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
scomandu@ucsc.edu

V. 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. Indeed, even a highly tuned enumeration code takes more than a day on a graph with millions of edges.

We provide an algorithm that exactly counts *all* 5-vertex (connected) subgraphs. We build a framework of 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. Our algorithm can be proven to run in $O(m\alpha^3+n)$ time where n, m are the number of vertices, edges, and α is the degeneracy of the graph.

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.

*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.

ACM ISBN TBD.
DOI: TBD

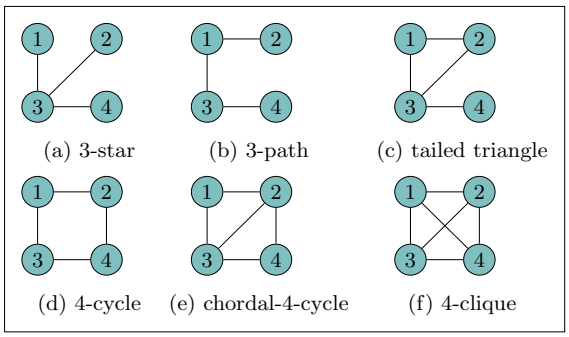


Figure 1: 4-vertex patterns

Keywords

motif analysis, subgraph counting, graph orientations

1. INTRODUCTION

Subgraph counting is a fundamental network analysis technique used across diverse domains: bioinformatics, social sciences, and infrastructure networks studies [19, 11, 27, 26, 9, 28, 14, 20, 5, 15, 37, 42, 35]. The high frequencies of certain subgraphs in real networks gives a quantifiable method of proving they are not Erdős-Rényi [19, 41, 26]. Distributions of small subgraphs are used to evaluate network models, to summarize real networks, and classify vertex roles, among other things [19, 28, 9, 20, 14, 5, 33, 13, 40].

The main challenge of motif counting is combinatorial explosion. As we see in our 5-vertex counts Tab. 2 and Tab. 3, the subgraph counts 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 scale. The key insight of this paper is *counting without enumeration* (or more precisely, counting with minimal enumeration). Most existing methods [20, 6, 45, 29] work for graphs of at most 100K edges, much smaller than the massive social networks we encounter. A notable exception is recent work on counting 4-vertex patterns, that scales to hundreds of millions of edges [3].

1.1 The problem

Our aim is simple: to exactly count the number of all connected vertex subgraphs (aka patterns, motifs, and graphlets) up to size 5 on massive graphs. There are 21 such subgraphs, as show in Fig. 2. Throughout the paper, we refer to these subgraphs/motifs by their number. (Our algorithm

also counts all connected 3 and 4 vertex patterns.) We give a formal description in §2.1.

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.

Fast and scalable. Our algorithm can provide counts for all connected patterns with up to 5-vertices, rapidly even for massive networks. For instance, processing the `as-skitter` graph with 1.7M vertices and 11.1M edges took less than 23 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 Essentially, we discover (building on existing theoretical work in pattern count) that it suffices to exhaustively enumerate a select subset of patterns to actually count all 5-vertex patterns. This is absolutely critical for exact counting, since enumeration is clearly infeasible even for graphs with a million edges. For instance, 11.1M edges of as-skitter generate 10^{14} instances of pattern 13. The mere size of the output will turn enumeration into a daunting task, while exact counts are achievable with clever data structures and combinatorial counting arguments.

Exploiting orientations. We show that orienting edges in a degeneracy style ordering can help reduce the runtimes drastically. Such techniques have been successfully applied triangle counting before. Here we show how this technique can be extended to general pattern counting. Furthermore, we can derive a formal analysis of our algorithms, and prove that five-vertex counting can be done in $O(m\alpha^3 + n)$ time, where α is the degeneracy (max core number) of the graph.

Improvements for 4-vertex patterns counting. The recent PGD package of Ahmed et al [3] has advanced the state of art significantly with better 4-vertex pattern algorithms. ESCAPE is significantly faster, even by a factor of thousands in many instances.

Trends in 5-vertex pattern counts: Our ability to count 5-vertex patterns provide a powerful graph mining tool. While a thorough analysis of 5-vertex counts is beyond the scope of this paper, we show a few examples on how our results can be potentially used for edge prediction and graph classification.

1.3 Significance of ESCAPE

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 [20, 6, 46], MCMC based sampling algorithms [7], edge sampling algorithms [29, 44, 43]. 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.

Need for exact algorithms: Motif-counting is an extremely popular research topic, and has led to wide variety of results in the past years. As mentioned above, there are numerous approximate algorithms that have been proposed, and we expect this rich area of research to yield even more results. For the purposes of validation at scale, it is critical to have a scalable, exact algorithm so that these approaches can be properly validated.

Ability to count without enumeration: A perceived bottleneck for pattern counting has been the size of the output, which is a lower bound on runtime for enumeration based algorithms. We overcome this bottleneck by replacing enumeration with counting combinations whenever possible, which helps us achieve drastic improvements in runtime.

Comprehensive results for 5-vertex patterns. We get detailed results for all 4-vertex counts on a large number of graphs, and believe this is useful for further data analysis (as done in [40]). Previous work usually focuses on a small, specific set of larger motifs [20, 6, 46], or gives coarser approximations for more motifs [7].

2. PRELIMINARIES

2.1 Formal description of the problem

Our input is an undirected simple graph $G = (V, E)$, with n vertices and m edges. We distinguish subgraphs from *induced subgraphs* [12]. A subgraph is a subset of edges. An induced subgraph is obtained by taking a subset V' of vertices, and considers *all edges* among these vertices. For convenience, we use ith 4-pattern to be the ith subgraph in Fig. 1 and analogously, refer to the jth 5-patterns. For example, the 6th 4-pattern in the four-clique and the 8th 5-pattern is the five-cycle.

It is convenient to focus on outputting induced subgraph counts. We use C_i (resp. $nonind_i$) 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 we will provide it in an extended form of this paper.

2.2 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 of a vertex by $d(i)$. We denote the number of triangles by $T(G)$, and use $T(i)$, $T(e)$ to denote the number of triangles incident to vertex i and edge e respectively. Analogously, we use $C_4(G)$, $C_4(i)$, and $C_4(e)$ to denote the number of 4-cycles (pattern (d) in Fig. 1). Similarly, $K_4(\cdot)$ is used to denote 4-clique counts. We use $CC(\cdot)$ to denote the number of chordal-cycles (pattern (e) in Fig. 1). For all patterns in Fig. 2, we use N_i to denote the non-induced count of pattern i.

