

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

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.

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

1. INTRODUCTION

*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.

Subgraph counting is a fundamental network analysis technique used across diverse domains: bioinformatics, social sciences, and infrastructure networks studies [25, 14, 33, 32, 11, 34, 20, 26, 6, 21, 45, 54, 41]. The high frequencies of certain subgraphs in real networks gives a quantifiable method of proving they are not Erdős-Rényi [25, 53, 32]. Distributions of small subgraphs are used to evaluate network models, to summarize real networks, and classify vertex roles, among other things [25, 34, 11, 26, 20, 6, 39, 16, 51].

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 [26, 8, 57, 35] 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 [3].

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:5vertex]Fig. ??fig:5vertex. Additionally, there are 11 disconnected patterns, which we discuss in [sec:disc]§??sec:disc of the Appendix. 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 [sec:problem]§??sec:problem.

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 that have been proposed for this problem [56, 49, 9, 40, 35, 27]. 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

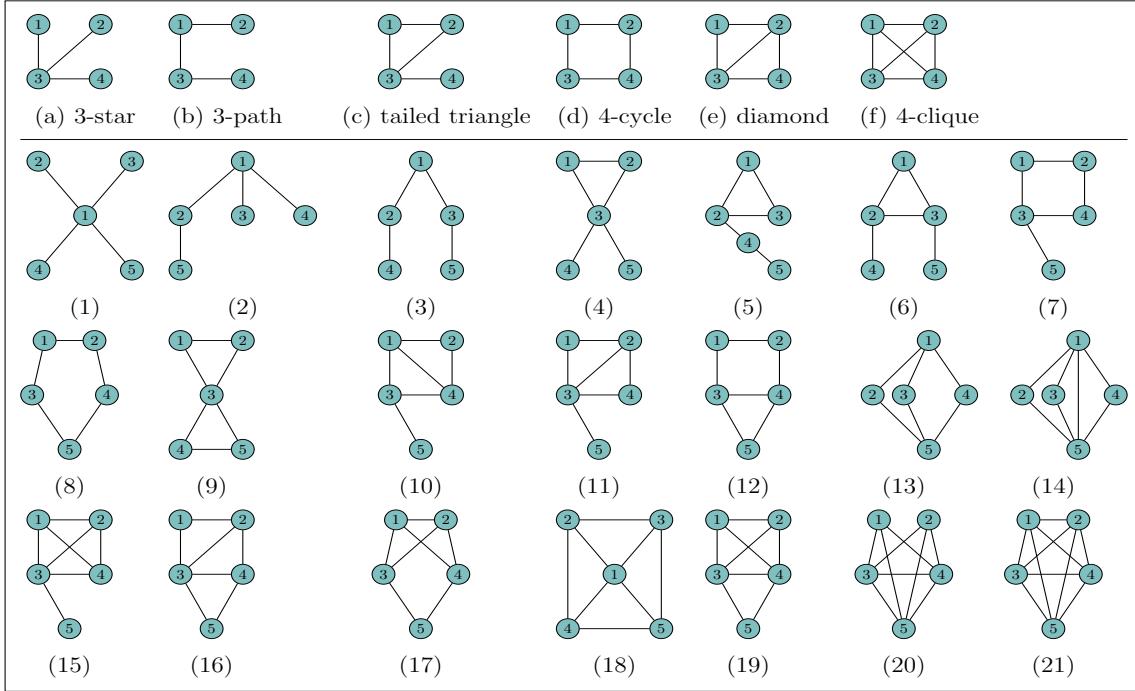


Figure 1: Connected 4 and 5-vertex patterns

machine using 64GB memory.

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 [26, 8, 58], MCMC based sampling algorithms [9], edge sampling algorithms [35, 56, 55]. 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 [24, 3, 18, 19] 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 autonomous systems graph with 11M edges has 10^{11} instances of pattern 17 (in [fig:5vertex]Fig. ??fig:5v). 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, 43]. 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. [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 some 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.

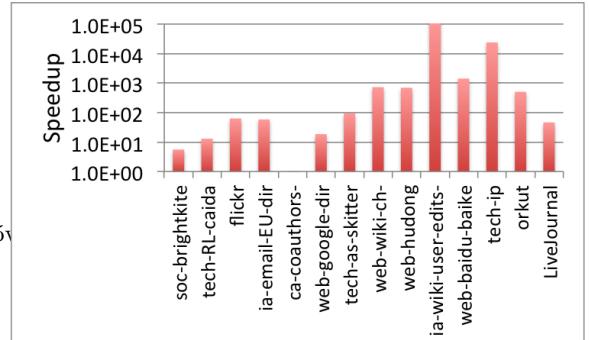


Figure 2: Speedup achieved by Escape over PGD (computed as runtime of PGD/runtime of ESCAPE).

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 [25, 14, 52, 53, 17, 32, 11, 20, 6, 54, 21, 44, 45, 41]. Specifically in bioinformatics, pattern counts have significant relevance in graph classification [32, 34, 23].

In social networks, Ugander et al. [51], 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 [36, 48, 7, 50].

