Linear Time Subgraph Counting, Graph Degeneracy, and the Chasm at Size Six

Suman K. Bera
University of California, Santa Cruz, CA 95064, USA
sbera@ucsc.edu

Noujan Pashanasangi
University of California, Santa Cruz, CA 95064, USA
npashana@ucsc.edu

C. Seshadhri
University of California, Santa Cruz, CA 95064, USA
sesh@ucsc.edu

Abstract

We consider the problem of counting all k-vertex subgraphs in an input graph, for any constant k. This problem (denoted SUB-CNT$_k$) has been studied extensively in both theory and practice. In a classic result, Chiba and Nishizeki (SICOMP 85) gave linear time algorithms for clique and 4-cycle counting for *bounded degeneracy graphs*. This is a rich class of sparse graphs that contains, for example, all minor-free families and preferential attachment graphs. The techniques from this result have inspired a number of recent practical algorithms for SUB-CNT$_k$. Towards a better understanding of the limits of these techniques, we ask: for what values of k can SUB-CNT$_k$ be solved in linear time?

We discover a chasm at $k = 6$. Specifically, we prove that for $k < 6$, SUB-CNT$_k$ can be solved in linear time. Assuming a standard conjecture in fine-grained complexity, we prove that for all $k \geqslant 6$, SUB-CNT$_k$ cannot be solved even in near-linear time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms; Theory of computation → Graph algorithms analysis

Keywords and phrases Subgraph counting, bounded degeneracy graphs, fine-grained complexity

Digital Object Identifier 10.4230/LIPIcs.ITCS.2020.38

Funding All authors are supported by NSF TRIPODS grant CCF-1740850 NSF CCF-1813165, and ARO Award W911NF1910294.

Acknowledgements We would like to thank David Helmbold for insightful discussions. In particular, David pointed out that the lower bound for counting 8-cycles does not follow from our construction.

1 Introduction

The subgraph counting problem asks for the number of occurrences of a (typically connected) "pattern" subgraph H in a connected input graph G. It is a fundamental algorithmic problem with a rich theory [34, 51, 16, 53, 5, 24, 69, 19], and widely used in practice [32, 18, 58, 15, 60, 33, 59, 40, 68, 62, 67, 8]. With the explosion of network science, subgraph counting is now a fundamental tool used for analyzing real-world graphs. Thus, the search for fast algorithms for subgraph counting is not just a theoretical problem, but one that has many applications in bioinformatics, social sciences, and computer science.

Especially for the many of the practical applications, a common version of subgraph counting is to count the frequency of *all connected subgraphs* with k vertices [57, 54, 2, 25, 26, 31, 47, 61, 37, 71, 71]. We will denote this problem as SUB-CNT$_k$. Even in the theory literature, it is common to parametrize running time by n (vertices in G) and k, so it is natural to study SUB-CNT$_k$. There is a rich line of theoretical work on getting $n^{\mu k}$ time

algorithms, for $\mu < 1$, using matrix multiplication and tree decomposition methods [34, 53, 5, 13, 44, 70, 45, 20, 14, 19]. Unfortunately, SUB-CNT$_k$ is (a generalization of) the canonical #W[1]-hard problem, and it is not believed that there exist $f(k) \cdot n^{o(k)}$ algorithms for SUB-CNT$_k$. From an application standpoint, these algorithms are typically not practical, and do not provide algorithmic guidance. Real-world graphs are massive, and one typically desires linear-time algorithms.

An alternate perspective is to look for faster algorithms for restricted graph classes, and hope that these classes correspond to real-world graphs. A seminal result of Chiba-Nishizeki gave $O(m\kappa^{k-2})$ algorithms for k-clique counting and an $O(m\kappa)$ algorithm for 4-cycle counting, where m is the number of edges in G and κ is the *graph degeneracy* [16]. We leave the technical definitions for later; but κ can be thought of as the maximum average degree of any subgraph of G. Chiba-Nishizeki implicitly prove linear-time algorithms for SUB-CNT$_k$ for $k = 3, 4$ (explicitly shown in [57, 54]). The class of bounded (constant) degeneracy graphs is immensely rich: it contains all minor-closed families, preferential attachment graphs, and bounded expansion graphs. The graph degeneracy appears heavily in network science, and real-world graphs have typically low degeneracy (though maybe not constant).

But most importantly for subgraph counting, the techniques from Chiba-Nishizeki have inspired a number of recent practical subgraph counting algorithms [57, 54, 37, 35].

The problems of SUB-CNT$_k$ for $k \leqslant 5$ have been successfully tackled in practice using these approaches. These algorithms are often tailored for k (using, for example, specific tricks to count individual 4-vertex subgraphs) and it is not clear how far they will extend for larger k.

Towards a better theoretical understanding, we pose the following question.

For what values k, does the SUB-CNT$_k$ *problem admit a linear time algorithm in bounded degeneracy graphs?*

1.1 Our Results

The question above has a surprisingly clean resolution, assuming conjectures from fine-grained complexity. For simplicity, we assume that the input graph G is connected. We assume Las Vegas randomized algorithms, so we talk of expected running times.

Our main theorem asserts linear time algorithms for counting (up to) 5-vertex subgraphs in bounded degeneracy graphs. For counting 6-vertex subgraphs and beyond, it is unlikely that even near-linear time algorithms exists.

▶ **Theorem 1** (The chasm at size 6)**.** *For $k \leqslant 5$, there is an expected $O(m\kappa^{k-2})$ time algorithm for* SUB-CNT$_k$.

Assume the Triangle Detection Conjecture *(Conj. 2). There exists an absolute constant $\gamma > 0$ such that the following holds. For any $k \geqslant 6$ and any function $f : \mathbb{N} \to \mathbb{N}$, there is no (expected) $o(m^{1+\gamma} f(\kappa))$ algorithm for* SUB-CNT$_k$.

The Triangle Detection Conjecture was first stated by Abboud and Williams [1]. They proved many lower bounds for the dynamic version of many well known graph problems such as bipartite perfect matching, single source reachability etc. It is actually believed that the constant γ could be as large as 1/3.

▶ **Conjecture 2** (Triangle Detection Conjecture [1])**.** There exists a constant $\gamma > 0$ such that in the word RAM model of $O(\log n)$ bits, any algorithm to detect whether an input graph on m edges has a triangle requires $\Omega(m^{1+\gamma})$ time in expectation.

1.2 Main Ideas

Conditional Lower Bounds

It is instructive to look at the conditional lower bounds. The reduction of triangle detection to subgraph counting in bounded degeneracy graphs is actually quite simple. Suppose we want to detect (or even count) triangles in an input graph G. Get graph G' by subdividing each edge into two, so a triangle in G becomes a $\mathcal{C}_6$ (6-cycle) in G'. But the degeneracy of G' is just 2! (In any induced subgraph of G', the minimum degree is at most 2, proving the bound.) Thus, if there exists $o(f(\kappa)m^{1+\gamma})$ time algorithms for counting 6-cycles, that would violate the Triangle Detection Conjecture.

It is fairly straightforward to generalize this idea for larger cycles, by replacing edges in G by short paths. Assuming Triangle Detection Conjecture, for all $k \geqslant 6$ and $k \neq 8$, we can rule out linear time algorithms for counting $\mathcal{C}_k$ in bounded degeneracy graphs. Our reduction does not work for $\mathcal{C}_8$; instead we consider a different subgraph for the case of $k = 8$ ($\mathcal{C}_7$ with a tail). We give the details in Section 5.

This reduction fails for counting 5-cycles and in general, it does not work for counting any 5-vertex subgraph. For good reason, as we discovered an efficient algorithm for this problem. This is the more technical part of our paper.

Algorithmic Framework

We present an algorithmic framework for solving the SUB-CNT$_k$ problem, that generalizes the core idea of Chiba and Nishizeki [16]. It is known from past work that their ideas basically provide an $O(m\kappa^{k-2})$ algorithm for SUB-CNT$_k$, for $k = 3, 4$. The main challenge is to get such an algorithm for $k = 5$, thereby nailing down the chasm of Theorem 1. This leads to new results for counting various 5-vertex subgraphs. Perhaps more than these individual results, our main contribution lies in identifying structural decompositions of the pattern subgraphs that allows for efficient algorithms. This decomposition also sheds light on why certain k-vertex subgraphs, for $k \geqslant 6$, does not seem to have any efficient algorithms in bounded arboricity graphs. We give an outline of our framework next, and present it formally in Section 4.

The key idea that comes from Chiba-Nishizeki is to perform subgraph counting on $G^{\rightarrow}$, an acyclic orientation of G where the out degree of each vertex is bounded by $O(\kappa)$[1]. The classic clique and 4-cycle counting algorithms enumerate directed stars and directed paths of length 2 to count subgraphs. We note that the algorithm does *not* enumerate 4-cycles, since there can be $\Omega(n^2)$ 4-cycles. It requires clever indexing to solve this problem, which we generalize in our algorithm.

The crucial generalization of this idea is to enumerate directed rooted trees. Specifically, we count occurrences of a connected pattern H by counting occurrences of all possible acyclic orientations (up to isomorphism) $H^{\rightarrow}$ of H in $G^{\rightarrow}$. The main idea is to find the largest directed rooted tree in $H^{\rightarrow}$, with edges directed away from the root. Call this tree T. Since outdegrees in $G^{\rightarrow}$ are bounded, we can efficiently enumerate all copies of T. Any copy of $H^{\rightarrow}$ in $G^{\rightarrow}$ is formed by extending a copy of T, but $H^{\rightarrow}$ may contain vertices that are not in T. Thus, the extensions could be expensive to compute. But when H has at most 5 vertices, we can prove that $H^{\rightarrow} \setminus T$ is itself either a collection of rooted stars or paths. We can create hash tables that store information about the occurrences of the latter. The final count of $H^{\rightarrow}$ is obtained by enumerating T and carefully combining counts from the hash tables.