We denote the *degree ordering* of G by $\prec$. Thus, 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* D by orienting all edges in G according to $\prec$.

A *wedge* is a path of length 2. The total number of wedges is denoted by W, and $W(i, j)$ is the number of wedges with vertices i and j as endpoints. In D, there are three types of wedges: out-wedges, inout-wedges, and in-wedges (refer to Fig. 3 and Fig. 4) The number of out-wedges is denoted by $W_{++}(D)$, and the number of out-wedges with endpoints i, j is denoted as $W_{++}(i, j)$. Analogously, the number of inout-wedges and in-wedges are denoted by $W_{+-}(\cdot)$ and $W_{--}(\cdot)$.

We use a classic result to enumerate triangles.

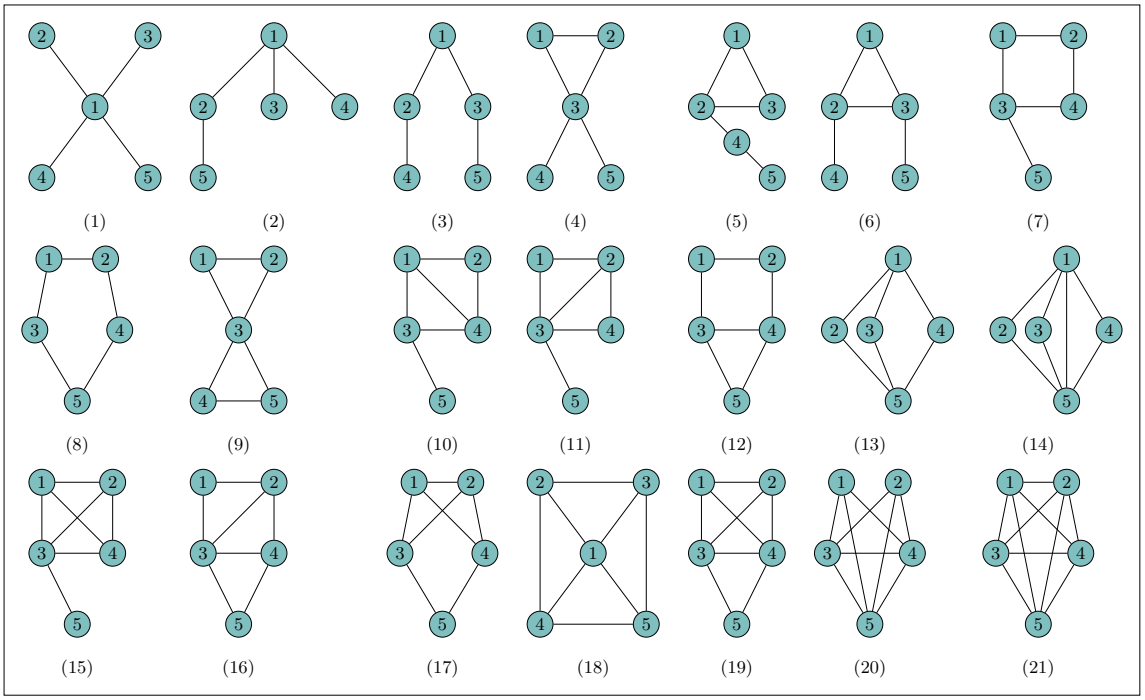


Figure 2: 5-vertex patterns

THEOREM 1. *[10, 32] Triangles can be enumerated in $O(W_{++}(G)+degree\ counting$, which has gained a lot of attention recently $n+m)$ time and $O(m+n)$ space. (Thus, all values $T(i)$ and $T(e)$ can computed in this time.)*

2.3 Related Work

The seminal work of Milo *et al.* [26] was one of the first to emphasize significance of motif counting in bioinformatics. This work has led to many studies based on pattern counts in biological networks. For example, Przulj *et al* [28] use pattern counts to argue that protein-protein interaction networks resemble a particular network model than previously thought. Similarly, in [20], motifs distribution is described as an important characteristic for assessment of network models. Hales and Arteconi [17] found that motifs profile of peer-to-peer networks are similar to those of protein networks. Motif counts has also been used for hierarchal network decomposition [21]. Refer to [6, 45] for more details.

Recently, Sariyuce et al. [30] nucleus decomposition, which generalizes k-core and k-truss decompositions to identify all dene subgraphs of a graph and identifying a hierarchy among these subgraphs. Their algorithm is based how many times a smaller cliques appear as a part of a larger clique, and thus ability to enumerate these cliques is a key part of the performance of their algorithm.

Triangle counting has a rich history in network analysis, that we simply refer the reader to the related work sections of [39, 34]. Ugander et al. [40], 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).

Gonen and Shavitt [16] 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*

counting, which has gained a lot of attention recently (see [25, 36, 16, 8]). Marcus and Shavitt [24] 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 [44, 43], which takes about 3 hours.

A breakthrough in exact 4-vertex pattern counting was recently achieved by Ahmed et al [3, 4]. Using techniques on graph transitions based on edge addition/removal, their PGD (Parametrized Graphlet Decomposition) package handles extremely large graphs (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 [38, 18]. We consider PGD to be the state-of-the-art for 4-vertex pattern counting. (We would also like to commend the authors on making their code public [2].) ESCAPE is significantly faster than PGD, as we show in our experimental results.

As an alternative exact counting, Jha et al. [22] proposed 3-path sampling to estimate all 4-vertex counts. Their technique builds on *wedge sampling* [31, 34, 23] and samples paths of length 3 to estimate various 4-vertex statistics. Their techniques come with error bars for the estimation, and can provide accurate estimates with in seconds even for the graphs with hundreds of millions of edges. However, these techniques do not provide per vertex statistics.

3. MAIN THEOREMS

Our final algorithms are quite complex and use a variety of combinatorial methods for efficiency. Nonetheless, the fi-

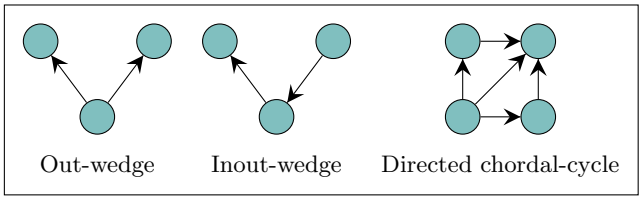


Figure 3: Fundamental patterns for 4-vertex pattern counting

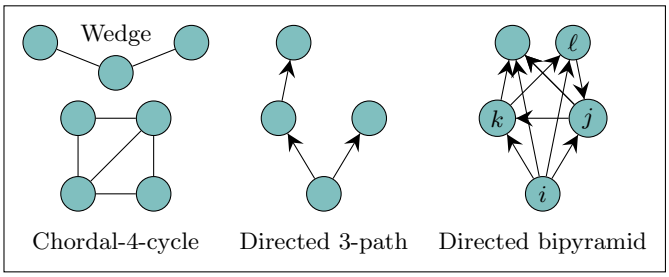


Figure 4: Fundamental patterns for 5-vertex pattern counting

nal 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.

