Efficiently Counting Vertex Orbits of All 5-vertex Subgraphs, by EVOKE

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

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

ABSTRACT

Subgraph counting is a fundamental task in network analysis. Typically, algorithmic work is on total counting, where we wish to count the total frequency of a (small) pattern subgraph in a large input data set. But many applications require *local counts* (also called vertex orbit counts) wherein, for every vertex v of the input graph, one needs the count of the pattern subgraph involving v. This provides a rich set of vertex features that can be used in machine learning tasks, especially classification and clustering. But getting local counts is extremely challenging. Even the easier problem of getting total counts has received much research attention. Local counts require algorithms that get much finer grained information, and the sheer output size makes it difficult to design scalable algorithms.

We present EVOKE, a scalable algorithm that can determine vertex orbits counts for all 5-vertex pattern subgraphs. In other words, EVOKE exactly determines, for every vertex v of the input graph and every 5-vertex subgraph H, the number of copies of H that v participates in. EVOKE can process graphs with tens of millions of edges, within an hour on a commodity machine. EVOKE is typically hundreds of times faster than previous state of the art algorithms, and gets results on datasets beyond the reach of previous methods.

Theoretically, we generalize a recent "graph cutting" framework to get vertex orbit counts. This framework generate a collection of polynomial equations relating vertex orbit counts of larger subgraphs to those of smaller subgraphs. EVOKE carefully exploits the structure among these equations to rapidly count. We prove and empirically validate that EVOKE only has a small constant factor overhead over the best (total) 5-vertex subgraph counter.

CCS CONCEPTS

• **Theory of computation → Graph algorithms analysis**; *Social networks*; • **Information systems** → *Data mining*.

*Both authors are supported by NSF Awards CCF-1740850, CCF-1813165, and ARO Award W911NF1910294.

KEYWORDS

motif analysis; subgraph counting; orbit counting; pattern cutting; graph orientations

ACM Reference Format:
Noujan Pashanasangi and C. Seshadhri. 2020. Efficiently Counting Vertex Orbits of All 5-vertex Subgraphs, by EVOKE. In *The Thirteenth ACM International Conference on Web Search and Data Mining (WSDM '20), February 3–7, 2020, Houston, TX, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3336191.3371773

1 INTRODUCTION

One of the most important algorithmic techniques in network analysis is *subgraph counting*, also referred to as motif counting or graphlet analysis. Subgraph counting is basically the problem of counting the frequency of small pattern subgraphs in a large input graph. These techniques have found applications in bioinformatics and biological networks [21, 31, 32], social networks [17, 26, 39, 40, 44], community and dense subgraph detection [8, 36, 42, 43], social sciences [11, 13, 20, 30], and many other applications [5, 7, 16, 18, 46]. (Refer to the tutorial [37] for more details on applications.)

Let G denote the input graph, that we wish to analyze. While the typical description of subgraph counting asks for the total count of a pattern subgraph in G, many applications require *local* counts. (These are also referred to as graphlet distributions, orbit counts, or k-profiles.) For a given set of patterns, the aim is to find, *for every vertex v of G*, the number of patterns that v participates in. This is a much finer grained description of the graph, and can be used to generate features for vertices. A compelling application of these local counts are the *graphlet kernel*, where local counts are used to construct vector representations of vertices for machine learning [38]. In many applications (documented in §1.3), one typically wants local counts for all pattern subgraphs of up to a given size.

Subgraph counting is an extremely challenging problem. As shown in previous work, even for a moderate graph with a few million edges, counts of (say) 5-vertex pattern subgraphs can be in the order of billions to trillions [6, 22, 29]. This combinatorial explosion is often tamed by clever counting methods that avoid enumeration, but these are tailored to global counts in G. There has been recent work on randomized methods for local counting, but these require large parallel hardware even for graphs with tens of millions of edges [15].

1.1 Problem Description

The input $G = (V, E)$ is a simple, undirected graph. Our aim is to get local counts, for every vertex in G, for all the patterns given in Fig. 2. Fig. 2 shows all connected subgraphs with at most 5 vertices. We will refer to these as *patterns*. (We do not focus on disconnected patterns; results in [29] imply that these can be easily determined from connected subgraph counts.) Within each pattern, vertices are present in different "roles" or *orbits*. In some patterns like the 5-cycle (H_{15}) and 5-clique (H_{29}), there is just one orbit. In contrast, H_{10} has four different orbits, indicated by the different colors. Thus, a vertex of G can participate in a copy of H_{10} in four different ways, and we wish to determine all of these four different counts. We delay the exact formalism of orbits to §2. But hopefully, Fig. 2 gives a clear pictorial representation of the 73 different orbits, numbered individually.

Our aim is to design an algorithm that: for every vertex v in V and every orbit θ, exactly outputs the number of times that v occurs in a copy of θ. Thus, the output is a set of $73|V|$ counts. (Technically, we ask for induced counts, but can also get non-induced counts. Details in §2.) For example, the count of orbit 17 is the number of times that v is the middle of a 4-path, while the count of orbit 15 is the number of 4-paths that start/end at v. Analogously, the count of orbit 34 is the number of 5-cycles that v participates in. For a fixed orbit, we refer to these numbers as the *vertex orbit counts* (VOCs). Collectively (over all orbits), we wish to determine *VOCs for all* 5*-vertex subgraphs*. For convenience, we refer to this as simply 5-VOCs. We refer to the total subgraph count as "global" counts, which is clearly a much easier problem.

As can be seen, the desired output is an immensely rich local description of the vertices of G. This output subsumes a number of recent subgraph counting problems in the data mining community [6, 14, 15, 29].

Main challenges: To the best of our knowledge, there is no algorithm that (even approximately) computes all 5-VOCs even for graphs with tens of millions of edges. Results on global counting are much faster, but it is not clear how to implement these ideas for VOCs [6, 29]. The ORCA package is the only algorithm that actually computes all 5-VOCs, but it does not terminate after days for graphs with tens of millions of edges. We give more details of previous work in §1.3.

From a mathematical standpoint, the challenge is to get all 5-VOCs without an expensive enumeration. The total number of orbit counts is easily in the order of trillions, and a fast algorithm should ideally avoid touching each 5-vertex subgraph in G. On the other hand, VOCs are an extremely fine-grained statistic, so purely global methods do not work.

1.2 Main Contributions

Our primary result is the Efficient Vertex Orbit pacKagE (EVOKE), an algorithm to compute all 5-VOCs.

Practical local counting: EVOKE advances the state of the art of subgraph counting. It is the first algorithm that can feasibly obtain all 5-VOCs on graphs with tens of millions of edges. We do comprehensive tests on many public data sets. We observe that EVOKE gets counts on graphs with millions of edges in just minutes, and on graphs with tens of millions of edges within an hour. This is on a single commodity machine with 64GB memory, without any parallelization. In contrast, for the larger instances, the previous state of the art ORCA package takes more than two days or runs out of memory, on a more powerful machine (384GB RAM). Even on instances where ORCA terminates, EVOKE is about a hundred times faster. We show the speedup of EVOKE over ORCA in Fig. 1. EVOKE is also able to get 5-VOCs in a social network with 100M edges, in less than two days. (ORCA runs of out memory in such instances.) All the blue bars in Fig. 1 denote instances where ORCA runs of out of memory (in two days) or is a thousand times slower than EVOKE.

