Some Vignettes on Subgraph Counting Using Graph Orientations

C. Seshadhri
Department of Computer Science & Engineering, University of California, Santa Cruz, CA, USA

Abstract

Subgraph counting is a fundamental problem that spans many areas in computer science: database theory, logic, network science, data mining, and complexity theory. Given a large input graph G and a small pattern graph H, we wish to count the number of occurrences of H in G. In recent times, there has been a resurgence on using an old (maybe overlooked?) technique of orienting the edges of G and H, and then using a combination of brute-force enumeration and indexing. These orientation techniques appear to give the best of both worlds. There is a rigorous theoretical explanation behind these techniques, and they also have excellent empirical behavior (on large real-world graphs). Time and again, graph orientations help solve subgraph counting problems in various computational models, be it sampling, streaming, distributed, etc. In this paper, we give some short vignettes on how the orientation technique solves a variety of algorithmic problems.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases subgraph counting, graph degeneracy, homomorphism counting, graph algorithms

Digital Object Identifier 10.4230/LIPIcs.ICDT.2023.3

Category Invited Talk

Funding NSF DMS-2023495, CCF-1740850, 1908384, 1909790

1 Introduction

A central problem in computer science is to count or enumerate the occurrences of a small pattern graph H in a large input graph G. The applications of graph pattern counts occur across numerous scientific areas, including logic, biology, statistical physics, database theory, social sciences, machine learning, and network science [36, 13, 17, 12, 24, 9, 30, 55, 48, 21, 47]. The tutorial [51] has more details on applications. A rich and deep theory has emerged from the study of graph pattern counting [41, 14, 32, 19, 42, 2, 18, 48, 49].

Let us formalize this problem through the language of graph homomorphisms (or graph mappings). The pattern simple graph is denoted $H = (V(H), E(H))$, and is thought of constant-sized. The input simple graph is denoted $G = (V(G), E(G))$. An H-homomorphism is a map $f : V(H) \to V(G)$ that preserves edges. Formally, $\forall (u, v) \in E(H)$, $(f(u), f(v)) \in E(G)$. If the map is 1-1, then it is called a *subgraph*. If the map also preserves non-edges, then it is an *induced* subgraph/homomorphism. For this high-level survey, we will not commit to any specific problem variant. We use "subgraph counting" an umbrella terms that refers to all of these problems.

The study of efficient algorithms for subgraph counting is almost a subfield in of itself [37, 3, 12, 24, 22, 19, 9, 18, 10, 49]. It would take us too far out to survey the state of this area. Even the simplest version, when H is a triangle, receives much attention. Let $n = |V(G)|$ and $k = |V(H)|$. The trivial algorithm that simply tries all k-tuples of vertices runs in $O(n^k)$ time. By $\#W[1]$-hardness even for H being a k-clique, we do not expect $n^{o(k)}$ algorithms for general H [19]. The algorithmic study of subgraph counting focuses on understanding conditions on H and G when the trivial n^k running time bound can be beaten.

The algorithmic technique of *graph orientations* has repeatedly helped in solving such counting problems. It is also practically viable and is a key tool in subgraph counting applications for real-world graphs. The study of this technique has led to a rich hoard of mathematical results, which further inspire empirical work. In this paper, we will describe a few short vignettes, describing the application of the technique and specific results.

2 Triangle counting through graph orientations

Let us begin with the basic problem of triangle counting, where H is a 3-clique. A fairly direct algorithm is the *wedge enumeration* procedure. For each vertex u, list all pairs of neighbors v, w. If (v, w) is an edge, then (u, v, w) form a triangle. Observe that we enumerate two-paths (or wedges) v, u, w; hence the name wedge enumeration.

Let the graph G have n vertices and m edges. For a vertex v, let d_v denote its degree. The running time of the above procedure is $O(\sum_v d_v^2)$. Not surprisingly, high-degree vertices greatly affect the running time.

Graph orientations can be thought of as a technique to cut down the running time of wedge enumeration. This method has been rediscovered many times, but the earliest reference is by Chiba-Nishizeki [14]. Chrobak-Eppstein use this idea to deal with planar graphs [15]. It has been rediscovered by Schank-Wagner [50] and Cohen [16].