Consider input graph G, with an associated DAG D.

THEOREM 2. *Let the number of the directed chordal-cycle in D be DCC. There is an algorithm for exactly counting all connected 4-vertex patterns in G whose runtime is $O(W_{++} + W_{+-} + DCC + m + n)$ and storage is $O(n + m)$.*

THEOREM 3. *Let the number of the specific directed 3-path and the specific directed bipyramid shown in Fig. 3 be DP and DBP respectively. There is an algorithm for exactly counting all connected 5-vertex patterns in G whose runtime is $O(W + CC + DP + DBP + m + n)$ and storage complexity is $O(n + m + T)$.*

How it works: Note that two of the three patterns used in Theorem 3 have fewer than 5 vertices. One may wonder how we can generate counts for all patterns from enumerating just these ones. The algorithm builds numerous indices and data structures from this enumeration, and carefully pieces together this information to get all counts. Furthermore, the algorithm uses this information to perform even more enumerations, but the running time is guaranteed to be bounded by the expressions in the theorems.

Closed form bounds: We can give closed form (but not tight) bounds using the *degeneracy* (commonly called the maximum core number) of the graph, denoted α. This is a classic graph theoretical quantity, and known to be relatively small compared to n. If D is given by the degeneracy/core-decomposition ordering, then the out-degree of each vertex is at most α. We can immediately bound the number of patterns using α. The bounds also hold for the degree ordering, using results of Chiba-Nishizeki [10]. We note that the same bound for 4-vertex counting has also been achieved in [10] (by an alternate algorithm). ESCAPE is the first practical algorithm that gets this time bound.

COROLLARY 4. *Let α be the degeneracy/max core-number of G. The counts of all connected 4-vertex patterns can be determined in $O(m\alpha^2 + n)$ time. The counts of all connected 5-vertex patterns can be determined in $O(m\alpha^3 + n)$ time.*

A classic bound is $\alpha \leq \sqrt{m}$, thus for *any* graph, we can count all 5-vertex patterns in $O(m^{5/2} + n)$ time.

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 in Tab. 2 and Tab. 3 are upwards of many billions for most of the graphs (which are 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* that the number of patterns. This is because we have to find *all potential* patterns, which is often much higher. 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. Every pattern, besides the clique, has a non-trivial cut set. Let H be the pattern. 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 copies in H.

- For each set of k vertices in G, the number of copies of $H_1, H_2, \ldots$ that involve S.
- The number of copies 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 formulas for counting most of the 5-vertex patterns. These formulas can be efficiently evaluated with appropriate data structure.

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 a DAG, and search for directed triangles [10, 32]. Thus, we actually search for *all* non-isomorphic DAG versions of the pattern H. When H is not a clique, we can apply Idea 1 to design algorithms to count each of these directed version. The key insight is that the resulting enumeration problems are significantly faster to perform, since the direction cuts down the combinatorial explosion.

When H is a clique, we simply perform an enumeration that builds smaller directed cliques, and tries to piece them together to get larger ones. This gives major speedup over an undirected enumeration. This is well documented for counting triangles [32], and we observe it again for counting 4 and 5 cliques.

5. THE CUTTING FRAMEWORK

Let H be a pattern we wish to count in G. For any set of vertices S in H, $H|_S$ is the subgraph of H induced on S. Only in this section, we will consider G and H to be labeled. This is critical to provide a formal analysis of the cutting framework.

We formally define a match and a partial match of the patterm 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.

If π is only an injection (so $|S| < |V(H)|$), then π is a *partial match*. The set of distinct matches of H in G is denoted $\text{match}(H)$. The set of distinct partial matches is denoted $\text{pmatch}(H)$.

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. The set of distinct matches of H in H is the automorphism set of H, denoted by $\text{Aut}(H)$.

DEFN. 3. Let σ be a partial match of H in G. The H-degree of σ is the number of matches of H that are extensions of σ.

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

DEFN. 4. Consider H with some non-trivial cut set C (so $|C| < |V(H)|$), who removal leads to connected components $S_1, S_2, \ldots$. The *C-fragments* of H are the graphs 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. 5. Consider graphs H, H', and a non-trivial cut set H 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.
- Each $\pi_i : F_i \to H'$ is a partial match.
- Each π_i extends σ.
- For each edge (u, v) of H', there is some i and vertices $a, b \in F_i$ such that $\pi_i(a) = u$ and $\pi_i(b) = v$.

The set of graphs 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 valid 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. (The quantity $\text{numSh}_C(H, H')$ is an absolute constant, independent of G.) Note that all F and shrinkages have less vertices than H, and thus we can build upon counts of smaller vertex patterns to count H.

LEMMA 5. *Consider pattern H with cut set C. Then,*

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

PROOF. Consider any copy σ of $H|_C$. Take all tuples of the form $(\pi_1, \pi_2, \ldots, \pi_{|\text{Frag}_{(H)}|})$ 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 are 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.

Note that to get the final *unlabeled* frequency, we must normalize and use $\text{match}(H)/|\text{Aut}(H)|$.

To demonstrate this lemma, let us derive counts for 5-pattern (2). We use the labeling in Fig. 2.

Pattern 2: Let us use edge $(1, 2)$ as the cut set S. Thus, the fragments are F_1, the wedge $\{(1, 2), (2, 5)\}$ and F_2, the three-star $\{(1, 2), (1, 3), (1, 4)\}$. Any match of $H|_S$ is simply every edge. Consider (u, v) with match $\sigma(u) = 1$ and $\sigma(v) = 2$. The degree $\deg_{F_1}(\sigma)$ is $d(v) - 1$. The degree $\deg_{F_2}(\sigma)$ is $(d(u) - 1)(d(u) - 2)$.

The only possible shrinkage is into a tailed triangle. Furthermore, $\text{numSh}_C(H, H')$ is 2, as given by the following maps. 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.

