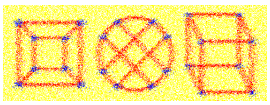

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 30, no. 1, pp. 295–321 (2026)
DOI: 10.7155/jgaa.v30i1.2980

Computing Complexity Measures of Degenerate Graphs

Pål Grønås Drange[1] *Patrick Greaves*[2] *Irene Muzi*[2] *Felix Reidl*[2]

[1]University of Bergen, Norway
[2]Birkbeck, University of London, UK

Submitted: October 2024 Accepted: June 2026 Published: July 2026

Article type: Regular paper Communicated by: A. Anagnostopoulos

Abstract. We show that the VC-dimension of a graph can be computed in time $n^{\lceil \log d+1 \rceil} d^{O(d)}$, where d is the degeneracy of the input graph. The core idea of our algorithm is a data structure to efficiently query the number of vertices that see a specific subset of vertices inside a (small) query set. The construction of this data structure takes time $O(d2^d n)$, after which queries can be computed efficiently using fast Möbius inversion.

This data structure turns out to be useful for a range of tasks, especially for finding bipartite patterns in degenerate graphs, and we outline an efficient algorithm for counting the number of times specific patterns occur in a graph. The largest factor in the running time of this algorithm is $O(n^c)$, where c is a parameter of the pattern we call its *left-covering number*.

Concrete applications of this algorithm include counting the number of (non-induced) bicliques in linear time, the number of co-matchings in quadratic time, as well as a constant-factor approximation of the ladder index in linear time.

Finally, we supplement our theoretical results with several implementations and run experiments on more than 200 real-world datasets—the largest of which has 8 million edges—where we obtain interesting insights into the VC-dimension of real-world networks.

Pål Grønås Drange received support from the Research council of Norway, grant number 329745: *Machine Teaching for Explainable AI*.

A preliminary version of this article was presented at the 18th International Symposium on Parameterized and Exact Computation (IPEC) [9].

E-mail addresses: Pal.Drange@uib.no (Pål Grønås Drange) p.greaves@bbk.ac.uk (Patrick Greaves) i.muzi@bbk.ac.uk (Irene Muzi) f.reidl@bbk.ac.uk (Felix Reidl)

1 Introduction

In this paper, we investigate efficient algorithms for computing complexity measures in sparse graphs, with a particular focus on degenerate graphs frequently encountered in real-world networks. Our primary contribution is an algorithm that computes the Vapnik–Chervonenkis (VC)-dimension of a graph (Definition 1) in time $n^{\lceil \log d+1 \rceil} d^{O(d)}$, where d is the degeneracy of the input graph. Additionally, we show that under natural complexity assumptions, any general improvement over the $n^{\log d}$ running time is unlikely, establishing the near-optimality of our approach for sparse graph classes. The VC-dimension of a graph is the size of the largest *shattered* set of vertices S, i.e., a set such that for every subset $X \subseteq S$, there is a vertex whose neighborhood in S is exactly X. A high VC-dimension indicates that the graph contains a large set of vertices on which neighborhoods realize all possible incidence patterns. Thus, the neighborhood structure is combinatorially complex and lacks the uniformity typically exploited by algorithms for sparse graph classes.

Our result is significant because the VC-dimension, a key parameter from learning theory, has gained prominence as a measure of structural complexity in sparse graphs [12]. Existing general algorithms for computing the VC-dimension are computationally expensive, taking $O(n^{\log n})$ time, and the problem is known to be LOGNP-complete[1] [21]. Our work explores how the structure of degenerate graphs can be exploited to develop more efficient algorithms for this and related tasks.

We investigate whether a more efficient algorithm can be designed under the assumption that the input graph is sparse, specifically d-degenerate. This focus is motivated by the fact that the degeneracy of most real-world networks tends to be small [7], combined with the fact that d-degeneracy can be computed in linear time [20].

Our first achievement is an algorithm that computes the VC-dimension of a d-degenerate graph in time $O(n^{\lceil \log d+1 \rceil} d^{O(d)})$. A core concept is a novel data structure which enables us to efficiently query the size of the intersection of several neighbourhoods for a small set of vertices, described in Section 3, which we use to quickly determine whether a given candidate set is shattered by its neighbours (see Section 2 for the definition of *shattered*).