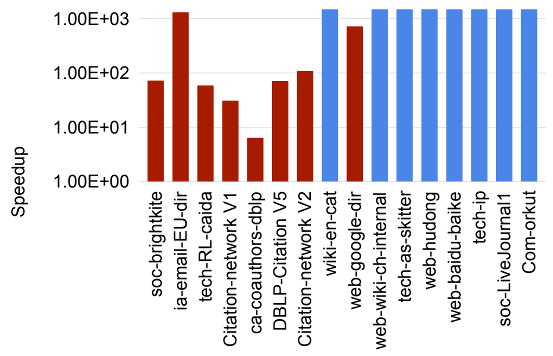


Figure 1: Runtime speedup for computing all 5-VOCs achieved by EVOKE over ORCA (computed as runtime of ORCA/runtime of EVOKE). Graphs are sorted by increasing number of edges from left to right. For the blue bars, ORCA ran out of memory or did not terminate after 1000 times the EVOKE running time. EVOKE is significantly faster than ORCA, and makes 5-VOC counting feasible for large graphs.

EVOKE has a large number of independent sub-algorithms. It is straightforward to run them in parallel, and we get about a factor two speedup. We do not consider this a significant novelty of EVOKE, but it does allow for an even faster running time.

Local counting without enumeration: Our work builds on the ESCAPE framework of Pinar-Seshadhri-Vishal [29]. One of their main insights is a combination of graph orientations and a "pattern cutting" technique. Larger patterns are carefully cut into smaller patterns. It is then shown that local counts of *smaller* patterns can be combined into *global* (total) counts of larger patterns. We formally prove that, for the orbits in Fig. 2, one can generalize their method to VOCs. This is mathematically quite technical and requires manipulations of various pattern automorphisms (which is not required for total counts). But the final result is a large collection of polynomial formulas to compute individual VOCs through some specialized local counts of smaller subgraphs. EVOKE exploits the structure among these formulas to count all VOCs efficiently.

Somewhat surprisingly, we mathematically prove that the running time is only a constant factor more than that of ESCAPE (which only computes total counts). This is borne out empirically where the running time of EVOKE is typically twice that of ESCAPE. Our result demonstrates the power of the cutting framework introduced in [29].

Fast computation of orbit frequency distributions: The distribution of VOCs is a useful tool in graph analysis, often called *graphlet degree distribution* in bioinformatics [31]. EVOKE makes it feasible to compute these distributions over real data. As a small demonstration of EVOKE, we observe interesting behavior in VOCs

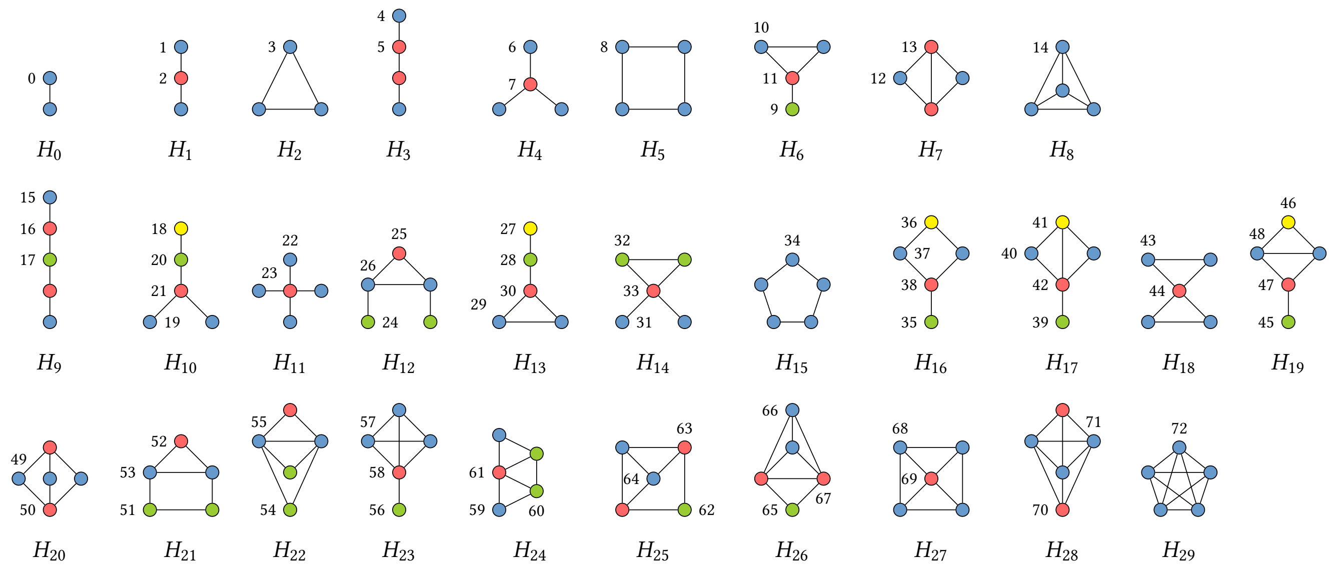


Figure 2: All vertex orbits for 5-vertex patterns. Within any pattern, vertices of the same color form an orbit.

across graphs from different domains. Also, the VOCs of different orbits within the same pattern behave differently, showing the importance of getting such fine-grained information.

On 4-VOCs: We do not consider this as a new contribution, but a salient observation for those interested in subgraph counting. EVOKE determines all 4-VOCs as a preprocessing step, based on ideas in [29] and Ortmann-Brandes [27]. As stated in these results, the key insight is an implementation of an old algorithm of Chiba-Nishizeki for 4-cycle counting [12]. This method is incredibly fast, and computes 4-VOCs in minutes. (Even for the largest instance of more than 100M edges, it took less than an hour.) For example, for a LiveJournal social network with 42M edges, EVOKE took ten minutes on a commodity machine (we got the same time even on a laptop). Contrast this with previous results for counting 4-VOCs for the same graph, which used a MapReduce cluster [15]. (We note that EVOKE, and the other results, are technically computing edge orbit counts, a more general problem.)

1.3 Related Work

Subgraph counting is an immensely rich area of study, and we refer the reader to a tutorial for more details [37]. Here, we only document results relevant to our problem. For this reason, we do not discuss the extremely large body of work on triangle counting (the most basic subgraph counting problem).

Vertex orbit counts beyond triangles have found significant uses in network analysis and machine learning. Notably, Shervashidze-Vishwanathan-Petri-Mehlhorn-Borgwardt defined the *graphlet kernel*, that uses vertex orbits counts to get embeddings of vertices in a network [38]. Ugander-Backstrom-Kleinberg showed that 4-vertex orbit counts can be used for role discovery and distinguishing different types of graph neighborhoods [44]. In an exciting recent use of orbit counts, Rotabi-Kamath-Kleinberg-Sharma showed that four and five cycle counts can be used for weak tie discovery in the Twitter network [35]. Yin-Benson-Leskovec have defined higher-order clustering coefficients, which are ratios of specific orbit counts [48, 49]. There is a line of work on the surprising benefits of using cycle and clique counts as vertex or edge weights, to find denser and more relevant communities in networks [8, 9, 36, 42, 43].

