ESCAPE: Efficiently Counting All 5-Vertex Subgraphs

Ali Pinar
Sandia National Laboratories*
Livermore, CA
apinar@sandia.gov

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

Vaidyanathan Vishal
ONU Technology
Cupertino, CA
vishal@onutechnology.com

ABSTRACT

Counting the frequency of small subgraphs is a fundamental technique in network analysis across various domains, most notably in bioinformatics and social networks. The special case of triangle counting has received much attention. Getting results for 4-vertex or 5-vertex patterns is highly challenging, and there are few practical results known that can scale to massive sizes.

We introduce an algorithmic framework that can be adopted to count any small pattern in a graph and apply this framework to compute exact counts for *all* 5-vertex subgraphs. Our framework is built on cutting a pattern into smaller ones, and using counts of smaller patterns to get larger counts. Furthermore, we exploit degree orientations of the graph to reduce runtimes even further. These methods avoid the combinatorial explosion that typical subgraph counting algorithms face. We prove that it suffices to enumerate only four specific subgraphs (three of them have less than 5 vertices) to exactly count all 5-vertex patterns.

We perform extensive empirical experiments on a variety of real-world graphs. We are able to compute counts of graphs with tens of millions of edges in minutes on a commodity machine. To the best of our knowledge, this is the first practical algorithm for 5-vertex pattern counting that runs at this scale. A stepping stone to our main algorithm is a fast method for counting all 4-vertex patterns. This algorithm is typically ten times faster than the state of the art 4-vertex counters.

Keywords

motif analysis, subgraph counting, graph orientations

*Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

WWW 2017, April 3–7, 2017, Perth, Australia.
ACM 978-1-4503-4913-0/17/04.
http://dx.doi.org/10.1145/3038912.3052597

1. INTRODUCTION

Subgraph counting is a fundamental network analysis technique used across diverse domains: bioinformatics, social sciences, and infrastructure networks studies [23, 32, 30, 11, 33, 24, 6, 19]. The high frequencies of certain subgraphs in real networks gives a quantifiable method of proving they are not Erdős-Rényi [23, 49, 30]. Distributions of small subgraphs are used to evaluate network models, to summarize real networks, and classify vertex roles, among other things [23, 33, 11, 24, 6, 38, 47].

The main challenge of motif counting is combinatorial explosion. As we see in our experiments, the counts of 5-vertex patterns are in the orders of billions to trillions, even for graphs with a few million edges. An enumeration algorithm is forced to touch each occurrence, and cannot terminate in a reasonable time. The key insight of this paper is to design a formal framework of *counting without enumeration* (or more precisely, counting with minimal enumeration). Most existing methods [24, 8, 51, 34] work for graphs of at most 100K edges, limiting their uses to (what we would now consider) fairly small graphs. A notable exception is recent work by Ahmed et al on counting 4-vertex patterns, that scales to hundreds of millions of edges [4].

1.1 The problem

Our aim is simple: to exactly count the number of all vertex subgraphs (aka patterns, motifs, and graphlets) up to size 5 on massive graphs. There are 21 such connected subgraphs, as shown in Fig. 1. Additionally, there are 11 disconnected patterns, shown in the full version [31]. Throughout the paper, we refer to these subgraphs/motifs by their number. (Our algorithm also counts all 3 and 4 vertex patterns.) We give a formal description in §2.

Motif-counting is an extremely popular research topic, and has led to wide variety of results in the past years. As we shall see, numerous approximate algorithms have been proposed for this problem [50, 45, 9, 39, 34, 25]. Especially for validation at scale, it is critical to have a scalable, exact algorithm. ESCAPE directly addresses this issue.

1.2 Summary of our contributions

We design the Efficient Subgraph Counting Algorithmic PackagE (ESCAPE), that produces exact counts of all ≤ 5-vertex subgraphs. We provide a detailed theoretical analysis and run experiments on a large variety of datasets, including web networks, autonomous systems networks, and social networks. All experiments are done on a single commodity machine using 64GB memory.

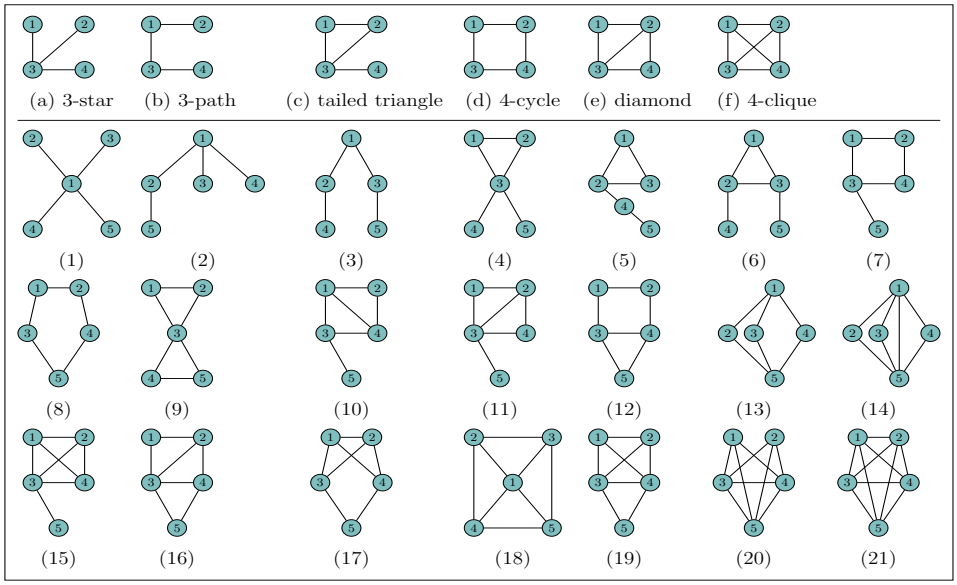


Figure 1: Connected 4 and 5-vertex patterns

Scalability through careful algorithmics. Conventional wisdom is that 5-vertex pattern counting is not feasible because of size. There are a host of approximate methods, such as color coding [24, 8, 52], MCMC based sampling algorithms [9], edge sampling algorithms [34, 50]. We challenge that belief. ESCAPE can do exact counting for patterns up to 5 vertices on graphs with tens of millions of edges in a matter of minutes. (As shown in the experimental section of the above results, they do not scale graphs of such sizes.) For instance, ESCAPE computes all 5 vertex counts on an router graph with 22M edges in under 5 minutes.

Avoiding enumeration by clever counting. One of the key insights into ESCAPE is that it suffices to enumerate a very small set of patterns to compute all 5 vertex counts. Essentially, we build a formal framework of "cutting" a pattern into smaller subpatterns, and show that it is practically viable. From this theoretical framework, we can show that it suffices to exhaustively enumerate a special (small) subset of patterns to actually count all 5-vertex patterns. Counting ideas to avoid enumeration have appeared in the past practical algorithms [22, 4, 17, 18] but in a more ad hoc manner (and never for 5-vertex patterns.)

The framework is absolutely critical for exact counting, since enumeration is clearly infeasible even for graphs with a million edges. For instance, an automonous systems graph with 11M edges has 10^{11} instances of pattern 17 (in Fig. 1). We achieve exact counts with clever data structures and combinatorial counting arguments.

