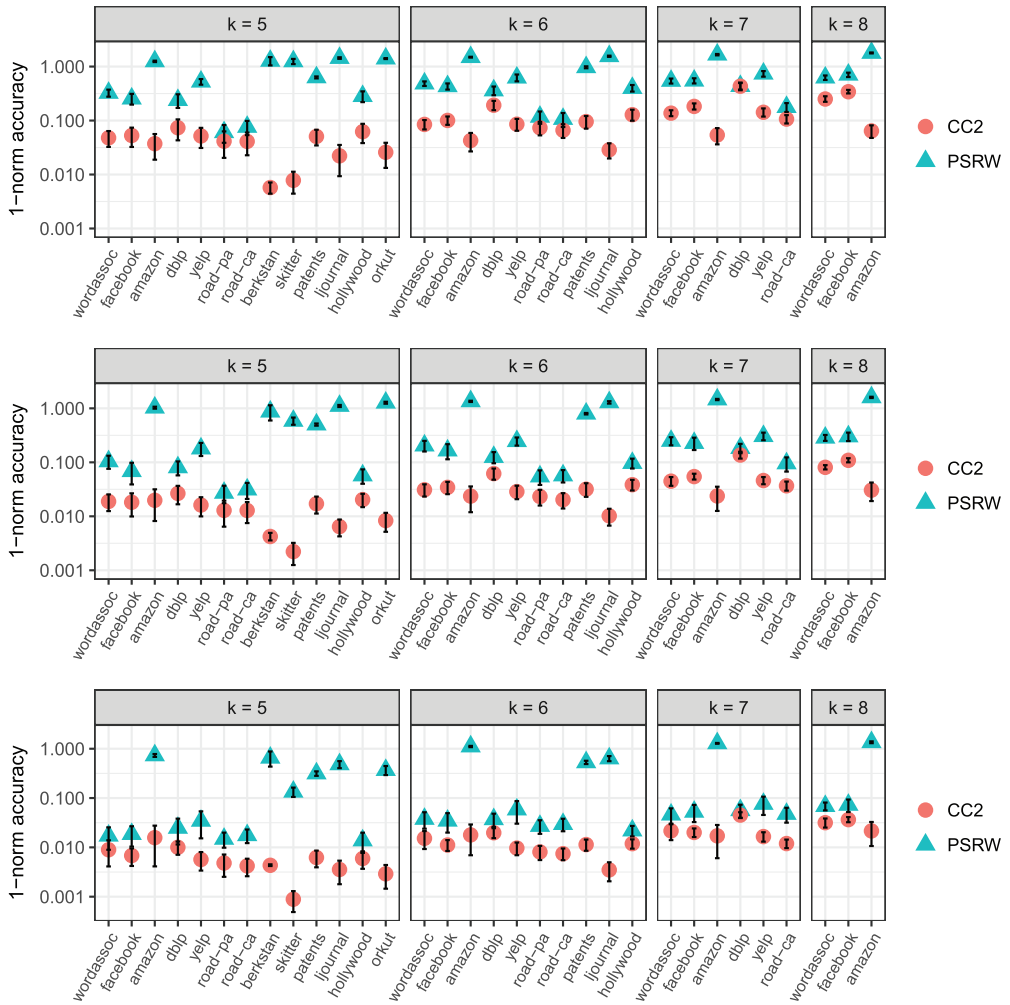

Fig. 1. Accuracy of CC2 and PSRW: average distance from ground truth, ± one standard deviation, after $1k$ samples (top), $10k$ samples (middle), and $100k$ samples (bottom).

and all graphlets, for $s = 1k, 10k$, and $100k$. The NRMSE of CC2 is always significantly lower than that of PSRW—and the gap increases with the number of samples. Note that NRMSE penalises large errors, so these results confirm that CC2 is reliably accurate on all graphs. Observe also that, while the NRMSE of CC2 increases monotonically with s, the NRMSE of PSRW *decreases* with s. This looks counterintuitive, but can be again be explained by high mixing time—if the walk gets "stuck" in a part of the graph where a graphlet is particularly scarce, then the estimates will progressively worsen.