We now discuss the literature on algorithms for subgraph counting. Ahmed-Neville-Rossi-Duffield gave the first algorithm that could count (total) 4-vertex subgraph counts for graphs with millions of edges [6]. Their PGD package was a significant improvement over past practical work for this problem [18]. Pinar-Seshadhri-Vishal designed the ESCAPE algorithm for practical (total) 5-vertex subgraph counting [29]. While these algorithms employed many clever combinatorial ideas, they did not focus on vertex orbit counting. There was concurrent development of sampling algorithms that are orders of magnitude faster, such as path-sampling [22] and the MOSS package [45].

Elenberg-Shanmugam-Borokhovich-Dimakis gave algorithms for 3, 4-vertex orbit counting [14, 15]. They employed a randomized algorithm, and proved convergence through polynomial concentration inequalities. The number of samples required for concentration was large, and they used Map-Reduce clusters to process graphs with tens of millions of edges. It was observed implicitly in the ESCAPE package and explicitly, by Ortmann-Brandes [27] that ideas from a classic result of Chiba-Nishizeki [12] gave a faster, exact algorithm for 4-vertex orbits.

The state of the art for local counting of 5-vertex orbits is the ORCA package of Hočevar-Demšar [19]. The algorithm is based on a method to build sets of linear equations relating various orbit counts. This saves computing all orbit counts independently. With some careful choices, ORCA tries to perform enumeration on the "easier" counts, and get the "harder" counts through the linear equations. There were also results on generating these linear equations auotmatically [24, 25]. We note that ORCA also has algorithms to generate 5-edge orbit counts, but this takes even longer than

5-VOCs. We leave the generalization of EVOKE to edge orbit counts as future work.

Rossi-Ahmed-Carranza-Arbour-Rao-Kim-Koh proposed a parallel algorithm for counting typed graphlets (subgraph patterns), which are a generalization of subgraph patterns to heterogeneous networks [34].

2 PRELIMINARIES

The input is an undirected simple graph $G = (V, E)$, with n vertices and m edges. The patterns of interest are all connected subgraphs with at most 5 vertices, denoted $H_0, \ldots, H_{29}$, as shown in Fig. 2. Previous results in [29] show that disconnected pattern counts can be determined by inclusion-exclusion from all connected pattern counts. Hence, we only focus on connected pattern subgraphs.

We now formally define orbits. The definitions below are taken from Bondy and Murty (Chapter 1, Section 2) [10].

Defn. 1. *Fix labeled graph $H = (V(H), E(H))$. An* automorphism *is a bijection $\sigma : V(H) \to V(H)$ such that (u, v) is in $E(H)$ iff $(\sigma(u), \sigma(v))$ is in $E(H)$.*

Define an equivalence relation among $V(H)$ as follows. We say that $u \sim v$ ($u, v \in V(H)$) iff there exists an automorphism that maps u to v. The equivalence classes of the relation are called orbits.

Fig. 2 shows the 73 different orbits. Within any H_i, all vertices in an orbit are colored the same. For example, in H_{28}, there are two different orbits (blue and red). The blue (resp. red) vertices can be mapped to each other by automorphisms, and are therefore "equivalent".

Technically, we denote orbits as pairs (H, S), where H is a (labeled) pattern subgraph and S is the subset of vertices forming the orbit. Consider pattern H and orbit $\theta = (H, S)$. We denote:

- orb(H): The set of orbits in the pattern H.
- sz(θ): $|S|$, the number of vertices in the orbit θ.

Induced vs non-induced: A non-induced subgraph is obtained by taking a subset of edges. An induced subgraph is obtained by taking a subset of vertices and considering all edges and non-edges among them. (A clique contains all non-induced subgraphs of smaller sizes, but the only induced subgraphs it contains are smaller cliques.) A theorem in [29] proves that the vector of non-induced subgraph (up to a given size) counts can be converted to the corresponding induced counts, through a linear transformation. A directed generalization of the arguments holds for k-VOCs, in that non-induced orbit counts (for each vertex) can be converted to induced orbit counts by a linear transformation. For space reasons, we omit details and give the proof in the full version [28]. It is a small linear transformation of the 73-dimensional orbit count for each vertex, and is efficient to do on all vertices.

EVOKE computes both non-induced and induced counts. Algorithmically, it is easier to compute non-induced counts first; hence we shall only refer to them in the technical description.

We are ready to define VOCs.

Defn. 2. *Fix an orbit $\theta = (H, S)$ and a vertex $v \in V$ (in the input graph G). A* match *of θ involving v is a non-induced copy of H in G such that v is mapped to a vertex in S. Call two matches* equivalent, *if one can be obtained from the other by applying an automorphism. We define $DM(v, \theta)$ to be the number of distinct matches of θ involving v.*

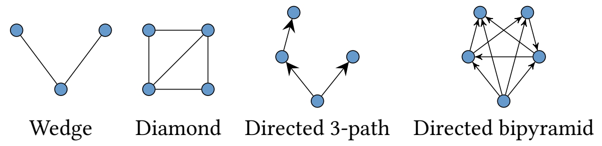


Figure 3: Fundamental patterns enumerated for orbit counting

Our aim is to compute the entire list of numbers $\{UM(v, \theta)\}$, over all $v \in V$ and all θ in Fig. 2.

Degree ordering: We will use the *degree orientation*, a fundamental tool for subgraph counting that was pioneered by Chiba-Nishizeki [12]. We will convert G into an DAG $G^{\rightarrow}$ as follows. Let $\prec$ denote the *degree ordering* of G. For vertices i, j, we say $i \prec j$, if either $d(i) < d(j)$ or $d(i) = d(j)$ and $i < j$ (ties broken by vertex id). The DAG $G^{\rightarrow}$ is obtained by orienting the edges with respect to $\prec$ ordering. In both the algorithm and analysis, all references to directed structures are with respect to $G^{\rightarrow}$.

Notation for subgraph counts: In formulas for orbit counts, we will use the following notation. We use $d(v)$ for the degree of vertex v. We will use $W(G)$, $D(G)$, $DP(G^{\rightarrow})$, and $DBP(G^{\rightarrow})$ for the total count of wedges, diamonds, directed 3-paths, and directed bipyramids respectively. These subgraphs are shown in Fig. 3.

2.1 Main theorem

Theorem 3. *There is an algorithm for exactly counting all VOCs for orbits 0-72, whose running time is $O(W(G) + D(G) + DP(G^{\rightarrow}) + DBP(G^{\rightarrow}) + m + n)$.*

This theorem is analogous to that of ESCAPE ([29]) which gives the same asymptotic running time for just total counting of 5-vertex subgraphs. We consider it quite significant that one gets the same asymptotic running time, despite the output being much larger and far more fine-grained. We stress that the EVOKE algorithm is significantly different than ESCAPE, since the orbit counts behave differently from total subgraph counts. The final proof is long and technical, so we will only provide the high-level ideas here and leave the final proof to the full version [28]. [1]

3 MAIN IDEAS

EVOKE builds off the ideas in ESCAPE for total subgraph counts. First, we explain difficulties in directly applying previous techniques.

Pattern cutting: Intuitively, a 5-vertex pattern can be "cut" into smaller patterns that can be explicitly enumerated. An enumeration over these smaller patterns can then be used to get a subgraph count. As an example, consider the 4-path (H_9). By cutting at the center (green) vertex, one gets two wedges. Thus, we can basically square the number of wedges that end at a vertex, and then sum this to get the total number of 4-paths. (Not quite, there is some inclusion-exclusion required to "correct" this count, but it is fairly easy to

[1]The full version is available at https://arxiv.org/abs/1911.10616

work out.) But this fails for orbit counting. The 4-path has three distinct orbits, and the idea above only works for the green orbit.