[1] Technically, this is *not* the idea of Chiba-Nishizeki, who use the degree orientation. But it was somewhat of a folklore result that it is easy to get the same result using the degeneracy orientation. Arguably the first such reference is Schank-Wagener [63].

2 Related Work

Subgraph counting problems has a long and rich history. More than three decades ago, Itai and Rodeh [34] gave the first non-trivial algorithm for the triangle detection and counting problems with $O(m^{3/2})$ runtime. Subsequently, Chiba and Nishizeki [16] gave an elegant algorithm based on the degree based vertex ordering that solves triangle counting, 4-cycle counting and ℓ-clique counting with running times of $O(m\kappa)$, $O(m\kappa)$, and $O(m\kappa^{\ell-2})$ respectively (κ denotes the degeneracy). In comparison, our algorithm exploits the *degeneracy ordering* of the vertices (see Section 3 for a formal definition); this enables us to create a uniform framework for any k-vertex subgraph for $k \in \{4, 5\}$. In dense graphs, the best bounds for the clique counting problem are achieved by fast matrix multiplications based algorithms [53, 24]; Vassilevska [69] gave combinatorial algorithm with significantly reduced space requirement. For general subgraphs, there is a rich line of research based on matrix multiplication, tree decomposition and vertex cover methods [34, 53, 5, 13, 44, 70, 45, 20, 14, 19] – these works focus on getting $n^{\mu k}$ time algorithmis, for $\mu < 1$.

Subgraph counting problems, specifically triangle counting, clique counting and cycle counting problems, has also been studied extensively in various Big Data models such as property testing model [22, 23, 6], MapReduce settings [17, 66, 42], and streaming model [7, 38, 46, 39, 3, 36, 56, 49, 9]. Most of these work focuses on an approximate count, rather than an exact count. In the applied world, there are many efficient algorithms that are based on clever sampling techniques [11, 10, 33, 75, 61, 73, 72, 37, 71]. Exact counting has also been studied extensively in the applied world [2, 57, 54, 12, 28, 50, 65, 47, 29, 31, 30, 25, 26]. In particular, Ahmed et al. [2] presented an algorithmic framework for solving the SUB-CNT$_4$ problem, called PGD (Parametrized Graphlet Decomposition), which scales to graphs with tens of millions of edges. Pinar et al. [57] studied the SUB-CNT$_5$ problem, and gave the current state of the art ESCAPE library based on degree ordering techniques. However, the provable runtime of their algorithm for certain 5-vertex subgraphs is quadratic, $O(n^2)$. For a deeper exploration of related applied work, refer to the tutorial on subgraph counting by Seshadhri and Tirthapura [64].

The subgraph detection problem, which asks whether an input graph has a copy of the subgraph, is a well-studied problem [34, 51, 4, 5, 41, 45, 74]. For the triangle detection problem, the best known algorithm is based on fast matrix multiplication and it runs in time $O(\min\{n^\omega, m^{2\omega/(\omega+1)}\})$ [5]. If $\omega = 2$, this would give us $O(\min\{n^2, m^{4/3}\})$ algorithm for the triangle detection problem. Hence, to falsify the Triangle Detection Conjecture, it would require a major breakthrough result in the algorithmic graph theory world. For a more detailed discussion on the Triangle Detection Conjecture and its implications, refer to the paper by Abboud and Williams [1].

In the subgraph enumeration problem, the goal is to output each occurrences of the target subgraph. Chiba and Nishizeki [16] showed that it is possible to enumerate all the triangles in a graph along with counting the total number of triangles in $O(m\kappa)$ time. For enumerating all the triangles, $O(m\kappa)$ time is effectively optimal assuming the 3SUM Conjecture [55, 43]. Eppstein [27] studied the bipartite subgraph enumaration problem in bounded arboricity graphs.

3 Preliminaries

In this paper, we study the SUB-CNT$_k$ problem which asks for the number of occurrences of each k-vertex subgraph H, in an input graph G with n vertices and m edges. We consider k to be a constant. For a fixed subgraph H, we use SUB-CNT$_H$ to denote the problem of

counting all occurrences of H in the input graph G. When H is the triangle subgraph, we denote the corresponding counting problem as TRI-CNT. In the context of the SUB-CNT$_k$ problem, we always use G to denote the input graph and H to denote the subgraph to be counted. Both G and H are simple, connected, undirected and unweighted.

In our algorithmic framework, directed graphs play a crucial role. We use $N^+_G(u)$ and $N^-_G(u)$ to denote the out-neighborhood and in-neighborhood of a vertex u in a directed graph G, respectively. We define $d^+_G(u) = |N^+_G(u)|$ and $d^-_G(u) = |N^-_G(u)|$. If the graph is clear from the context, we drop the subscript G.

A graph G is k-degenerate if every subgraph of G has a vertex of degree at most k. The *degeneracy* of a graph G (also called coloring number, refer to Sec. 5.2 of [21]), denoted as $\kappa(G)$, is the smallest integer k such that G is k-degenerate. The *arboricity* of a graph G, denoted as $\alpha(G)$, is the smallest integer k such that the edge set $E(G)$ can be partitioned into k forests. When the graph G is clear from the context, we simply write κ, and α, instead of $\kappa(G)$ and $\alpha(G)$. A classic theorem of Nash-Williams shows that the degeneracy and arboricity are closely related. All our results can be stated in terms of either of the parameters.

▶ **Theorem 3** (Nash-Williams [52]). *In every graph G, $\alpha(G) \leqslant \kappa(G) \leqslant 2\alpha(G) - 1$.*

Vertex ordering is central to many subgraph counting algorithms. In this paper, we work with the *degeneracy ordering* of G, which is defined as follows.

▶ **Definition 4.** Degeneracy ordering *of a graph G, denoted by $\lhd$, is obtained by repeatedly removing the vertex with minimum degree. The ordering is defined by the removal time.*

For example, if $u \lhd v$, then u is removed before v according to the above process. *Degeneracy ordering* can be found in linear time [48].

Using any vertex ordering $\prec$ of an undirected graph G, we construct a directed graph $G^{\rightarrow}_{\prec}$ as follows: for each edge $\{u, v\} \in E(G)$, direct the edge from u to v iff $u \prec v$. We denote this directed edge as (u, v). Observe that $G^{\rightarrow}_{\prec}$ is necessarily acyclic. We denote the directed graph obtained from *degeneracy ordering* $\lhd$ as $G^{\rightarrow}_{\lhd}$. The following two are folklore results about vertex ordering and degeneracy, and can be derived from Prop. 5.2.2 of [21].

▶ **Lemma 5.** *For each vertex $v \in G^{\rightarrow}_{\lhd}$, $d^+(v) \leqslant \kappa$.*

▶ **Lemma 6.** *If there exists a vertex ordering $\prec$ of G such that in the corresponding directed graph $G^{\rightarrow}_{\prec}$, $d^+(v) \leqslant k$ for each vertex v, then $\kappa(G) \leqslant k$.*

Next, we formally define a match (occurrence) of the target subgraph H in the input graph G. We also define a match in the context of directed graphs H' and G'.

▶ **Definition 7.** *A match of H in G is a bijection $\pi : S \to V(H)$ where $S \subseteq V(G)$ and for any two vertices u and v in S, $\{u, v\} \in E(G)$ if $\{\pi(u), \pi(v)\} \in E(H)$.*

▶ **Definition 8.** *A match of H' in G' is a bijection $\pi : S \to V(H')$ where $S \subseteq V(G')$ and for any ordered pair of vertices (u, v) where u and v are in S, $(u, v) \in E(G')$ if $(\pi(u), \pi(v)) \in E(H')$.*

Our algorithm counts matches of H in G by counting matches of all possible acyclic orientations $H^{\rightarrow}$ of H in $G^{\rightarrow}_{\lhd}$. In general, whenever we use '$\rightarrow$' to denote a directed graph, such as in $G^{\rightarrow}_{\lhd}$ and $H^{\rightarrow}$, the directed graph is a DAG.

We denote the number of matches of H in G by $\mathrm{M}(G, H)$. An incomplete match of H in G is an injection $\pi : S \to V(H)$ (so $|S| < |V(H)|$), that has the same properties of a match except being surjective. Consider two incomplete matches (injections) of H, $\pi_1 : S_1 \to V(H)$, and $\pi_2 : S_2 \to V(H)$. Let $V_{\pi_1} = \{\pi_1(u) \mid u \in S_1\}$ and $V_{\pi_2} = \{\pi_2(u) \mid u \in S_2\}$. We say that π_2 completes π_1 to be a match of H, when $V(H) = V_{\pi_1} \cup V_{\pi_2}$ (surjective), $V_{\pi_1} \cap V_{\pi_2} = \emptyset$ (injective), and for any two vertices $u \in S_1$ and $v \in S_2$, $\{u, v\} \in E(G)$ if $\{\pi_1(u), \pi_2(v)\} \in E(H)$. In case of directed graphs, it should hold that $(u, v) \in E(G')$ if $(\pi_1(u), \pi_2(v)) \in E(H')$ and $(v, u) \in E(G')$ if $(\pi_2(v), \pi_1(u)) \in E(H')$.

Two matches are distinct if they are not authomorphims of a match. In other words, two matches π_1 and π_2 of H are equivalent, if they map two automorphisms of the exact same subgraph of G to H. We denote the number of distinct matches of H in G by $\mathrm{DM}(G, H)$. In the SUB-CNT$_k$ problem, we are interested in $\mathrm{DM}(G, H)$ for all k-vertex subgraphs H.

4 Subgraph Counting Through Orientation and Directed Trees

In this section, we discuss our algorithmic framework for solving the SUB-CNT$_k$ problem. Instead of directly counting the number of occurrences of a k-vertex subgraph H in the input graph G, we count the occurrences of all possible DAG $H^{\rightarrow}$ (up to isomorphism) of H in the graph $G^{\rightarrow}_{\triangleleft}$. To achieve this, our main idea is to find the largest directed tree of $H^{\rightarrow}$, enumerate all matches of this tree, and then count matches of the remaining vertices using structures we save in a hash table. In Section 4.1, we show that our framework solves the SUB-CNT$_5$ problem in expected $O(m\kappa^3)$ time. In Section 4.2, we demonstrate the limitation of our framework as it fails to solve the SUB-CNT$_{\mathcal{C}_6}$ problem *efficiently*.