Thus

$$\begin{aligned} \text{match}(H) = \sum_{(u,v) \in E} \big[&(d(v) - 1)(d(u) - 1)(d(u) - 2) \\ &+ (d(u) - 1)(d(v) - 1)(d(v) - 2)\big] - 2 \cdot \text{match}(\text{tailed triangle}) \end{aligned}$$

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 $\sum_v t(v)(d(v)-2)$ (we can also derive this from Lemma 5). Thus,

$$N_2 = \sum_{\langle u,v \rangle \in E} (d(v) - 1) \binom{d(u) - 1}{2} - 2 \sum_v t(v)(d(v) - 2)$$

6. COUNTING 4-VERTEX PATTERNS

A good introduction to our techniques is our algorithm for counting 4-vertex patterns. The following formulas have been proven in [22, 3], but can be derived using the framework of Lemma 5.

THEOREM 6.

$$\begin{aligned} \#\ \textit{3-stars} &= \sum_v \binom{d(v)}{3} \\ \#\ \textit{3-paths} &= \sum_{(u,v) \in E} (d(u) - 1)(d(v) - 1) - 3 \cdot T(G) \\ \#\ \textit{tailed-triangles} &= \sum_v t(v)(d(v) - 2) \\ CC(G) &= \sum_e \binom{t(e)}{2} \end{aligned}$$

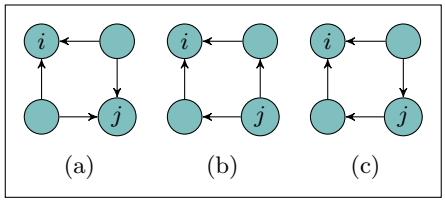


Figure 5: All acyclic orientations of the 4-cycle

For counting 4-cycles, we can use the cutting framework. Note that any set of opposite vertices (like 1 and 4 in 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 it requires a complete wedge enumeration. Employing the direction, we can prove a significant improvement. With a little care, we can get counts per edge.

THEOREM 7. *$C_4(G) = \sum_{i \succ j} \binom{W_{++}(i,j)+W_{+-}(i,j)}{2}$. For edge (i,k) where $i \succ k$, $C_4(i,k) = \sum_{j \succ k} [W_{++}(i,j) + W_{+-}(i,j)] + \sum_{j \prec k} W_{+-}(i,j)$.*

PROOF. Consider all DAG versions of the 4-cycle, as given in Fig. 5. 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 outwedges, 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 have to look at the wedges using this wedge. Consider some wedge involving j. When we take $j \succ k$, we could be in type (a) or type (c). If $j \prec k$, we are considering type (b). Summing over the possibilities of the other wedge, we complete the proof.

THEOREM 8. *(We remind the reader that DCC is the number of directed chordal-cycles, as given in Fig. 3.) The number of four-cliques per-vertex and per-edge can be found in time $O(W_{++} + W_{+-} + DCC)$ and $O(m)$ additional space.*

PROOF. Let H denote the directed chordal-cycle 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, where i is the smallest vertex according to $\prec$. It is possible to enumerate all such patterns in time linear in DCC. 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. 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.

The main 4-vertex counting theorem, Theorem 2, follows directly from the previous theorems. It is useful to state a stronger version involving more counts. This is a direct consequence of the ability to enumerate triangles. This will be used for 5-vertex counting.

THEOREM 9. *In $O(W_{++}+W_{+-}+DCC+m)$ 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.*

7. ONTO 5-VERTEX COUNTS

With the cut framework of §5, we can count all 5-vertex patterns, barring the 5-cycle (pattern 8) and the 5-clique (pattern 21). We give the formulas that Lemma 5 yields. It is cumbersome and space-consuming to give proofs of all of these, so we omit them. In some cases, we give some minor explanation. We break the formulas into four groups, depending on whether the cut chosen in a vertex, edge, triangle, or wedge.

THEOREM 10. *[Cut is vertex]*

$$N_1 = \sum_i \binom{d(i)}{4}$$

$$N_3 = \sum_i \sum_{(i,j)\in E} (d(j)-1) - 4\cdot C_4(G) - 2\cdot TT(G) - 3\cdot T(G)$$

$$N_4 = \sum_i t(i)(d(i)-2)$$

$$N_7 = \sum_i C_4(i)(d(i)-2) - 2\cdot CC(G)$$

$$N_9 = \sum_i \binom{t_v}{2} - 2\cdot CC(G)$$

$$N_{15} = \sum_i K_4(i)(d(i)-3)$$

THEOREM 11. *[Cut is edge]*

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

$$N_5 = \sum_{e=\langle i,j\rangle\in E} (d(i)-1)(d(j)-T(e)) - 4\cdot CC(G)$$

$$N_6 = \sum_{e=(i,j)\in E} t_e(d(i)-2)(d(j)-2) - 2\cdot CC(G)$$

$$N_{11} = \sum_{e=\langle i,j\rangle\in E} \binom{T(e)}{2}(d(i)-3)$$

$$N_{12} = \sum_{e\in E} C_4(e)T(e) - 4\cdot CC(G)$$

$$N_{14} = \sum_e \binom{T(e)}{3}$$

$$N_{19} = \sum_e K_4(e)(t(e)-2)$$

THEOREM 12. *[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. 2 for labels. For pattern 10, we use vertices 1, 2, 4 as the cut. For pattern 16, we use vertices 2, 3, 4 as the cut. For pattern 20, we use vertices 3, 4, 5 as the cut. The formulas can be derived directly, or by using Lemma 5.

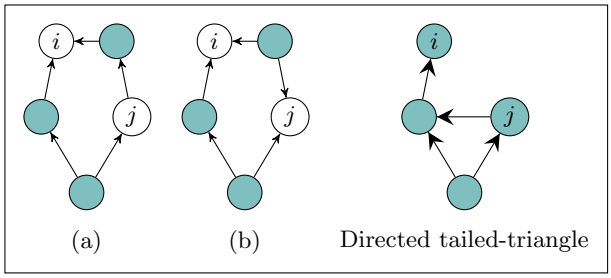


Figure 6: Directed patterns for 5-cycle and 5-clique counting

THEOREM 13. *[Cut is pair or wedge] Define $CC(i,j)$ to be the number of chordal-cycles 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 chordal-cycles where i maps to 1, j maps to 2 and k maps to 4.*

$$N_{13} = \sum_{i \succ j} \binom{W(i,j)}{2}$$

$$N_{17} = \sum_{i \succ j} (W(i,j) - 2) CC(i,j)$$

$$N_{18} = \sum_{i,j,k} \binom{CC(i,j,k)}{2}$$

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 chordal-cycles. For the wheel (pattern 18), we use the "diagonal" 2, 1, 5 as the cut. The fragments are both chordal-cycles sharing those vertices.