Accuracy versus time. Finally, we look at the accuracy versus the running time. Note that there are some subtleties in this comparison. On the one hand, PSRW starts sampling immediately, while CC2 first performs an (expensive) building phase. On the other hand, when sampling, CC2 immediately produces unbiased samples, while PSRW must wait until the walk mixes. Keeping this in mind, it makes sense to compare the accuracy of the two algorithms after the same elapsed

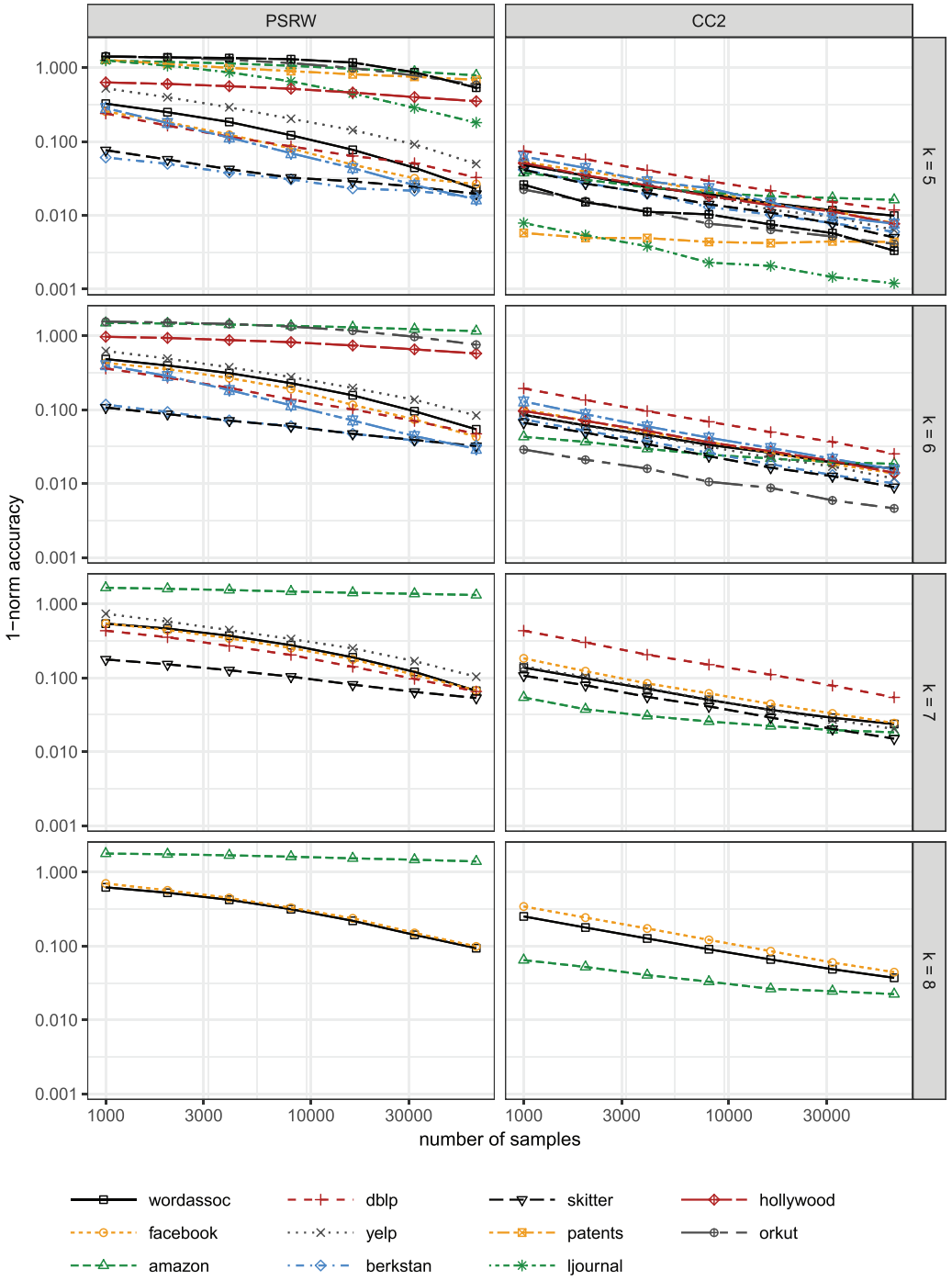

Fig. 2. Convergence of PSRW (left) and CC2 (right) to the ground truth distribution.