▶ **Definition 1.** *Given any undirected, simple graph G, an* acyclic orientation *of G is a DAG D such that (u, v) is an edge in D iff (u, v) is an edge in G.*

Let the partial order on vertices induced by D be denoted $\prec_D$.

We can also construct a DAG by defining a total order π on the vertices, and then orienting the edges from lower to higher vertex.

We consider an acycle orientiation D, and instead enumerate all (directed) triangles in D. Observe that a triangle has a unique acyclic orientation. Moreover, from every u, we will only find triangles (u, v, w) such that $u \prec_D v, w$. This is what allows for the major savings in computation.

Formally, the meta-algorithm is:

1. Compute an acyclic orientation D of the input graph G.
2. For every vertex u:
   a. For every pair of *outneighbors* v, w, check if edge (v, w) is present.

The key difference in the enumeration method is to only look at the outneighbors of u, which is at most the degree of u. Suppose the outdegree of a vertex v is denoted d_v^+. Then the running time of the meta-algorithm is $O(\sum_v (d_v^+)^2)$.

So how do we choose the orientation to make this sum small? We will consider two schemes: *degree orientations* and *degeneracy orientations*.

Degree orientations. We order vertices by degree, breaking ties by vertex id. Formally, $u \prec_D v$ iff $d(u) < d(v)$ or $d(u) = d(v)$ and u has smaller id. This was the ordering proposed in Chiba-Nishizeki's original paper [14]. It is implicitly used in the `forward` algorithm of Schank-Wagner [50].

Degeneracy orientations. This is a more sophisticated approach. Think of the "peeling process", where we repeatedly remove a vertex of minimum degree. (Note that the degree keeps decreasing as more vertices are removed.) The order of removal is the degeneracy ordering, and one simply creates a DAG from this ordering.

The degeneracy orientation is a byproduct of the classic core decomposition of Matula and Beck [43]. It was first used for subgraph counting by Chrobak-Eppstein [15]. Schank-Wagner independently give the equivalent `node-iterator-core` algorithm [50].

2.1 Graph orientations and degeneracy

There is a remarkable connection between the graph orientations given above, the concept of *graph degeneracy*, and measures of graph density. Let us begin with a classic definition from graph theory that directly ties into our problem. For a directed graph D, we use $d_v^+(D)$ to denote the outdegree of v in D.

▶ **Definition 2.** *The graph degeneracy, denoted $\kappa(G)$, is defined as follows.*

$$\kappa(G) = \min_{D \text{ orientation of } G} \max_v d_v^+(D)$$

In plain English, the graph degeneracy is the smallest possible maximum outdegree of an acyclic orientation of G. This quantity is also called the *coloring number*, due to connections with graph coloring (Sec. 5.2 of [23]). For convenience, we will simply denote the degeneracy of G as κ.

Matula-Beck gave a simple linear time algorithm to compute the graph degeneracy, which is exactly the peeling process [43]. Quite surprisingly, the peeling process (or core decomposition) discovers the orientation that minimizes the maximum outdegree. Observe that the running time of the triangle enumeration process can be bounded as $O(\sum_v (d_v^+)^2) = O(\max_v d_v^+ \sum_v d_v^+) = O(m \max_v d_v^+)$. If we choose the degeneracy orientation, then the running time is $O(m\kappa)$. This algorithm is somewhat folklore, and explicitly stated by Schank-Wagner [50].

The $O(m\kappa)$ bound was first achieved by Chiba-Nishizeki, using degree orientations [14]. They (implicitly) proved the following theorem, which is stronger than what is required.

▶ **Theorem 3** ([14]). *For the degree orientation, $\sum_v d_v d_v^+ = O(m\kappa)$.*

Asymptotically, both degree and degeneracy orientations provide the same running time benefit for triangle counting. This result of Chiba-Nishizeki was expressed in terms of the graph *arboricity*, a closely related parameter. But this result sparked off an entire subarea of algorithms, where the running time is parameterized by the graph degeneracy.

To get more context, let us dig deeper into the meaning of degeneracy and its connection to other graph parameters.

2.2 Degeneracy and graph density

The (half) average degree of a graph, $m(G)/n(G)$, is a natural graph parameter. Yet it appears to be a weak measure of the density of a graph. One may have a graph with a linear number of edges, but containing a clique of size $\sqrt{n}$. A stronger notion of sparsity would be the minimum average degree over all subgraphs of G.