Furthermore, using standard inclusion-exclusion arguments, we prove that the counts of all connected patterns can be used to get the counts of all (possibly disconnected) patterns. This is done without any extra work on the input graph.

Exploiting orientations. A critical idea developed in ESCAPE is orienting edges in a degeneracy style ordering. Such techniques have been successfully applied to triangle counting before [12, 13, 40]. Here we show how this technique can be extended to general pattern counting. This is what allows ESCAPE to be feasible for 5-vertex pattern counting, and makes it much faster for 4-vertex pattern counting.

Improvements for 4-vertex patterns counting. The recent PGD package of Ahmed et al. [4] has advanced the state of art significantly with better 4-vertex pattern algorithms. We show the speedup of ESCAPE over PGD in Fig. 2. ESCAPE is significantly faster, by a factor of hundreds almost half the instances. Notably, on the `orkut` social network with 200M edges, ESCAPE took less than 20 minutes, while PGD ran for days.

Trends in 5-vertex pattern counts: Our ability to count 5-vertex patterns provides a powerful graph mining tool. We discover some interesting trends in the pattern counts. Most surprisingly, both patterns (18) and (19) have the same number of edges, but (18) is significantly less frequent than (19). Also, by studying ratios of induced and non-induced counts, we discover some patterns are highly unlikely to be induced. We believe these results can potentially be used for edge prediction and graph classification, and for doing subgraph frequency analysis of Ugander et al [47].

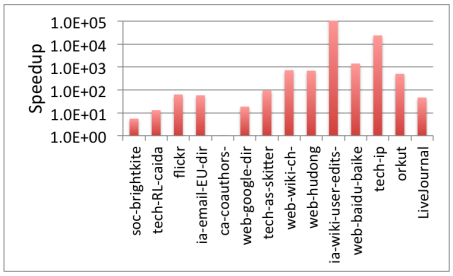


Figure 2: Speedup achieved by Escape over PGD [4] for 4-vertex pattern counting (runtime of PGD/runtime of ESCAPE).

1.3 Related Work

In fields as varied as social sciences, biology, and physics, it has been observed that the frequency of small pattern subgraphs plays an important role in graph structure [23, 14, 48,

49, 16, 30, 11, 6, 41]. Specifically in bioinformatics, pattern counts have significant relevance in graph classiciation [30, 33, 21].

In social networks, Ugander et al. [47], underlined the significance of 4-vertex patterns by proposing a "coordinate system" for graphs based on the motifs distribution. This was applied to classification of comparatively small networks (thousands of vertices). We stress that this was useful even without graph attributes, and thus the structure itself was enough for classification purposes. A number of recent results have used small subgraph counts for detecting communities and dense subgraphs [35, 44, 7, 46].

From the practical algorithmics standpoint, triangle counting has received much attention. We simply refer the reader to the related work sections of [43, 39]. Gonen and Shavitt [20] propose exact and approximate algorithms for computing *non-induced* counts of some 4-vertex motifs. They also consider counting number of motifs that a vertex participates in, an instance of a problem called *motif degree counting*, which has gained a lot of attention recently (see [29, 20, 10]). Marcus and Shavitt [27] give exact algorithms for computing all 4-vertex motifs running in time $O(d \cdot m + m^2)$. Here d is the maximum vertex degree and m is the number of edges. Their package RAGE does not scale to large graphs. The largest graph processed has 90K edges and takes 40 minutes. They compare with the bioinformatics FANMOD package [50], which takes about 3 hours.

A breakthrough in exact 4-vertex pattern counting was recently achieved by Ahmed et al. [4]. Using techniques on graph transitions based on edge addition/removal, their PGD (Parametrized Graphlet Decomposition) package handles graphs with tens of millions of edges and more, and is many orders of magnitude faster than RAGE. It routinely processes 10 million edge graphs in under an hour. There are other results on counting 4-vertex patterns, but none achieve the scalability of PGD [42, 22]. We consider PGD to be the state-of-the-art for 4-vertex pattern counting. They do detailed comparisons and clearly outperform previous work. (Notably, the authors made their code public [2].)

Elenberg et al. [17, 18] give algorithms for computing pattern *profiles*, which involve computing pattern counts per vertex and edge. This is a significantly harder problem, and Elenberg et al. employ approximate and distributed algorithms. The maximum graph size they handle is in order of tens of millions of edges.

Many of the results above [4, 42, 18] use combinatorial strategies to cut down enumeration, which our cutting framework tries to formalize. For the special case of vertex and edge profiles, Melckenbeeck et al. give an automated method to generate combinatorial equations for profile counting [28]. These results only generate linear equations, and do not prescribe the most efficient method of counting. In contrast, our cutting framework generates polynomial formulas, and we deduce the most efficient formula for 5-vertex patterns.

As an alternative exact counting, Jha et al. [25] proposed 3-path sampling to estimate all 4-vertex counts. Their technique builds on *wedge sampling* [36, 39] and samples paths of length 3 to estimate various 4-vertex statistics.

To the best of our knowledge, there is no method (approximate or exact) that can count all 5-vertex patterns for graphs with millions of edges.

Table 1: Notation for various subgraph counts

Notation	Count
$d(i)$	degree of i
$W(G)$	wedge
$W_{++}(G^{\rightarrow}), W_{+-}(G^{\rightarrow})$	out-wedge, inout-wedge
$W(i,j)$	wedge between i,j
$W_{++}(i,j)$	outwedge between i,j
$W_{+-}(i,j)$	wedge from i to j
$T(G), T(i), T(e)$	triangle
$C_4(G), C_4(i), C_4(e)$	4-cycle
$K_4(G), K_4(i), K_4(e)$	4-clique
$D(G)$	diamond
$DD(G^{\rightarrow})$	directed diamond
$DP(G^{\rightarrow})$	directed 3-path
$DBP(G^{\rightarrow})$	directed bipyramid

2. PRELIMINARIES

Our input is an undirected simple graph $G = (V, E)$, with n vertices and m edges. We distinguish subgraphs from *induced subgraphs* [15]. A subgraph is a subset of edges. An induced subgraph is obtained by taking a subset V' of vertices, and taking *all edges* among these vertices.

We wish to get induced and non-induced counts for all patterns up to size 5. As shown later in Theorem 3, it suffices to get counts for only connected patterns, since all other counts can be obtained by simple combinatorics. The connected 4-vertex and 5-vertex patterns are shown in Fig. 1. For convenience, the *ith 4-pattern* refers to ith subgraph with 4 vertices in Fig. 1. For example, the 6th 4-pattern in the four-clique and the 8th 5-pattern is the five-cycle.

Without loss of generality, we focus on computing induced subgraph counts. We use C_i (resp. N_i) to denote the induced (resp. non-induced) count of the ith 5-pattern. *Our aim is to compute all C_i values.* A simple (invertible) linear transformation gives all the C_i values from the N_i values. This matrix is not included here due to space limitations, but is provided in the full version [31].

2.1 Notation

The input graph $G = (V, E)$ is undirected and has n vertices and m edges. For analysis, we assume that the graph is stored as an adjacency list, where each list is a hash table. Thus, edge queries can be made in constant time.