Table 2. Accuracy of Individual Graphlet Estimates: Average NRMSE of PSRW and CC2

	$1k$ samples				$10k$ samples				$100k$ samples			
	$k=5$	$k=6$	$k=7$	$k=8$	$k=5$	$k=6$	$k=7$	$k=8$	$k=5$	$k=6$	$k=7$	$k=8$
PSRW	3.84	1.93	1.61	1.23	5.52	2.11	1.98	3.11	5.74	2.61	2.25	3.76
CC2	0.19	0.27	0.44	0.47	0.11	0.15	0.26	0.29	0.07	0.09	0.15	0.17

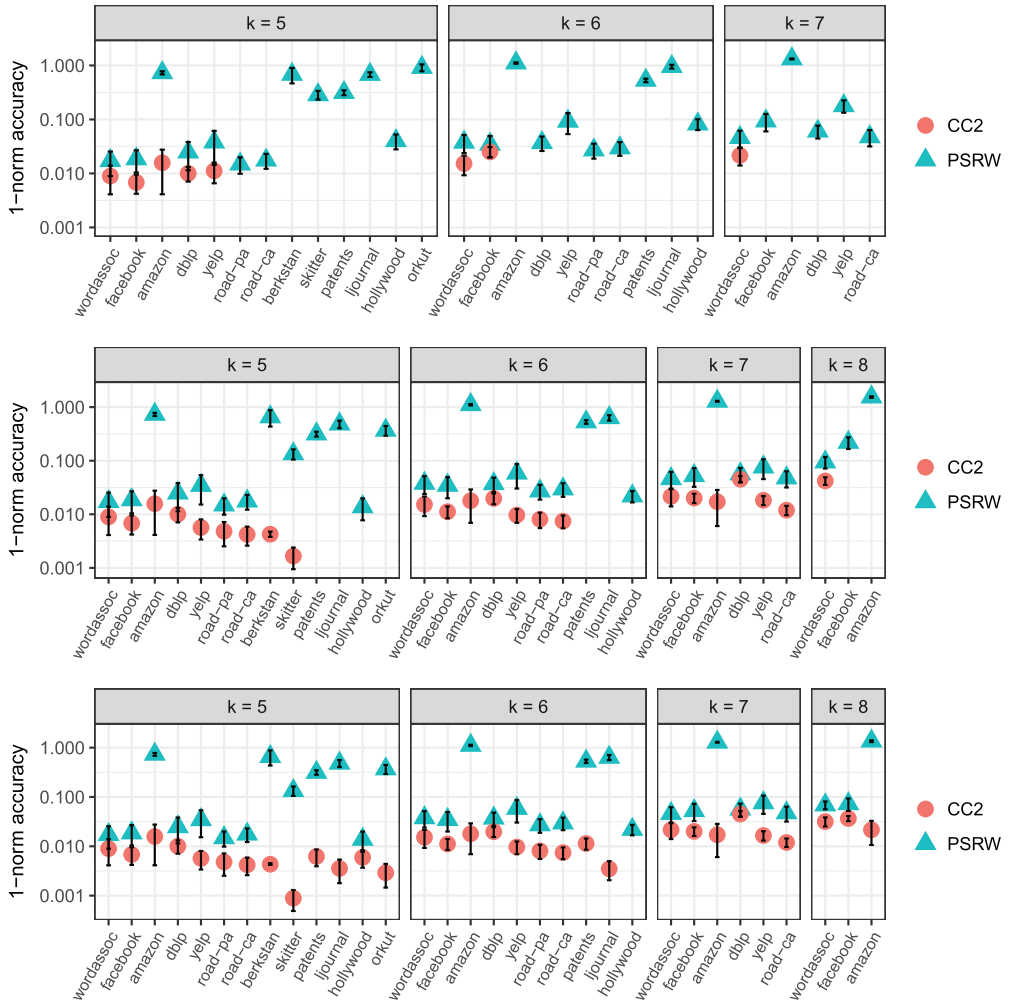

Fig. 3. Accuracy of CC2 and PSRW: average distance from ground truth, ± one standard deviation, after running for 10 seconds (top), 100 seconds (middle), 1,000 seconds (bottom). For CC2, the time includes the building phase.

running time. Figure 3 shows the average distance from ground truth, ± one standard deviation, measured after 10 seconds (top), 100 seconds (middle), and 1,000 seconds (bottom). For CC2, the time includes the building phase. Missing data means no run of the algorithm had produced at least 1,000 samples at that time. As we expected, in many cases CC2 takes longer than PSRW to produce samples. However, when it does so, it consistently provides higher accuracy. In fact, it