From the practical algorithmics standpoint, triangle counting has received much attention. We simply refer the reader to the related work sections of [47, 40]. Gonen and Shavitt [22] 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 [31, 42, 22, 10]). Marcus and Shavitt [29] 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 [56, 55], 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 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 [46, 24]. 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. [18, 19] 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 [3, 46, 19] 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 [30]. 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. [27] proposed 3-path sampling to estimate all 4-vertex counts. Their technique builds on *wedge sampling* [37, 40, 28] and samples paths of length 3 to estimate various 4-vertex statistics.

To the best of our knowledge, there is no method (ap-

proximate or exact) that can count all 5-vertex patterns for graphs with millions of edges.

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* [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 [thm:disc]Theorem ??thm:disc, 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:5vertex]Fig. ??fig:5vertex. For convenience, the i th 4-pattern refers to i th subgraph with 4 vertices in [fig:5vertex]Fig. ??fig:5vertex. 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 i th 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. we provide details in [sec:transform]§??sec:transform of the Appendix.

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 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:notation]Tab. ??tab:notation. Counts of certain patterns, especially those in [fig:5vertex-basis]Fig. ??fig:5vertex-basis, 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.

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

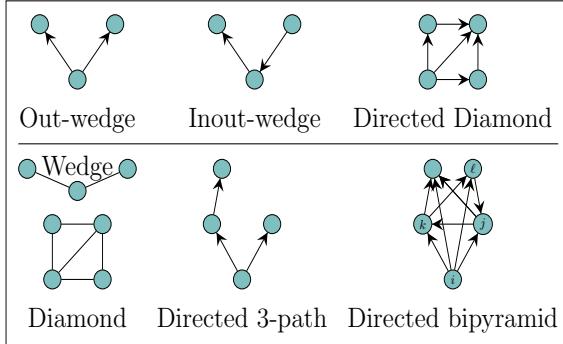


Figure 3: Fundamental patterns for 4-vertex (above) and 5-vertex (below) pattern counting

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 $O(n + m + T(G))$.*

Note that previous theorems only handle connected patterns. But a routine inclusion-exclusion argument yields the following theorem.

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: [sec:ideas]§??sec:ideas gives a high level overview of our main techniques. [sec:cut]§??sec:cut discusses the cutting framework used to reduce counting all patterns into enumeration of some specific patterns (namely, those in [fig:5vertex-basis]Fig. ??fig:5vertex-basis). In [sec:fourvertex]§??sec:fourvertex we apply this framework to 4-vertex pattern count, and prove [thm:fourvertex]Theorem ??thm:fourvertex. In [sec:fivevertex]§??sec:fivevertex we work towards 5-vertex pattern counting and prove [thm:fivevertex]Theorem ??thm:fivevertex.

The proof and discussion of [thm:disc]Theorem ??thm:disc is omitted because of space constraints and appears in [sec:disc]§??sec:disc of the Appendix. In the remainder of this main body of the paper, we only focus on connected patterns.

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, \dots . 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, \dots 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_i s, 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, 38, 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 can be used for algorithm design and 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 basi-

cally an induced subgraph of G that is exactly H .

DEFN. 1. A *match* of H is a bijection $\pi : S \rightarrow V(H)$ 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 $\text{match}(H)$.

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

A match $\pi : S \rightarrow V(H)$ extends a partial match $\sigma : T \rightarrow V(H)$ if $S \supseteq T$ 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, \dots . The C -*fragments* of H are the subgraphs of H induced by $C \cup S_1, C \cup S_2, \dots$. 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, \dots . A C -*shrinkage* of H into H' is a set of maps $\{\sigma, \pi_1, \pi_2, \dots, \pi_{|\text{Frag}_C(H)|}\}$ with the following properties.

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

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 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,

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

PROOF. Consider any copy σ of $H|_C$. Take all tuples of the form $(\pi_1, \pi_2, \dots, \pi_{|\text{Frag}_C(H)|})$ 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 ≤ 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 [lem:cut]Lemma ??lem:cut. 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:5vertex][Fig. ??fig:5vertex]. 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 $\text{match } \sigma(i) = 1$ and $\sigma(j) = 2$. The degree $\deg_{F_1}(\sigma)$ is $d(j) - 1$. The degree $\deg_{F_2}(\sigma)$ is $(d(i) - 1)(d(j) - 2)$.