We denote the *degree ordering* of G by $\prec$. For vertices i, j, we say $i \prec j$, if either $d(i) < d(j)$ or $d(i) = d(j)$ and $i < j$ (comparing vertex id). We construct the *degree ordered DAG* $G^{\rightarrow}$ by orienting all edges in G according to $\prec$.

Our results and proofs are somewhat heavy on notation, and important terms are provided in Tab. 1. Counts of certain patterns, especially those in Fig. 3, will receive special notation. Note that some of these patterns are directed, since we will require the count of them in $G^{\rightarrow}$.

We will also need *per-vertex, per-edge* counts for some patterns. For example, $T(G)$ denotes the total number of triangles, while $T(i), T(e)$ denote the number of triangles incident to vertex i and edge e respectively.

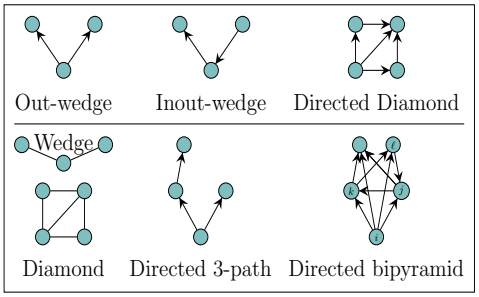


Figure 3: Fundamental patterns for 4-vertex (above) and 5-vertex (below) pattern counting

3. MAIN THEOREMS

Our final algorithms are quite complex and use a variety of combinatorial methods for efficiency. Nonetheless, the final asymptotic runtimes are easy to express. (While we do not focus on this, the leading constants in the $O(\cdot)$ are quite small.) Our main insight is: despite the plethora of small subgraphs, it suffices to enumerate a very small, carefully chosen set of subgraphs to count everything else. Furthermore, these subgraphs can themselves be enumerated with minimal overhead.

THEOREM 1. *There is an algorithm for exactly counting all connected 4-vertex patterns in G whose runtime is $O(W_{++}(G^{\rightarrow}) + W_{+-}(G^{\rightarrow}) + DD(G^{\rightarrow}) + m + n)$ and storage is $O(n+m)$.*

THEOREM 2. *There is an algorithm for exactly counting all connected 5-vertex patterns in G whose runtime is $O(W(G) + D(G) + DP(G^{\rightarrow}) + DBP(G^{\rightarrow}) + m + n)$ and storage complexity is $O(n + m + T(G))$.*

A routine inclusion-exclusion argument yields counts for all patterns.

THEOREM 3. *Fix a graph G. Suppose we have counts for all connected r-vertex patterns, for all $r \leq k$. Then, the counts for all (even disconnected) k-vertex patterns can be determined in constant time (only a function of k).*

Outline of remaining paper: §4 gives a high level overview of our main techniques. §5 discusses the cutting framework used to reduce counting all patterns into enumeration of some specific patterns (namely, those in Fig. 3). In §6, we apply this framework to 4-vertex pattern count, and prove Theorem 1. In §7, we work towards 5-vertex pattern counting and prove Theorem 2.

The proof and discussion of Theorem 3 is omitted because of space constraints and appears in our full paper [31].

4. MAIN IDEAS

The goal of ESCAPE is to avoid the combinatorial explosion that occurs in a typical enumeration algorithm. For example, the `tech-as-skitter` graph has 11M edges, but *2 trillion* 5-cycles. Most 5-vertex pattern counts are upwards of many billions for most of the graphs (which have only 10M edges). Any algorithm that explicitly touches each such pattern is bound to fail. The second difficulty is that the time for enumeration is significantly *more* than the count of patterns. This is because we have to find *all potential* patterns, the number of which is more than the count of patterns. A standard method of counting triangles is to enumerate wedges, and check whether it participates in a triangle. The number of wedges in a graph is typically an order of magnitude higher than the number of triangles.

Idea 1: Cutting patterns into smaller patterns. For a pattern H, a cut set is a subset of vertices whose removal disconnects H. Other than the clique, every other pattern has a cut set that is a strict subset of the vertices (we call this a non-trivial cut set). Formally, suppose there is some set of k vertices S, whose removal splits H into connected components $C_1, C_2, \ldots$. Let the graphs induced by the union of S and C_i be H_i. The key observation is that if we determine the following quantities, we can count the number of occurrences of H.

- For each set S of k vertices in G, the number of occurrences of $H_1, H_2, \ldots$ that involve S.
- The number of occurrences of H', for all H' with fewer vertices than H.

The exact formalization of this requires a fair bit of notation and the language of graph automorphisms. This gives a set of (polynomial) formulas for counting most of the 5-vertex patterns. These formulas can be efficiently evaluated with appropriate data structures.

There is some art in choosing the right S to design the most efficient algorithm. In most of the applications, S is often just a vertex or edge. Thus, if we know the number of copies of H_i incident to every vertex and edge of G, we can count H. This information can be determined by enumerating all the H_is, which is a much simpler problem.

Idea 2: Direction reduces search. A classic algorithmic idea for triangle counting is to convert the undirected G into the DAG $G^{\rightarrow}$, and search for directed triangles [12, 37, 13]. We extend this approach to 4 and 5-vertex patterns. The idea is to search for *all* non-isomorphic DAG versions of the pattern H in $G^{\rightarrow}$. This is combined with Idea 1, where we break up patterns in smaller ones. These smaller patterns are enumerated through $G^{\rightarrow}$, since the direction significantly cuts down the combinatorial expansion of the enumeration procedure. The use of graph orientations has been employed in theoretical algorithms for subgraph counting [5]. We bring this powerful technique to practical counting of 4 and 5-vertex patterns.

5. THE CUTTING FRAMEWORK

This section introduces the framework of our algorithms. We start with introducing the theory, and then discuss how it applies to algorithm design. We finally present its application to 5-pattern 2 counting.

Let H be a pattern we wish to count in G. For any set of vertices C in H, $H|_C$ is the subgraph of H induced on C. For this section, it is convenient to consider G and H as labeled. This makes the formal analysis much simpler. (Labeled counts can be translated to unlabeled counts by pattern automorphism counts.)

We formally define a match and a partial match of the pattern $H = (V(H), E(H))$. As defined, a match is basically an induced subgraph of G that is exactly H.

DEFN. 1. A *match* of H is a bijection $\pi : S \to V(H)$ where $S \subseteq V$ and $\forall s_1, s_2 \in S$, (s_1, s_2) is an edge of G iff $(\pi(s_1), \pi(s_2))$ is an edge of H. The set of distinct matches of H in G is denoted match(H).

If π is only an injection (so $|S| < |V(H)|$), then π is a *partial match*.

A match $\pi : S \to V(H)$ *extends* a partial match $\sigma : T \to V(H)$ if $S \supset T$ and $\forall t \in T$, $\pi(t) = \sigma(t)$.

DEFN. 2. Let σ be a partial match of H in G. The H-degree of σ, denoted $\deg_H(\sigma)$, is the number of matches of H that extend σ.

We now define the *fragment* of G that is obtained by cutting H into smaller patterns.