This is even more problematic for patterns like H_{21}, H_{25}, H_{27}, H_{28}, where the removal of certain vertices does not "cut" the pattern into convenient smaller pieces. The main insight in ESCAPE was that all 5-vertex patterns have a convenient cutset of vertices, whose removal leads to fragments that can be easily enumerated. This is not true for orbits. We do have the freedom of choosing the convenient cutset.

From 4-edge orbit counts to 5-VOCs: Our main insight is that the suitable generalization of the pattern cutting approach connects 5-VOCs to 4-*edge* orbit counts. We essentially prove that nearly all the orbit counts in Fig. 2 for a vertex v can be related (by non-trivial polynomial equations) to the *edge* orbits counts (of 4-vertex subgraphs) on edges incident to v. The edge orbits of 4-vertex subgraphs are given in Fig. 5b. These edge orbits counts can be obtained by implementations of the Chiba-Nishizeki clique and 4-cycle counter [12], with extra inclusion-exclusion tricks to get all counts. EVOKE uses this as a preprocessing step.

Careful indexing during enumeration: Even with the previous ideas, we still need an efficient implementation that can generate all the counts. We design a collection of vertex and edge indexed data structures, that are updated by an enumeration of the patterns shown in Fig. 3. Somewhat surprisingly, we show that as these patterns are enumerated, one can quickly update these data structures and generate all the orbit counts. This leads to Theorem 3. The final proof is quite technical and has many parts (due to the large number of orbits).

4 THE CUTTING FRAMEWORK FOR ORBITS

In this section, we describe the cutting framework for orbits. As mentioned earlier, this is a generalization of ideas in [29].

First, we formally define a match, which is a non-induced copy of H. For a set C where $C \subseteq V(H)$, we use $H|_C$ to denote the subgraph of H induced on C. We also denote the remaining graph after removing C from H, by $H \setminus C$.

Defn. 4. *A* match *of H in G is a bijection $\pi : T \to V(H)$ where $T \subset V$ and for any two vertices t_1 and t_2 in T, $(t_1, t_2) \in E$ if $(\pi(t_1), \pi(t_2)) \in E(H)$.*

Defn. 5. *Fix an orbit $\theta = (H, S)$ and a vertex $v \in V$. We define $\mathcal{M}(v, \theta)$ to be the set of all (not necessarily distinct) matches $\pi : T \to V(H)$ of H, where $T \subset V$, such that $v \in T$ and $\pi(v) \in S$. We use $M(v, \theta)$ to denote $|\mathcal{M}(v, \theta)|$.*

Defn. 6. *For any orbit $\theta = (H, S)$ we define $\lambda = (H, i)$, where i is a vertex in S, as a representative of θ.*
We use $r(\theta)$ to denote its representative (H, j), where j is the vertex with the smallest id in S.

Let $\lambda = (H, i)$ be a representative of an orbit θ. Abusing notation, for a vertex $v \in V$, we use $\mathcal{M}(v, \lambda)$ to denote the set of matches $\pi \in \mathcal{M}(v, \theta)$ where $\pi(v) = i$. Analogously, we use $\mathrm{M}(v, \lambda)$ to show $|\mathcal{M}(v, \lambda)|$. We can see that $\mathrm{M}(v, \theta) = \mathrm{sz}(\theta) \cdot \mathrm{M}(v, \lambda)$. Next, we define *fragments* in H, which are the result of cutting H using a cut set.

Defn. 7. *Let H be a subgraph pattern and consider a non-trivial cut set $C \subsetneq V(H)$. Let $S_1, S_2, \ldots$ be connected components of $H \setminus C$. The* fragments *of H obtained by removing C are the subgraphs of H induced by $C \cup S_1, C \cup S_2, \ldots$. We denote the set of these fragments by $Frag_C(H)$.*

A partial match $\pi : T \to V(H)$ is similar to a match, except that it is an injection, and is not surjective, thus $|T| < |V(H)|$.

Defn. 8. *A match $\pi : T \to V(H)$* extends *a partial match $\sigma : T' \to V(H)$ if $T' \subset T$ and for any vertex t in T, $\pi(t) = \sigma(t)$. We denote the number of matches π of H that extend σ, by $\deg_H(\sigma)$.*

Consider a match σ of $H|_C$. For σ to extend to a match of H, it is sufficient that it extends to disjoint matches of all fragments in $\mathrm{Frag}_C(H)$. Merging these extensions leads to a match of H. If extension of σ to these fragments are not disjoint, merging them leads to a match of a different pattern H', which we call a *shrinkage.*

Defn. 9. *Let H, H' be subgraph patterns, $C \subsetneq V(H)$ be a cut set of H, and $Frag_C(H) = \{F_1, F_2, \ldots, F_{|Frag_C(H)|}\}$. Let $\tau : H|_C \to H'$ be a partial match of H'. For each $F_i \in Frag_C(H)$, let $\pi_i : F_i \to H'$ be a partial match of H' in H that extends τ. We call $\{\tau, \pi_1, \pi_2, \ldots, \pi_{|Frag_C(H)|}\}$ a C-shrinkage of H into H' if for each edge $(s, t) \in E(H')$, there exists an edge (a, b) in fragment $F_j \in Frag_C(H)$ such that $\pi_j(a) = s$ and $\pi_j(b) = t$.*
We use $Shrink_C(H)$ to denote the set of patterns (up to isomorphism) H', to which there exist at least a C-shrinkage from H.

Defn. 10. *Consider graph $H, H' \in Shrink_C(H)$, $\lambda = (H, i)$, and $\lambda' = (H', j)$. We define $numSh_C(\lambda, \lambda')$ to be the number of distinct C-shrinkages of H into H' where $\tau(i) = j$.*

Lemma 11. *Consider a pettern H, an orbit $\theta = (H, S)$, a representative $\lambda = (H, i)$ of θ, and a cut set C in H such that $i \in C$. Then,*

$$M(v,\lambda) = \sum_{\sigma \in \mathcal{M}(v,(H|_C,i))} \prod_{F \in Frag_C(H)} \deg_F(\sigma) - \sum_{H' \in Shrink_C(H)} \sum_{\substack{\theta' \in orb(H'),\\ \lambda' = r(\theta')}} numSh_C(\lambda,\lambda') \cdot DM(v,\lambda')$$

Proof. Consider any match σ of $H|_C$ in $\mathcal{M}(v, (H|_C, i))$, and all sets of maps $\{\pi_1, \ldots, \pi_{|\mathrm{Frag}_C(H)|}\}$ where π_ℓ is a copy of $F_\ell \in \mathrm{Frag}_C(H)$ that extends σ. The number of such sets is exactly:

$$\sum_{\sigma \in \mathcal{M}(v,(H|_C,i))} \prod_{F \in \mathrm{Frag}_C(H)} \deg_F(\sigma) \tag{1}$$

Consider one of these sets of maps $\{\pi_1, \ldots, \pi_{|\mathrm{Frag}_C(H)|}\}$, let $V(\pi_\ell)$ be the set of vertices that π_ℓ maps to F_ℓ. If all $V(\pi_\ell) \setminus V(C)$ are disjoint, we get a match in $\mathcal{M}(v, \lambda)$. Therefore, Each match of H in $\mathcal{M}(v, \lambda)$ is counted exactly one time in (1). But for each orbit $\theta' = (H', S')$ where $H' \in \mathrm{Shrink}_C(H)$, we have also counted some matches in $\mathcal{M}(v, \theta')$. The number of distinct matches of θ' involving v is $\mathrm{DM}(v, \theta')$. Let $\lambda' = (H', j)$ be $r(\theta')$. The number of distinct C-shrinkages of H into H', where $\tau(i) = j$, is $\mathrm{numSh}_C(\lambda, \lambda')$. Thus, per each orbit θ', we have counted $\mathrm{numSh}_C(\lambda, \lambda') \cdot \mathrm{DM}(v, \lambda')$ matches which should now be subtracted from (1).