The only possible shrinkage of the patterns is into a tailed triangle. 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

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

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 $\sum_i t(i)(d(i) - 2)$ (we can also derive this from [lem:cut]Lemma ??lem:cut). Thus,

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

6. COUNTING 4-VERTEX PATTERNS

A good introduction to these techniques is counting 4-vertex patterns. The following formulas have been proven in [27, 3], but can be derived using the framework of [lem:cut]Lemma ??lem:

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

For counting 4-cycles, note any set of opposite vertices (like 1 and 4 in 4-cycle of [fig:5vertex]Fig. ??fig:5vertex) 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:4cycle-DAG]Fig. ??fig:4cycle-DAG. 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 j s, 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:5vertex-basis]Fig. ??fig:5vertex-basis.) 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:5vertex-basis]Fig. ??fig:5vertex-basis. 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, [thm:fourvertex]Theorem ??thm:fourvertex. 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

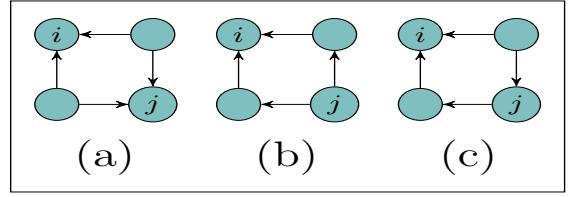


Figure 4: All acyclic orientations of the 4-cycle

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, 38]. (We used the same argument to prove [thm:4clique]Theorem ??thm:4clique.) By [thm:4vertex-easy]Theorem ??thm:4vertex-easy, once we have per-vertex and per-edge triangle counts, we can count everything other than 4-cycles and 4-cliques in linear time. By [thm:4cycle]Theorem ??thm:4cycle, enumerating outout and inout wedges suffices for 4-cycle counting. We add the bound on [thm:4clique]Theorem ??thm:4clique to complete the proof.

7. ONTO 5-VERTEX COUNTS

With the cut framework of [sec:cut]§??sec:cut, 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 [lem:cut]Lemma ??lem:cut 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 in 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_i s.

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

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

THEOREM 11. [Cut is triangle]

$$N_{10} = \sum_{t=(i,j,k) \text{ triangle}} [(t(i,j)-1)(d(k)-1)] - 4 \cdot K_4(G)$$

$$N_{16} = \sum_{t=(i,j,k) \text{ triangle}} (t(i,j)-1)(t(i,k)-1)$$

$$N_{20} = \sum_t \binom{K_4(t)}{2} - 4 \cdot K_4(G)$$

PROOF. Refer to [fig:5vertex]Fig. ??fig:5vertex for labels. We will apply [lem:cut]Lemma ??lem:cut, where C will be

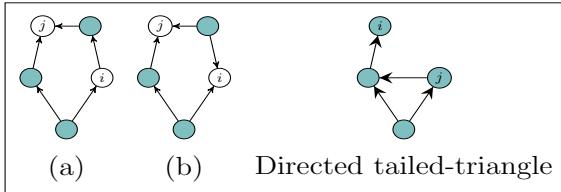


Figure 5: Directed patterns for 5-cycle and 5-clique counting

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 [lem:cut]Lemma ??lem:cut.

THEOREM 12. [*Cut is pair or wedge*] Define $D(i, j)$ to be the number of diamonds involving i and j where i and j are not connected to the chord (in [fig:5vertex]Fig. ??fig:5vertex, 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.

$$\begin{aligned} N_{13} &= \sum_{i \prec j} \binom{W(i, j)}{3} \\ N_{17} &= \sum_{i \prec j} (W(i, j) - 2)D(j, i) \\ N_{18} &= \sum_{i, j, k} \binom{D(i, j, k)}{2} \end{aligned}$$

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. The following theorem (and proof) show an algorithm that uses the formulas given above.

THEOREM 13. Assume we have all the information from [thm:4vertex-tri]Theorem ??thm:4vertex-tri. All counts in [thm:5cutvertex]Theorem ??thm:5cutvertex, [thm:5cutedge]Theorem ??thm:5cutedge, and [thm:5cuttri]Theorem ??thm:5cuttri can be computed in time $O(W(G) + D(G) + n + m)$ and $O(n + m + T(G))$ storage.

PROOF. The counts of [thm:5cutvertex]Theorem ??thm:5cutvertex, [thm:5cutedge]Theorem ??thm:5cutedge, and [thm:5cuttri]Theorem ??thm:5cuttri 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 [thm:4vertex-tri]Theorem ??thm:4vertex-tri). 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:5cycle]Fig. ??fig:5cycle, 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:5cycle]Fig. ??fig:5cycle. The number of 5-cycles is $\sum_{i \prec j} P(i, j) \cdot (W_{++}(i, j) + W_{+-}(i, j)) - Z$.

PROOF. [fig:5cycle]Fig. ??fig:5cycle 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 outwedge 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:5cycle]Fig. ??fig:5cycle(d).

This pattern is also counted exactly once in the product above. (One can formally derive this relation using [lem:cut]Lemma ??lem:cut.) Thus, we subtract 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:5vertex-basis]Fig. ??fig:5vertex-basis.) 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 [thm:4vertex-tri]Theorem ??thm:4vertex-tri, 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:5vertex-basis]Fig. ??fig:5vertex-basis.

Suppose triangle t consists of vertices i, j, k . We enumerate every other triangle incident to j, k using the data structure of [thm:4vertex-tri]Theorem ??thm:4vertex-tri. 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 $\sum_{t=(i,j,k)} t(j, k) \leq \sum_{j,k} t(j, k)^2 = O(D(G) \cdot 5CT(G))$. Once these lists are generated, the additional time is exactly DBP to generate each directed pattern.