DEFN. 3. Consider H with some non-trivial cut set C (so $|C| < |V(H)|$), whose removal leads to connected components $S_1, S_2, \ldots$. The *C-fragments* of H are the subgraphs of H induced by $C \cup S_1, C \cup S_2, \ldots$. This set is denoted by $\text{Frag}_C(H)$.

Before launching into the next definition, it helps to explain the main cutting lemma. Suppose we find a copy σ of $H|_C$ in G. If σ extends to a copy of every possible $F_i \in \text{Frag}_C(H)$ *and* all these copies are disjoint, then they all combine to give a copy of H. When these copies are not disjoint, we end up with another graph H', which we call a *shrinkage*.

DEFN. 4. Consider graphs H, H', and a non-trivial cut set C for H. Let the graphs in $\text{Frag}_C(H)$ be denoted by $F_1, F_2, \ldots$. A *C-shrinkage of H into H'* is a set of maps $\{\sigma, \pi_1, \pi_2, \ldots, \pi_{|\text{Frag}_C(H)|}\}$ with the following properties.

- $\sigma : H|_C \to H'$ is a partial match of H'.
- Each $\pi_i : F_i \to H'$ is a partial match of H'.
- Each π_i extends σ.
- For each edge (u, v) of H', there are some index $i \in |\text{Frag}_C(H)|$ and vertices $a, b \in F_i$ such that $\pi_i(a) = u$ and $\pi_i(b) = v$.

The set of graphs $H' \neq H$ such that there exists some C-shrinkage of H in H' is denoted $\text{Shrink}_C(H)$. For $H' \in \text{Shrink}_C(H)$, the number of distinct C-shrinkages is $\text{numSh}_C(H, H')$.

The main lemma tells us that if we know $\deg_F(\sigma)$ for every copy σ of $H|_C$ and for every C-fragment F, and we know the counts of every possible shrinkage, we can deduce the count of H.

LEMMA 4. *Consider pattern H with cut set C. Then,*

$$\mathit{match}(H) = \sum_{\sigma \in \mathit{match}(H|_C)} \prod_{F \in \mathit{Frag}_C(H)} \deg_F(\sigma) - \sum_{H' \in \mathit{Shrink}_C(H)} \mathit{numSh}_C(H, H') \mathit{match}(H')$$

PROOF. Consider any copy σ of $H|_C$. Take all tuples of the form $(\pi_1, \pi_2, \ldots, \pi_{|\text{Frag}_C(H)|})$ where π_i is a copy $F_i \in \text{Frag}_C(H)$ that extends σ. The number of such tuples is exactly $\sum_{\sigma \in \text{match}(H|_C)} \prod_{F \in \text{Frag}_C(H)} \deg_F(\sigma)$.

Abusing notation, let $V(\pi_i)$ be the set of vertices that π_i maps to F_i. If all $V(\pi_i) \setminus V(C)$ are distinct, by definition, we get a copy of H. If there is any intersection, this is a C-shrinkage of H into some H'. Consider aggregating the above argument over all copies σ. Each match of H is counted exactly once. Each match of $H' \in \text{Shrink}_C(H)$ is counted for every distinct C-shrinkage of H into H', which is exactly $\text{numSh}_C(H, H')$. This completes the proof.

Algorithmically using this lemma: Suppose H is a 5-vertex pattern, and counts for all $\leq$ 4-vertex patterns are known. In typical examples, C is either a vertex or an edge. Thus, each σ in the formula is simply just every possible vertex or edge. If we can enumerate all matches of each $F \in \text{Frag}_C(H)$, then we can store $\deg_F(\sigma)$ in appropriate data structures. Each F has strictly less than 5-vertices (and in most cases, just 2 or 3), and thus, we can hope to enumerate F.

Once all $\deg_F(\sigma)$ are computed, we can iterate over all σ to compute the first term in Lemma 4. We need to subtract out the summation over $\text{Shrink}_C(H)$. Observe that $\text{numSh}_C(H, H')$ is an absolute constant independent of G, so it can be precomputed. Each $H' \in \text{Shrink}_C(H)$ has less than 5 vertices, so we already know $\text{match}(H')$.

This yields $\text{match}(H)$. To get the final *unlabeled* frequency, we must normalize to $\text{match}(H)/|\text{Aut}(H)|$. (Here, $\text{Aut}(H)$ is the set of automorphisms of H. The same unlabeled pattern can be counted multiple times as a labeled match. For example, every triangle gets counted three times in match, and we divide this out to get the final unlabeled frequency.)

Application of lemma for pattern 2: To demonstrate this lemma, let us derive counts for 5-pattern (2). We use the labeling in Fig. 1. Let edge $(1, 2)$ be the cut set S. Thus, the fragments are F_1, the wedge $\{(1, 2), (2, 5)\}$ and F_2, the three-star $\{(1, 2), (1, 3), (1, 4)\}$. Every edge in G is a match of $H|_S$. Consider (i, j) with match $\sigma(i) = 1$ and $\sigma(j) = 2$. (Note that σ maps from vertices in G to $H|_S$.) The degree $\deg_{F_1}(\sigma)$ is $d(j) - 1$. The degree $\deg_{F_2}(\sigma)$ is $(d(i) - 1)(d(i) - 2)$.

The only possible shrinkage of the patterns is into a tailed triangle. (Vertex 3 or 4 "merges" with 5; any other merging of vertices also merges an edge. So these are not valid shrinkages.) Let H be the 5-pattern (2), and H' be the tailed triangle. Note that $\text{numSh}_C(H, H')$ is 2. In both cases, we set $\sigma'(1) = 3$ and $\sigma'(2) = 1$. Set $\pi_1(5) = 2$ and $\pi_2(3) = 2$, $\pi_2(4) = 4$. Alternately, we can change $\pi_2(4) = 2$ and $\pi_2(3) = 4$. The set of maps $\{\sigma', \pi_1, \pi_2\}$ in both cases forms a C-shrinkage of this pattern into tailed triangles.

This

$$\text{match}(H) = \sum_{(i,j) \in E} \big[(d(j) - 1)(d(i) - 1)(d(i) - 2) + (d(i) - 1)(d(j) - 1)(d(j) - 2)\big] - 2 \cdot \text{match}(\text{tailed triangle})$$

Note that H has two automorphisms, as does the tailed triangle. Thus, the number of tailed triangle matches (as a labeled graph) is twice the frequency. A simple argument shows that the number of tailed triangles is $\sum_i t(i)(d(i) - 2)$ (we can also derive this from Lemma 4). Thus,

$$N_2 = \sum_{\langle i,j \rangle \in E} (d(j) - 1)\binom{d(i) - 1}{2} - 2\sum_i t(i)(d(i) - 2)$$

6. COUNTING 4-VERTEX PATTERNS

A good introduction to these techniques is counting 4-vertex patterns. The following formulas have been proven in [25, 4], but can be derived using the framework of Lemma 4.