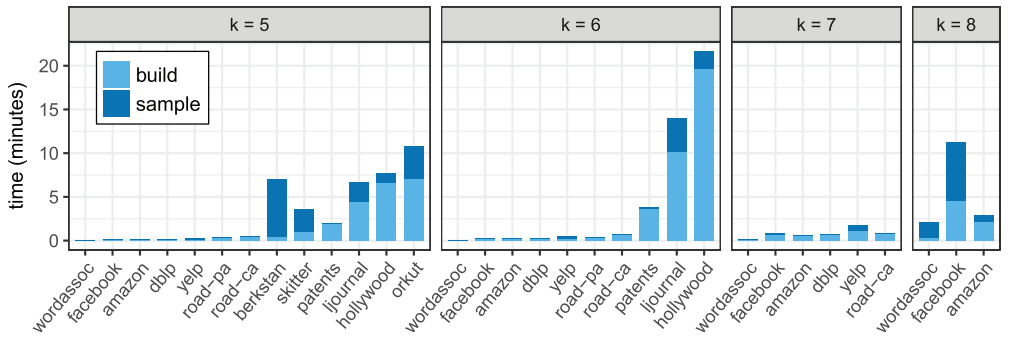

Fig. 4. Running time of CC2 (10^5 samples, average of 50 runs).

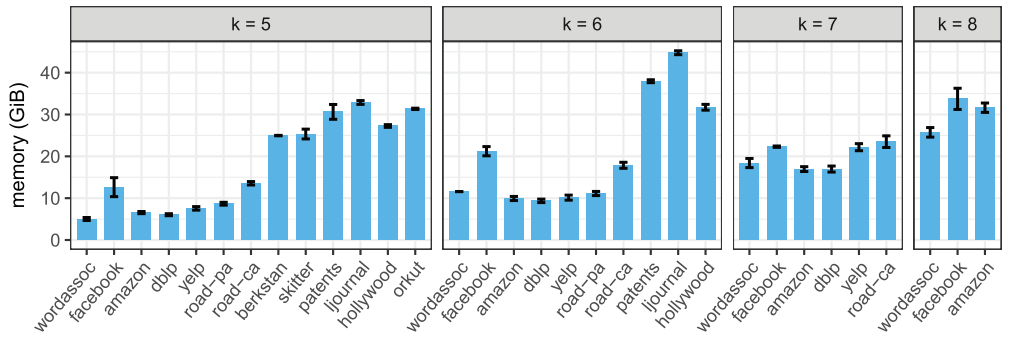

Fig. 5. Memory footprint of CC2 (10^5 samples, average ± one standard deviation of 50 runs).

seems that the instances where CC2 takes longer are those where PSRW yields more inaccurate estimates.

6.3 Performance Analysis of CC2

We analyze the performance of CC2 in terms of running time, memory footprint, and sampling speed. Running time was measured as described above, separately for the building phase and the sampling phase. The sampling speed is simply the ratio of the total number of samples, $100k$, to the sampling time. For memory, we report the value returned by `Runtime.getRuntime().totalMemory()` just after drawing the last sample; this is the JVM heap size used in that moment by our process. We note that the memory footprint depends on the behavior of JVM's Garbage Collector, and that one can reduce it at the expense of time. The measurements are summarized in Figures 4–6.

Two observations are in order. First, consistently with the complexity of CC2, the build time grows with the number of edges in the graph. Similarly, memory grows with the size of the graph, but in a more complex way (recall that the memory used by CC2 actually depends on the number of colorful graphlets). Furthermore, for the instances on which it ran successfully, CC2 managed to take $100k$ samples in a matter of minutes; and this includes instances with tens or hundreds of millions of edges, like `LiveJournal` or `Orkut`, for $k = 5$ and $k = 6$, and many others for $k > 6$. Even a non-optimized implementation of CC2, then, allows us to scale graphlet counting to a larger k than it was possible before, and on just a commodity machine; and an optimized rewriting in a