The reason we considered only distinct matches of λ' involving v is that the shrinkage from H to H' gives us the labeling of H' and the set of maps $\{\pi_1, \ldots, \pi_{|\mathrm{Frag}_C(H)|}\}$, which resulted in counting this match, dictates the match. Also, notice that the shrinkage

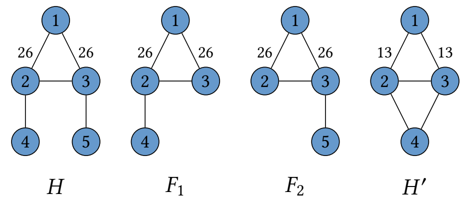


Figure 4: Application of Lemma 11 for vertex orbit 26

determines the vertex in H' that v is mapped to. That is why we consider number of shrinkages for a representative of θ'. □

Corollary 12. *As mentioned, $M(v,\theta) = sz(\theta) \cdot M(v,\lambda)$. Therefore, we can derive $DM(v,\theta)$, which is the number of distinct matches of θ, as follows: $DM(v,\theta) = sz(\theta) \cdot M(v,\lambda)/|Aut(H)|$.*

Application of Lemma 11 for vertex orbit 26: We will show how this lemma works applying it to H_{12} and computing VOCs for a vertex $v \in V$. Let $\theta_{26} = (H,S)$, where $S = \{2,3\}$ and H is as shown in Fig. 4, denote orbit 26. Let the representative λ_{26} be $(H,2)$.

Let triangle $\{1,2,3\}$ be the cut set C. So, $\text{Frag}_C(H) = \{F_1, F_2\}$ as we can see in Fig. 4. Let $\hat{\lambda} = (H|_C, 2)$ be a representative of orbit 3 (the only orbit in the cut set). Every triangle in G incident to v is a match in $\mathcal{M}(v,\hat{\lambda})$. Each such triangle has two mappings to $H|_C$. consider triangle $\{u,v,w\}$ in G. Vertex v has to be matched to vertex 2, therefore one match (A) is $\sigma(u) = 1$, $\sigma(v) = 2$, and $\sigma(w) = 3$, and the other match (B) is $\sigma(u) = 3$, $\sigma(v) = 2$, and $\sigma(w) = 1$. For match (A), $\deg_{F_1}(\sigma) \cdot \deg_{F_2}(\sigma) = (d(v)-2)(d(w)-2)$, and $\deg_{F_1}(\sigma) \cdot \deg_{F_2}(\sigma) = (d(v)-2)(d(u)-2)$ for match (B).

The only possible shrinkage of H is to a diamond H', as shown in Fig. 4. Let orbit $\theta_{13} = (H', S')$, where $S' = \{2,3\}$, show orbit 13. We can see that in any C-shrinkage of H into H', $\tau(2) \in S'$. Let $\lambda_{13} = (H',2)$ be a representative of θ_{13}. Notice that $\text{numSh}_C(\lambda_{26}, \lambda_{13}) = 2$. In one case we set $\tau(1) = 1$, $\tau(2) = 2$, $\tau(3) = 3$, $\pi_1(4) = 4$, and $\pi_2(5) = 4$. In the other case, we set $\tau(1) = 4$, $\tau(2) = 2$, $\tau(3) = 3$, $\pi_1(4) = 1$, and $\pi_2(5) = 1$. The set of maps $\{\tau, \pi_1, \pi_2\}$ in both cases forms a C-shrinkage of H into H' where $\tau(2) = 2$.

$$\begin{aligned} \text{M}(v,\lambda_{26}) = \sum_{t=\langle u,v,w\rangle \text{ triangle}} &[(d(v)-2)((d(u)-2) \\ &+ (d(w)-2))] - 2\cdot \text{DM}(v,\lambda_{13}) \end{aligned} \tag{2}$$

Note that $\text{sz}(\theta_{26}) = 2$ and H has two automorphisms, so (by Corollary 12) $\text{DM}(v,\theta_{26}) = \text{M}(v,\lambda_{26})$.

5 GETTING ORBIT COUNTS

Given space constraints, it is not possible to describe EVOKE completely or give a full proof of Theorem 3. Formally, there is a collection of more than fifty equations similar to (2). For each of them, we verify that they can be computed through an enumeration of the patterns in Fig. 3, assuming that all edge orbits of Fig. 5b are available. We leave these (tedious) details to the full version [28], and give a few examples here.

Getting edge orbit counts of 4-vertex subgraphs: There are eleven edge orbits for 4-vertex subgraphs as shown in Fig. 5b. Formally, one can prove the following. (Actually, one can get a much better running time, but this is not important for our main theorem. Details in full version. [28]) For an edge (u,v), we use $E_i(u,v)$ to denote the count of the ith edge orbit (where i is from Fig. 5b).

Theorem 13. *All vertex and edge orbit counts for 4-vertex patterns can be obtained in time $O(W(G) + D(G) + m + n)$.*

Getting VOCs: We demonstrate the main ideas through a number of examples.

Orbit 26: The pattern cutting framework gives (2). We can precompute and store degrees at all vertices. During an enumeration of all triangles, one can compute the summand for each triangle. The triangles can be enumerated in $O(W(G))$ time (indeed, it can be done even faster using orientations). Orbit 13 belongs to a 4-vertex pattern, so $\text{DM}(2,\lambda_{13})$ is obtained from Theorem 13.

Orbit 37: let $\lambda_{37} = r(\theta_{37})$ and $\lambda_{12} = r(\theta_{12})$, then

$$\text{DM}(u,\lambda_{37}) = \sum_{v\in N(u)} [E_5(u,v)(d(v)-2)] - 2\text{DM}(u,\lambda_{12}). \tag{3}$$

After storing E_5-values on each edge, one can get this VOC by a triangle enumeration. Orbit 12 belongs to a 4-vertex pattern.

Overall, this technique can analogously handle all orbits, barring 5-cycle and 5-clique (each of which as a single orbit). 5-cliques can be directly enumerated in time $O(DBP(G^{\rightarrow}))$, a consequence of the classic Chiba-Nishizeki algorithm [12] and explicitly proven in [29].

Dealing with 5-cycles: This special case is handled in the following theorem, which gives a significant strengthening of the 5-cycle counter in ESCAPE, which only gave a global count in the same running time.