The following theorem builds on classic results of Nash-Williams [44]. It relates the degeneracy to strong notions of graph sparsity.

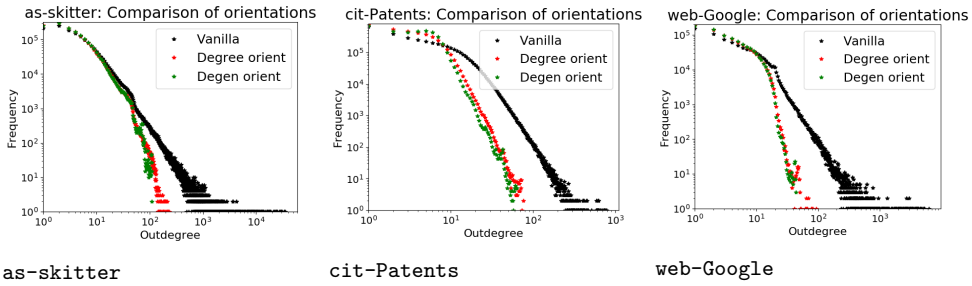

Figure 1 We plot the outdegree distributions of the degree and degeneracy orientation for different real-world graphs. For context, the plots also give the original (vanilla) degree distribution, to see how the orientations cut down the heavy tail. Observe that both orientations do quite well, though the degeneracy orientation leads to a smaller maximum degree.

▶ **Theorem 4.** *Let* $\alpha(G) = \max_{G' \text{subgraph of } G} \frac{m(G)}{n(G)-1}$. *Then* $\alpha(G) \leq \kappa(G) \leq 2\alpha(G)$.

Ignoring constant factors, a low degeneracy graph is one where *all* subgraphs have low average degree. One can show that $\kappa(G) \leq \sqrt{2m}$, which shows that triangle counting for any graph can be done in $O(m^{3/2})$ time.

This concept motivates *bounded degeneracy graph classes*. These are graph classes with constant degeneracy, or alternately, graphs where all subgraphs are sparse. Bounded degeneracy graph classes are immensely rich; they contain all minor-closed families. Preferential attachment graphs have constant degeneracy. Real-world graphs typically have a small degeneracy, comparable to their average degree ([33, 39, 53, 4, 8], also Table 2 in [4]). The repeated occurrence of bounded degeneracy graphs across many scenarios underscores the importance of graph orientations as an algorithmic technique.

2.3 Taming real-world heavy tails

The heavy-tailed degree distribution is one of the hallmarks of real-world graphs. While these graphs are sparse, their degrees show high variance. These heavy tails pose particular challenges for subgraph counting and other algorithmic tasks. Orientations give a simple and effective method to cut down these tails.

In Figure 1, we plot the (out)degree distributions for three different real-world networks with millions of edges [56]. The degree distribution is the number of vertices of a given degree, plotted in log-log scale. The "vanilla" points, marked in black, give the original degree distribution. One can see the characteristic heavy tail in all cases.

We then plot the outdegree distributions of the degree and degeneracy orientations, in red and green respectively. Observe how both these orientations dramatically reduce the tail. The degeneracy orientation is only slight lower than the degree orientation. As expected the maximum degree of the degeneracy orientation is smaller than that of the degree orientation. In general, the quantity $\sum_v (d_v^+)^2$ is similar for both orientations.

These observations explain why the orientation technique has so much practical utility. The original algorithm for triangle counting is immensely effective in practice. A well-engineered implementation can count triangles in real-world graphs with hundreds of millions of edges within minutes on a commodity machine [48, 1]. As the plots in Figure 1 show, for triangle counting, the degree orientation is as effective as the degeneracy orientation. Degree orientations have the additional benefit of being locally computable and easily parallelizable. Cohen [16] and Suri-Vassilvitskii [54] independently proposes this orientation for Map-Reduce listing of triangles.

2.4 Practical clique counting

The power of degeneracy orientations is central to most practical clique counting algorithms. Following the template for triangle counting, k-clique counting can be done by searching all $(k-1)$-tuples of outneighbors. So, for each vertex v, we consider $\binom{d_v^+}{k-1}$ tuples. This leads to a total running time of $O(\sum_v (d_v^+)^{k-1})$. For the degeneracy orientation, $\max_v d_v^+ = \kappa$. Hence, $\sum_v (d_v^+)^{k-1} \leq \kappa^{k-2} \sum_v d_v^+ = m\kappa^{k-2}$. Thus, we can get a $O(m\kappa^{k-2})$ time algorithm for counting all k-cliques.