At long last, we can prove [thm:fivevertex]Theorem ??thm:fivevertex.

PROOF. (of [thm:fivevertex]Theorem ??thm:fivevertex) We simply combine all the relevant theorems: [thm:4vertex-tri]Theorem ??thm:4vertex-tri, [thm:5formula]Theorem ??thm:5formula, [thm:5cycle]Theorem ??thm:5cycle, and [thm:5clique]Theorem ??thm:5clique. The runtime of [thm:4vertex-tri]Theorem ??thm:4vertex-tri is $O(W_{++}(G^\rightarrow) + W_{+-}(G^\rightarrow) + DD(G^\rightarrow) + m + n)$. The overhead of [thm:4vertex-tri]Theorem ??thm:4vertex-tri 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 [thm:5cycle]Theorem ??thm:5cycle, 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:5cycle]Fig. ??fig:5cycle. Thus, we pay an additional cost of $DP(G^\rightarrow)$. We add in the time of [thm:5clique]Theorem ??thm:5clique to get the main runtime bound of $O(W + D + DP(G^\rightarrow) + DBP(G^\rightarrow) + m + n)$. The storage is dominated by [thm:4vertex-tri]Theorem ??thm:4vertex-tri, 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 [59] and SNAP [60]. In all cases, directionality is ignored, and duplicate edges and self loops are omitted. [tab:properties]Tab. ??tab:properties 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].

4-vertex pattern counting: We compare ESCAPE with the Parallel Parameterized Graphlet Decomposition (PGD) Library [3]. 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:properties]Tab. ??tab:properties and the speedups, computed as ratio of PGD runtime to ESCAPE runtime, are presented in [fig:4speedup]Fig. ??fig:4speedup. 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 for these graphs. The only instance where PGD was faster is ca-coauthors dblp, where the runtimes were comparable. 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 more than 500 times faster than PGD. We should also 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:properties]Tab. ??tab:properties. For instance, computing exact counts on the as-skitter graph with 1.7M vertices and 11.1M edges took only 21.79 seconds. We assert that exact 4-vertex pattern counting is quite feasible, with reasonable runtimes, for even massive graphs. We present counts of all 5-patterns in [sec:fullcounts]§??sec:fullcounts of the Appendix.

5-vertex Pattern Counting: ESCAPE runtimes for counting 5-patterns are also presented in [tab:properties]Tab. ??tab:properties. 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 [27] for an analysis of 4-cliques, and to [3] for comparisons to PGD) that existing methods cannot scale for 10M edge graphs: FANMOD [56], edge sampling methods [49, 35], ORCA [24].

Runtimes Predictions [thm:fourvertex]Theorem ??thm:fourvertex and [thm:fivevertex]Theorem ??thm:fivevertex claim that the runtime of the ESCAPE algorithm is bounded by the counts of specific patterns (as shown in [fig:5vertex-basis]Fig. ??fig:5vertex basis). 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

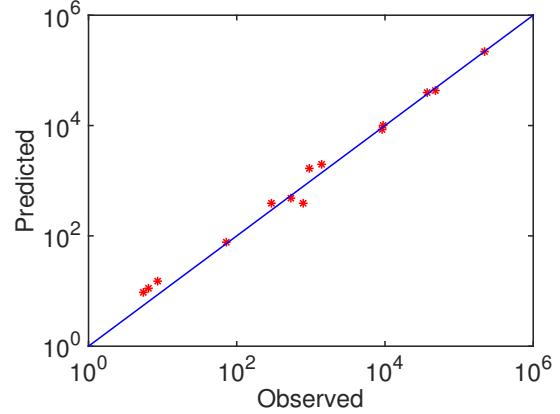


Figure 6: Predicting ESCAPE runtime for 5-patterns. The line is defined by $1.0E - 4 * (1.39W(G) + 1.09cc(G) + 24.28DP(G^\rightarrow) + 1.1DBP(G^\rightarrow))$.

[fig:predict]Fig. ??fig:predict, which shows that the runtime can be accurately predicted as a function of counts for base patterns as described in [thm:fivevertex]Theorem ??thm:fivevertex.

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

- *Induced 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:inducedratio]Fig. ??fig:inducedratio. The surprising observation that across all the graphs, certain patterns are extremely rarely induced. It is extremely infrequent to observe 5-patterns 16–20 as induced patterns, which can be good tool for edge prediction. Note that wedges/triangles are commonly used for edge prediction, but across all graphs wedge-to-triangle closure is not frequent at all. This ratio can be high for some graphs, for an arbitrary graph wedges by themselves are not good edge predictors. Also note that 5-pattern 1 frequently remains as a induced pattern, even though it has 6 potential missing edges. These results show that going beyond 3 and 4-vertex patterns can reveal more interesting structures and provide predictive power.