Algorithm 1 Counting distinct matches of all 5-vertex subgraphs in G (SUB-CNT$_5$).

procedure COUNT-ALL-5(G)
  Derive $G^{\rightarrow}_{\triangleleft}$ by orienting $E(G)$ with respect to degeneracy ordering.
  for all connected 5-vertex subgraphs H except 4-star **do**
    Run COUNT-MATCH($G^{\rightarrow}_{\triangleleft}, H$) and save the result for H.
  Save $\sum_{u \in V(G)} \binom{d(u)}{4}$ for 4-star.

Algorithm 2 Counting distinct matches of H in G (SUB-CNT$_H$).

procedure COUNT-MATCH($G^{\rightarrow}_{\triangleleft}, H$)
  $\mathrm{DM}(G, H) \leftarrow 0$
  for all possible DAGs (up to isomorphism) $H^{\rightarrow}$ of H **do**
    $\mathrm{M}(G^{\rightarrow}_{\triangleleft}, H^{\rightarrow}) \leftarrow 0$
    Find one of the largest DRTSs in $H^{\rightarrow}$, and call it $T_{\max}$.
    for all match π of $T_{\max}$ in $G^{\rightarrow}_{\triangleleft}$ **do**
      if π is a match of $H^{\rightarrow}$ **then** ▷ $V(T_{\max}) = V(H^{\rightarrow})$. Lemma 14
        $\mathrm{M}(G^{\rightarrow}_{\triangleleft}, H^{\rightarrow}) \leftarrow \mathrm{M}(G^{\rightarrow}_{\triangleleft}, H^{\rightarrow}) + 1$
      else if π is an incomplete match of $H^{\rightarrow}$ **then** ▷ Lemma 14
        $k \leftarrow$ number of ways to complete π to a match of $H^{\rightarrow}$. ▷ Lemma 16
        $\mathrm{M}(G^{\rightarrow}_{\triangleleft}, H^{\rightarrow}) \leftarrow \mathrm{M}(G^{\rightarrow}_{\triangleleft}, H^{\rightarrow}) + k$
    $\mathrm{DM}(G, H) \leftarrow \mathrm{M}(G^{\rightarrow}_{\triangleleft}, H^{\rightarrow}) / |Aut(H^{\rightarrow})|$
  return $\mathrm{DM}(G, H)$

4.1 5-vertex Subgraph Counting

Our main algorithmic result is given in the following theorem.

▶ **Theorem 9.** *There is an algorithm that solves the* SUB-CNT$_5$ *problem in* $O(m\kappa^3)$ *time.*

Our strategy is to count matches of all possible DAGs (up to isomorphism) $H^{\rightarrow}$ of H in $G^{\rightarrow}_{\triangleleft}$, to obtain the number of distinct matches of H in G. Alg. 2 demonstrates this subroutine of our algorithm for SUB-CNT$_5$, which is shown in Alg. 1. First, we find one of the largest directed rooted tree subgraphs (*DRTS*), which we define as follows, in $H^{\rightarrow}$.

▶ **Definition 10.** *Given any directed graph D, a directed rooted tree subgraph (DRTS) of D, is a subgraph T of D, where the underlying undirected graph of T is a rooted tree, and edges are oriented away from the root in T.*

The following lemma shows that we can find all matches of any DRTS in $H^{\rightarrow}$ in the desired time.

▶ **Lemma 11.** *Let T be a directed tree with k vertices. All matches of T in $G^{\rightarrow}_{\triangleleft}$ can be enumerated in $O(m\kappa^{k-2})$.*

Proof. Let $t_1, \ldots, t_k$ be a BFS ordering of T starting at the root t_1. Fix an edge $(u, v) \in E(G^{\rightarrow}_{\triangleleft})$ and map u to t_1 and v to t_2. There are m possible matches for (t_1, t_2), which we can find by enumerating the edges of $G^{\rightarrow}_{\triangleleft}$. Now, we will choose vertices to map to $t_3, \ldots, t_k$, one by one, in this order. Since the out-degree of each vertex in $G^{\rightarrow}_{\triangleleft}$ is at most κ, if we have already mapped vertices to $t_1, \ldots, t_i$, there are at most κ vertices that could be mapped to t_{i+1}. Therefor $\mathrm{M}(G^{\rightarrow}_{\triangleleft}, T) = O(m\kappa^{k-2})$, and we can enumerate all of them by first choosing (u, v) to map to (t_1, t_2) and then choosing vertices to map to $t_3, \ldots, t_k$, in this order and one by one. ◀

▶ **Observation 12.** *Call a vertex v of a directed graph a* source *vertex, if $d^-(v) = 0$. Consider T to be one of the largest DRTSs of a DAG D. T has to have a source vertex of D as the root, otherwise the root has an in-neighbor v, which is not in T as it would create a cycle. Adding v to T creates a new DRTS which has one more vertex than T. This contradicts the fact that T is one of the largest DRTSs of D. Hence, the root of T has to be a source vertex of D.*

Given a 5-vertex DAG $H^{\rightarrow}$, we can find a DRTS that has the most number of vertices among all DRTSs of $H^{\rightarrow}$ in constant time. First, find all source vertices, and then apply a Breath First Search (BFS) starting from each of these vertices and pick a BFS tree with the most number of vertices among all. The following lemma shows that the largest DRTS has at least 3 vertices for a 5-vertex connected subgraphs, except 4-star. Notice that, the largest DRTS of a 4-star with all the edges oriented towards the center has two vertices.

▶ **Lemma 13.** *Let H be a connected undirected 5-vertex graph that is not a 4-star. Each largest DRTS of any DAG $H^{\rightarrow}$, which is an acyclic orientation of H, has at least three vertices.*

Proof. We prove this lemma by contradiction. Assume that any DRTS of $H^{\rightarrow}$ has at most two vertices. A directed 2-path, or any vertex with at least two outgoing edges result in a DRTS with three vertices. Therefore,

(a) $H^{\rightarrow}$ does not have a 2-path,

(b) each vertex in $H^{\rightarrow}$ has at most one outgoing edges.

Notice that, since $H^{\rightarrow}$ is a DAG, it has at least one source vertex. Consider a source vertex u. Since H is connected, u has at least one neighbor, and by (b) it should have exactly one neighbor. Let $N^+(u) = \{v\}$, then $N^+(v) = \emptyset$, by (a). So, v should have at least one incoming neighbor w. By (a), w has no incoming edges, and it has no outgoing edges by (b). Call the other two vertices x and y. As H is connected, there should be a connection between $\{u, v, w\}$ and $\{x, y\}$. u and w cannot have any neighbor other than v, so x and y could only be connected to v. Since H is not a star, there should be an edge between x and y. Without loss of generality, let (x, y) be that edge. By (a), $(y, v) \notin E(H^{\rightarrow})$ and by (b) $(x, v) \notin E(H^{\rightarrow})$. So, $\{u, v, w\}$ is not connected to $\{x, y\}$, and H is disconnected, which is a contradiction. Thus, the assumption that any DRTS of $H^{\rightarrow}$ has at most two vertices is wrong, and each largest DRTS of $H^{\rightarrow}$ has at least three vertices. ◀

So far, we know that we can find one of the largest DRTSs of H, which has at least 3 vertices. We use $T_{\max}$ to denote this DRTS. By Lemma 11, we can enumerate all matches of $T_{\max}$ in $G^{\rightarrow}_{\triangleleft}$ in $O(m\kappa^3)$ time. For each such match, we need to validate whether it is a (incomplete) match of $H^{\rightarrow}$ or not. If it is not, then it could not be completed to a match of $H^{\rightarrow}$. The following lemma shows that we can perform this validation efficiently. In the remaining part of this section, "constanct expected time", refers to constant amortized time access to hash maps that we use.

▶ **Lemma 14.** *Let T be a DRTS of a DAG $H^{\rightarrow}$ of a connected k-vertex graph H. Assume edges of $G^{\rightarrow}_{\triangleleft}$ are saved in a hash table. For each match π of T in $G^{\rightarrow}_{\triangleleft}$, it takes $O(|E(H^{\rightarrow})|)$ expected time to validate whether π is a (incomplete) match of $H^{\rightarrow}$ or not.*

Proof. Since π is a bijection, it has an inverse which we denote by π^{-1}. Let $H^{\rightarrow}[V(T)]$ denote the subgraph of $H^{\rightarrow}$ induced on $V(T)$. Observe that, there could be edges in $H^{\rightarrow}[V(T)]$ not present in T. For π to be a match (if $V(T) = V(H^{\rightarrow})$) or incomplete match of $H^{\rightarrow}$, these edges have to be present between corresponding vertices in $G^{\rightarrow}_{\triangleleft}$ mapped to T by π. Formally, consider all ordered pairs of vertices $(a, b) \in V(T) \times V(T)$ such that $(a, b) \in E(H^{\rightarrow})$ and $(a, b) \notin E(T)$, π is a match or incomplete match of $H^{\rightarrow}$ iff $(\pi^{-1}(a), \pi^{-1}(b)) \in E(G^{\rightarrow}_{\triangleleft})$ for all such pairs of vertices. To validate this, we enumerate all edges (a, b) of $H^{\rightarrow}[V(T)]$ which are not present in T, and search for $(\pi^{-1}(a), \pi^{-1}(b))$ in hashed edges of $G^{\rightarrow}_{\triangleleft}$ in expected constant time. So this only requires $O(|E(H^{\rightarrow})|)$ expected time. ◀

If $V(T_{\max}) = V(H^{\rightarrow})$, then a match of $T_{\max}$ could be a match of $H^{\rightarrow}$ too, which could be verified as explained. If there is a vertex in $H^{\rightarrow}$ which is not present in $T_{\max}$, then after validating that a match of $T_{\max}$ is an incomplete match of $H^{\rightarrow}$, we need to find the number of ways to complete it to a match of $H^{\rightarrow}$. For this we need to count matches of each possible structures that $T_{\max}$ does not cover in $H^{\rightarrow}$. We save the count of these structures in $G^{\rightarrow}_{\triangleleft}$, in hash tables. The following lemma shows that this can be done efficiently.