The general idea behind this algorithm can be extended to other bipartite "patterns", such as bicliques, co-matchings, and ladders (defined in Section 2.2). These structures are also strongly connected to various notions of "graph complexity" and are relevant in studies of graph width measures [11] and algorithm design for sparse graph classes [14]. Our pattern-finding algorithm, presented in Section 3, can count bicliques in linear time, co-matchings in quadratic time, and find partial ladders in linear time. See Section 4 for these and additional results.

At a high level, our algorithms exploit the fact that in a d-degenerate ordering every vertex has at most d neighbours to its left. This allows us to preprocess all subsets of left-neighbourhoods, which remain small when d is small. The resulting data structure supports the following type of query: given a small set S of vertices, and a subset $X \subseteq S$, how many vertices have neighbourhood inside S exactly equal to X? Once such queries can be answered efficiently, many pattern-finding tasks reduce to checking whether the required neighbourhood traces occur on one side of a bipartite pattern. In particular, shattered sets, bicliques, co-matchings, and ladders can all be recognised by prescribing which subsets of a small vertex set must appear as neighbourhoods. Graphs with bounded c-closure are defined by the largest common neighbourhood of two non-adjacent vertices, which can be viewed as searching for a $K_{2,c}$-like configuration with an additional forbidden edge on the side of size 2.

[1] LOGNP refers, informally, to the class of decision problems whose certificates consist of a logarithmic-size set or sequence [21]. Problems such as Log Dominating Set, Log Clique, and Log Path, and generally problems asking for sets or sequences of logarithmic size, are in LOGNP.

Dense structures like cliques and bicliques are well-known for their significance in network analysis, and we propose that co-matchings and ladders could hold similar potential. However, without efficient algorithms to compute these structures, their practical utility remains largely unexplored. Our work aims to provide the computational tools necessary to enable the investigation of these lesser-studied patterns in real-world applications. We therefore implement algorithms to compute the VC-dimension, ladder index, maximum biclique[2] and maximum co-matching of a graph. To establish their practicality, we ran these four algorithms on 206 real-world networks from various sources, see Section 5. The VC-dimension algorithm in our experiments terminated within 10 minutes on networks with up to ∼33k vertices, the other three on networks up to ∼93k vertices. This is already squarely in the region of "practical" for certain types of networks and we believe that with further engineering—in particular to improve space efficiency—our implementation can be used to compute these statistics on much larger networks.

As a final application of our data structure, we implement an algorithm to compute the c-closure of a graph. The c-closure is a complexity measure for social networks [15], capturing the largest set of common neighbours between any two non-adjacent vertices. In social networks, this value is expected to be low due to the principle of triadic closure [22]. Furthermore, social networks are often observed to be d-degenerate[3], possibly due to an upper bound on meaningful connections any one person can maintain, as discussed by Dunbar [10]. We provide two algorithms with different running times, both achieving $O(nd^3)$ when the c-closure is lower bounded by the graph's degeneracy, a condition met by almost all non-bipartite graphs[4] (the exceptions are a few well-clustered networks with large cliques) in our experiments. An FPT algorithm with a linear dependence on the input size is sometimes referred to as a fixed-parameter linear (FPL) algorithm [17], and a parameterized algorithm running in polynomial time is sometimes called an FPT-in-P algorithm [16], thereby making our algorithm a linear FPT-in-P algorithm when the c-closure is lower bounded by the degeneracy. The previously best known running time for computing the c-closure of a graph is $O(cn^2 + m^{3/2})$ [19]. Using our data structure, our algorithm runs in time $O(n \cdot d(d^2 + 2^d))$, hence we expect it to be faster when d is $o(\log n)$, and in most cases, where $c \geqslant d$, our algorithm is faster when d is $o(\sqrt{n})$.

Prior work. We briefly mention a few relevant previous articles on the subject. Eppstein, Löffler, and Strash [13] gave an algorithm for enumerating maximal cliques in d-degenerate graphs in $O(dn3^{d/3})$ time, i.e., fixed-parameter tractable time when parameterized by the degeneracy. They also give experimental results showing that their algorithm works well on large real-world networks. Bera, Pashanasangi, and Seshadhri [1], extending the classic result by Chiba and Nishizeki [6], show that for all patterns H of size less than six, we can count the number of appearances of H in a d-degenerate graph G in time $O(m \cdot d^{k-2})$, where m is the number of edges in G and k is the number of vertices in H. Recently, Bressan, and Roth [3] gave algorithms for counting copies of a graph H in a d-degenerate graph G in time $f(d,k) \cdot n^{\mathbf{im}(H)} \log n$, for some function f, where k again is the number of vertices in H, n the number of vertices in G, and $\mathbf{im}(H)$ is the size of a largest induced matching in H.