THEOREM 5. *# 3-stars* $= \sum_i \binom{d(i)}{3}$, *# diamonds* $= \sum_e \binom{t(e)}{2}$, *# 3-paths* $= \sum_{(i,j) \in E} (d(i) - 1)(d(j) - 1) - 3 \cdot T(G)$, *# tailed-triangles* $= \sum_i t(i)(d(i) - 2)$

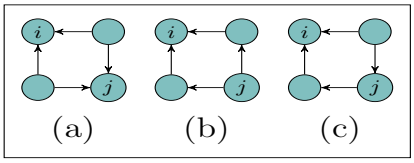


Figure 4: All acyclic orientations of the 4-cycle

For counting 4-cycles, note any set of opposite vertices (like 1 and 4 in 4-cycle of Fig. 1) form a cut. It is easy to see that $C_4(G) = \sum_{i<j} \binom{W(i,j)}{2}/2$. These values are potentially expensive to compute as a complete wedge enumeration is required. Employing the degree ordering, we can prove a significant improvement. With a little care, we can get counts per edge. (We remind the reader that $\succ$ refers to the degree ordering.)

THEOREM 6. *$C_4(G) = \sum_{i\succ j} \binom{W_{++}(i,j)+W_{+-}(i,j)}{2}$. For edge (i,k) where $i \succ k$,*
$C_4((i,k)) = \sum_{j\succ k} [W_{++}(i,j) + W_{+-}(i,j) + W_{+-}(j,i) - 1] + \sum_{j\prec k}[W_{+-}(j,i) + W_{++}(i,j) - 1]$.

PROOF. Consider all DAG versions of the 4-cycle, as given in Fig. 4. Let i denote the highest vertex according to $\prec$, and let j be the opposite end (as shown in the figure). The key observation is that wedges between i and j are either 2 out-wedges, 2 inout wedges, or one of each. Summing over all possible js, we complete the proof of the basic count.

Consider edge (i,k) where $i \succ k$. To determine the 4-cycles on this edge, we look at all wedges that involve (i,k). Suppose the third vertex on such a wedge is j. We have two possibilities. (i) $i \succ k \prec j$: Thus, (i,k,j) is an out wedge, and could be a part of a 4-cycle of type (a) or (c). Any out or inout wedge between i and j creates a 4-cycle. We need a -1 term to subtract out the wedge (i,k,j) itself.

(ii) $i \succ k \succ j$: This is an inout wedge and can be part of a 4-cycle of type (b) or (c). Again, any other out or inout wedge between i and j forms a 4-cycle. A similar argument to the above completes the proof.

Now, we show how to count 4-cliques.

THEOREM 7. *(We remind the reader that DD is the number of directed diamonds, as shown in Fig. 3.) The number of four-cliques per-vertex and per-edge can be found in time $O(W_{++}(G^{\rightarrow}) + DD(G^{\rightarrow}))$ and $O(m)$ additional space.*

PROOF. Let H denote the directed diamond of Fig. 3. The key observation is that every four-clique in the original graph must contain one (and exactly one) copy of H as a subgraph. It is possible to enumerate all such patterns in time linear in $DD(G^{\rightarrow})$. We simply loop over all edges (i,j), where $i \prec j$. We enumerate all the outout wedges involving (i,j), and determine all triangles involving (i,j) with i as the smallest vertex. Every pair of such triangles creates a copy of H, where (i,j) forms the diagonal. For each such copy, we check for the missing edge to see if it forms a four-clique. Since we enumerate all four-cliques, it is routine to find the per-vertex and per-edge counts.

We state below a stronger version of the 4-vertex counting theorem, Theorem 1. This will be useful for 5-vertex pattern counting.

THEOREM 8. *In $O(W_{++} + W_{+-} + DD + m)$ time and $O(T)$ additional space, there is an algorithm that computes (for all vertices i, edges e, triangle t): all $T(i)$, $T(e)$, $C_4(i)$, $C_4(e)$, $K_4(i)$, $K_4(e)$, $K_4(t)$ counts, and for every edge e, the list of triangles incident to e.*

PROOF. A classic theorem basically states that all triangles can be enumerated in $O(W_{++}(G^{\rightarrow}))$ time [12, 37]. (We used the same argument to prove Theorem 7.) By Theorem 5, once we have per-vertex and per-edge triangle counts, we can count everything other that 4-cycles and 4-cliques in linear time. By Theorem 6, enumerating outout and inout wedges suffices for 4-cycle counting. We add the bound on Theorem 7 to complete the proof.

7. ONTO 5-VERTEX COUNTS

With the cut framework of §5, we can generate *efficient* formulas for all 5-vertex patterns, barring the 5-cycle (pattern 8) and the 5-clique (pattern 21). We give the formulas that Lemma 4 yields. It is cumbersome and space-consuming to give proofs of all of these, so we omit them. We break the formulas into four groups, depending on whether the cut chosen is a vertex, edge, triangle, or wedge. We use $TT(G)$ to denote the tailed triangle count in G. After stating these formulas, we will later explain the algorithm that computes the various N_is.

THEOREM 9. *[Cut is vertex]* $N_1 = \sum_i \binom{d(i)}{4}$
$N_3 = \sum_i \sum_{(i,j)\in E}(d(j)-1) - 4\cdot C_4(G) - 2\cdot TT(G) - 3\cdot T(G)$
$N_4 = \sum_i t(i)(d(i)-2)$
$N_7 = \sum_i C_4(i)(d(i)-2) - 2\cdot D(G)$
$N_9 = \sum_i \binom{t_i}{2} - 2\cdot D(G)$
$N_{15} = \sum_i K_4(i)(d(i)-3)$

THEOREM 10. *[Cut is edge]* $N_2 = \sum_{\langle i,j\rangle\in E}(d(j)-1)\binom{d(i)-1}{2} - 2\cdot TT(G)$
$N_5 = \sum_{e=\langle i,j\rangle\in E}(d(i)-1)(d(j)-T(e)) - 4\cdot D(G)$
$N_6 = \sum_{e=\langle i,j\rangle\in E} t_e(d(i)-2)(d(j)-2) - 2\cdot D(G)$
$N_{11} = \sum_{e=\langle i,j\rangle\in E}\binom{T(e)}{2}(d(i)-3)$
$N_{12} = \sum_{e\in E} C_4(e)T(e) - 4\cdot D(G)$
$N_{14} = \sum_e \binom{T(e)}{3}$
$N_{19} = \sum_e K_4(e)(t(e)-2)$

For the next theorem, we give a short proof sketch of how the formulas are obtained.

THEOREM 11. *[Cut is triangle]*
$N_{10} = \sum_{t=\langle i,j,k\rangle\ triangle}[(t(i,j)-1)(d(k)-1)] - 4\cdot K_4(G)$
$N_{16} = \sum_{t=\langle i,j,k\rangle\ triangle}(t(i,j)-1)(t(i,k)-1)$
$N_{20} = \sum_{t\ triangle}\binom{K_4(t)}{2} - 4\cdot K_4(G)$

PROOF. Refer to Fig. 1 for labels. We will apply Lemma 4, where C will be a triangle. For pattern 10, we use vertices $\{1,2,4\}$ as C; for pattern 16, the cut is $\{2,3,4\}$; for pattern 20, the cut is $\{3,4,5\}$. The formulas can be derived using Lemma 4.