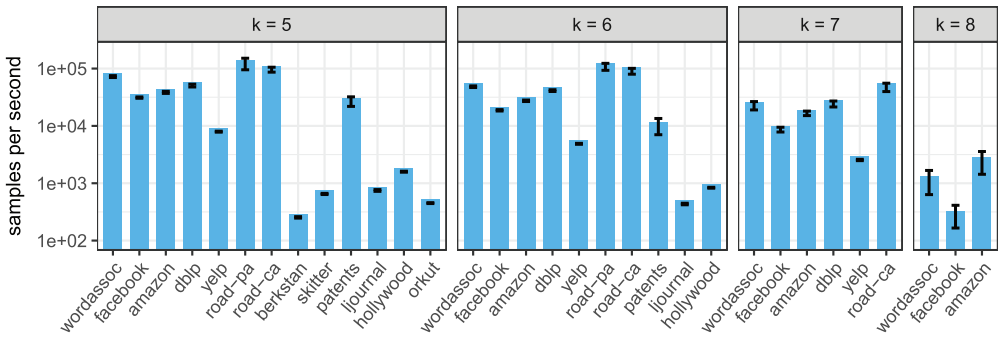

Fig. 6. Sampling speed of CC2 (10^5 samples, average ± one standard deviation of 50 runs).

memory-efficient language (e.g., C++) could lead to significantly better performance. Second, the performance of CC2 appears very stable. The relative standard deviation of the building phase time was below 0.1 for all instances except `WordAssoc` on $k = 5$ and $k = 6$, for which however the average building time was below 1.5seconds and thus inevitably noisy. The relative standard deviation of the memory footprint was less than 0.1 in all cases except `Facebook` for $k = 5$. The relative standard deviation of the sampling time was below 0.3 save for five instances.

Finally, a fact that may look surprising is the large variation in the sampling speed across different graphs, even for the same k (Figure 6). We believe this has to do with how we sample colored treelets in the graphs. Recall that, in our implementation, we sample a treelet by building on-the-fly the distribution of colorful subtrees over the neighbors of a node. If a graph contains a high-degree node, thus, sampling treelets containing it will be rather inefficient. In addition, that node will be likely included in a large fraction of all the graphlets, and therefore we will encounter it often in the sampling phase. Therefore, it is likely that sampling is slower in graphs of higher degree.

6.4 Scaling Graphlet Sampling Beyond Five Nodes

We conclude by showing the distribution of k-graphlets, for $k = 5, 6, 7, 8$, according to our ground truth (see above). Figure 7 shows the distributions, graphlets sorted from the highest to the lowest average frequency. For readability, we include only the 15 most frequent graphlets, and we exclude the distribution of the road graphs, `Road-CA` and `Road-PA`, which are similar and contain mostly trees. Figure 7 tells some interesting facts. First, the most frequent graphlets on average are K_{k-1} (a star) and the graphlet K^+_{k-2} obtained by attaching an extra node to a leaf of K_{k-2}. The three top 5-graphlets, the six top 6-graphlets, the eight top 7-graphlets, and the 14 top 8-graphlets are trees. In addition, many of these trees have a single *branching* node, i.e., a single node with degree more than two, which we conjecture is mapped to a high-degree node (a *hub*) of the graph. These properties might be rooted in social phenomena. Take for instance `LiveJournal` and `Amazon`; these two graphs contain many more copies of K_{k-1} than of K^+_{k-2}. A possible explanation is that the readers of the most frequently read blogs will tend not to know each other and, since the `LiveJournal` graph was made undirected, the neighbors of the high-degree nodes will tend to not induce many edges. The same can hold for `Amazon`, which is a co-purchasing network—in some sense, in both graphs users are buying products. On the other hand, look at the `Facebook` graph (containing only the users from the New Orleans area). Facebook imposes an upper bound on the number of friends of a user. This may increase the likelihood that two neighbors of a node are actually friends, so the ego-network of most nodes will tend to have a significant number of edges. This clearly decreases the fraction of induced K_{k-1}'s that can be found in this graph. Finally,