Our contribution. We summarize our results here: For a d-degenerate n-vertex graph G, we show: (i) The VC-dimension of G can be computed in time $n^{\lceil \log d+1 \rceil} d^{O(d)}$, and that the $\log d$ in

[2]There are probably faster programs to compute bicliques in practice, we compute this statistic here as a baseline.
[3]In our network corpus, 108 of 206 have degeneracy less than 10 and only ten graphs have degeneracy above 100.
[4]The c-closure of a bipartite graph is the size of the largest common neighbourhood, and is thus expected to be high.

the exponent of n is likely necessary. (ii) The number of copies of a bipartite pattern H can be counted in time $n^{\mathrm{lc}(H)} \cdot f_H(d)$, where the dependence on H is governed by its left-covering number and half-ordering asymmetry. (iii) As concrete corollaries, bicliques can be counted in linear time, co-matchings in quadratic time, and the ladder index can be approximated within a factor of 2 in time $O(d^2 8^d \cdot n)$. (iv) The c-closure can be computed in time $O(nd^3 + n^2 d)$, and in fact, in $O(nd^3)$ when the c-closure is at least d.

In particular, for VC-dimension (i), this improves over the generic $O(n^{\log n})$ running time by replacing the exponent $\log n$ with $\lceil \log d + 1 \rceil$ on d-degenerate graphs, which is a substantial improvement whenever $d \ll n$.

2 Preliminaries

For an integer k, we use $[k]$ as a shorthand for the set $\{0, 1, 2, \ldots, k-1\}$. We use blackboard bold letters like $\mathbb{X}$ to denote sets X associated with a total order $<_{\mathbb{X}}$. The *index function* $\iota_{\mathbb{X}} \colon X \to \mathbb{N}$ maps elements of X to their corresponding position in $\mathbb{X}$. We extend this function to sets via $\iota_{\mathbb{X}}(S) = \{\iota_{\mathbb{X}}(s) \mid s \in S\}$. For any integer $i \in [|X|]$ we write $\mathbb{X}[i]$ to mean the ith element in the ordered set. An *index set* I for $\mathbb{X}$ is simply a subset of $[|X|]$ and we extend the index notation to sets via $\mathbb{X}[I] \coloneqq \{\mathbb{X}[i] \mid i \in I\}$. We write $\pi(H)$ for the set of all permutations of H. All logarithms in this article are *base 2*, i.e., we will write $\log d$ to mean $\log_2 d$.

For a graph G we use $V(G)$ and $E(G)$ to refer to its vertex- and edge-set, respectively. We use the shorthands $|G| \coloneqq |V(G)|$ and $\|G\| \coloneqq |E(G)|$.

An *ordered graph* is a pair $\mathbb{G} = (G, <)$ where G is a graph and $<$ a total ordering of $V(G)$. We write $<_{\mathbb{G}}$ to denote the ordering for a given ordered graph and extend this notation to the derived relations $\leqslant_{\mathbb{G}}$, $>_{\mathbb{G}}$, $\geqslant_{\mathbb{G}}$.

We use the same notations for graphs and ordered graphs and additionally we write

$$N^-(u) \coloneqq \{v \in N(u) \mid v <_{\mathbb{G}} u\}$$

for the *left neighbourhood* and

$$N^+(u) \coloneqq \{v \in N(u) \mid v >_{\mathbb{G}} u\}$$

for the *right neighbourhood* of a vertex $u \in \mathbb{G}$. We further use $d^-_{\mathbb{G}}(u)$ and $d^+_G(u)$ for the left and right degree, as well as $\Delta^-(\mathbb{G}) \coloneqq \max_{u \in \mathbb{G}} d^-_{\mathbb{G}}(u)$ and $\Delta^+(\mathbb{G}) \coloneqq \max_{u \in \mathbb{G}} d^+_{\mathbb{G}}(u)$. We omit the graphs in the subscripts if clear from the context.