In practice, this is a remarkably powerful tool for clique counting. Instead of enumerating within outneighborhoods, observe that k-clique counting on the input graph G is reduced to $(k-1)$-clique counting on the n outneighborhoods. Each outneighborhood is potentially small (at most size κ). Each "outneighborhood problems" can be parallelized or distributed; being small problems, one can fit each of them into the memory of a small machine. This idea is central to almost all state-of-the-art practical clique counting algorithms [29, 31, 38, 20, 52].

3 Beyond clique counting

It is natural to ask whether the power of orientations goes beyond counting cliques. A nice twist on the triangle counting algorithm can be used to count 4-cycles. As before, we will orient our input graph G using the degree or degeneracy orientation. Each 4-cycle of G will become an oriented version, and there are three possible non-isomorphic orientations of the cycle. These are shown in Figure 2.

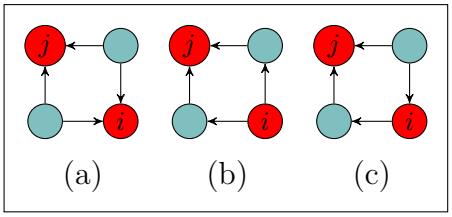

Figure 2 All acyclic orientations of the 4-cycle.

Notice that for all the three cases, the directed wedge between i and j (marked in red) is either an out-wedge or an inout-wedge. These wedges are given in Figure 3. Hence, one can enumerate only these wedges, index them appropriately, and get the total 4-cycle count.

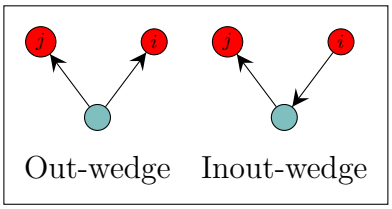

Figure 3 Directed wedges.

For two vertices i, j, let W_{ij}^{++} and W_{ij}^{+-} be the number of out-wedges and inout wedges respectively between i and j. The algorithm is:

1. Compute an acyclic (degree or degeneracy) orientation D of the input graph G.
2. Enumerate all out-wedges and inout-wedges (shown in Figure 3). Through this enumeration, compute, for each pair (i, j) of vertices, compute the numbers W_{ij}^{++} and W_{ij}^{+-}.
3. Output the sum $\sum_{i,j} \left(\binom{W_{ij}^{++}}{2} + \binom{W_{ij}^{+-}}{2} + W_{ij}^{+-} \cdot W_{ij}^{++} \right)$.

A few comments. By appropriate indexing and use of data structures, the entire running time can be made linear in the total number of out-wedges and inout wedges. The sum given above separately computes the various directed 4-cycles. There are three terms, each corresponding to one pattern in Figure 2. Observe that the algorithm gets an exact count *without* enumeration of 4-cycles. This leads to a large savings in running time.

The total number of wedges enumerated is at most $\sum_v d_v d_v^+$. This is somewhat larger than triangle counting, where only out-wedges are enumerated. Nonetheless, for the degeneracy ordering, $\max_v d_v^+ = \kappa$. So the running time is $O(m\kappa)$. For the degree orientation, by Theorem 3, we also get the $O(m\kappa)$ running time.

This bound was first achieved by Chiba-Nishizeki, but through a more complicated algorithm and analysis. The presentation given here is from Pinar et al. [48]. An equivalent formulation was given earlier by Cohen [16].

The grand generalization. How far can this technique go? The overall template for counting H-subgraphs is to first construct all acyclic orientations of H, and count each of them in the degeneracy (or degree) oriented G. For each acyclic orientation of H, we break it up into a collection of directed rooted trees. By the outdegree bounds of the degeneracy orientation, we can enumerate all these directed rooted trees in G. These directed trees needed to indexed appropriately so the overall H-subgraph count can be efficiently computed (as in the case of 4-cycles, by the three terms).