▶ **Lemma 15.** *In $O(m\kappa^3)$ time and space, we can save all the following key and value pairs in hash maps $\mathcal{HM}_1$, $\mathcal{HM}_2$, and $\mathcal{HM}_3$.*

1. $\mathcal{HM}_1 : ((u, v), 1)$ *where* $(u, v) \in E(G^{\rightarrow}_{\triangleleft})$
2. $\mathcal{HM}_2 : (S, k)$ $\forall S \subseteq V(G^{\rightarrow}_{\triangleleft})$ *where* $1 \leqslant |S| \leqslant 4$*, and k is the number of vertices u such that* $S \subseteq N^+(u)$
3. $\mathcal{HM}_3 : ((S_1, S_2), \ell)$ $\forall S_1, S_2 \subseteq V(G^{\rightarrow}_{\triangleleft})$*, where* $1 \leqslant |S_1 \cup S_2| \leqslant 3$*, and ℓ is the number of edges $e = (u, v) \in E(G^{\rightarrow}_{\triangleleft})$ such that $S_1 \subseteq N^+(u)$ and $S_2 \subseteq N^+(v)$.*

Proof. We show how to enumerate and save all these structures in $\mathcal{HM}_1$, $\mathcal{HM}_2$, and $\mathcal{HM}_3$.

1. $\mathcal{HM}_1$: We can easily do this in $O(m)$ by enumerating the out-neighbors of each vertex
2. $\mathcal{HM}_2$: For each edge $e = (u, v)$, we can enumerate all subsets T of the set $\{w \in N^+(u) \mid v \lhd w\}$, where $|T| \leqslant 3$, in $O(\kappa^3)$ time, and increment the value for the key $T \cup \{v\}$ in the hash map by one.
3. $\mathcal{HM}_3$: For each edge $e = (u, v)$ $(v \in N^+(u))$, we enumerate all possible subset $S_1 \subseteq N^+(u) \setminus \{v\}$ where $|S_1| \leqslant 3$. And, for each S_1 we enumerate all possible $S_2 \setminus S_1$ in subsets of $N^+(v)$, such that $1 \leqslant |S_1 \cup S_2| \leqslant 3$. This takes $O(\kappa^3)$ as the out-degree of each vertex is at most κ, and we choose up to three vertices. All possible $S_1 \cap S_2$ can be determined by checking the connection between v and each vertex in S_1 using the hashed edges of $G^{\rightarrow}$ in $\mathcal{HM}_1$. ◀

The following lemma shows that we can count the number of ways to complete a match of $T_{\max}$, which is also an incomplete match of $H^{\rightarrow}$, to a match of $H^{\rightarrow}$ efficiently.

▶ **Lemma 16.** *Let H be a 5-vertex connected graph, $H^{\rightarrow}$ be a DAG of H, and $T_{\max}$ be one of the largest DRTSs in $H^{\rightarrow}$. Assume $\mathcal{HM}_1$, $\mathcal{HM}_2$, and $\mathcal{HM}_3$ are given. For each match π of $T_{\max}$ in $G^{\rightarrow}_{\lhd}$ which is an incomplete match of $H^{\rightarrow}$, we can count the number of ways to complete π to a match of $H^{\rightarrow}$ in expected constant time.*

Proof. By Lemma 13, $T_{\max}$ has at least 3 vertices, and since π is an incomplete match (not a match) of $H^{\rightarrow}$, we can assume that $|V(T_{\max})| < 5$. Observe that, $T_{\max}$ is a maximal DRTS. Any vertex in $H^{\rightarrow}$ which is not in $T_{\max}$ can only be connected to vertices of $T_{\max}$ by outgoing edges, otherwise they could be added to $T_{\max}$ to create a larger DRTS of $H^{\rightarrow}$, which contradicts the maximality of $T_{\max}$. We consider two cases where $T_{\max}$ has three or four vertices.

Let $|V(T_{\max})| = 4$, and i be the only vertex in $H^{\rightarrow}$ that is not in $T_{\max}$. To complete π to a match of $H^{\rightarrow}$, we need to choose a vertex in $G^{\rightarrow}_{\lhd}$, that is connected by outgoing edges to vertices mapped to the out-neighborhood of i in $H^{\rightarrow}$. Let $S_i = \{\pi^{-1}(t) \mid t \in N^+_{H^{\rightarrow}}(i)\}$. $\mathcal{HM}_2(S_i)$ is the number of vertices that could be mapped to i, but some of them may be already mapped to a vertex in $T_{\max}$, by π. Let r_i denote the number of vertices $v \in \{\pi^{-1}(t) \mid t \in V(T_{\max})\}$, where $S_i \subseteq N^+_{G^{\rightarrow}_{\lhd}}(v)$. We can obtain r_i in expected constant time, by enumerating vertices mapped to $V(T_{\max})$, and counting vertices that are connected to all vertices in S_i. For any vertex, we can check the connection to each vertex of S_i using $\mathcal{HM}_1$ in expected constant time. The number of ways to complete π to a match of $H^{\rightarrow}$ in this case is $\mathcal{HM}_2(S_i) - r_i$.

Now we consider the case where $|V(T_{\max})| = 3$. Let $V(H^{\rightarrow}) \setminus V(T_{\max}) = \{i, j\}$. To complete π to a match of $H^{\rightarrow}$, we only need to choose two vertices of $G^{\rightarrow}_{\lhd}$ to map to i and j. Let $S_i = \{\pi^{-1}(t) \mid t \in V(T_{\max}) \cap N^+_{H^{\rightarrow}}(i)\}$ and $S_j = \{\pi^{-1}(t) \mid t \in V(T_{\max}) \cap N^+_{H^{\rightarrow}}(j)\}$. We consider two cases, where i and j are connected or not. If they are connected, without loss of generality, assume $(i, j) \in E(H^{\rightarrow})$. If $(i, j) \in E(H^{\rightarrow})$, then we can use $\mathcal{HM}_3$ in Lemma 15, to find the number of edges (u, v) where u and v could be mapped to i and j, respectively. Let $r_{(i,j)}$ be the number of edges $e = (w, x) \in E(G^{\rightarrow}_{\lhd})$, where w and x are mapped to vertices in $T_{\max}$ by π, such that, $S_i \subseteq N^+_{G^{\rightarrow}_{\lhd}}(w)$, and $S_j \subseteq N^+_{G^{\rightarrow}_{\lhd}}(x)$. We can obtain $r_{(i,j)}$ in expected constant time using $\mathcal{HM}_1$. Then the number of edges (u, v) that could be mapped to (i, j) is $\mathcal{HM}_3((S_i, S_j)) - r_{(i,j)}$. Next case is when $(i, j) \notin E(H^{\rightarrow})$. In this case, we use $\mathcal{HM}_2$ to find the number of pair of vertices of $G^{\rightarrow}_{\lhd}$ which could be mapped to i and j. Let r_i (r_j resp.) denote the number of vertices $v \in V(G^{\rightarrow}_{\lhd})$ where v is mapped to a vertex in $T_{\max}$ and $S_i \subseteq N^+_{G^{\rightarrow}_{\lhd}}(v)$ ($S_j \subseteq N^+_{G^{\rightarrow}_{\lhd}}(v)$ resp.). Also, we use $r_{i,j}$ to denote the number of vertices $v \in V(G^{\rightarrow}_{\lhd})$ that are counted in both r_i and r_j, meaning

$S_i \cup S_j \subseteq N^+_{G^{\rightarrow}_{\lhd}}(v)$. We can obtain r_i, r_j, and $r_{i,j}$ easily in expected constant time using $\mathcal{HM}_1$. The number of pairs of vertices which could be mapped to i and j is equal to $(\mathcal{HM}_2(S_i) - r_i) \cdot (\mathcal{HM}_2(S_j) - r_j) - (\mathcal{HM}_2(S_i \cup S_j) - r_{i,j})$. ◀

Now, we have all the tools to efficiently count distinct matches of a DAG of $H^{\rightarrow}$ in $G^{\rightarrow}_{\lhd}$. The following lemma shows that we can do this in $O(m\kappa^3)$ expected time.

▶ **Lemma 17.** *There is an algorithm which counts distinct matches for each possible DAG (up to isomorphism) $H^{\rightarrow}$ of a 5-vertex connected subgraphs H, in $O(m\kappa^3)$ expected time.*

Proof. Fix a DAG $H^{\rightarrow}$ of H. If H is a 4-star and $H^{\rightarrow}$ has ℓ incoming neighbors, then the number of distinct matches of $H^{\rightarrow}$ is $\sum_{u \in V(G^{\rightarrow}_{\lhd})} \binom{d^-(u)}{\ell}\binom{d^+(u)}{4-\ell}$. Assume that H is not a 4-star. Find a DRTS of $H^{\rightarrow}$ with the most number of vertices among all its DRTSs, and call it $T_{\max}$. This can be done in constant time for $H^{\rightarrow}$. By Lemma 13, $T_{\max}$ has at least three vertices. We will now enumerate all matches of $T_{\max}$ in $G^{\rightarrow}_{\lhd}$. By Lemma 11, this step requires $O(m\kappa^3)$ expected time. For each match π of $T_{\max}$ in $G^{\rightarrow}_{\lhd}$, we can verify whether π is a match (if ($|V(T_{\max})| = 5$) or incomplete match of $H^{\rightarrow}$ in expected constant time, by Lemma 14. If $|V(T_{\max})| = 5$, while enumerating all matches of $T_{\max}$, we only count them if they are a match of $H^{\rightarrow}$. So in this case we can count $\mathrm{M}(G^{\rightarrow}_{\lhd}, H^{\rightarrow})$ in $O(m\kappa^3)$ expected time.