THEOREM 14. *Assume we have all the information from Theorem 9. All counts in the above theorems can be computed in time $O(W + CC + n + m)$.*

PROOF. The counts of Theorem 10, Theorem 11, and Theorem 12 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 $CC(i,j)$ and $CC(i,j,k)$ 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 9). 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 chordal-cycle 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 chordal-cycles. 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 D, analogous to 4-cycles and 4-cliques.

THEOREM 15. *Consider the 3-path in Fig. 6, 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. 6. The number of 5-cycles is $\sum_{i,j} P(i,j) \cdot (W_{++}(i,j) + W_{+-}(i,j)) - Z$.*

PROOF. Fig. 6 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. The vertices have the same directed three-path between them. They also have either an outwedge or inout-wedge connecting them. Thus, the product $\sum_{i,j} P(i,j) \cdot (W_{++}(i,j) + W_{+-}(i,j))$ counts each 5-cycle exactly once. The shrinkage of either directed 5-cycle yields the directed tailed-triangle of Fig. 6(d). This pattern is also counted exactly once in the product above. (One can formally derive this relation using Lemma 5.) Thus, we substract Z out to get the number of 5-cycles.

THEOREM 16. *(We remind the reader that DBP is the count of the directed bipyramid in Fig. 4.) The number of 5-cliques can be counted in time $O(DBP + CC + T + n + m)$.*

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 9, 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. 4.

Suppose triangle t consists of vertices i,j,k. We enumerate every other triangle incident to j,k using the data structure of Theorem 9. 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 $\sum_{t=(i,j,k)} t(j,k) \leq \sum_{j,k} t(j,k)^2 = O(CC(G) + T(G))$. Once these lists are generated, the additional time is exactly DBP to generate each directed pattern.

At long last, we can prove Theorem 3.

PROOF. (of Theorem 3) We combine Theorem 9, Theorem 14, Theorem 15, and Theorem 16. The runtime of Theorem 9 is $O(W_{++} + W_{+-} + DCC + m + n)$. The overhead of Theorem 9 is $O(W + CC + m + n)$. Note that this dominates the previous runtime, since it involves undirected counts. To generate $P(i,j)$ counts, as in Theorem 15, 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. 6. Thus, we pay an additional cost of DP. We add in the time of Theorem 16 to get the main runtime bound of $O(W + CC + DP + DBP + m + n)$. The storage is dominated by Theorem 9, 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 performed our experiments on 20 graphs collected from the Network Repository. [1] In all cases, directionality is ignored, and duplicate edges and self loops are omitted. Tab. 1 has the properties of these graphs, where $|V|$ and $|E|$ are the numbers of vertices and edges, respectively.

The entire ESCAPE package is available as open source code (including the code used in these results) at BitBucket [1].

[1] http://www.networkrepository.com/

8.1 Results on 4-vertex Patterns

To assess the effectiveness of our algorithmic techniques, we compared our runtimes with Parallel Parameterized Graphlet Decomposition (PGD) Library [3]. PGD can exploit parallelism using multiple threads, but our focus is on the algorithms, and thus to compare the algorithmic techniques we ran PGD on a single thread (our algorithms can be parallelized the same way as in PGD, but this is not yet implemented in the code).

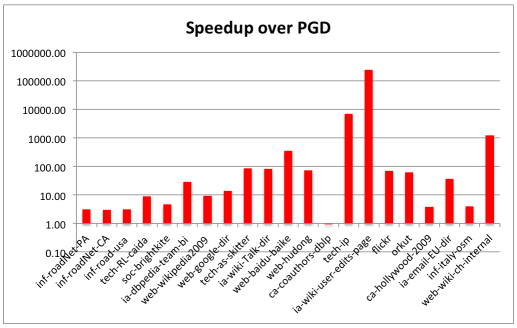


Figure 7: Speedup achieved by Escape over PGD (computed as runtime of ESCAPE/runtime of PGD).

Fig. 7 shows the speedup achieved by ESCAPE over PGD. Note that the vertical axis in this figure uses logarithmic scale. In our experiments, PGD failed to complete in two graphs, tech-ip and wiki-user-edits-page after 658K seconds, and we stopped these processes and used this time as the completion time. The figure shows that ESCAPE offers significant speedups over PGD. The only exception is `ca-coauthors-dblp`, where PGD was 8% faster than ESCAPE. One of the algorithmic insights of ESCAPE is using the edge orientation as explained earlier, but as Tab. 1 shows, edge orientation offers little in this case (W_{++} and W_{+-} wedges make up a significant portion of all wedges), which explains the ESCAPE performance for this graph. We should also note that overall runtimes are in the order of seconds for these very large graphs, as displayed on the right most column in Tab. 1. For instance, computing exact counts on the `as-skitter` graph with 1.7M vertices and 11.1M edges took only 21.79 seconds.

8.2 Results on 5-vertex Patterns

After validating the effectiveness of our algorithmic approach on 4-vertex vertices, we applied our techniques to counting 5-vertex patterns. We report the induced counts for all connected 5-vertex patterns in Tab. 2 and Tab. 3. The runtime for each instance is also reported on the right most column of Tab. 3. In the next section, we will discuss the information revealed by these numbers. Here we want to note that our code enables computation of exact counts for 5-vertex patterns on massive graphs in reasonable times. For instance, processing the `as-skitter` graph with 1.7M vertices and 11.1M edges took less than 23 minutes. The worst instance was `ca-coauthors-dblp` (with 30M edges), which took 10 hours. We want to note that this is a computation once deemed intractable beyond graphs with thousands of vertices.

In Fig. 8, we break down the runtime into different segments of the algorithm. The preprocess step corresponds to triangle and 4-vertex pattern enumeration. The figure shows that the bottleneck segment varies based the properties of the graph. On average, 5-cycle and 5-clique times dominates the times of any other pattern, but this is not always the case. These numbers raises the question of how our algorithms can be tailored for properties of the given graph.

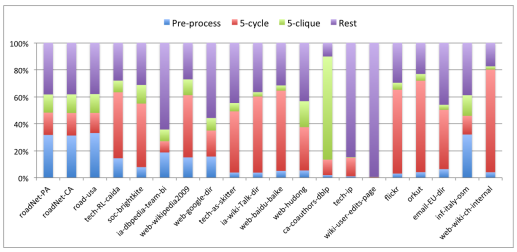


Figure 8: Breakdown of runtime into segments of the algorithm

8.3 What do the pattern counts reveal?

A thorough analysis of 5-vertex counts will be interesting and valuable, but it is beyond the scope of this paper. Here, we will only give examples of potential analyses. Fig. 9(a) shows the ratio of noninduced counts for near cliques (pattern 20) to cliques (pattern 21). This means, if we know of the presence of all edges of a 5-clique but one, how often is that remaining edge also present? As this figure shows, the remaining edge of a near clique is almost always present, which may be a great way to predict missing edges. Fig. 9(b) shows the ratio of induced 5-cycles to noninduced 5-cycles. The results show two distinct modes, which can be used for graph classification. This ratio is very high for infrastructure networks, but very low for networks with a social structure, where we expect to see transitivity. Fig. 9(c) shows the ratio of noninduced counts of triplets of wedges (pattern 13) to triplets of triangles (pattern 14). We see a similar two mode distribution here with a low ratio for infrastructure networks. Moreover, this ratio can be used as a missing edge predictor, akin to Jaccard index like methods.