Theorem 14. *Vertex orbit counts for the 5-cycle can be computed in time $O(W(G) + DP(G^{\rightarrow}) + m + n)$.*

Proof. As shown in Fig. 5a, there are three different 5-cycle DAGs up to isomorphism, and each has exactly one directed 3-path, such that the remaining wedge is not an in-in wedge. In Fig. 5a, this directed 3-path is labeled i,j,k,l, and w is the center vertex of the wedge. By a directed wedge enumeration, we can precompute the number of such wedges between all pairs of vertices. We enumerate over the directed 3-paths: for every directed 3-path we get between vertices i and l, we already know the number of relevant directed wedges between i and l. This allows us to increment the orbit counts for vertices i,j,k,l, by the number of wedges. (There is some inclusion-exclusion to fix the count; details in the full version [28].)

This process does not update the orbit count for vertex w. Let $P(i,l)$ be the number of directed 3-paths from i to l. To compute the orbit counts for vertex w, we enumerate in-out and out-out wedges between i and l, and add $P(i,l)$ to the orbit count of vertex w. Again, there is a possible inclusion-exclusion error to be fixed, since the 3-paths (corresponding to $P(i,l)$) potentially intersect with the wedge under consideration. We need to subtract out the counts of specific directed tailed triangles (Details in full version [28].) Overall, we can get VOCs for 5-cycle in the stated time. □

6 EXPERIMENTAL RESULTS

We implement EVOKE in C++. We ran experiments on a commodity machine from AWS EC2: R5d.2xlarge, which has Intel Xeon Platinum 8175M CPU @ 2.50GHz with 4 cores and 1024K L2 cache (per

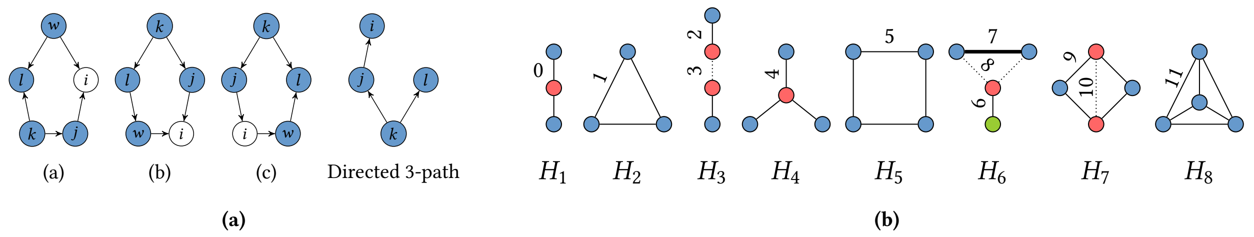


Figure 5: (a) All different 5-cycle DAGs up to isomorphism. (b) All edge orbits of 4-vertex patterns.

core), 34MB L3 cache, and 64GB memory. For running EVOKE on the `com-orkut` graph (117M edges), we used the more powerful R5d.12xlarge EC2 instance (with 384GB RAM). We actually run ORCA for 5-vertex patterns on the larger machine for any instance with more than 1M edges. The EVOKE package is available at [2] as open source code.

We used large graph datasets from the Network Repository [33], SNAP [23], and Citation Network Dataset [1, 41]. We removed directions from edges, and omitted duplicates and self loops. Tab. 1 includes the number of nodes, edges, and triangles for all the graphs we used. We also run EVOKE on `wiki-en-cat`, a bipartite graph from the KONECT network repository [3, 4, 47].

As mentioned earlier, we compare our results with ORCA [19] which is the state of the art algorithm for computing all 5-VOCs. The runtimes of ESCAPE, EVOKE, and ORCA is given in Tab. 1. We also state the time for just counting 4-VOCs. When we do not report a time for ORCA, it implies that either ORCA ran out of memory or ran more than 1000 times the EVOKE running time. In all the results, the time includes the I/O, so we account for the time required to print the (large) output into files. As mentioned later, there is a parallel implementation of EVOKE, but all run times reported are of the sequential implementation (to have a fair comparison with ORCA).

Running time of EVOKE: As seen in Tab. 1, for many instances of counting 5-VOCs, we simply cannot get results with ORCA. For all graphs larger than `web-google-dir`, ORCA-5 runs out of memory even on the more powerful EC2 instance, or was stopped after a thousand times the corresponding EVOKE running time has passed (shown by blue bars in Fig. 1). When ORCA does give results, the speedup of EVOKE is easily in the orders of hundreds. Fig. 1 gives the speedup as a chart. EVOKE makes 5-VOCs computation feasible, for graphs with tens of millions of edges. ORCA is unable to process any graph in that size range. Even for the large `com-orkut` graph with over 100M edges, EVOKE gets all counts in two days.

As an aside, for counting 4-VOCs, EVOKE runs typically in minutes, consistent with previous work [27, 29].

Comparison with ESCAPE: Theorem 3 shows that the asymptotic upper bound given for ESCAPE in [29] is also an asymptotic upper bound for EVOKE run time. We are able to validate this in practice. Fig. 6a shows the ratio of runtime of EVOKE over ESCAPE for 5-vertex patterns. Note that ESCAPE counts subgraphs and EVOKE computes orbit counts for orbits in those subgraphs. As we can see in Fig. 6a, in all our experiments the ratio is typically below 2 and never more than 4. We believe this finding to be significant, since obtaining the richer information of 5-VOCs is just as feasible as getting exact total counts.

Runtime distribution and parallel speedup: Typically, a few orbits take the lion's share of the running time. Fig. 6b shows the split-up of running time over the various orbits. We group them into four classes: the 5-clique, the 5-cycle, the orbits of H_{25} and H_{27} (these require diamond enumerations), and everything else. By and large, just the 5-cycle and 5-clique orbits account for half the time.

It is straightforward to parallelize the computation of these different groups. For the non-induced setting, these are simply independent computations. We perform this parallelism, and present the speedup we achieve in Fig. 6c. As expected, there is roughly a 1.5-2 factor speedup, corresponding to the most expensive orbit to compute.

VOC distributions: As a demonstration of EVOKE, we plot the VOC distribution (also called graphlet degree distribution) of various graphs. To get cleaner figures, we plot the Complementary Cumulative Distribution (CCD): for x, we plot the fraction of vertices whose orbit count is at least x. This is plotted for Orbit 70 (in induced 5-clique minus edge) in Fig. 7a and for Orbit 17 (center of induced 4-path) in Fig. 7b. We stress that these induced counts are typically harder to obtain than the non-induced counts.

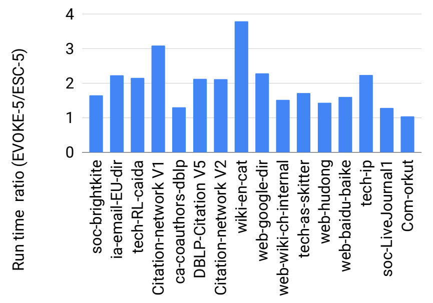
For Orbit 17, we observe that the largest count is more than trillions, showing the challenges in exact counting. Also the distribution of `tech-as-skitter` has a bigger dropoff in the tail, which may be indicative of the path structures in AS networks. The `web-google-dir` graph has a sharp dropoff at the end as well. We see that Orbit 70 distributions are quite different over the graphs, unlike Orbit 17, where the tails are similar for three of the graphs. The counts in `Citation-network V2` are much smaller, suggesting there are not many 5-cliques missing edges.

In Fig. 7c, for the graph `web-google-dir`, we plot the VOC of the three different orbits (15-17) of the induced 4-path. Observe how the distribution for Orbit 15 (the start/end) is significantly different from Orbit 17 (the center), underscoring the fine-grained information that orbits provide over vanilla counts.