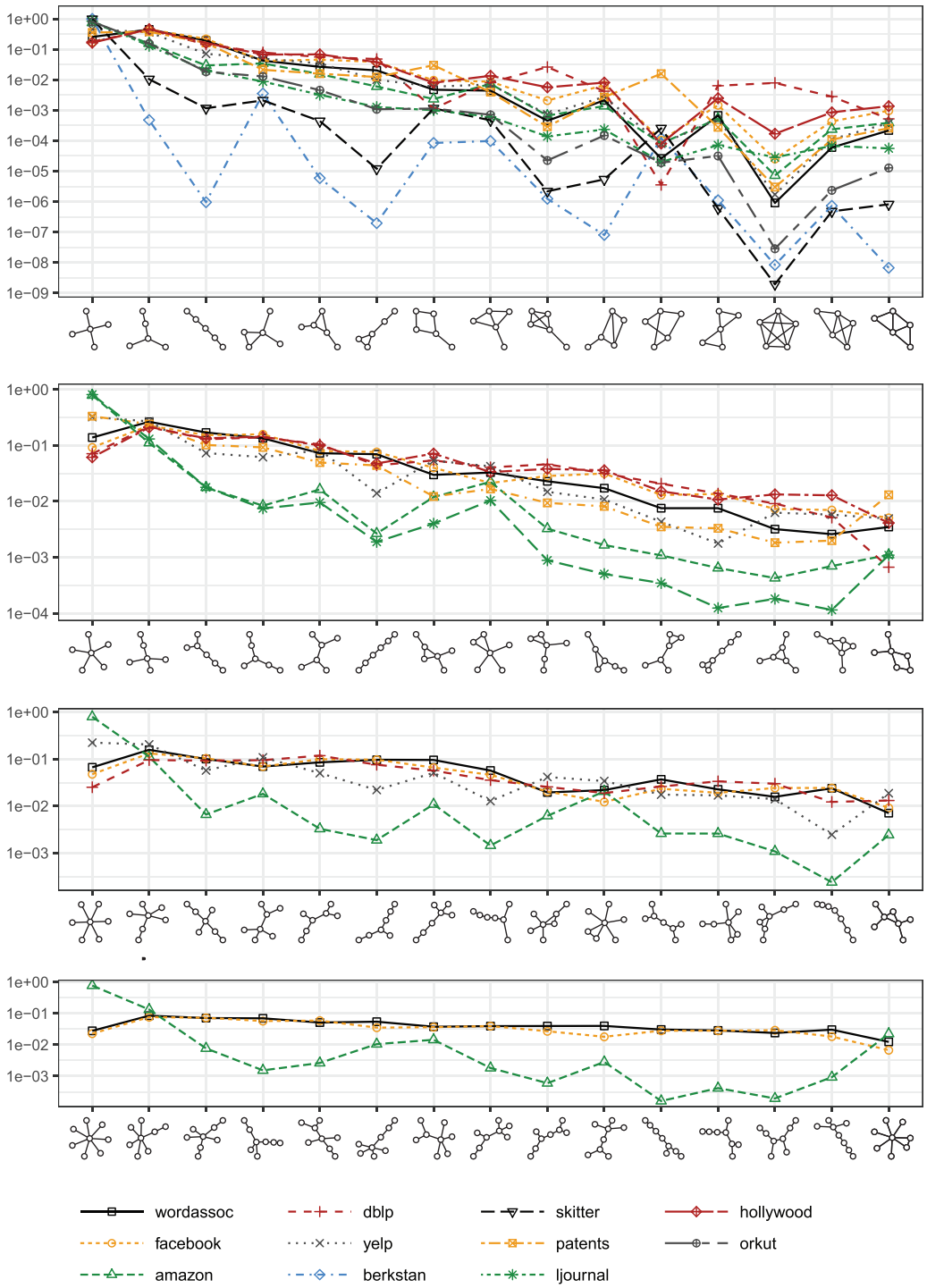

Fig. 7. Estimated frequency distribution of k-graphlets, from $k = 5$ (top) to $k = 8$ (bottom).

the Hollywood graph is the union of cliques, since the actors who starred in a movie are pairwise friends; hence, this graph does not contain many induced K_{k-1}'s.

Second, there seem to be different families of distributions followed by different graphs. This is especially evident in the 6-graphlets distribution, where on the one side, we have Amazon and LiveJournal, with skewed distributions that are strikingly similar; on the other hand, we have the remaining graphs, with much flatter distributions that are again quite close to each other. An intriguing question then one can explain this separation in terms of differences between the processes that have formed the networks.

Third, a surprising fact is that the graphlet distributions of WordAssoc and Facebook are extremely close for all $k = 5, 6, 7, 8$; but Facebook is a social graph, while WordAssoc is a graph resulting from a "free word association" experiment in psychology. Understanding whether such an almost perfect overlapping is just a casual correlation would be an interesting research direction.

Finally, in relation to the algorithm of [8], we note that 4/15 of the top 6-graphlets, 11/15 of the top 7-graphlets, and 14/15 of the top 8-graphlets do not contain a simple path on $(k-1)$ nodes and therefore would be unobservable with their algorithm (see Section 4.3).