THEOREM 12. *[Cut is pair or wedge] Define $D(i,j)$ to be the number of diamonds involving i and h where i and j are not connected to the chord (in Fig. 1, i maps 1 and j maps to 4). Let $CC(i,j,k)$ be the number of diamonds where i maps to 1, j maps to 2 and k maps to 4.*
$N_{13} = \sum_{i\prec j}\binom{W(i,j)}{3}$
$N_{17} = \sum_{i\prec j}(W(i,j)-2)D(j,i)$
$N_{18} = \sum_{i,j,k}\binom{D(i,j,k)}{2}$

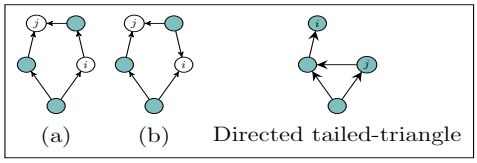


Figure 5: Directed patterns for 5-cycle counting

PROOF. The formula for N_{13} is straightforward. For pattern 17, we choose vertices 3 and 4 as the cut. Observe that vertices 1, 2, 3, and 4 form a diamond. For the wheel (pattern 18), we use the "diagonal" 2, 1, 5 as the cut. The fragments are both diamonds sharing those vertices.

Finally, we put everything together.

THEOREM 13. *Assume we have all the information from Theorem 8. All counts in Theorem 9, Theorem 10, and Theorem 11 can be computed in time $O(W(G) + D(G) + n + m)$ and $O(n + m + T(G)$ storage.*

PROOF. The counts of Theorem 9, Theorem 10, and Theorem 11 can be computed in $O(n)$, $O(m)$, and $O(T)$ time respectively. We can obviously count N_{13} in $O(W)$ time, by enumerating all wedges. For the remaining, we need to generate $D(i, j)$ and $D(i, j, k)$ counts.

Let us describe the algorithm for N_{17}. Fix a vertex i. For every edge (i, k), we have the list of triangles incident to (i, k) (from Theorem 8). For each such triangle (i, k, ℓ), we can get the list of triangles incident to (k, ℓ). For each such triangle (k, ℓ, j), we have generated a diamond with i and j at opposite ends. By performing this enumeration over all (i, k), and all (i, k, ℓ), we can generate $CC(i, j)$ for all j. By doing a 2-step BFS from i, we can also generate all $W(i, j)$ counts. Thus, we compute the summand, and looping over all i, we compute N_{17}. The total running time is the number of diamonds plus wedges. An identical argument holds for N_{18} and is omitted.

7.1 The 5-cycle and 5-clique

The final challenge is to count the 5-cycle and the 5-clique. The main tool is to use the DAG $G^{\rightarrow}$, analogous to 4-cycles and 4-cliques.

THEOREM 14. *Consider the 3-path in Fig. 5, and let $P(i, j)$ be the number of* directed 3-paths *between i and j, as oriented in the figure. Let Z be the number of directed tailed-triangles, as shown in Fig. 5. The number of 5-cycles is $\sum_{i \prec j} P(i, j) \cdot (W_{++}(i, j) + W_{+-}(i, j)) - Z$.*

PROOF. Fig. 5 shows the different possible 5-cycle DAGs. There are only two (up to isomorphism). In both cases, we choose i and j (as shown) to be the cut. (Wlog, we assume that $i \prec j$.) The vertices have the same directed three-path between them. They also have either an out-wedge or inout-wedge connecting them. Thus, the product $\sum_{i \prec j} P(i, j) \cdot (W_{++}(i, j) + W_{+-}(i, j))$ counts each 5-cycle exactly once. The shrinkage of either directed 5-cycle yields the directed tailed-triangle of Fig. 5(d). This pattern is also counted exactly once in the product above. (One can formally derive this relation using Lemma 4.) Thus, we substract Z out to get the number of 5-cycles.

THEOREM 15. *(We remind the reader that DBP is the count of the directed bipyramid in Fig. 3.) The number of 5-cliques can be counted in time $O(DBP(G) + D(G) + T(G) + n + m)$.*

PROOF. First observe that every 5-clique in D contains one of these directed bipyramids. Thus, it suffices to enumerate them to enumerate all 5-cliques. The key is to enumerate this pattern with minimal overhead. From Theorem 8, we have the list of triangles incident to every edge. For every triangle t, we determine all of these patterns that contain t as exactly the triangle (i, j, k) in Fig. 3.

Suppose triangle t consists of vertices i, j, k. We enumerate every other triangle incident to j, k using the data structure of Theorem 8. Such a triangle has a third vertex, say ℓ. We check if i, j, k, ℓ form the desired directed configuration. Once we generate all possible ℓ vertices, every pair among them gives the desired directed pattern.

The time required to generate the list of ℓ vertices over all triangles is at most $\sum_{t=(i,j,k)} t(j, k) \leq \sum_{j,k} t(j, k)^2 = O(D(G) + T(G))$. Once these lists are generated, the additional time is exactly DBP to generate each directed pattern.

At long last, we can prove Theorem 2.

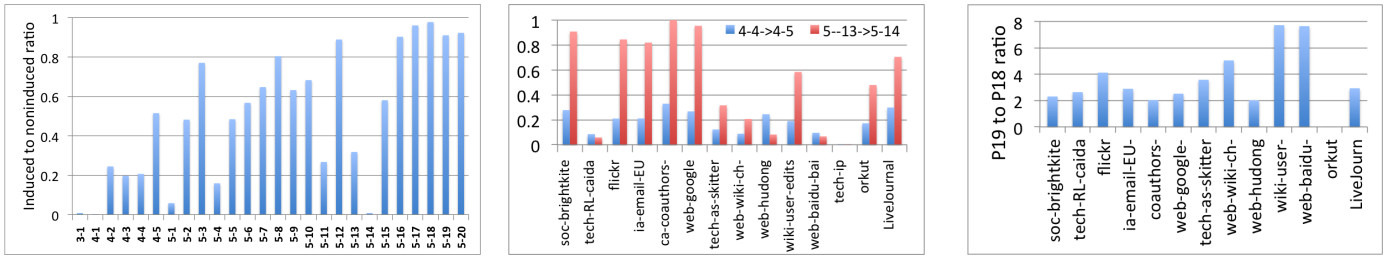
PROOF. (of Theorem 2) We simply combine all the relevant theorems: Theorem 8, Theorem 13, Theorem 14, and Theorem 15. The runtime of Theorem 8 is $O(W_{++}(G^{\rightarrow}) + W_{+-}(G^{\rightarrow}) + DD(G^{\rightarrow}) + m + n)$. The overhead of Theorem 8 is $O(W(G) + D(G) + m + n)$. Note that this dominates the previous runtime, since it involves undirected counts. To generate $P(i, j)$ counts, as in Theorem 14, we can easily enumerate all such three-paths from vertex i. We can also generate $W(i, j)$ counts to compute the product, and the eventual sum. Enumerating these three-paths will also find all of the directed tailed triangles of Fig. 5. Thus, we pay an additional cost of $DP(G^{\rightarrow})$. We add in the time of Theorem 15 to get the main runtime bound of $O(W + D + DP(G^{\rightarrow}) + DBP(G^{\rightarrow}) + m + n)$. The storage is dominated by Theorem 8, since we explicitly store every triangle of G.