A series of papers performed these generalizations [48, 6, 7, 5], and most significant is probably Bressan's notion of DAG treewidth [11]. By combining various results, one arrives at the following dichotomy theorem (technically for homomorphisms).

▶ **Theorem 5** ([7, 5]). *Suppose the longest induced cycle of H has length at most 5. Then, there is an algorithm exactly computing the H-homomorphism count that runs in $O(m\,poly(\kappa)\log n)$ time.*

Suppose the longest induced cycle of H has length strictly greater than 5. Assume the strong Triangle Detection Conjecture from fine-grained complexity. Then, for all (computable) functions $g : \mathbb{N} \to \mathbb{N}$ and all $\delta > 0$, there does not exist an algorithm computing H-homomorphism counts is $O(m^{4/3-\delta} g(\kappa))$ time.

This is surprisingly precise dichotomy theorem for when bounded degeneracy helps in subgraph/homomorphism counting. The limits of the orientation technique remarkably match up with the hardness result. The strong form of the Triangle Detection Conjecture states that there is no algorithm that can find a triangle in a graph in $O(m^{4/3-\delta})$ time. (The best upper bound is much larger, and would become $m^{4/3}$ if the matrix multiplication exponent is 2.)

Many practical algorithms for large-scale graph pattern counting use versions of these algorithms for bounded degeneracy graphs [2, 40, 48, 46, 39, 47]. While they may not be explicitly stated in the language above, the algorithmic techniques combine orientations and indexing. The concept of DAG treewidth captures the essence of the algorithms, and the upper bound of Theorem 5 subsumes all the applications.

4 A sublinear application

Let us consider a seemingly unrelated problem. We are given access to the adjacency list of a massive graph G. We can sample a uniform at random (uar) vertex, query the degree of a vertex, and can sample uar neighbors of a given vertex.

Our aim is to estimate the average degree $\sum_v d_v/n = 2m/n$, with the fewest queries to the graph. An obvious approach is to sample a set of uar vertices and compute the average degree of the sample. While this is an unbiased estimator, the variance can be extremely high. As an extreme example, suppose the graph is a star. So all vertices except the center have degree one, while the center vertex has degree $n-1$. The average degree is $2(n-1)/n = 2 - o(1)$. But the sampled average will be 1, with extremely high probability.

We have observed that orientations provide a way of "cutting down the tails". So consider the following algorithm.

1. Pick a uar vertex u.
2. Pick a uar neighbor v of u.
3. If $d_u < d_v$, output $2d_u$. If $d_u = d_v$ and the ID of u is less than the ID of v, output $2d_u$. Otherwise, output 0.

To analyze this algorithm, it is convenient to think of the degree orientation. When the procedure picks a directed edge leaving u, then it outputs $2d_u$. The expected output of this procedure is

$$\frac{1}{n}\sum_u \frac{d_u^+}{d_u} \cdot 2d_u = \frac{2\sum_u d_u^+}{n} = 2m/n$$

Thus, this procedure is also an unbiased estimator for the average degree. Observe what this procedure does for the star. The central vertex has no neighbors of high degree and thus, does not contribute to the estimator. Hence, the variance of the estimator is much smaller.

Remarkably, the variance can be bounded by the degeneracy.

▶ **Theorem 6** ([26]). *With high probability, the average of $O(\epsilon^3\kappa)$ samples is a $(1+\epsilon)$-approximation to the average degree.*

Since κ is at most $\sqrt{2m}$, this shows that the average degree of a graph can be approximated with (the optimal) $O(\sqrt{m})$ samples. The algorithm given above is substantially simpler than existing procedures that achieved this bound [35]. (Refer to Chapter 10.3 of [34] for more details.)

This idea of exploiting orientations by a sampling process has succeeded in solving a number of sublinear graph estimation problems [25, 26, 27, 28]. For such algorithms, we can only afford to use the degree orientation since it is locally computable. One of the challenges in these results is related properties of the degree orientation to desired properties of the degeneracy orientation.

5 Conclusion

These vignettes show the varied algorithmic uses of orientations for subgraph counting problems. Given the relative simplicity of the orientation technique, it is surprisingly effective in designing efficient algorithms. And as Theorem 5 shows, these algorithms are often optimal. Each of the above sections merely scratches the surface of what orientations can achieve. We discussed four related applications: triangles, cliques, four-cycles, and degree estimation.