Otherwise, $T_{\max}$ has 3 or 4 vertices. In this case, for each match π of $T_{\max}$, we first verify that it is also an incomplete match of $H^{\rightarrow}$. Then, we count the number of ways to complete π to a match of $H^{\rightarrow}$, which we can do in expected constant time, by Lemma 16. To obtain $\mathrm{M}(G^{\rightarrow}_{\lhd}, H^{\rightarrow})$, we simply sum the ways to complete each incomplete match we have found, to a match of $H^{\rightarrow}$.

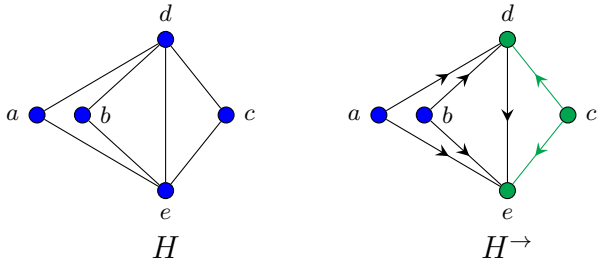
This approach gives us the number of all (not necessarily distinct) matches of $H^{\rightarrow}$ in $G^{\rightarrow}_{\lhd}$. Let $H^{\rightarrow}_{\pi}$ be a subgraph of $G^{\rightarrow}_{\lhd}$ that π maps to $H^{\rightarrow}$. Each automorphism of $H^{\rightarrow}$, gives a new match π' which is not distinct from π, as it is still mapping H_π (the same copy of H) to $H^{\rightarrow}$ (example in Fig. 1b). As each match of $H^{\rightarrow}$, also maps vertices to $T_{\max}$, resulting in a match of $T_{\max}$ and an (incomplete) match of $H^{\rightarrow}$, we will find all distinct matches of $H^{\rightarrow}$ and count each one exactly $|Aut(H^{\rightarrow})|$ times. We want the number of distinct matches, which we can obtain by dividing the count of all matches by $|Aut(H^{\rightarrow})|$.

Thus, it requires $O(m\kappa^3)$ expected time to create $\mathcal{HM}_1$, $\mathcal{HM}_2$, and $\mathcal{HM}_3$ by Lemma 15, $O(m\kappa^3)$ time for enumerating matches of $T_{\max}$, expected constant time to validate these matches, and expected constant time for counting ways to complete each such match, that is verified to be an incomplete match of $H^{\rightarrow}$, to a match of $H^{\rightarrow}$. So overall, we can find $\mathrm{DM}(G^{\rightarrow}_{\lhd}, H^{\rightarrow})$ in $O(m\kappa^3)$ expected time.

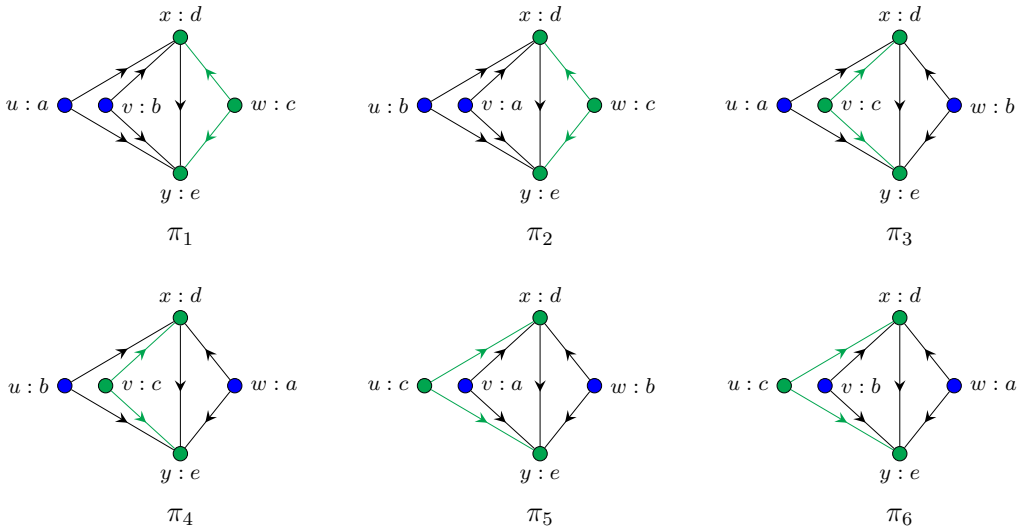
This completes the proof of this lemma. ◀

Lastly, we can prove Theorem 9 as follows.

Proof of Theorem 9. Given a 5-vertex connected subgraph H, we can count all distinct matches of each possible DAG $H^{\rightarrow}$ of H, in $G^{\rightarrow}_{\lhd}$ in $O(m\kappa^3)$ expected time, by Lemma 17. To count all distinct matches of H in G, we just need to sum the number of distinct matches of all possible DAGs (up to isomorphism) of H. The number of such DAGs is constant for H. There are 21 different connected 5-vertex subgraphs (illustrated in [57]), and we perform this process on all of them. This completes the proof of the theorem. ◀



(a) H is 5-vertex connected subgraph and $H^{\rightarrow}$ is one possible acyclic orientation of it. $T_{\max}$ (largest DRTS of $H^{\rightarrow}$) is shown in green and contains three vertices.



(b) All six figures show exactly the same subgraph in $G^{\rightarrow}_{\triangleleft}$. $\pi_1, \ldots, \pi_6$ are six equivalent matches of $H^{\rightarrow}$ in $G^{\rightarrow}_{\triangleleft}$, one for each automorphism of $H^{\rightarrow}$. Notice (u, v, w) being mapped to all permutations of (a, b, c).

Figure 1 Application of Alg. 2 on a DAG $H^{\rightarrow}$ of an example 5-vertex connected subgraph H.

4.2 Limitations of Our Framework for a Six Vertex Subgraph

Consider $\mathcal{C}_6$, shown as H in Fig. 2. Then $H^{\rightarrow}$, shown in the right side of Fig. 2, is a possible DAG of H. In $H^{\rightarrow}$, s_1, s_2, and s_3 are the source vertices, and t_1, t_2, and t_3 are the sink vertices. Any DRTS of $H^{\rightarrow}$ has at most three vertices, and there are three such DRTS, T_1, T_2, and T_3 rooted at s_1, s_2 and s_3, respectively. T_1 is shown by red in Fig. 2. For each of T_1, T_2, and T_3, the remaining vertices include a vertex, with two incoming edges, which we call an in-in wedge. For example, t_2 is such a vertex for T_1. Even graphs with bounded degeneracy can have $\Omega(n^2)$ in-in wedges. We cannot hash the count of such structures in expected time bounded by m and κ. So, Alg. 2 fails to count occurrences of $\mathcal{C}_6$ in the desired time. In the next section, we discuss why such limitations are natural to any framework for the SUB-CNT$_k$ problem at and beyond $k = 6$.

5 A Chasm at Six

At the end of the previous section, we showed the limitations of our framework in counting certain 6-vertex subgraphs. In this section, we show that perhaps such limitations are fundamental to any subgraph counting algorithms. In particular, the landscape of SUB-CNT$_k$ problem in the bounded degeneracy graphs changes dramatically as we move beyond $k = 5$.

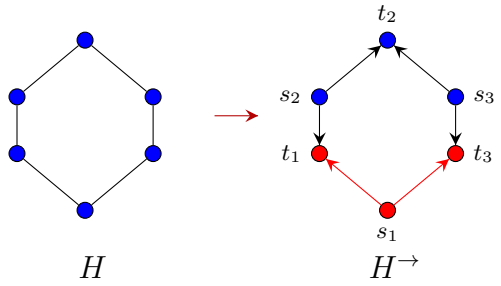


Figure 2 Let $H^{\rightarrow}$ be a DAG of H ($\mathcal{C}_6$). Considering any largest DRTS of $H^{\rightarrow}$, the remaining vertices include a vertex with two incoming edges (in-in wedge). Even graphs with bounded degeneracy can have $\Omega(n^2)$ in-in wedges. So hashing in Alg. 2 will not be bounded by m and κ for H.

We prove that for every integer $k \geqslant 6$, there exists a k-vertex subgraph H such that, the running time of any algorithm for the SUB-CNT$_H$ problem does not depend on the degeneracy of the input graph, assuming the Triangle Detection Conjecture. In contrast, for $k \leqslant 5$, $O(m\kappa^{k-2})$ algorithms exists for SUB-CNT$_k$ (see Section 4). The following theorem captures the main result of this section.

▶ **Theorem 18.** *Assume the* Triangle Detection Conjecture *(Conj. 2). There exists an absolute constant $\gamma > 0$ such that the following holds. For any $k \geqslant 6$ and any function $f : \mathbb{N} \to \mathbb{N}$, there exists a k-vertex subgraph H such that there is no (expected) $o(m^{1+\gamma} f(\kappa))$ algorithm for* SUB-CNT$_H$.

Outline of the Proof. For each $k \geqslant 6$ and $k \neq 8$, the subgraph of interest will be the k-cycle graph, $\mathcal{C}_k$. For $k = 8$, the subgraph of interest will be the $\mathcal{C}_7$ with a tail (see Figure 3). We first give a proof outline. Fix some $k \geqslant 6$ and let H_k denote the target subgraph of size k. Recall the TRI-CNT problem — count the number of triangles in a graph with m edges. Conjecture 2 asserts that for any algorithm $\mathcal{A}$ for the TRI-CNT problem, $T(\mathcal{A}) = \omega(m)$ where $T(\mathcal{A})$ denotes the worst case time complexity of the algorithm $\mathcal{A}$. Our strategy is to reduce from the TRI-CNT problem to the SUB-CNT$_{H_k}$ problem. To this end, we construct a new graph G_k from the input instance G of the TRI-CNT problem such that G_k has $O(m)$ edges, and has degeneracy at most 2. More importantly, the number of triangles in G is a simple linear function of the number of H_k in G_k. Hence, we can derive the number of triangles in G by counting the number of H_k in G_k. As $\kappa(G_k) \leqslant 2$, any $O(mf(\kappa))$ algorithm for the SUB-CNT$_{H_k}$ problem translates to a $O(m)$ algorithm for the TRI-CNT problem, contradicting the Triangle Detection Conjecture. We remark that, for $k = 8$, our proof strategy will be slightly different — instead of reducing from the TRI-CNT problem, we shall reduce from the triangle detection problem itself. However, the gadget construction will follow the same basic principle.