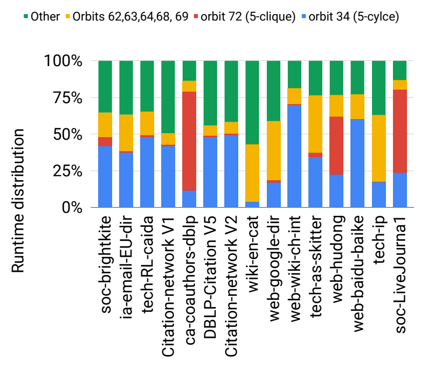
Graph mining through orbit counts: As another demonstration, we focus on the citation network `DBLP-Citation-network V5`, where we have metadata associated with vertices (papers). We found that the paper with the largest count of Orbit 17 (center of induced 4-path) is the classic book "C4.5: Programs for Machine Learning" by Ross Quinlan. On the other hand, the paper participating in the most 5-cliques is the highly cited VLDB 94 paper "Fast Algorithms for Mining Association Rules in Large Databases"

Table 1: Properties of the graphs and runtime of ESCAPE, EVOKE, and ORCA

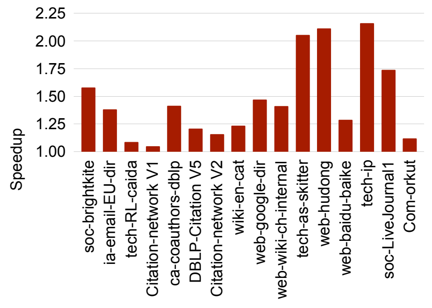
				Runtimes in seconds					
Dataset (sorted by increasing $\|E\|$)	$\|V\|$	$\|E\|$	$\|T\|$	ESC-4	EVOKE-4	ORCA-4	ESC-5	EVOKE-5	ORCA-5
soc-brightkite	56.7K	213K	494K	0.43	0.59	1.77	4.69	7.74	562.84
ia-email-EU-dir	265K	364K	267K	0.49	1.29	9.38	5.91	13.18	17.36K
tech-RL-caida	191K	607K	455K	0.68	1.29	2.99	4.65	10.03	595.44
Citation-network V1	2.17K	631K	248K	0.69	2.57	42.91	2.89	8.93	275.15
ca-coauthors-dblp	540K	1.52M	444M	266.81	287.89	510.77	20.69K	26.91K	171.32K
DBLP-Citation-network V5	470K	2.08M	1.38M	2.59	10.18	13.04	19.17	40.76	2.92K
Citation-network V2	660K	3.02M	1.9M	4.11	11.57	28.42	32.78	69.36	7.52K
wiki-en-cat	2.04M	3.8M	0	3.13	12.61	114.31	22.85	86.58	-
web-google-dir	876K	4.32M	13.4M	4.76	10.03	45.40	45.86	104.88	76.37K
web-wiki-ch-internal	1.93M	8.95M	18.19M	30.11	65.45	655.15	1.22K	1.87K	-
tech-as-skitter	1.69M	11.1M	28.8M	28.91	68.25	827.46	853.21	1.46K	-
web-hudong	1.98M	14.43M	21.61M	48.20	85.83	1.78K	2.41K	3.45K	-
web-baidu-baike	2.14M	17.01M	25.2M	61.4	148.11	2.92K	2.66K	4.27K	-
tech-ip	2.25M	21.64M	2.3M	92.03	277.87	79.96K	18.14K	40.57K	-
soc-LiveJournal1	4.85M	42.85M	285.73M	401.07	599.43	1.30K	28.46K	36.57K	-
com-orkut	3.72M	117.18M	627.58M	1.23K	2.77K	7.37K	137.73K	143.41K	-



(a) Ratio of runtime represented as EVOKE-5/ESC-5 demonstrates Theorem 3

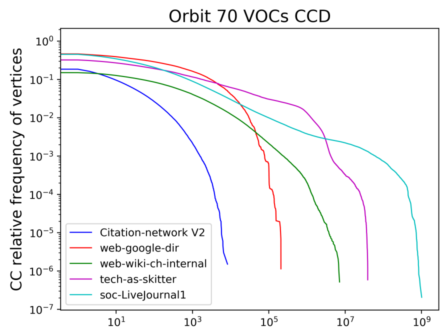


(b) Runtime distribution over 5-vertex orbits

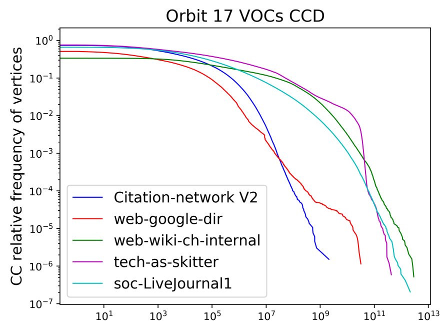


(c) Speedup achieved by parallel computation of 5-VOCs

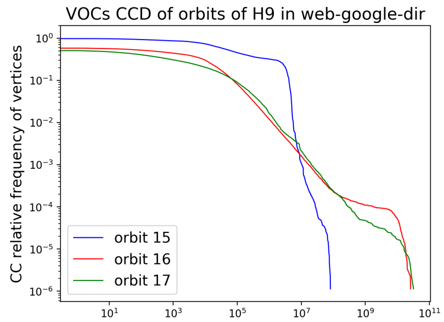
Figure 6: Empirical analysis of EVOKE runtime



(a) Orbit 70 VOCs CCD



(b) Orbit 17 VOCs CCD



(c) VOCs CCD of orb(H_9)

Figure 7: (a), (b): VOCs comp. cum. distribution (CCD) of orbits. For count x, we plot the fraction of vertices with orbit count at least x. (c) For `web-google-dir`, we plot the VOC CCD for all orbits of the 4-path. Observe that the distributions for the start/end (orbit 15) and the center (orbit 17) behave differently.

by Agarwal and Srikant.It is interesting that the orbit counts can immediately give us semantically significant vertices.

ACKNOWLEDGMENTS

We thank Akul Goyal for helpful discussions and his help on verifying the correctness of the output of the EVOKE package.