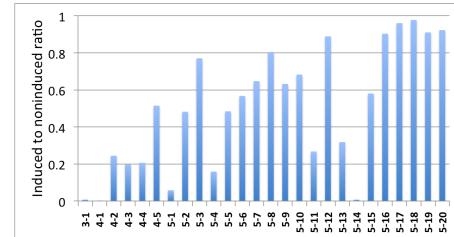


Figure 7: Likelihood that a copy of pattern-i contains another edge, measured as $1 - C_i/N_i$ across all graphs. Patterns are labeled as i-j for j-th i-pattern. 3-1 refers to a two path.

- *A measure of transitivity:* What is the likelihood that to vertices with two neighbors are connected by an edge? What if it was three neighbors instead? An alternate (not equivalent) method to measure this is the see the fraction of 4-cycles that form diamonds, and the fraction of (13) that form

Table 1: Properties of the graphs.

	$ V $	$ E $	$ T $	Runtimes in seconds		
				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

(14). (The latter is basically taking a pattern where two vertices have three neighbors, and see how often those vertices have an edge.) [fig:transitivity] Fig. ??fig:transitivity shows that having 3 common neighbors significantly increases likelihood of an edge, especially for social networks.

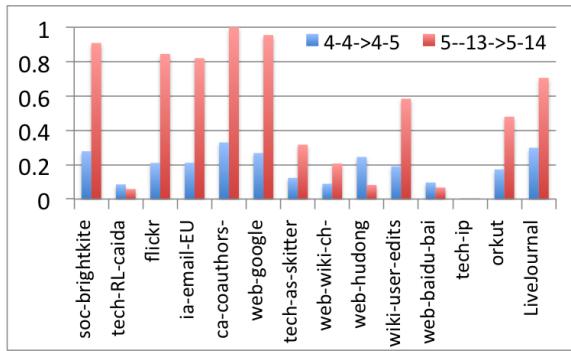


Figure 8: Comparing transitivity of 3 common neighbors and 2 common neighbors

- *The lack of wheels:* The intriguing fact is that wheels (pattern (18)) are much rarer than one would expect. It appears to be an “unstable” pattern. [fig:inducedratio] Fig. ??fig:inducedratio already shows that they are infrequent as induced patterns. Here we will go a step further and how often pattern 18 has an additional edge to turn into pattern 20, and as a basis for comparison we will compare it with that of pattern 19, which has the same number of edges. Results are presented in [fig:wheel] Fig. ??fig:wheel, which shows that P18 is more than twice as likely to turn into Pattern 20 compared to pattern 19.

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. Technical Report 1506.04322, Arxiv, 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 algorithms 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.

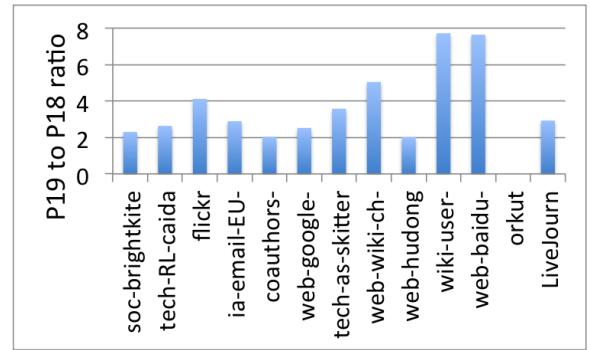


Figure 9: Ratio of 5-pattern 19 to 5-pattern 18.

- [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] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] G. Fagiolo. Clustering in complex directed networks. *Phys. Rev. E*, 76:026107, Aug 2007.
- [21] K. Faust. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Social Networks*, 32(3):221–233, 2010.
- [22] M. Gonen and Y. Shavitt. Approximating the number of network motifs. *Internet Mathematics*, 6(3):349–372, 2009.
- [23] D. Hales and S. Arteconi. Motifs in evolving cooperative networks look like protein structure networks. *NHM*, 3(2):239–249, 2008.
- [24] T. Hocevar and J. Demsar. A combinatorial approach to graphlet counting. *Bioinformatics*, 2014.
- [25] P. Holland and S. Leinhardt. A method for detecting structure in sociometric data. *American Journal of Sociology*, 76:492–513, 1970.
- [26] 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.
- [27] 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.
- [28] T. G. Kolda, A. Pinar, T. Plantenga, C. Seshadhri, and C. Task. Counting triangles in massive graphs with mapreduce. *SIAM J. Scientific Computing*, 36(5), 2014.
- [29] D. Marcus and Y. Shavitt. Efficient counting of network motifs. In *ICDCS Workshops*, pages 92–98. IEEE Computer Society, 2010.
- [30] 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.
- [31] T. Milenkovic and N. Przulj. Uncovering Biological Network Function via Graphlet Degree Signatures. *arXiv*, q-bio.MN, Jan. 2008.
- [32] 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.
- [33] A. Portes. Social capital: Its origins and applications in modern sociology. *Annual Review of Sociology*, 24(1):1–24, 1998.
- [34] N. Przulj, D. G. Corneil, and I. Jurisica. Modeling interactome: scale-free or geometric?. *Bioinformatics*, 20(18):3508–3515, 2004.
- [35] 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.
- [36] A. E. Sarıyuce, 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.
- [37] T. Schank and D. Wagner. Approximating clustering coefficient and transitivity. *Journal of Graph Algorithms and Applications*, 9:265–275, 2005.
- [38] 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.
- [39] C. Seshadhri, T. G. Kolda, and A. Pinar. Community structure and scale-free collections of Erdős-Rényi graphs. *Physical Review E*, 85(5):056109, May 2012.
- [40] C. Seshadhri, A. Pinar, and T. G. Kolda. Fast triangle counting through wedge sampling. In *Proceedings of the SIAM Conference on Data Mining*, 2013.
- [41] 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(4):e33918, 04 2012.
- [42] A. Stoica and C. Prieur. Structure of neighborhoods in a large social network. In *CSE (4)*, pages 26–33. IEEE Computer Society, 2009.
- [43] S. Suri and S. Vassilvitskii. Counting triangles and the curse of the last reducer. In *World Wide Web (WWW)*, pages 607–614, 2011.
- [44] 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.
- [45] M. Szell and S. Thurner. Measuring social dynamics in a massive multiplayer online game. *Social Networks*, 32:313–329, 2010.
- [46] J. D. Tomaz Hocevar. Combinatorial algorithm for counting small induced graphs and orbits. Technical report, arXiv, 2016. <http://arxiv.org/abs/1601.06834>.
- [47] C. Tsourakakis, M. N. Kolountzakis, and G. Miller. Triangle sparsifiers. *J. Graph Algorithms and Applications*, 15:703–726, 2011.
- [48] C. E. Tsourakakis. The k-clique densest subgraph problem. In *Conference on World Wide Web (WWW)*, pages 1122–1132, 2015.
- [49] 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.