9. REFERENCES

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

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

[3] 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.

[4] N. K. Ahmed, J. Neville, R. A. Rossi, and N. Duffield, *Fast parallel graphlet counting for large networks*, Tech. Report 1506.04322, Arxiv, 2015.

[5] L. Becchetti, P. Boldi, C. Castillo, and A. Gionis, *Efficient semi-streaming algorithms for local triangle counting in massive graphs*, in KDD'08, 2008, pp. 16–24, doi:10.1145/1401890.1401898.

[6] 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 (2011), pp. 1296–1308.

[7] 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, 2012, pp. 91–100.

[8] E. Birmel, *Detecting local network motifs*, Electron. J. Statist., 6 (2012), pp. 908–933, doi:10.1214/12-EJS698, http://dx.doi.org/10.1214/12-EJS698.

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

Table 1: Properties of the graphs. Columns show counts or relative frequencies of patterns with up to 4-vertices. The right most column reports the 4-vertex counting time by ESCAPE in seconds.

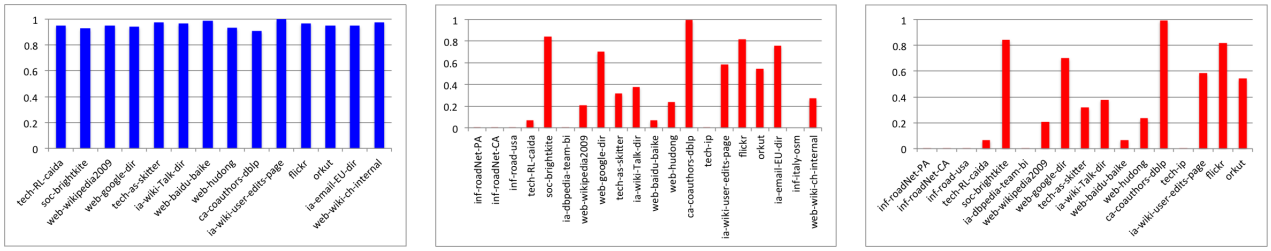
	V	E	$\frac{W_{++}}{W}$	$\frac{W_{+-}}{W}$	T	3-star	3-path	tailed triangle	4-cycle	Chordal 4-cycle	4-clique	Time (secs)
roadNet-PA	1.1e+6	3.1e+6	0.217	0.398	6.7e+4	1.4e+6	6.2e+6	2.9e+5	1.5e+5	5.7e+3	16	0.23
roadNet-CA	2e+6	5.5e+6	0.215	0.402	1.2e+5	2.4e+6	1.1e+7	5.2e+5	2.5e+5	1.3e+4	40	0.41
road-usa	2.4e+7	5.8e+7	0.212	0.345	4.4e+5	1.8e+7	8.1e+7	1.5e+6	1.6e+6	2.1e+4	90	5.26
tech-RL-caida	1.9e+5	1.2e+6	0.082	0.173	4.5e+5	1.7e+9	5.8e+8	7.7e+7	4e+7	7.4e+6	4.2e+5	0.33
soc-brightkite	5.7e+4	4.3e+5	0.076	0.194	4.9e+5	1.3e+9	5.3e+8	1.1e+8	2.7e+6	1.2e+7	2.9e+6	0.24
ia-dbpedia-team-bi	3.7e+5	8.4e+5	0.011	0.010	5.2e+2	3.9e+9	1.9e+8	3.5e+5	5.5e+5	2e+3	0	0.17
web-wikipedia2009	1.9e+6	9e+6	0.064	0.175	2.2e+6	1e+10	4.3e+9	4e+8	6.4e+7	3e+7	1.5e+6	3.76
web-google-dir	8.8e+5	4.7e+6	0.025	0.035	2.5e+6	8.3e+10	1.5e+9	5.1e+8	1.4e+7	2.4e+7	2e+6	4.38
tech-as-skitter	1.7e+6	2.2e+7	0.006	0.013	2.9e+7	9.6e+13	8.2e+11	1.6e+11	4.3e+10	2e+10	1.5e+8	21.79
ia-wiki-Talk-dir	2.4e+6	4.8e+6	0.004	0.025	2e+6	2.4e+13	2e+11	7.2e+9	1.6e+8	9.6e+7	3.9e+6	28.58
web-baidu-baike	2.1e+6	1.7e+7	0.008	0.021	3.6e+6	7.2e+13	4.8e+11	2.3e+10	5.9e+8	1.8e+8	7.1e+5	62.76
web-hudong	2e+6	1.5e+7	0.008	0.015	5.1e+6	1.5e+13	2.2e+11	6.3e+9	2.8e+8	2.5e+8	8.3e+7	134.92
ca-coauthors-dblp	5.4e+5	3e+7	0.232	0.313	4.4e+8	2.7e+10	4.2e+10	6.7e+10	3.1e+7	3.4e+9	1.5e+10	596.3
tech-ip	2.3e+6	2.2e+7	0.001	0.000	3e+5	1.3e+17	1.4e+13	8.6e+9	8e+11	3e+7	2	93.84
wiki-user-edits-page	2.1e+6	1.1e+7	0.000	0.000	6.7e+6	8.8e+16	4.8e+12	2e+12	4.4e+10	7e+10	1e+7	12.17
flickr	1.7e+6	3.6e+6	0.013	0.029	1.6e+7	1.1e+13	4.9e+11	1.4e+11	2.4e+9	4.7e+9	1.2e+8	3.02
orkut	3.1e+6	3.6e+7	0.016	0.034	6.1e+7	7e+13	1.5e+12	2.2e+11	8.4e+9	7.4e+9	3.3e+8	57.74
ia-email-EU-dir	2.7e+5	7.3e+5	0.005	0.023	2.7e+5	2.2e+11	4.4e+9	3.4e+8	6.7e+6	1e+7	5.8e+5	0.33
italy-osm	6.7e+6	1.4e+7	0.098	0.692	7.4e+3	9.9e+5	9.9e+6	2.7e+4	4.7e+4	2.4e+2	0	0.77
web-wiki-ch-internal	1.9e+6	1.8e+7	0.015	0.047	1.8e+7	3.1e+13	1.3e+12	1.3e+11	4.2e+9	2.1e+9	3e+7	4.38