The construction of G_k from G is rather simple. The details of the construction depends on whether k is a multiple of 3 or not. We take two examples to describe the construction.

First, we take $k = 6$, and the target subgraph $H_6 = \mathcal{C}_6$. For each edge e in $E(G)$, we replace e with a length two path $\{e_1, e_2\}$ in $E(G_6)$. To accomplish this, we add a new vertex v_e for each edge: $V(G_6) = V(G) \cup \{v_e\}_{e \in E(G)}$. This is shown in Figure 4a. Each triangle in G creates a $\mathcal{C}_6$ in G_6. We formally prove in Lemma 20 that the number of triangles in G is same as the number of $\mathcal{C}_6$ in G_6. In Lemma 19, we bound the degeneracy of G_6 by 2. This construction can be generalized for any $k = 3\ell$ where $\ell \geqslant 2$, by replacing each edge in $E(G)$ with ℓ-length path.

Next consider the case $k = 7$. For each edge $e \in E(G)$, we first create two parallel copies of e, and then replace the first one with a length two path $\{e_{1,1}, e_{1,2}\}$, and the second one with a length three path $\{e_{2,1}, e_{2,2}, e_{2,3}\}$. So in $E(G_7)$, we have 5 edges for each edge in $E(G)$. We create 3 new vertices per edge to accomplish this, and denote them as v_e, u_{e_1}, u_{e_2}. See Figure 4b for a pictorial demonstration. In Lemma 20, we argue that the number of $\mathcal{C}_7$ is exactly 3 times the number of triangles in G. In Lemma 19, we bound the degeneracy of G_7 by 2. This construction generalizes to any $k = 3\ell + i$ where $\ell \geqslant 2$ and $i \in \{1, 2\}$ (except for the case when $k = 8$, that is $\ell = 2$ and $i = 2$) by splitting each edge into ℓ and $\ell + 1$ many parts respectively.

Finally, we consider the case of $k = 8$. Note that the target subgraph H_8 is the 7-cycle with a tail in this case (see Figure 3). It is natural to wonder why do we not simply take $H_8 = \mathcal{C}_8$? After all, for all other values of k, taking $H_k = \mathcal{C}_k$ suffices. At a first glance, it seems like if we consider the same graph G_7 as described above (and in Figure 4b) the number of $\mathcal{C}_8$ would be a simple linear function of the number of triangles in G — for each triangle in G, there will be exactly three $\mathcal{C}_8$ in G_7. However, each $\mathcal{C}_4$ in G would also lead to a $\mathcal{C}_8$ in G_8. Observe that for $k > 8$, we do not run into this problem. A more formal treatment of this issue appear in Section 5.

So instead, we take H_8 to be the subgraph $\mathcal{C}_7$ with a tail to prove our conditional lower bound for SUB-CNT$_8$. The construction of the graph G_8 remains exactly the same as that of G_7. We show in Lemma 21 that, there exists a $\mathcal{C}_7$ with a tail in G_8 if and only if there exist a triangle in G.

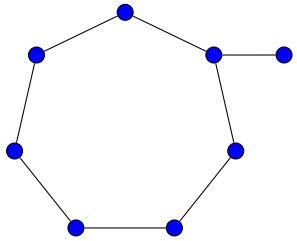


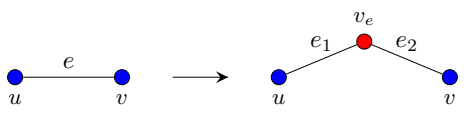
Figure 3 Target subgraph for proving conditional lower bounds for SUB-CNT$_8$: the $\mathcal{C}_7$ with a tail

We now present the proof of Theorem 18 in full details.

Proof of Theorem 18. Fix some $k \geqslant 6$. Let the subgraph H_k denote the target subgraph of size k. For $k \neq 8$, H_k is $\mathcal{C}_k$, and for $k = 8$, H_k is $\mathcal{C}_7$ with a tail (see Figure 3). We reduce from the TRI-CNT problem to the SUB-CNT$_{H_k}$. Let $G = (V, E)$ be the input instance for the TRI-CNT problem with $|V| = n$ and $|E| = m$. We construct an input instance $G_k = (V_k, E_k)$ for the SUB-CNT$_{H_k}$ problem from G. The construction of G_k differs based on whether k is divisible by 3 or not. We next consider these two cases separately.

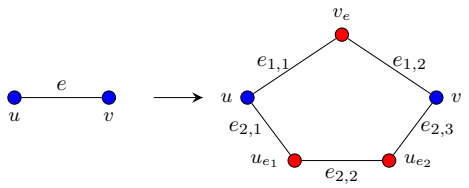
Details of the Reduction. First assume $k = 3\ell$ for some integer $\ell \geqslant 2$. We first define the vertex set V_k. For each vertex in V, we add a vertex in V_k. For each edge $e \in E$, we add a set of $\ell - 1$ many vertices, denoted as $V_e = \{v_{e,1}, v_{e,2}, \ldots, v_{e,\ell-1}\}$. We collect all these second type of vertices into the set V_E. Formally, we have

$$V_k = V \cup V_E\,,$$
$$\text{where } V_E = \bigcup_{e \in E} V_e\,,$$
$$\text{for } V_e = \{v_{e,1}, v_{e,2}, \ldots, v_{e,\ell-1}\}\,.$$



$e \in E(G)$ $\quad e_1, e_2 \in E(G_6)$

(a) Construction of the edge set $E(G_6)$ from the edge set $E(G)$. The red colored nodes are only present in $V(G_6)$, and not in $V(G)$.



$e \in E(G)$ $\quad e_{i,j} \in E(G_7)$

(b) Construction of the edge set $E(G_7)$ from the edge set $E(G)$. The red colored nodes are only present in $V(G_7)$, and not in $V(G)$.

Figure 4 Reduction from the TRI-CNT problem to the SUB-CNT$_{\mathcal{C}_k}$ problem for $k = 6$ (left) and $k = 7$ (right).

We now describe the edge set E_k. We treat each edge $e = \{u, v\} \in E$ as an ordered pair (u, v) where the ordering can be arbitrary of the vertices (for example, assume lexicographical ordering). Now for each edge $e = (u, v)$ construct an ℓ-length path between u and v in V_k by connecting the vertices in $\{u\} \cup V_e \cup \{v\}$ sequentially. More precisely, we define E_k as follows.

$$E_k = \bigcup_{e \in E} E_e \,,$$
$$\text{where } E_e = \{\{u, v_{e,1}\}, \{v_{e,1}, v_{e,2}\}, \ldots, \{v_{e,\ell-2}, v_{e,\ell-1}\}, \{v_{e,\ell-1}, v\}\} \text{ for } e = (u, v)\,.$$

This completes the construction of the graph $G_k = (V_k, E_k)$. We give an example in Figure 4a for $k = 6$.

Now assume $k = 3\ell + i$ for some some integer $\ell \geqslant 2$ and $i \in \{1, 2\}$. In the previous case, we added a set of $\ell - 1$ many vertices for each edge in E. But now, for each edge $e \in E$, we add two sets of vertices, one with $\ell - 1$ many vertices and the other with ℓ many vertices. We denote the first set as $V_e = \{v_{e,1}, v_{e,2}, \ldots, v_{e,\ell-1}\}$, and the second set as $U_e = \{u_{e,1}, u_{e,2}, \ldots, u_{e,\ell}\}$. We also add the set of vertices in V to V_k. Formally, we have

$$V_k = V \cup V_E \,,$$
$$\text{where } V_E = \bigcup_{e \in E} V_e \cup U_e \,,$$
$$\text{for } V_e = \{v_{e,1}, v_{e,2}, \ldots, v_{e,\ell-1}\} \,,$$
$$\text{and } U_e = \{u_{e,1}, u_{e,2}, \ldots, u_{e,\ell}\}\,.$$

To construct the edge set E_k, as before we treat each edge in $e = \{u, v\} \in E$ as an ordered pair (u, v) according to some arbitrary ordering of the vertices. Now, for each edge $e = (u, v)$, construct an $2\ell + 1$-length cycle between u and v in V_k by creating a ℓ-length path via the vertices in V_e and another $\ell + 1$-length path via the vertices in U_e. We denote the corresponding edge sets as $E_{V,e}$ and $E_{U,e}$ respectively. Formally, we define E_k as follows.

$$E_k = \bigcup_{e \in E} (E_{V,e} \cup E_{U,e}) \,,$$
$$\text{where } E_{V,e} = \{\{u, v_{e,1}\}, \{v_{e,1}, v_{e,2}\}, \ldots, \{v_{e,\ell-2}, v_{e,\ell-1}\}, \{v_{e,\ell-1}, v\}\} \,,$$
$$\text{and } E_{U,e} = \{\{u, u_{e,1}\}, \{u_{e,1}, u_{e,2}\}, \ldots, \{u_{e,\ell-1}, u_{e,\ell}\}, \{u_{e,\ell}, v\}\} \text{ for } e = (u, v)\,.$$

This completes the construction of the graph $G_k = (V_k, E_k)$. Note that the construction is independent of the value of i. Hence, we produce the same graph G_k for $k = 3\ell + 1$ and $k = 3\ell + 2$. We give an example in Figure 4b for $k = 7$.