A graph G is d-degenerate if there exists an ordering $\mathbb{G}$ such that $\Delta^-(\mathbb{G}) \leqslant d$. An equivalent definition is that a graph is d-degenerate if every subgraph has a vertex of degree at most d. The number of edges in a d-degenerate graph is bounded by dn and many important sparse graph classes—bounded treewidth, planar graphs, graphs excluding a minor—have finite degeneracy. Indeed, planar graphs have degeneracy at most 5, whereas graphs with treewidth t have degeneracy at most t. The degeneracy ordering of a graph can be computed in time $O(n + m)$ [20], which is $O(dn)$ when G is a d-degenerate graph. We note here that we can, in the same time bound, rearrange the neighbourhoods of each vertex in such a way that each vertex v has two ordered lists $N^-_G(v)$ and $N^+_G(v)$ corresponding to the left and right neighbourhoods, respectively, and that both lists are sorted by their position in the degeneracy ordering. After finding the degeneracy ordering, we go through each vertex v from left to right and keep a set of already processed vertices L such that when we process v and its neighbour u, we know that $u \in L$ if and only if u is left of v:

```
input 𝔾[a]
for v ∈ 𝔾 do
    N⁻(v) ← []
    N⁺(v) ← []
for v ∈ 𝔾 do
    for u ∈ N_𝔾(v) do
        if v <_𝔾 u then
            N⁻(u).push_back(v)
        else
            N⁺(u).push_back(v)
```

[a] In all our algorithms, we iterate through the vertices in the degeneracy ordering.

Clearly the algorithm runs in time $O(n+m)$, and $N^{\pm}(\bullet)$ is ordered according to the degeneracy ordering, since elements are added in that order. Each edge uv (suppose $u <_{\mathbb{G}} v$) is processed twice; The first time, u is added to $N^-(v)$ and the second time, v is added to $N^+(u)$. Note that this also allows us to check if $u \in N^-(v)$ in time $O(\log d)$, and hence we can check whether $uv \in E$ in $O(\log d)$ time.

Let $\mathcal{F} \subseteq 2^U$ be a set family over U. We define the intersection of a set family with set $X \subseteq U$ as $\mathcal{F} \cap X \coloneqq \{F \cap X \mid F \in \mathcal{F}\}$. A set $X \subseteq U$ is *shattered* by $\mathcal{F}$ if $\mathcal{F} \cap X = 2^X$. The *graph representation* of a set family $\mathcal{F}$ is the bipartite graph $G(\mathcal{F}) = (\mathcal{F}, U, E)$ where for each $F \in \mathcal{F}$ and $x \in U$ we have the edge $Fx \in E$ iff $x \in F$. In the other direction, we define for a graph G its *neighbourhood set system* $\mathcal{F}(G) \coloneqq \{N(v) \mid v \in G\}$.

Definition 1 (VC-dimension)**.** *The Vapnik–Chervonenkis dimension (VC-dimension) of a set family $\mathcal{F} \subseteq 2^U$ is the size of the largest set in U that is shattered by $\mathcal{F}$ and we write this quantity as $\mathbf{vc}(\mathcal{F})$. The VC-dimension of a graph G, called the* graph VC-dimension*, is defined as the VC-dimension of its neighbourhood set system, i.e. $\mathbf{vc}(G) \coloneqq \mathbf{vc}(\mathcal{F}(G))$.*

2.1 Set dictionaries

In the following we will make heavy use of data structures that model functions of the form $f\colon 2^U \to \mathbb{Z}$ for some universe U. Since the arguments in our use-case are assumed to be small, we use prefix-tries [23] in our theoretical analysis (see notes on practical implementations below):

Definition 2 (Subset dictionary)**.** *Let U be a set and let $\mathbb{U}$ be an arbitrary total order of U. A* subset dictionary *D over U associates a key $X \subseteq U$ with an integer $D[X]$ by storing the sequence $\mathbb{X}$ of X under $<_{\mathbb{U}}$ in a prefix trie.*

Accordingly, insertion/update/deletion of a value for a key X takes time $O(|X|)$ *if* we can assume the key X to be present in some canonical order. Our algorithms all work on graphs imbued with a (degeneracy) ordering and we will sort the left-neighbourhood $N^-(\bullet)$ of each vertex according to this global ordering, which we will simply call "sorting the left-neighbourhoods" for brevity. Subsets of these left-neighbourhoods are assumed to inherit this ordering, which covers all operations that we will need in our algorithms, which in conclusion means that we can assume that all sets used as keys in subset dictionaries have a canonical ordering.

Unless otherwise noted, we will use the convention that $D[X] = 0$ for all keys X that have not been inserted into D.