REFERENCES

[1] Citation network dataset. Available at https://aminer.org/citation.
[2] Evoke. https://bitbucket.org/nojan-p/orbit-counting.
[3] The koblenz network collection. Available at http://konect.uni-koblenz.de/.
[4] Wikipedia (en) network dataset – KONECT, Oct. 2016.
[5] Agrawal, M., Zitnik, M., and Leskovec, J. Large-scale analysis of disease pathways in the human interactome. In *Pacific Symposium on Biocomputing* (2018), vol. 23, World Scientific, p. 111.
[6] Ahmed, N. K., Neville, J., Rossi, R. A., and Duffield, N. Efficient graphlet counting for large networks. In *Proceedings of International Conference on Data Mining (ICDM)* (2015).
[7] Becchetti, L., Boldi, P., Castillo, C., and Gionis, A. Efficient semi-streaming algorithms for local triangle counting in massive graphs. In *Conference on Knowledge Data and Discovery (KDD)* (2008), pp. 16–24.
[8] Benson, A., Gleich, D. F., and Leskovec, J. Higher-order organization of complex networks. *Science 353*, 6295 (2016), 163–166.
[9] Berry, J. W., Hendrickson, B., LaViolette, R. A., and Phillips, C. A. Tolerating the community detection resolution limit with edge weighting. *Phys. Rev. E 83* (May 2011), 056119.
[10] Bondy, J., and Murty, U. Graph theory (2008). *Grad. Texts in Math* (2008).
[11] Burt, R. Structural holes and good ideas. *American Journal of Sociology 110*, 2 (2004), 349–399.
[12] Chiba, N., and Nishizeki, T. Arboricity and subgraph listing algorithms. *SIAM J. Comput. 14* (1985), 210–223.
[13] Coleman, J. Social capital in the creation of human capital. *American Journal of Sociology 94* (1988), S95–S120.
[14] Elenberg, E. R., Shanmugam, K., Borokhovich, M., and Dimakis, A. G. Beyond triangles: A distributed framework for estimating 3-profiles of large graphs. In *Conference on Knowledge Data and Discovery (KDD)* (2015), pp. 229–238.
[15] Elenberg, E. R., Shanmugam, K., Borokhovich, M., and Dimakis, A. G. Distributed estimation of graph 4-profiles. In *The Web Conference (WWW)* (2016), pp. 483–493.
[16] Fagiolo, G. Clustering in complex directed networks. *Phys. Rev. E 76* (Aug 2007), 026107.
[17] Faust, K. A puzzle concerning triads in social networks: Graph constraints and the triad census. *Social Networks 32*, 3 (2010), 221–233.
[18] Gonen, M., and Shavitt, Y. Approximating the number of network motifs. *Internet Mathematics 6*, 3 (2009), 349–372.
[19] Hočevar, T., Demšar, J., et al. Computation of graphlet orbits for nodes and edges in sparse graphs. *Journ. Stat. Soft 71* (2016).
[20] Holland, P., and Leinhardt, S. A method for detecting structure in sociometric data. *American Journal of Sociology 76* (1970), 492–513.
[21] Hormozdiari, F., Berenbrink, P., Prulj, N., and Sahinalp, S. C. Not all scale-free networks are born equal: The role of the seed graph in ppi network evolution. *PLoS Computational Biology 118* (2007).
[22] Jha, M., Seshadhri, C., and Pinar, A. Path sampling: A fast and provable method for estimating 4-vertex subgraph counts. In *The Web Conference (WWW)* (2015).
[23] Leskovec, J., and Krevl, A. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.
[24] Melckenbeeck, I., Audenaert, P., Colle, D., and Pickavet, M. Efficiently counting all orbits of graphlets of any order in a graph using autogenerated equations. *Bioinformatics 34*, 8 (2018), 1372–1380.
[25] Melckenbeeck, I., Audenaert, P., Michoel, T., Colle, D., and Pickavet, M. An algorithm to automatically generate the combinatorial orbit counting equations. *PLoS ONE 11*, 1 (01 2016), 1–19.
[26] O'Callaghan, D., Harrigan, M., Carthy, J., and Cunningham, P. Identifying discriminating network motifs in youtube spam. *arXiv preprint arXiv:1202.5216* (2012).
[27] Ortmann, M., and Brandes, U. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied network science 2*, 1 (2017), 13.
[28] Pashanasangi, N., and Seshadhri, C. Efficiently counting vertex orbits of all 5-vertex subgraphs, by evoke. *arXiv preprint arXiv:1911.10616* (2019).
[29] Pinar, A., Seshadhri, C., and Vishal, V. Escape: Efficiently counting all 5-vertex subgraphs. In *The Web Conference (WWW)* (2017), International World Wide Web Conferences Steering Committee, pp. 1431–1440.
[30] Portes, A. Social capital: Its origins and applications in modern sociology. *Annual Review of Sociology 24*, 1 (1998), 1–24.
[31] Przulj, N. Biological network comparison using graphlet degree distribution. *Bioinformatics 23*, 2 (2007), 177–183.
[32] Przulj, N., Corneil, D. G., and Jurisica, I. Modeling interactome: scale-free or geometric?. *Bioinformatics 20*, 18 (2004), 3508–3515.
[33] Rossi, R. A., and Ahmed, N. K. The network data repository with interactive graph analytics and visualization. In *AAAI* (2015).
[34] Rossi, R. A., Ahmed, N. K., Carranza, A., Arbour, D., Rao, A., Kim, S., and Koh, E. Heterogeneous network motifs. *arXiv preprint arXiv:1901.10026* (2019).
[35] Rotabi, R., Kamath, K., Kleinberg, J. M., and Sharma, A. Detecting strong ties using network motifs. In *The Web Conference (WWW)* (2017), pp. 983–992.
[36] Sariyuce, A. E., Seshadhri, C., Pinar, A., and Catalyurek, U. V. Finding the hierarchy of dense subgraphs using nucleus decompositions. In *The Web Conference (WWW)* (2015), pp. 927–937.
[37] Seshadhri, C., and Tirthapura, S. Scalable subgraph counting: The methods behind the madness: WWW 2019 tutorial. In *Proceedings of the Web Conference (WWW)* (2019).
[38] Shervashidze, N., Vishwanathan, S. V. N., Petri, T., Mehlhorn, K., and Borgwardt, K. M. Efficient graphlet kernels for large graph comparison. In *AISTATS* (2009), pp. 488–495.
[39] Son, S., Kang, A., Kim, H., Kwon, T., Park, J., and Kim, H. Analysis of context dependence in social interaction networks of a massively multiplayer online role-playing game. *PLoS ONE 7*, 4 (04 2012), e33918.
[40] Szell, M., and Thurner, S. Measuring social dynamics in a massive multiplayer online game. *Social Networks 32* (2010), 313–329.
[41] Tang, J., Zhang, J., Yao, L., Li, J., Zhang, L., and Su, Z. Arnetminer: extraction and mining of academic social networks. In *Conference on Knowledge Data and Discovery (KDD)* (2008), ACM, pp. 990–998.
[42] Tsourakakis, C. E. The k-clique densest subgraph problem. In *The Web Conference (WWW)* (2015), pp. 1122–1132.
[43] Tsourakakis, C. E., Pachocki, J., and Mitzenmacher, M. Scalable motif-aware graph clustering. In *The Web Conference (WWW)* (2017), pp. 1451–1460.
[44] Ugander, J., Backstrom, L., and Kleinberg, J. M. Subgraph frequencies: mapping the empirical and extremal geography of large graph collections. In *The Web Conference (WWW)* (2013), pp. 1307–1318.
[45] Wang, P., Zhao, J., Zhang, X., Li, Z., Cheng, J., Lui, J. C. S., Towsley, D., Tao, J., and Guan, X. MOSS-5: A fast method of approximating counts of 5-node graphlets in large graphs. *IEEE Transactions on Knowledge and Data Engineering (TKDE) 30*, 1 (2018), 73–86.
[46] Welles, B., Van Devender, A., and Contractor, N. Is a friend a friend?: Investigating the structure of friendship networks in virtual worlds. In *CHI-EA'10* (2010), pp. 4027–4032.
[47] Wikimedia Foundation. Wikimedia downloads. http://dumps.wikimedia.org/, January 2010.
[48] Yin, H., Benson, A. R., and Leskovec, J. Higher-order clustering in networks. *Phys. Rev. E 97* (2018), 052306.
[49] Yin, H., Benson, A. R., and Leskovec, J. The local closure coefficient: A new perspective on network clustering. In *ACM International Conference on Web Search and Data Mining (WSDM)* (2019), pp. 303–311.