Note that although our target subgraph for the case $k = 8$ is a 7-cycle with a tail instead of 8-cycle, our construction is still the same.

Correctness of the Reduction. In Lemma 19, we prove that G_k has degeneracy at most 2. In Lemma 20, we show that, for $k \neq 8$, the number of $\mathcal{C}_k$ in the graph G_k is a linear function of the number of triangles in G. In Lemma 21, we show that G_8 is H_8 free if and only if G is triangle free.

▶ **Lemma 19.** $\kappa(G_k) \leqslant 2$.

Proof. To prove the lemma it is sufficient to exhibit a vertex ordering $\prec$ such that in the corresponding directed graph $G^{\rightarrow}_{\prec}$, $d^+(v) \leqslant 2$ for all $v \in V_k$ (application of Lemma 6). We use an ordering $\prec$ where $V_E \prec V$ and the ordering within each set is arbitrary. Observe that each vertex $v \in V_E$ has degree exactly 2 and no two vertices in V are connected to each other. Hence, $d^+(v) \leqslant 2$ for all $v \in V_k$. ◀

▶ **Lemma 20.** *Let $\ell \geqslant 2$ be some integer. For $k = 3\ell$, $\mathrm{DM}(G_k, \mathcal{C}_k) = \mathrm{DM}(G, \mathcal{C}_3)$. For $k = 3\ell + i$ with $i \in \{1, 2\}$ and $k \neq 8$, $\mathrm{DM}(G_k, \mathcal{C}_k) = 3 \cdot \mathrm{DM}(G, \mathcal{C}_3)$.*

Proof. Let $\mathcal{T}$ be the set of triangles in G and $\mathcal{C}$ be the set of $\mathcal{C}_k$ in G_k. Note that a triangle in $\mathcal{T}$ and a k-cycle in $\mathcal{C}$ can be uniquely identified by a set of three and k edges, respectively.

We first take up case of $k = 3\ell$ for some $\ell \geqslant 2$. Let g be the mapping between the sets $\mathcal{T}$ and $\mathcal{C}$, $g : \mathcal{T} \to \mathcal{C}$, defined as follows: $g(\{e_1, e_2, e_3\}) = E_{e_1} \cup E_{e_2} \cup E_{e_3}$. To prove the lemma, it is sufficient to exhibit that g is a bijection. To this end, note that if $g(\tau_1) = g(\tau_2)$, then $\tau_1 = \tau_2$. This follows immediately from the definition of g, since $E_{e_1} \cap E_{e_2} = \emptyset$ for all $e_1 \neq e_2$. We now show that every k-cycle in $\mathcal{C}$ has an inverse mapping in g. Let ξ be a k-cycle in $\mathcal{C}$. Fix some edge $e \in E$. By construction, either all the edges from the set E_e are present in ξ, or none of them are. Hence, ξ must be of the form $E_{e_1} \cup E_{e_2} \cup E_{e_3}$ for some three distinct edges e_1, e_2, and e_3. Clearly, $\{e_1, e_2, e_3\}$ forms a triangle in G.

Now assume $k = 3\ell + i$ for some $\ell \geqslant 2$ and $i \in \{1, 2\}$, and $k \neq 8$. It is not difficult to see that each triangle in $\mathcal{T}$ leads to exactly three k-cycles in $\mathcal{C}$. The non-trivial direction is to show that for each k-cycles in $\mathcal{C}$ there is an unique triangle in $\mathcal{T}$. Let ξ be a k-cycle in $\mathcal{C}$. Fix some edge $e \in E$. By construction, exactly one of the following must be true: (i) all the ℓ edges from the set $E_{V,e}$ are present in ξ, (ii) all the $\ell + 1$ edges from the set $E_{U,e}$ are present in ξ, (iii) none of the edges from the set $E_{V,e} \cup E_{U,e}$ are present in ξ. First assume $i = 1$. Since ξ has $3\ell + 1$ many edges, and $\ell \geqslant 2$, it must consist of one $E_{U,e}$ set of size $\ell + 1$, and two $E_{V,e}$ sets of size ℓ. When $i = 2$ and $\ell > 2$, ξ must consist of two $E_{U,e}$ set of size $\ell + 1$, and one $E_{V,e}$ sets of size ℓ. Clearly, the three edges corresponding to these sets form a unique triangle in G. (When $k = 8$, that is $\ell = 2$ and $i = 2$, taking four distinct sets $E_{V,e}$ creates a copy of $\mathcal{C}_8$, and hence the argument does not work.) ◀

▶ **Lemma 21.** *The input graph G is triangle free if and only if G_8 does not have any $\mathcal{C}_7$ with a tail.*

Proof. Observe that, if there exists a triangle τ in G, then in G_8, there would be at least one $\mathcal{C}_7$ with a tail (in fact, the exact number would depend on the degree of the involved vertices). In the proof of Lemma 20, we argued that each 7-cycle in G_7 (which is isomorphic to G_8) corresponds to a triangle in G. Also, by our construction, if G_8 has a $\mathcal{C}_7$, then that 7-cycle necessarily has a tail. Therefore, existence of $\mathcal{C}_7$ with a tail in G_8 implies existence of a triangle in G. This completes the proof of the lemma. ◀

Lemmas 19 to 21 together prove the theorem: if there exists an algorithm $\mathcal{A}$ for the SUB-CNT$_{\mathcal{C}_k}$ problem with $T(\mathcal{A}) = O(mf(\kappa))$, then $\mathcal{A}$ is an algorithm for the TRI-CNT problem (or the triangle detection problem in the case of $k = 8$) with $T(\mathcal{A}) = O(m)$, where $T(\mathcal{A})$ denotes the worst case time complexity of the algorithm $\mathcal{A}$. ◀

6 Future Directions

Although our algorithmic framework fails to produce a linear time algorithm for SUB-CNT$_{\mathcal{C}_6}$ in bounded degeneracy graphs, there are certain other 6-vertex subgraphs where it indeed succeeds. An easy example is SUB-CNT$_{\mathcal{K}_6}$. In fact, our framework gives a linear time algorithm for counting any constant size clique in bounded degeneracy graphs – for each acyclic orientation of a clique, the source vertex construct a DRTS covering all the remaining vertices. There exists other non-clique 6-vertex subgraphs as well, where Alg. 2 succeeds. Consider the subgraph H shown in Fig. 5. It is easy to see that, any acyclic orientation of H such as $H^{\rightarrow}$ has at least one source vertex s that is a root of a DRTS with four vertices. Thus, we can solve SUB-CNT$_H$ in $O(m\kappa^3)$ expected time.

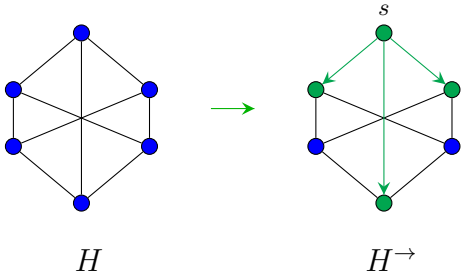


Figure 5 Alg. 2 succeeds to count the number of distinct matches of H in linear time for bounded (constant) degeneracy graphs. Each acyclic orientation of H has a source vertex s, which is connected to exactly three vertices, as in $H^{\rightarrow}$. So, the largest DRTS has at least four vertices (shown in green). Number of matches of the remaining vertices (shown in blue) could be counted using $\mathcal{HM}_2$.

Despite the chasm at six, there exist subgraphs H with 6-vertices (or more) such that SUB-CNT$_H$ admits a linear time algorithm in bounded degeneracy graph. We end this exposition with the following natural problem:

Characterize all subgraphs H such that SUB-CNT$_H$ *has a linear time algorithm in bounded degeneracy graphs.*

References

1 Amir Abboud and Virginia Vassilevska Williams. Popular conjectures imply strong lower bounds for dynamic problems. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science*, 2014.

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 Kook Jin Ahn, Sudipto Guha, and Andrew McGregor. Graph sketches: sparsification, spanners, and subgraphs. In *Proc. 31st ACM Symposium on Principles of Database Systems*, pages 5–14. ACM, 2012.

4 Noga Alon, Raphael Yuster, and Uri Zwick. Color-coding. *J. ACM*, 42(4):844–856, 1995.

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

6 Sepehr Assadi, Michael Kapralov, and Sanjeev Khanna. A Simple Sublinear-Time Algorithm for Counting Arbitrary Subgraphs via Edge Sampling. In *Proc. 10th Conference on Innovations in Theoretical Computer Science*, 2018.

7 Ziv Bar-Yossef, Ravi Kumar, and D. Sivakumar. Reductions in Streaming Algorithms, with an Application to Counting Triangles in Graphs. In *Proc. 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2002.

8 A. Benson, D. F. Gleich, and J. Leskovec. Higher-order organization of complex networks. *Science*, 353(6295):163–166, 2016.

9 Suman K Bera and Amit Chakrabarti. Towards tighter space bounds for counting triangles and other substructures in graph streams. In *Proc. 34th International Symposium on Theoretical Aspects of Computer Science*, 2017.

10 Nadja Betzler, Rene Van Bevern, Michael R Fellows, Christian Komusiewicz, and Rolf Niedermeier. Parameterized algorithmics for finding connected motifs in biological networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 8(5):1296–1308, 2011.

11 M. Bhuiyan, M. Rahman, M. Rahman, and M. Al Hasan. GUISE: Uniform Sampling of Graphlets for Large Graph Analysis. In *International Conference on Data Mining*, pages 91–100, 2012.

12 Etienne Birmele et al. Detecting local network motifs. *Electronic Journal of Statistics*, 6:908–933, 2012.

13 Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Counting paths and packings in halves. In *Proc. 17th Annual European Symposium on Algorithms*, pages 578–586, 2009.

14 Andreas Björklund, Petteri Kaski, and Łukasz Kowalik. Counting thin subgraphs via packings faster than meet-in-the-middle time. *ACM Transactions on Algorithms (TALG)*, 13(4):48, 2017.