- [50] C. E. Tsourakakis, J. W. Pachocki, and M. Mitzenmacher. Scalable motif-aware graph clustering. *CoRR*, abs/1606.06235, 2016.
- [51] 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.
- [52] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.
- [53] D. Watts and S. Strogatz. Collective dynamics of ‘small-world’ networks. *Nature*, 393:440–442, 1998.
- [54] 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*, pages 4027–4032, 2010.
- [55] S. Wernicke. Efficient detection of network motifs. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 3(4):347–359, 2006.
- [56] S. Wernicke and F. Rasche. Fanmod: a tool for fast network motif detection. *Bioinformatics*, 22(9):1152–1153, 2006.
- [57] 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.
- [58] 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.
- [59] The network repository. Available at <http://www.networkrepository.com/>.
- [60] Stanford Network Analysis Project (SNAP). Available at <http://snap.stanford.edu/>.

APPENDIX

A. DISCONNECTED PATTERNS

Disconnected patterns can be easily counted in extra *constant* time, if we have all connected patterns up to a given size.

THEOREM 16. *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).*

PROOF. Basically, we show that the counts of disconnected patterns is just a polynomial function of counts of connected patterns with fewer vertices. Formally, we prove by induction on c , the number of connected components in the pattern. The base case, $c = 1$, is trivially true by assumption. For induction, suppose we have counts for all patterns (of at most k vertices) with at most c connected components. Consider pattern H with connected components H_1, H_2, \dots, H_{c+1} . The count is the number of 1-1 matches $f : V(H) \rightarrow V(G)$ mapping edges of H to edges of G . Observe that the function “partitions” into 1-1 functions $f_i : V(H_i) \rightarrow V(G)$ ($\forall i \leq c+1$) such that all images are disjoint. But the number of f_i ’s is exact the number of matches of H_i . We now use a one-step inclusion-exclusion

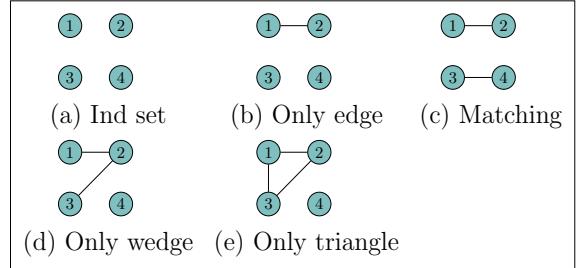


Figure 10: Disconnected 4-vertex patterns

to determine $\text{match}(H)$.

$$\text{match}(H) = \prod_{i=1}^{c+1} \text{match}(H_i) - |\mathcal{F}|$$

where \mathcal{F} is the set of maps $f : V(H) \rightarrow V(G)$ such that each $f|_{V(H_i)}$ is 1-1, but f is not 1-1. This can only happen if f maps to a single vertex some set $S \subseteq V(H)$ that is *not* contained in a single $V(H_i)$.

We now partition \mathcal{F} according to the following scheme. Consider a non-trivial partition \mathcal{S} of $V(H)$ with the following condition: a non-singleton in \mathcal{S} cannot be contained in some $V(H_i)$. Define $\mathcal{F}_{\mathcal{S}}$ to be the set of maps where each $S \in \mathcal{S}$ is mapped to a single vertex, but these vertices are different for distinct \mathcal{S} . Observe that for any $f \in \mathcal{F}_{\mathcal{S}}$, $f|_{V(H_i)}$ is 1-1 for all $i \leq c+1$. Thus, \mathcal{F} is partitioned into the $\mathcal{F}_{\mathcal{S}}$, and $|\mathcal{F}| = \sum_{\mathcal{S}} |\mathcal{F}_{\mathcal{S}}|$.