7 CONCLUSIONS

In this article, we compared random walks and CC as the two most powerful algorithmic methods available to efficiently count graphlets in massive graphs. Our theoretical mixing time analysis cautions the blind use of random walks on real graphs, if statistical accuracy is paramount; on the other hand, we show that CC can be extended into a graphlet-sampling algorithm with statistical guarantees. In our experiments, CC appears to outperform the state-of-the-art random walk methods, yielding accurate counts for graphlets on up to eight nodes. Investigating the properties of graphs that lead to high mixing times for random walks is an interesting direction of future research. For CC, it will be interesting to see if the dynamic program table can somehow be compressed without sacrificing statistical guarantees much, which would make the method applicable for even larger graphlets; however, such an endeavour appears very challenging.

REFERENCES

[1] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *Proceedings of the 15th IEEE International Conference on Data Mining (ICDM'15)*. 1–10. DOI: http://dx.doi.org/10.1109/ICDM.2015.141

[2] Noga Alon, Raphael Yuster, and Uri Zwick. 1995. Color-coding. *Journal of the ACM* 42, 4 (Jul. 1995), 844–856. DOI: http://dx.doi.org/10.1145/210332.210337

[3] Mansurul A. Bhuiyan, Mahmudur Rahman, Mahmuda Rahman, and Mohammad Al Hasan. 2012. GUISE: Uniform sampling of graphlets for large graph analysis. In *Proceedings of the 12th IEEE International Conference on Data Mining (ICDM'12)*. 91–100. DOI: http://dx.doi.org/10.1109/ICDM.2012.87

[4] Paolo Boldi, Marco Rosa, Massimo Santini, and Sebastiano Vigna. 2011. Layered label propagation: A multiresolution coordinate-free ordering for compressing social networks. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. IW3C2, Geneva, Switzerland, 587–596. DOI: http://dx.doi.org/10.1145/1963405.1963488

[5] P. Boldi and S. Vigna. 2004. The webgraph framework I: Compression techniques. In *Proceedings of the 13th International Conference on World Wide Web (WWW'04)*. IW3C2, Geneva, Switzerland, 595–602. DOI: http://dx.doi.org/10.1145/988672.988752

[6] Marco Bressan, Flavio Chierichetti, Ravi Kumar, Stefano Leucci, and Alessandro Panconesi. 2017. Counting graphlets: Space vs time. In *Proceedings of the 10th ACM International Conference on Web Search and Data Mining (WSDM'17)*. ACM, New York, NY, 557–566. DOI: http://dx.doi.org/10.1145/3018661.3018732

[7] V. T. Chakaravarthy, M. Kapralov, P. Murali, F. Petrini, X. Que, Y. Sabharwal, and B. Schieber. 2016. Subgraph counting: Color coding beyond trees. In *Proceedings of the 30th IEEE International Parallel and Distributed Processing Symposium (IPDPS'16)*. 2–11. DOI: http://dx.doi.org/10.1109/IPDPS.2016.122

[8] Xiaowei Chen, Yongkun Li, Pinghui Wang, and John C. S. Lui. 2016. A general framework for estimating graphlet statistics via random walk. *Proceedings of the VLDB Endowment* 10, 3 (Nov. 2016), 253–264. DOI: http://dx.doi.org/10.14778/3021924.3021940

[9] F. Chung. 2007. Four proofs for the Cheeger inequality and graph partition algorithms. In *Proceedings of the Integrated Community Case Management (ICCM'07)*.

[10] Peter Floderus, Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. 2015. Detecting and counting small pattern graphs. *SIAM Journal of Discrete Mathematics* 29, 3 (Feb. 2015), 1322–1339. DOI: http://dx.doi.org/10.1137/140978211

[11] Guyue Han and Harish Sethu. 2016. Waddling random walk: Fast and accurate mining of motif statistics in large graphs. In *Proceedings of the 16th IEEE International Conference on Data Mining (ICDM'16)*. 181–190.

[12] Mark Jerrum and Kitty Meeks. 2015. The parameterised complexity of counting connected subgraphs and graph motifs. *Journal of Computer and System Sciences* 81, 4 (Nov. 2015), 702–716. DOI: http://dx.doi.org/10.1016/j.jcss.2014.11.015

[13] Madhav Jha, C. Seshadhri, and Ali Pinar. 2013. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. ACM, New York, NY, 589–597. DOI: http://dx.doi.org/10.1145/2487575.2487678

[14] Madhav Jha, C. Seshadhri, and Ali Pinar. 2015. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proceedings of the 24th International Conference on World Wide Web (WWW'15)*. IW3C2, Geneva, Switzerland, 495–505. DOI: http://dx.doi.org/10.1145/2736277.2741101

[15] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining (KDD'05)*. ACM, New York, NY, 177–187. DOI: http://dx.doi.org/10.1145/1081870.1081893

[16] Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. 2008. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web (WWW'08)*. IW3C2, Geneva, Switzerland, 695–704. DOI: http://dx.doi.org/10.1145/1367497.1367591

[17] David A. Levin, Yuval Peres, and Elizabeth L. Wilmer. 2009. *Markov Chains and Mixing Times*. American Mathematical Society. xviii+371 pages.