Table 2: Induced counts for 5 vertex patterns; Patterns 1–12

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
roadNet-PA	2.2e+5	7.5e+6	1.2e+7	9.2e+4	5.7e+5	4.2e+5	9e+5	8.5e+4	1.1e+4	1.7e+4	9.5e+3
roadNet-CA	3.5e+5	1.3e+7	2e+7	1.6e+5	1e+6	7.2e+5	1.4e+6	1.5e+5	2e+4	3.6e+4	2e+4
road-usa	2.4e+6	8.4e+7	1.3e+8	2.9e+5	2.4e+6	1.7e+6	9.4e+6	7.1e+5	1.6e+4	4.7e+4	2.2e+4
tech-RL-caida	2.4e+11	8e+10	2.1e+10	8.8e+9	2.1e+9	5.1e+9	1.1e+10	3.4e+7	7e+7	5.1e+8	1.8e+9
soc-brightkite	2.3e+11	1.3e+11	2.2e+10	1.7e+10	4e+9	9.7e+9	1.2e+9	3.5e+7	2.2e+8	1.2e+9	2.7e+9
ia-dbpedia-team-bi	6.3e+11	4.2e+10	1.3e+10	8.7e+7	2.6e+6	7.5e+7	3.6e+8	2.2e+5	3.9e+3	2.9e+4	2.3e+6
web-wikipedia2009	2.6e+12	7.6e+11	2.3e+11	4e+10	1.8e+10	4.9e+10	3.9e+10	6e+8	4.3e+8	6.4e+9	1.1e+10
web-google-dir	4e+13	1.1e+12	4.9e+10	3.6e+11	3.5e+9	2e+10	7.3e+9	2.7e+7	3.5e+8	1e+9	1e+10
tech-as-skitter	6e+17	6.4e+15	7.1e+14	1.3e+15	7.3e+12	2.7e+14	7e+14	3e+11	3.6e+11	3.3e+12	2.9e+14
ia-wiki-Talk-dir	2.7e+17	6e+14	3.7e+13	1.3e+13	8.9e+11	9e+12	9.3e+11	1.3e+10	8.5e+9	1.8e+11	3.2e+11
web-baidu-baike	7.6e+17	4.5e+15	1.5e+14	2.8e+14	1.4e+12	4.4e+13	1.9e+13	2e+10	2.6e+10	1e+11	5.4e+12
web-hudong	7.2e+16	1.2e+15	6.1e+13	4.2e+13	3.2e+11	1.2e+13	2.2e+12	3.7e+9	3.2e+9	3.2e+10	8e+11
ca-coauthors-dblp	5.7e+12	5e+12	2.8e+12	4.9e+12	4.2e+12	4.2e+12	1.1e+10	9.8e+8	1.7e+12	4.2e+11	3.6e+11
tech-ip	4.1e+18	4.2e+17	4.6e+16	1.4e+14	1.2e+12	9.7e+12	3e+16	1.7e+11	8.2e+8	1.3e+11	9.5e+11
wiki-user-edits-page	2.5e+18	1e+18	2.7e+16	4.7e+17	1.1e+13	6.7e+16	1.4e+16	1.2e+11	1.4e+12	5.5e+12	3.8e+16
flickr	4.6e+16	2.2e+15	1.3e+14	6.7e+14	8.9e+12	2.1e+14	2e+13	2e+11	7.4e+11	4.4e+12	5.8e+13
orkut	3.6e+17	1.2e+16	8.8e+14	1.4e+15	1.5e+13	7.1e+14	1.2e+14	2.6e+11	5.5e+11	6.9e+12	1.2e+14
ia-email-EU-dir	3.1e+14	4.3e+12	3.9e+11	1.9e+11	1.5e+10	1.4e+11	1e+10	1.4e+8	2e+8	4.7e+9	1.1e+10
italy-osm	1e+5	4e+6	1.2e+7	4.8e+3	3.8e+4	3.1e+4	2.3e+5	3.3e+4	3.8e+2	5.6e+2	4.4e+2
web-wiki-ch-internal	1.7e+17	8e+15	3.8e+14	1e+15	1.5e+13	2.2e+14	5.2e+13	2.4e+11	6.5e+11	2.6e+12	3.9e+13

Table 3: 5-vertex patterns; Patterns 13–21 and runtimes in seconds

	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21	Time
inf-roadNet-PA	2.9e+4	1.6e+2	1	35	6.8e+2	70	57	1	0	0	1.88
-roadNet-CA	5.4e+4	4.6e+2	1	1.2e+2	1.6e+3	1.4e+2	3.1e+2	1	0	0	3.47
-road-usa	1.1e+5	4.2e+2	1	1.7e+2	1e+3	4.2e+2	2.9e+2	1	0	0	45.98
tech-RL-caida	9.8e+7	4.9e+9	3e+8	1.4e+8	6.4e+7	4.1e+7	5.1e+6	3.1e+7	5.8e+6	6.5e+5	5.96
soc-brightkite	1.2e+8	1.4e+7	1.4e+8	6.2e+8	2.4e+8	2.4e+7	1.5e+7	1.9e+8	6.2e+7	1.9e+7	6.73
ia-dbpedia-team-bi	1.8e+4	2.7e+6	1.4e+4	0	2.2e+2	2.5e+2	1	0	0	0	0.88
web-wikipedia2009	1.7e+9	2.4e+9	7e+8	4.2e+8	6.2e+8	5.8e+8	1.3e+8	1e+8	1.7e+7	1.7e+6	50.7
web-google-dir	9.7e+7	3.5e+8	8.8e+8	3.2e+8	8.7e+7	2.5e+7	9e+6	3.7e+7	8.7e+6	1.4e+6	15.94
tech-as-skitter	4.6e+11	1.6e+14	7.2e+13	1.3e+12	5e+11	1.3e+11	2.4e+10	2.9e+11	3.8e+10	1.2e+9	1359.41
ia-wiki-Talk-dir	1.5e+10	3e+9	1.8e+9	2.1e+10	5.8e+9	1.4e+9	2.3e+8	9.3e+8	8.9e+7	4.5e+6	264.66
web-baidu-baike	1.5e+10	1.3e+12	9.4e+10	8e+9	4.4e+9	4.9e+8	2.9e+7	3.4e+8	6.9e+6	9.9e+4	793.91
web-hudong	3.9e+9	1.3e+11	1.2e+10	7.9e+9	5.4e+9	2.4e+9	2.6e+9	6e+9	4.6e+9	1.2e+9	506.9
ca-coauthors-dblp	1.1e+10	2e+7	2.6e+10	3.2e+12	5.2e+10	7.6e+8	3.2e+8	3.5e+11	3.6e+10	5.7e+11	37226.12
tech-ip	2.5e+11	3.2e+15	2.7e+9	4.4e+4	7.9e+8	4.7e+10	1.5e+8	2.1e+2	0	0	10339.12
wiki-user-edits-page	5.7e+11	1.6e+15	2.2e+15	3.4e+12	1.2e+12	6.6e+10	1.5e+10	4.6e+11	2e+10	1.4e+7	11255.04
flickr	5.8e+11	3.4e+11	1.8e+12	1.6e+12	6.9e+11	7.2e+10	1.8e+10	1.9e+11	1.5e+10	7.1e+8	917.57
orkut	8.1e+11	2.9e+12	3.4e+12	1.5e+12	6.3e+11	1.4e+11	3.2e+10	1.5e+11	1.7e+10	1.8e+9	3359.56
ia-email-EU-dir	2.7e+8	8.3e+7	3.8e+8	1.1e+9	2e+8	4.2e+7	9.8e+6	7e+7	9.9e+6	1.1e+6	8.37
italy-osm	1.2e+3	21	0	0	28	2	4	0	0	0	5.59
web-wiki-ch-internal	3.9e+11	2.7e+12	1e+12	4.3e+11	2e+11	4e+10	3.5e+9	3.1e+10	1.3e+9	4e+7	1630.92