The orientation technique has led to optimal and practical algorithms in each application. Moreover, there is a rich theory emerging on the basis of orientations. The connections to density in Section 2.2 form a starting point to much deeper inquiry into graph sparsity, developed by Nešetřil and Ossana de Mendez [45]. The sublinear subgraph counting results referenced in Section 4 have all emerged from understanding the power of degree orientations in reducing the variance of specific random variables.

References

1 Escape. `https://bitbucket.org/seshadhri/escape`.

2 Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. Efficient graphlet counting for large networks. In *International Conference on Data Mining*, 2015.

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

4 Suman K Bera, Amit Chakrabarti, and Prantar Ghosh. Graph coloring via degeneracy in streaming and other space-conscious models. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2020.

5 Suman K. Bera, Lior Gishboliner, Yevgeny Levanzov, C. Seshadhri, and Asaf Shapira. Counting subgraphs in degenerate graphs. *Journal of the ACM*, 69(3), 2022.

6 Suman K Bera, Noujan Pashanasangi, and C Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In *Innovations in Theoretical Computer Science*, 2020.

7 Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Near-linear time homomorphism counting in bounded degeneracy graphs: The barrier of long induced cycles. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 2315–2332, USA, 2021.

8 Suman K Bera and C Seshadhri. How the degeneracy helps for triangle counting in graph streams. In *Principles of Database Systems*, pages 457–467, 2020.

9 Christian Borgs, Jennifer Chayes, László Lovász, Vera T Sós, and Katalin Vesztergombi. Counting graph homomorphisms. In *Topics in discrete mathematics*, pages 315–371. Springer, 2006.

10 Marco Bressan. Faster subgraph counting in sparse graphs. In *International Symposium on Parameterized and Exact Computation (IPEC)*, 2019.

11 Marco Bressan. Faster algorithms for counting subgraphs in sparse graphs. *Algorithmica*, 83:2578–2605, 2021.

12 Graham R Brightwell and Peter Winkler. Graph homomorphisms and phase transitions. *Journal of combinatorial theory, series B*, 77(2):221–262, 1999.

13 Ashok K Chandra and Philip M Merlin. Optimal implementation of conjunctive queries in relational data bases. In *Symposium on Theory of Computing (STOC)*, pages 77–90, 1977.

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

15 Marek Chrobak and David Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theor. Comput. Sci.*, 86(2):243–266, 1991.

16 Jonathan Cohen. Graph twiddling in a MapReduce world. *Computing in Science & Engineering*, 11:29–41, 2009.

17 J. Coleman. Social capital in the creation of human capital. *American Journal of Sociology*, 94:S95–S120, 1988.

18 Radu Curticapean, Holger Dell, and Dániel Marx. Homomorphisms are a good basis for counting small subgraphs. In *Proceedings of the 49th Annual ACM SIGACT Symposium on Theory of Computing*, pages 210–223, 2017.

19 Víctor Dalmau and Peter Jonsson. The complexity of counting homomorphisms seen from the other side. *Theoretical Computer Science*, 329(1-3):315–323, 2004.

20 Maximilien Danisch, Oana Denisa Balalau, and Mauro Sozio. Listing k-cliques in sparse real-world graphs. In *Conference on the World Wide Web (WWW)*, pages 589–598, 2018.

21 Holger Dell, Marc Roth, and Philip Wellnitz. Counting answers to existential questions. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, 2019.

22 Josep Díaz, Maria Serna, and Dimitrios M Thilikos. Counting h-colorings of partial k-trees. *Theoretical Computer Science*, 281(1-2):291–309, 2002.

23 Reinhard Diestel. *Graph Theory, Fourth Edition*. Springer, 2010.

24 Martin Dyer and Catherine Greenhill. The complexity of counting graph homomorphisms. *Random Structures & Algorithms*, 17(3-4):260–289, 2000.

25 T. Eden, A. Levi, D. Ron, and C Seshadhri. Approximately counting triangles in sublinear time. In *Proceedings of the Symposium on Foundations of Computer Science (FOCS)*, pages 614–633, 2015.

26 T. Eden, D. Ron, and C. Seshadhri. Sublinear time estimation of degree distribution moments: The degeneracy connection. In *International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 7:1–7:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.7.