8. EXPERIMENTAL RESULTS

We implemented our algorithms in `C++` and ran our experiments on a computer equipped with a 2x2.4GHz Intel Xeon processor with 6 cores and 256KB L2 cache (per core), 12MB L3 cache, and 64GB memory. We ran ESCAPE on a large collection of graphs from the Network Repository [53] and SNAP [54]. In all cases, directionality is ignored, and duplicate edges and self loops are omitted. Tab. 2 has the properties of all these graphs.

The entire ESCAPE package is available as open source code (including the code used in these results) at [1]. ESCAPE is a complete list of frequencies all (even disconnected) $\leq k$-vertex patterns, for choice of $k \leq 5$. For timing purposes, we run ESCAPE to count patterms for just $k = 4$, and then consider runs with $k = 5$.

4-vertex pattern counting: We compare ESCAPE for just 4-vertex pattern counting with the Parallel Parameterized Graphlet Decomposition (PGD) Library [4]. PGD can exploit parallelism using multiple threads. But our focus is on the basic algorithms, so we ran ESCAPE and PGD on a single thread. The runtimes of PGD and ESCAPE are given in Tab. 2, and the speedups, computed as ratio of PGD runtime to ESCAPE runtime, are presented in Fig. 2. PGD has not completed after over 170 and 121 hours for tech-ip and ia-wiki-user-edits graphs, respectively and thus we use these times as lower bounds for runtimes of PGD on these graphs.



(a) (b) (c)

Figure 6: Trends in pattern counts. (a) likelihood that a copy of pattern-i contains another edge, measured as $1 - C_i/N_i$ across all graphs. Patterns are labeled as k-i for ith k-pattern. (4-1 refers to a three-path. (b) Comparing transitivity of 3 common neighbors and 2 common neighbors (c) ratio of pattern 5-19 to 5-18 (the wheel)

Table 2: Properties of graph datasets

				Runtimes in seconds		
	$\|V\|$	$\|E\|$	$\|T\|$	PGD	ESC-4	ESC-5
soc-brightkite	56.7K	426K	494K	1.20	0.22	6.54
tech-RL-caida	191K	1.22M	455K	3.21	0.25	5.47
flickr	244K	3.64M	15.9M	809K	12.9	961K
ia-email-EU-dir	265K	729K	267K	10.6	0.18	8.69
ca-coauth-dblp	540K	3.05M	444M	585	615	47.4K
web-google-dir	876K	8.64M	13.4M	54.5	2.94	71.8
tech-as-skitter	1,69M	22.2M	28.8M	1.90K	20.3	1.41K
web-wiki-ch-int	1.93M	9.16M	2.63M	4.91K	6.80	798
web-hudong	1.98M	14.6M	5.07M	9.40K	13.6	534
wiki-user-edits	2.09M	11.1M	6.68M	439K	2.92	9.15K
web-baidu-baike	2.14M	17.4M	3.57M	22.9K	16.2	9.46K
tech-ip	2.25M	21.6M	298K	613K	25.7	295
orkut	3.07M	234M	628M	598K	1.19K	217K
LiveJournal	4.84M	85.7M	286M	25.9K	538	37.1K

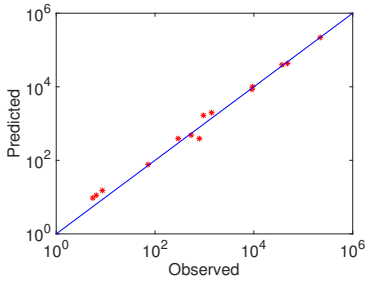


Figure 7: Predicting ESCAPE runtime for 5-patterns. The line is defined by $1.0E-4\times(1.39W(G)+1.09D(G)+24.28DP(G^{\rightarrow})+4.41DBP(G^{\rightarrow}))$.

The only instance where PGD was faster is `ca-coauthors-dblp`, where the runtimes were within 5% of each other. In almost all medium sized instances (< 10M edges), we observe a one order of magnitude of speedup on medium sized instances. For large instances (100M edges), ESCAPE gives two orders of magnitude speedup over PGD. For instance on the orkut graph with 234M edges, ESCAPE runs in less than 20 minutes, more than 500 times faster than PGD. We note that PGD is already a well-designed code based on strong algorithms. Most notably, overall runtimes are in the order of seconds for these very large graphs, as displayed on the right most column in Tab. 2. We assert that exact 4-vertex pattern counting is quite feasible, with reasonable runtimes, for even massive graphs.

5-vertex Pattern Counting: ESCAPE runtimes for counting 5-patterns are also presented in Tab. 2. We note that 5-vertex pattern counting can be done in minutes for graphs with less than 10M edges. For instance, ESCAPE computes all 5-patterns for tech-ip with 2.25M nodes and 21.6M edges in less than 5 minutes. Thus, randomization is quite unnecessary for graphs of such size. No other method we know of can handle even such medium size graphs for this problem. It is well-documented (refer to [25] for an analysis of 4-cliques, and to [4] for comparisons to PGD) that existing methods cannot scale for 10M edge graphs: FANMOD [50], edge sampling methods [45, 34], ORCA [22].

Runtime Predictions Theorem 1 and Theorem 2 claim that the runtime of the ESCAPE algorithm is bounded by the counts of specific patterns (as shown in Fig. 3). Here we present our validation for only 5-patterns due to space. We fit a line using coefficients for $W(G)$, $D(G)$, $DP(G^{\rightarrow})$ and $DBP(G^{\rightarrow})$. We do not use m and n to limit degrees of freedom. And the result is presented in Fig. 7, which shows that the runtime can be accurately predicted as a linear function of a few counts, as described in Theorem 2.

Trends in pattern counts: We analyze the actual counts of the various patterns, and glean the following trends.

- I*nduced vs non-induced:* For all patterns, we look at the ratio C_i/N_i, the fraction of non-induced matches of a pattern that are also induced. Conversely, one can interpret $1 - C_i/N_i$ as the "likelihood" that a copy of pattern (i) contains another edge. We present the results in Fig. 6(a). The surprising observation that across all the graphs, certain patterns are extremely rarely induced. It is extremely infrequent to observe patterns (16)–(20) as induced patterns. This can be a useful tool for link prediction. Note that triadic closure is commonly used for link prediction [3, 26], and our analysis here might provide higher-order link prediction tools.
- *A measure of transitivity*: What is the likelihood that vertices with two neighbors are connected by an edge? What if it was three neighbors instead? An alternate (not equivalent) measure is the see the fraction of 4-cycles that participate in diamonds, and the fraction of (13) that participate in (14). (The latter is basically taking a pattern where two vertices have three neighbors, and see how often those vertices have an edge.) Fig. 6(b) shows that having 3 common neighbors significantly increases likelihood of an edge, especially for social networks.
- *The lack of wheels*: The intriguing fact is that wheels (pattern (18)) are significantly less frequent that (19), despite having the same number of edges. Both (18) and

(19) are obtained by removing two edges from a 5-clique; in the latter, the removed edges are incident to one vertex. Fig. 6(c) plots the ratio between induced (19) and induced (18) frequencies. Why is pattern (19) much more likely to be present than wheels? Somehow, it is more common to "miss" a 5-clique by two edges incident to the same vertex than otherwise. We believe this could be the starting point for an intriguing investigation into social processes that reflect this trend.