15 R. Burt. Structural Holes and Good Ideas. *American Journal of Sociology*, 110(2):349–399, 2004.

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

17 Jonathan Cohen. Graph twiddling in a mapreduce world. *Computing in Science & Engineering*, 11(4):29, 2009.

18 J. Coleman. Social Capital in the Creation of Human Capital. *American Journal of Sociology*, 94:S95–S120, 1988. URL: `http://www.jstor.org/stable/2780243`.

19 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.

20 Radu Curticapean and Dániel Marx. Complexity of counting subgraphs: Only the boundedness of the vertex-cover number counts. In *Proc. 55th Annual IEEE Symposium on Foundations of Computer Science*, pages 130–139, 2014.

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

22 Talya Eden, Amit Levi, Dana Ron, and C Seshadhri. Approximately counting triangles in sublinear time. *SIAM Journal on Computing*, 46(5):1603–1646, 2017.

23 Talya Eden, Dana Ron, and C Seshadhri. On approximating the number of k-cliques in sublinear time. In *Proc. 50th Annual ACM Symposium on the Theory of Computing*, pages 722–734, 2018.

24 Friedrich Eisenbrand and Fabrizio Grandoni. On the complexity of fixed parameter clique and dominating set. *Theoretical Computer Science*, 326(1-3):57–67, 2004.

25 Ethan R Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G Dimakis. Beyond triangles: A distributed framework for estimating 3-profiles of large graphs. In *Proc. 12th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 229–238. ACM, 2015.

26 Ethan R Elenberg, Karthikeyan Shanmugam, Michael Borokhovich, and Alexandros G Dimakis. Distributed estimation of graph 4-profiles. In *Proc. 25th Proceedings, International World Wide Web Conference (WWW)*, pages 483–493. International World Wide Web Conferences Steering Committee, 2016.

27 David Eppstein. Arboricity and bipartite subgraph listing algorithms. *Information processing letters*, 51(4):207–211, 1994.

28 Mira Gonen and Yuval Shavitt. Approximating the number of network motifs. *Internet Mathematics*, 6(3):349–372, 2009.

29 Tomaž Hočevar and Janez Demšar. A combinatorial approach to graphlet counting. *Bioinformatics*, 30(4):559–565, 2014.

30 Tomaž Hočevar and Janez Demšar. Combinatorial algorithm for counting small induced graphs and orbits. *PloS ONE*, 12(2):e0171428, 2017.

31 Tomaž Hočevar, Janez Demšar, et al. Computation of graphlet orbits for nodes and edges in sparse graphs. *Journ. Stat. Soft*, 71, 2016.

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

33 F. Hormozdiari, P. Berenbrink, N. Prulj, and S. Cenk Sahinalp. Not All Scale-Free Networks Are Born Equal: The Role of the Seed Graph in PPI Network Evolution. *PLoS Computational Biology*, 118, 2007.

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

35 Shweta Jain and C Seshadhri. A Fast and Provable Method for Estimating Clique Counts Using Turán's Theorem. In *Proc. 26th Proceedings, International World Wide Web Conference (WWW)*, pages 441–449. International World Wide Web Conferences Steering Committee, 2017.

36 Madhav Jha, C Seshadhri, and Ali Pinar. A space efficient streaming algorithm for triangle counting using the birthday paradox. In *Proc. 19th Annual SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 589–597, 2013.

37 Madhav Jha, C Seshadhri, and Ali Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proc. 24th Proceedings, International World Wide Web Conference (WWW)*, pages 495–505. International World Wide Web Conferences Steering Committee, 2015.

38 Hossein Jowhari and Mohammad Ghodsi. New streaming algorithms for counting triangles in graphs. In *Computing and Combinatorics*, pages 710–716, 2005.

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

40 Arijit Khan, Nan Li, Xifeng Yan, Ziyu Guan, Supriyo Chakraborty, and Shu Tao. Neighborhood based fast graph search in large networks. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, 2011.

41 Ton Kloks, Dieter Kratsch, and Haiko Müller. Finding and counting small induced subgraphs efficiently. *Information Processing Letters*, 74(3-4):115–121, 2000.

42 Tamara G Kolda, Ali Pinar, Todd Plantenga, C Seshadhri, and Christine Task. Counting triangles in massive graphs with MapReduce. *SIAM Journal on Scientific Computing*, 36(5):S48–S77, 2014.

43 Tsvi Kopelowitz, Seth Pettie, and Ely Porat. Higher lower bounds from the 3SUM conjecture. In *Proc. 27th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2016.

44 Ioannis Koutis and Ryan Williams. Limits and applications of group algebras for parameterized problems. In *International Colloquium on Automata, Languages and Programming*, pages 653–664, 2009.

45 Mirosław Kowaluk, Andrzej Lingas, and Eva-Marta Lundell. Counting and detecting small subgraphs via equations. *SIAM Journal on Discrete Mathematics*, 27(2):892–909, 2013.

46 Madhusudan Manjunath, Kurt Mehlhorn, Konstantinos Panagiotou, and He Sun. Approximate Counting of Cycles in Streams. In *Proc. 19th Annual European Symposium on Algorithms*, pages 677–688, 2011.

47 Dror Marcus and Yuval Shavitt. Efficient counting of network motifs. In *IEEE 30th International Conference on Distributed Computing Systems Workshops*, pages 92–98. IEEE, 2010.

48 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.

49 Andrew McGregor, Sofya Vorotnikova, and Hoa T. Vu. Better Algorithms for Counting Triangles in Data Streams. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, pages 401–411, 2016.

50 Tijana Milenković and Nataša Pržulj. Uncovering biological network function via graphlet degree signatures. *Cancer informatics*, 6:CIN–S680, 2008.

51 Burkhard Monien. How to find long paths efficiently. In *North-Holland Mathematics Studies*, volume 109, pages 239–254. Elsevier, 1985.

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

53 Jaroslav Nešetřil and Svatopluk Poljak. On the complexity of the subgraph problem. *Commentationes Mathematicae Universitatis Carolinae*, 26(2):415–419, 1985.

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

55 Mihai Patrascu. Towards polynomial lower bounds for dynamic problems. In *Proc. 42nd Annual ACM Symposium on the Theory of Computing*, 2010.

56 Aduri Pavan, Kanat Tangwongsan, Srikanta Tirthapura, and Kun-Lung Wu. Counting and sampling triangles from a graph stream. *Proceedings of the VLDB Endowment*, 6(14):1870–1881, 2013.

57 Ali Pinar, C Seshadhri, and Vaidyanathan Vishal. Escape: Efficiently counting all 5-vertex subgraphs. In *Proceedings, International World Wide Web Conference (WWW)*, pages 1431–1440. International World Wide Web Conferences Steering Committee, 2017.

58 Alejandro Portes. Social Capital: Its Origins and Applications in Modern Sociology. *Annual Review of Sociology*, 24(1):1–24, 1998. doi:10.1146/annurev.soc.24.1.1.

59 Natasa Przulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):177–183, 2007.

60 Natasa Przulj, Derek G. Corneil, and Igor Jurisica. Modeling interactome: scale-free or geometric? *Bioinformatics*, 20(18):3508–3515, 2004.

61 M. Rahman, M. A. Bhuiyan, and M. Al Hasan. GRAFT: An Efficient Graphlet Counting Method for Large Graph Analysis. *IEEE Transactions on Knowledge and Data Engineering*, PP(99), 2014.

62 Ahmet Erdem Sariyuce, C. Seshadhri, Ali Pinar, and Umit V. Catalyurek. Finding the Hierarchy of Dense Subgraphs Using Nucleus Decompositions. In *Proceedings, International World Wide Web Conference (WWW)*, pages 927–937, 2015.

63 T. Schank and D. Wagner. Finding, Counting and Listing All Triangles in Large Graphs, an Experimental Study. In *Experimental and Efficient Algorithms*, pages 606–609. Springer, 2005.

64 C. Seshadhri and Srikanta Tirthapura. Scalable Subgraph Counting: The Methods Behind The Madness: WWW 2019 Tutorial. In *Proceedings, International World Wide Web Conference (WWW)*, 2019.

65 Alina Stoica and Christophe Prieur. Structure of neighborhoods in a large social network. In *International Conference on Computational Science and Engineering*, volume 4, pages 26–33. IEEE, 2009.

66 Siddharth Suri and Sergei Vassilvitskii. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*, pages 607–614, 2011.

67 Charalampos E. Tsourakakis. The K-clique Densest Subgraph Problem. In *Proceedings, International World Wide Web Conference (WWW)*, pages 1122–1132, 2015.

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

69 Virginia Vassilevska. Efficient algorithms for clique problems. *Information Processing Letters*, 109(4):254–257, 2009.

70 Virginia Vassilevska and Ryan Williams. Finding, minimizing, and counting weighted subgraphs. In *Proc. 41st Annual ACM Symposium on the Theory of Computing*, pages 455–464, 2009.

71 Pinghui Wang, Junzhou Zhao, Xiangliang Zhang, Zhenguo Li, Jiefeng Cheng, John CS Lui, Don Towsley, Jing Tao, and Xiaohong Guan. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering*, 30(1):73–86, 2017.

72 S. Wernicke and F. Rasche. FANMOD: a tool for fast network motif detection. *Bioinformatics*, 22(9):1152–1153, 2006.

73 Sebastian Wernicke. Efficient detection of network motifs. *IEEE/ACM Transactions on Computational Biology and Bioinformatics (TCBB)*, 3(4):347–359, 2006.

74 Virginia Vassilevska Williams, Joshua R Wang, Ryan Williams, and Huacheng Yu. Finding four-node subgraphs in triangle time. In *Proc. 26th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1671–1680, 2014.

75 Zhao Zhao, Guanying Wang, Ali R Butt, Maleq Khan, VS Anil Kumar, and Madhav V Marathe. Sahad: Subgraph analysis in massive networks using hadoop. In *IEEE 26th International Parallel and Distributed Processing Symposium*, pages 390–401. IEEE, 2012.