[18] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. 2017. ESCAPE: Efficiently counting all 5-vertex subgraphs. In *Proceedings of the 26th International Conference on World Wide Web (WWW'17)*. IW3C2, Geneva, Switzerland, 1431–1440. DOI: http://dx.doi.org/10.1145/3038912.3052597

[19] Tanay Kumar Saha and Mohammad Al Hasan. 2015. Finding network motifs using MCMC sampling. In *Proceedings of the 6th Workshop on Complex Networks (CompleNet'15)*. Springer International Publishing, 13–24.

[20] G. M. Slota and K. Madduri. 2013. Fast approximate subgraph counting and enumeration. In *Proceedings of the 42nd International Conference on Parallel Processing (ICPP'13)*. 210–219. DOI: http://dx.doi.org/10.1109/ICPP.2013.30

[21] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th International Conference on World Wide Web (WWW'11)*. IW3C2, Geneva, Switzerland, 607–614. DOI: http://dx.doi.org/10.1145/1963405.1963491

[22] Ngoc Hieu Tran, Kwok Pui Choi, and Louxin Zhang. 2013. Counting motifs in the human interactome. *Nature Communications* 4 (Aug. 2013), Article 2241.

[23] Charalampos E. Tsourakakis, U. Kang, Gary L. Miller, and Christos Faloutsos. 2009. DOULION: Counting triangles in massive graphs with a coin. In *Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'09)*. ACM, New York, NY, 837–846. DOI: http://dx.doi.org/10.1145/1557019.1557111

[24] W. T. Tutte. 2001. *Graph Theory*. Cambridge University Press. https://books.google.it/books?id=uTGhooU37h4C.

[25] Bimal Viswanath, Alan Mislove, Meeyoung Cha, and Krishna P. Gummadi. 2009. On the evolution of user interaction in Facebook. In *Proceedings of the 2nd ACM Workshop on Online Social Networks (WOSN'09)*. ACM, New York, NY, 37–42. DOI: http://dx.doi.org/10.1145/1592665.1592675

[26] M. D. Vose. 1991. A linear algorithm for generating random numbers with a given distribution. *IEEE Transactions on Software Engineering* 17, 9 (Sep. 1991), 972–975. DOI: http://dx.doi.org/10.1109/32.92917

[27] Pinghui Wang, John C. S. Lui, Bruno Ribeiro, Don Towsley, Junzhou Zhao, and Xiaohong Guan. 2014. Efficiently estimating motif statistics of large networks. *ACM Transactions on Knowledge Discovery from Data* 9, 2 (Sep. 2014), Article 8, 27 pages. DOI: http://dx.doi.org/10.1145/2629564

[28] Pinghui Wang, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John C. S. Lui, Don Towsley, Junzhou Zhao, Jing Tao, and Xiaohong Guan. 2016. A fast sampling method of exploring graphlet degrees of large directed and undirected graphs. arXiv:1604.08691

[29] Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John C. S. Lui, Don Towsley, Jing Tao, and Xiaohong Guan. 2018. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering* 30, 1 (2018), 73–86. https://doi.org/10.1109/TKDE.2017.2756836

[30] Virginia Vassilevska Williams and Ryan Williams. 2013. Finding, minimizing, and counting weighted subgraphs. *SIAM Journal on Computing* 42, 3 (2013), 831–854. DOI: http://dx.doi.org/10.1137/09076619X

[31] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (Jan. 2015), 181–213. DOI: http://dx.doi.org/10.1007/s10115-013-0693-z

[32] Z. Zhao, M. Khan, V. S. A. Kumar, and M. V. Marathe. 2010. Subgraph enumeration in large social contact networks using parallel color coding and streaming. In *Proceedings of the 39th International Conference on Parallel Processing (ICPP'10)*. 594–603. DOI: http://dx.doi.org/10.1109/ICPP.2010.67

Received June 2017; revised February 2018; accepted February 2018