9. REFERENCES

[1] Escape. https://bitbucket.org/seshadhri/escape.

[2] Parallel parameterized graphlet decomposition (pgd) library. http://nesreenahmed.com/graphlets/.

[3] L. Adamic and E. Adar. Friends and neighbors on the web. *Social Networks*, 25(3):211–230, 2003.

[4] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield. Efficient graphlet counting for large networks. In *Proceedings of International Conference on Data Mining (ICDM)*, 2015.

[5] N. Alon, R. Yuster, and U. Zwick. Color-coding: A new method for finding simple paths, cycles and other small subgraphs within large graphs. pages 326–335, 1994.

[6] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *KDD'08*, pages 16–24, 2008.

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

[8] N. Betzler, R. van Bevern, M. R. Fellows, C. Komusiewicz, and R. Niedermeier. Parameterized algorithmics for finding connected motifs in biological networks. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 8(5):1296–1308, 2011.

[9] M. Bhuiyan, M. Rahman, M. Rahman, and M. A. Hasan. Guise: Uniform sampling of graphlets for large graph analysis. In *Proceedings of International Conference on Data Mining*, pages 91–100, 2012.

[10] E. Birmel. Detecting local network motifs. *Electron. J. Statist.*, 6:908–933, 2012.

[11] R. Burt. Structural holes and good ideas. *American Journal of Sociology*, 110(2):349–399, 2004.

[12] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM J. Comput.*, 14:210–223, 1985.

[13] J. Cohen. Graph twiddling in a MapReduce world. *Computing in Science & Engineering*, 11:29–41, 2009.

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

[15] R. Diestel. *Graph Theory, Graduate texts in mathematics 173*. Springer-Verlag, 2006.

[16] J.-P. Eckmann and E. Moses. Curvature of co-links uncovers hidden thematic layers in the World Wide Web. *Proceedings of the National Academy of Sciences (PNAS)*, 99(9):5825–5829, 2002.

[17] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Beyond triangles: A distributed framework for estimating 3-profiles of large graphs. In *Knowledge Data and Discovery (KDD)*, pages 229–238, 2015.

[18] E. R. Elenberg, K. Shanmugam, M. Borokhovich, and A. G. Dimakis. Distributed estimation of graph 4-profiles. In *Conference on World Wide Web*, pages 483–493, 2016.

[19] K. Faust. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Social Networks*, 32(3):221–233, 2010.

[20] M. Gonen and Y. Shavitt. Approximating the number of network motifs. *Internet Mathematics*, 6(3):349–372, 2009.

[21] D. Hales and S. Arteconi. Motifs in evolving cooperative networks look like protein structure networks. *NHM*, 3(2):239–249, 2008.

[22] T. Hocevar and J. Demsar. A combinatorial approach to graphlet counting. *Bioinformatics*, 2014.

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

[24] F. Hormozdiari, P. Berenbrink, N. Prđulj, and S. C. Sahinalp. Not all scale-free networks are born equal: The role of the seed graph in ppi network evolution. *PLoS Computational Biology*, 118, 2007.

[25] M. Jha, C. Seshadhri, and A. Pinar. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *Proc. World Wide Web (WWW)*, number 1212.2264, 2015.

[26] D. Liben-Nowell and J. Kleinberg. The link prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

[27] D. Marcus and Y. Shavitt. Efficient counting of network motifs. In *ICDCS Workshops*, pages 92–98. IEEE Computer Society, 2010.

[28] I. Melckenbeeck, P. Audenaert, T. Michoel, D. Colle, and M. Pickavet. An algorithm to automatically generate the combinatorial orbit counting equations. *PLoS ONE*, 11(1):1–19, 01 2016.

[29] T. Milenkovic and N. Przulj. Uncovering Biological Network Function via Graphlet Degree Signatures. *arXiv*, q-bio.MN, Jan. 2008.

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

[31] A. Pinar, C. Seshadhri, and V. Vishal. Escape:efficiently counting all 5-vertex subgraphs. Technical report, 2016. https://users.soe.ucsc.edu/~sesh/escape.pdf.

[32] A. Portes. Social capital: Its origins and applications in modern sociology. *Annual Review of Sociology*, 24(1):1–24, 1998.

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

[34] M. Rahman, M. A. Bhuiyan, and M. A. Hasan. Graft: An efficient graphlet counting method for large graph analysis. *IEEE Transactions on Knowledge and Data Engineering*, PP(99), 2014.

[35] A. E. Sariyuce, C. Seshadhri, A. Pinar, and U. V. Catalyurek. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 927–937, New York, NY, USA, 2015. ACM.

[36] T. Schank and D. Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9:265–275, 2005.

[37] 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 Berlin / Heidelberg, 2005.

[38] C. Seshadhri, T. G. Kolda, and A. Pinar. Community structure and scale-free collections of Erdös-Rényi graphs. *Physical Review E*, 85(5):056109, May 2012.

[39] C. Seshadhri, A. Pinar, and T. G. Kolda. Fast triangle counting through wedge sampling. In *Proceedings of the SIAM Conference on Data Mining*, 2013.

[40] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *World Wide Web (WWW)*, pages 607–614, 2011.

[41] M. Szell, R. Lambiotte, and S. Thurner. Multirelational organization of large-scale social networks in an online world. *Proceedings of the National Academy of Sciences*, 107:13636–13641, 2010.

[42] J. D. Tomaz Hocevar. Combinatorial algorithm for counting small induced graphs and orbits. Technical report, arXiv, 2016. http://arxiv.org/abs/1601.06834.

[43] C. Tsourakakis, M. N. Kolountzakis, and G. Miller. Triangle sparsifiers. *J. Graph Algorithms and Applications*, 15:703–726, 2011.

[44] C. E. Tsourakakis. The k-clique densest subgraph problem. In *Conference on World Wide Web (WWW)*, pages 1122–1132, 2015.

[45] C. E. Tsourakakis, U. Kang, G. L. Miller, and C. Faloutsos. Doulion: counting triangles in massive graphs with a coin. In *Knowledge Data and Discovery (KDD)*, pages 837–846, 2009.

[46] C. E. Tsourakakis, J. W. Pachocki, and M. Mitzenmacher. Scalable motif-aware graph clustering. *CoRR*, abs/1606.06235, 2016.

[47] J. Ugander, L. Backstrom, and J. M. Kleinberg. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *WWW*, pages 1307–1318, 2013.

[48] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[49] D. Watts and S. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

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

[51] E. Wong, B. Baur, S. Quader, and C.-H. Huang. Biological network motif detection: principles and practice. *Briefings in Bioinformatics*, 13(2):202–215, 2012.

[52] Z. Zhao, G. Wang, A. Butt, M. Khan, V. S. A. Kumar, and M. Marathe. Sahad: Subgraph analysis in massive networks using hadoop. In *Proceedings of International Parallel and Distributed Processing Symposium (IPDPS)*, pages 390–401, 2012.

[53] The network repository. Available at http://www.networkrepository.com/.

[54] Stanford Network Analysis Project (SNAP). Available at http://snap.stanford.edu/.