(a) (b) (c)

Figure 9: Trends in 5-vertex counts. Ratio of (a) near cliques to cliques (b) Induced 5-cycles to non-induced 5-cycles (c) Triplets of wedges (pattern 13) to triplets of triangles (pattern 14)

[10] N. Chiba and T. Nishizeki, *Arboricity and subgraph listing algorithms*, SIAM J. Comput., 14 (1985), pp. 210–223.

[11] J. Coleman, *Social capital in the creation of human capital*, American Journal of Sociology, 94 (1988), pp. S95–S120, http://www.jstor.org/stable/2780243.

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

[13] N. Durak, A. Pinar, T. G. Kolda, and C. Seshadhri, *Degree relations of triangles in real-world networks and graph models*, in CIKM'12, 2012.

[14] G. Fagiolo, *Clustering in complex directed networks*, Phys. Rev. E, 76 (2007), p. 026107, doi:10.1103/PhysRevE.76.026107, http://link.aps.org/doi/10.1103/PhysRevE.76.026107.

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

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

[17] D. Hales and S. Arteconi, *Motifs in evolving cooperative networks look like protein structure networks.*, NHM, 3 (2008), pp. 239–249, http://dblp.uni-trier.de/db/journals/nhm/nhm3.html#HalesA08.

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

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

[20] F. Hormozdiari, P. Berenbrink, N. Prdulj, 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).

[21] S. Itzkovitz, R. Levitt, N. Kashtan, R. Milo, M. Itzkovitz, and U. Alon, *Coarse-graining and self-dissimilarity of complex networks*, 71 (2005).

[22] 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), no. 1212.2264, 2015.

[23] T. G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, and C. Task, *Counting triangles in massive graphs with MapReduce*, SIAM Journal of Scientific Computing, (2013), arXiv:1301.5887. To appear.

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

[25] T. Milenkovic and N. Przulj, *Uncovering Biological Network Function via Graphlet Degree Signatures*, arXiv, q-bio.MN (2008), http://arxiv.org/abs/0802.0556v1, arXiv:0802.0556v1.

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

[27] A. Portes, *Social capital: Its origins and applications in modern sociology*, Annual Review of Sociology, 24 (1998), pp. 1–24, doi:10.1146/annurev.soc.24.1.1.

[28] N. Przulj, D. G. Corneil, and I. Jurisica, *Modeling interactome: scale-free or geometric?.*, Bioinformatics, 20 (2004), pp. 3508–3515, http://dblp.uni-trier.de/db/journals/bioinformatics/bioinformatics20.html#PrzuljCJ04.

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

[30] 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, New York, NY, USA, 2015, ACM, pp. 927–937, doi:10.1145/2736277.2741640, http://doi.acm.org/10.1145/2736277.2741640.

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

[32] T. Schank and D. Wagner, *Finding, counting and listing all triangles in large graphs, an experimental study*, in Experimental and Efficient Algorithms, Springer Berlin / Heidelberg, 2005, pp. 606–609, doi:10.1007/11427186_54.

[33] C. Seshadhri, T. G. Kolda, and A. Pinar, *Community structure and scale-free collections of Erdös-Rényi graphs*, Physical Review E, 85 (2012), p. 056109, doi:10.1103/PhysRevE.85.056109.

[34] C. Seshadhri, A. Pinar, and T. G. Kolda, *Fast triangle counting through wedge sampling*, in Proceedings of the SIAM Conf. Data Mining, 2013, http://arxiv.org/abs/1202.5230.

[35] S. Son, A. Kang, H. Kim, T. Kwon, J. Park, and H. Kim, *Analysis of context dependence in social interaction networks of a massively multiplayer online role-playing game*, PLoS ONE, 7 (2012), p. e33918, doi:10.1371/journal.pone.0033918, http://dx.doi.org/10.1371%2Fjournal.pone.0033918.

[36] A. Stoica and C. Prieur, *Structure of neighborhoods in a large social network*, in CSE (4), IEEE Computer Society, 2009, pp. 26–33.

[37] M. Szell and S. Thurner, *Measuring social dynamics in a massive multiplayer online game*, Social Networks, 32 (2010), pp. 313–329.

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

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

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

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

[42] B. Welles, A. Van Devender, and N. Contractor, *Is a friend a friend?: Investigating the structure of friendship networks in virtual worlds*, in CHI-EA'10, 2010, pp. 4027–4032.

[43] S. Wernicke, *Efficient detection of network motifs*, IEEE/ACM Trans. Comput. Biology Bioinform., 3 (2006), pp. 347–359.

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

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

[46] Z. Zhao, G. Wang, A. Butt, M. Khan, V. S. A. Kumar, and M. Marathe, *Sahad: Subgraph analysis in massive networks using hadoop*, in Proc. Intl. Parallel and Distributed Processing Symposium (IPDPS), 2012, pp. 390–401.