27 T. Eden, D. Ron, and C. Seshadhri. On approximating the number of k-cliques in sublinear time. In *Symposium on Theory of Computing (STOC)*, pages 722–734, 2018.

28 T. Eden and W. Rosenbaum. On sampling edges almost uniformly. In *Symposium on Simplicity in Algorithms (SOSA)*, pages 1–9, 2018. doi:10.4230/OASIcs.SOSA.2018.7.

29 David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18, 2013.

30 G. Fagiolo. Clustering in complex directed networks. *Phys. Rev. E*, 76:026107, August 2007.

31 Irene Finocchi, Marco Finocchi, and Emanuele G. Fusco. Clique counting in mapreduce: Algorithms and experiments. *ACM Journal of Experimental Algorithmics*, 20, 2015.

32 Jörg Flum and Martin Grohe. The parameterized complexity of counting problems. *SIAM Journal on Computing*, 33(4):892–922, 2004.

33 G. Goel and J. Gustedt. Bounded arboricity to determine the local structure of sparse graphs. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 159–167. Springer, 2006.

34 O. Goldreich. *Introduction to Property Testing*. Cambridge University Press, 2017.

35 O. Goldreich and D. Ron. Approximating average parameters of graphs. *Random Structures and Algorithms*, 32(4):473–493, 2008.

36 P. Holland and S. Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76:492–513, 1970.

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

38 Shweta Jain and C. Seshadhri. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *Conference on the World Wide Web (WWW)*, pages 441–449, 2017.

39 Shweta Jain and C Seshadhri. A fast and provable method for estimating clique counts using turán's theorem. In *Conference on the World Wide Web (WWW)*, pages 441–449, 2017.

40 Madhav Jha, C Seshadhri, and Ali Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Conference on the World Wide Web (WWW)*, pages 495–505, 2015.

41 László Lovász. Operations with structures. *Acta Mathematica Academiae Scientiarum Hungarica*, 18(3-4):321–328, 1967.

42 László Lovász. *Large networks and graph limits*, volume 60. American Mathematical Soc., 2012.

43 David W Matula and Leland L Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM (JACM)*, 30(3):417–427, 1983.

44 C. St. J. A. Nash-Williams. Decomposition of finite graphs into forests. *Journal of the London Mathematical Society*, 39(1):12, 1964.

45 J. Nešetřil and P. Ossana de Mendez. *Sparsity: Graphs, Structures, and Algorithms*. Springer, 2012.

46 Mark Ortmann and Ulrik Brandes. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied network science*, 2(1), 2017.

47 Noujan Pashanasangi and C Seshadhri. Efficiently counting vertex orbits of all 5-vertex subgraphs, by evoke. In *International Conference on Web Search and Data Mining (WSDM)*, pages 447–455, 2020.

48 Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Conference on the World Wide Web (WWW)*, pages 1431–1440, 2017.

49 Marc Roth and Philip Wellnitz. Counting and finding homomorphisms is universal for parameterized complexity theory. In *Proceedings of the Symposium on Discrete Algorithms (SODA)*, pages 2161–2180, 2020.

50 Thomas Schank and Dorothea Wagner. Finding, counting and listing all triangles in large graphs, an experimental study. In *Experimental and Efficient Algorithms*, pages 606–609. Springer Berlin / Heidelberg, 2005.

51 C. Seshadhri and Srikanta Tirthapura. Scalable subgraph counting: The methods behind the madness: WWW 2019 tutorial. In *Conference on the World Wide Web (WWW)*, 2019.

52 Jessica Shi, Laxman Dhulipala, and Julian Shun. Parallel clique counting and peeling algorithms. In *Proceedings of the Conference on Applied and Computational Discrete Algorithms (ACDA)*, pages 135–146, 2021.

53 K. Shin, T. Eliassi-Rad, and C. Faloutsos. Patterns and anomalies in k-cores of real-world graphs with applications. *Knowledge and Information Systems*, 54(3):677–710, 2018.

54 Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Conference on the World Wide Web (WWW)*, pages 607–614, 2011.

55 Johan Ugander, Lars Backstrom, and Jon M. Kleinberg. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *Conference on the World Wide Web (WWW)*, pages 1307–1318, 2013.

56 Stanford Network Analysis Project (SNAP). Available at `http://snap.stanford.edu/`.