Crucially, we note that $\mathcal{F}_{\mathcal{S}}$ is exactly the set of matches for the pattern $H_{\mathcal{S}}$ created by merging $V(H)$ as follows: for each $S \in \mathcal{S}$, merge S into a single vertex. Since \mathcal{S} is non-trivial, it contains some non-singleton S . This non-singleton is not contained in any $V(H_i)$; thus, merging S reduces the number of connected components in H . Thus, $H_{\mathcal{S}}$ has strictly less than $c+1$ connected components, and by induction, we already know its count.

Thus, we write our main equation as follows, and observe that the right hand side involves counts that are already known (by induction).

$$\text{match}(H) = \prod_{i=1}^{c+1} \text{match}(H_i) - \sum_{\mathcal{S}} \text{match}(H_{\mathcal{S}})$$

B. CONVERSION BETWEEN INDUCED AND NONINDUCED COUNTS

We show how to convert induced pattern counts to non-induced counts, and vice versa. Algorithmically, it is often easier to get non-induced counts, and a linear transformation of these counts suffices to get induced counts. It is standard that each non-induced pattern count can be obtained as a linear combination of induced pattern counts (of that size). We use C_i (resp. N_i) to denote the induced (resp. non-induced) count of the i th 5-pattern. We can think of the list of counts as vectors C, N . There is a matrix \mathbf{A} such that for all graphs, $N = \mathbf{AC}$. The matrix is given in [fig:A]Fig. ??fig:A. It is constructed by noting that $\mathbf{A}_{i,j}$ is the number of copies of pattern j in pattern i . Naturally, $C = \mathbf{A}^{-1}N$, and that is how we get induced counts from non-induced counts. The inverse matrix is given in [fig:Ainv]Fig. ??fig:Ainv.

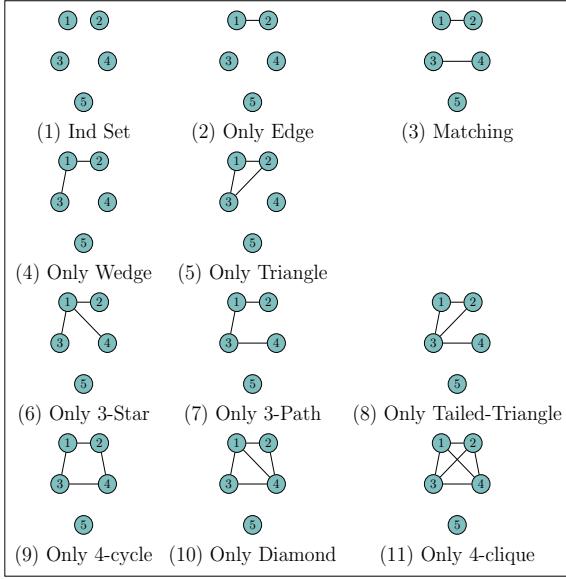


Figure 11: Disconnected 5-vertex patterns

C. DETAILED INDUCED COUNTS

Here we present detailed counts for various patterns unto 5 vertices. First, [tab:prop-full]Tab. ??tab:prop-full presents all connected 4-patterns and some of the 3-vertex patterns relevant to our algorithms. We also report the induced counts for all connected 5-vertex patterns in [tab:5vertex1]Tab. ??tab:5vertex1 and [tab:5vertex2]Tab. ??tab:5vertex2.

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 1 & 1 & 1 & 2 & 3 & 5 \\ 1 & 1 & 2 & 1 & 2 & 2 & 4 & 4 & 5 & 4 & 6 & 12 & 9 & 10 \\ 1 & 1 & 2 & 1 & 2 & 5 & 4 & 4 & 2 & 7 & 6 & 6 & 6 & 10 \\ 1 & 1 & 2 & 2 & 2 & 2 & 2 & 2 & 2 & 6 & 3 & 3 & 4 & 8 \\ 1 & 1 & 4 & 2 & 2 & 2 & 3 & 6 & 6 & 6 & 16 & 12 & 12 & 30 \\ 1 & 1 & 2 & 2 & 1 & 1 & 6 & 6 & 5 & 4 & 12 & 14 & 30 & 60 \\ 1 & 1 & 1 & 1 & 2 & 6 & 6 & 3 & 4 & 8 & 16 & 12 & 30 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 4 & 2 & 6 & 12 & 12 & 15 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 6 & 15 & 15 & 24 \\ 1 & 1 & 1 & 1 & 1 & 1 & 3 & 2 & 2 & 8 & 8 & 24 & 60 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 6 & 3 & 2 & 4 & 10 & 24 & 60 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 2 & 4 & 12 & 6 & 24 & 60 & 60 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 1 & 4 & 10 & 10 & 10 & 10 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 3 & 1 & 3 & 10 & 10 & 10 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 2 & 6 & 20 & 20 & 20 & 20 & 20 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 4 & 18 & 60 & 60 & 60 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 4 & 1 & 9 & 30 & 30 & 30 & 30 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 3 & 15 & 15 & 15 & 15 & 15 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 6 & 30 & 30 & 30 & 30 & 30 & 30 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \end{pmatrix}$$

Figure 12: Matrix transforming induced 5-vertex pattern counts to non-induced counts

$$A^{-1} = \begin{pmatrix} 1 & -1 & 1 & -2 & -1 & -2 & -2 & 4 & 4 & 5 & 4 & 6 & -12 & -9 & -10 & -10 & 20 & 20 & -36 & 60 \\ 1 & 1 & -2 & -2 & -1 & -2 & -5 & 4 & 4 & 2 & 7 & 6 & -6 & -6 & -10 & -14 & 24 & 18 & -36 & 60 \\ 1 & 1 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & -2 & 6 & 3 & 3 & 3 & -4 & -8 & 15 & -30 \\ 1 & 1 & -4 & -4 & -2 & -2 & -2 & -4 & -2 & -2 & -2 & -2 & 3 & 6 & 6 & 6 & -16 & -12 & 30 & -60 \\ 1 & 1 & -2 & -2 & -2 & -2 & -1 & -2 & -2 & -2 & -2 & -1 & 6 & 6 & 5 & 4 & -12 & -14 & 30 & -60 \\ 1 & 1 & -1 & -1 & -1 & -2 & -6 & -1 & -1 & -2 & -2 & -1 & 6 & 3 & 4 & 8 & -16 & -12 & 30 & -60 \\ 1 & 1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & 1 & 2 & 2 & 2 & -4 & -2 & 6 & -12 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & -1 & -1 & -1 & 2 & 2 & -6 & 15 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -3 & -2 & -2 & -2 & 8 & 8 & -24 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -6 & -3 & -2 & -2 & 4 & 10 & -24 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -2 & -4 & -4 & -4 & 12 & 6 & -24 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & 2 & 2 & 1 & -4 & 10 & -10 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -1 & 3 & -1 & -2 & 6 & -20 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -4 & -4 & 18 & -60 & 60 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -4 & -1 & 9 & -30 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & -3 & 15 & 15 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 6 & 30 & 30 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 10 & -10 & 1 \end{pmatrix}$$

Figure 13: Matrix transforming non-induced 5-vertex pattern counts to induced counts

Table 2: 4 and 5-vertex pattern (induced) counts for various graphs

	V	E	$\frac{W_{++}}{W}$	$\frac{W_{+-}}{W}$	T	3-star	3-path	tailed triangle	4-cycle	Chordal 4-cycle	4-clique
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
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
flickr	2.4e+5	3.6e+6	0.013	0.029	1.6e+7	1.1e+13	4.9e+11	1.4e+11	2.4e+09	4.7e+09	1.2e+08
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
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
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
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
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
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
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
web-baidu-baik	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
tech-ip	2.3e+6	2.2e+7	0.001	0.000	3.0e+5	1.3e+17	1.4e+13	8.6e+9	8e+11	3e+7	2
orkut	3.1e+6	2.3e+8	0.089	0.188	6.3e+8	9.8e+13	1.9e+13	1.5e+12	7.0e+10	4.8e+10	3.2e+9
LiveJournal	4.8e+6	8.6e+7	0.094	0.190	2.9e+8	6.6e+12	1.1e+12	1.2e+11	5.0e+9	1.7e+10	9.9e+9

Table 3: Induced counts for 5 vertex patterns; Patterns 1–12

	P1	P2	P3	P4	P5	P6	P7	P8	P9	P10	P11
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
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
flickr	4.6e+16	2.2e+15	1.3e+14	6.7e+14	8.9e+12	2.1e+14	2.0e+13	2.0e+11	7.4e+11	4.4e+12	5.8e+13
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
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
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
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
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
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
web-baidu-baik	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
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
orkut	4.5e+17	8.6e+16	1.0e+16	7.1e+15	6.0e+14	2.5e+15	6.3e+14	1.3e+13	1.8e+13	8.2e+13	3.9e+14
LiveJournal	1.9e+16	2.2e+15	3.1e+14	1.7e+14	2.2e+13	5.4e+13	1.5e+13	3.7e+11	1.2e+12	3.8e+12	1.0e+13

Table 4: 5-vertex patterns; Patterns 13–21 and runtimes in seconds

	P12	P13	P14	P15	P16	P17	P18	P19	P20	P21
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
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
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+08
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
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
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
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
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
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
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
web-baidu-baik	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
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
orkut	1.5e+13	1.1e+13	9.7e+12	1.2e+13	7.1e+12	1.4e+12	2.4e+11	1.3e+12	1.4e+11	1.6e+10
LiveJournal	5.0e+11	3.0e+11	7.1e+11	2.3e+12	7.1e+11	1.6e+11	1.3e+11	1.0e+12	6.3e+11	4.7e+11