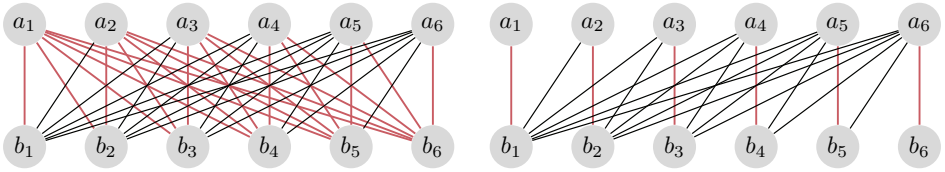
2.2 Bipartite patterns and left-covers

Definition 3 (Pattern)**.** *A* pattern H *is a complete graph whose edges are partitioned into sets* B*,* R*, and* W *(black, red and white). We say that a graph* G contains H *(or* H appears in G*) if there exists a vertex set* $S \subseteq V(G)$ *and a bijection* $\phi\colon V(H) \to S$ *such that* $uv \in B \implies \phi(u)\phi(v) \in E(G)$ *and* $uv \in R \implies \phi(u)\phi(v) \notin E(G)$*.*

We say that a pattern H *is* bipartite *if the vertex set of* H *can be partitioned into two sets* X, Y *such that all edges inside* X *and inside* Y *are white.*

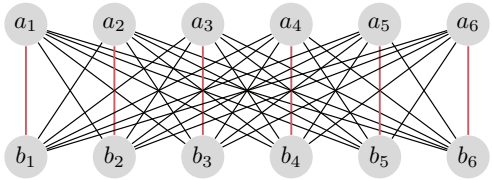
For a vertex $v \in V(H)$ we write $N(v)$ to denote its neighbours according to the black edge relation only. An *ordered* pattern $\mathbb{H}$ is a pattern whose vertex set comes with a linear order $<_{\mathbb{H}}$. Given a vertex $v \in V(\mathbb{H})$, we write $N^-(v) \coloneqq \{u \in N(v) \mid u <_{\mathbb{H}} v\}$.

A *ladder* (sometimes called a chain graph) L_n of size n is a bipartite pattern defined on two vertex sequences $V_A = (a_i)_{i\in[n]}$ and $V_B = (b_i)_{i\in[n]}$, where $a_ib_j \in B$ if $i > j$ and $a_ib_j \in R$ otherwise. Note that for any $1 \leqslant l \leqslant r \leqslant n$ the subgraph induced by the sequences $(a_i)_{i\in[l,r]}$ and $(b_i)_{i\in[l,r]}$ induces a ladder. A *semi-ladder* $\tilde{L}_n$ has the same black edges, but only the edges a_ib_i, $i \in [n]$ are red. All the remaining edges are white:

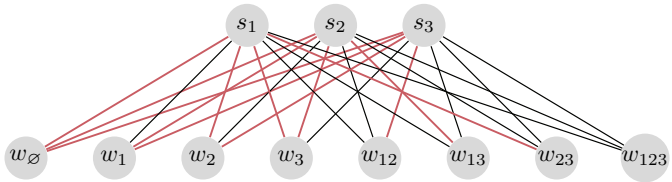


The *ladder index* of a graph G is the largest n such that G contains the pattern L_n.

A *co-matching* $\overline{M}_n$ (also called *crown*) has black edges a_ib_j for $i \neq j$ and red edges a_ib_i for $i \in [n]$.



Finally, the *shattered* pattern U_n of size n has a side S (the *shattered* set) of size n and a side W (the *witness* set) of size 2^n. We index the vertices of W by subsets $I \subseteq S$, then the vertex w_I has black edges into I and red edges into $S \setminus I$:



Definition 4 (Left-cover, left-covering number)**.** *Given an ordered bipartite pattern* $\mathbb{H}$ *with bipartition* (X, Y)*, a* left-cover *is a set of vertices* $C \subseteq V(\mathbb{H})$ *such that either* $X \subseteq N^-(C) \cup C$ *or*

$Y \subseteq N^-(C) \cup C$. *The* left-covering number $\mathrm{lc}(\mathbb{H})$ *is the minimum size of a left cover of* $\mathbb{H}$. *For an (unordered) pattern* H *we define its left-covering number as*

$$\mathrm{lc}(H) \coloneqq \max_{\mathbb{H} \in \pi(H)} \mathrm{lc}(\mathbb{H}).$$

Note that we include the covering set C itself in the cover; This is necessary since for a given ordering of a pattern some vertices might not have right neighbours and can therefore not be covered by left neighbourhoods.

The left-covering number of a pattern is the first important measure that will influence the running time of the main algorithm presented later. The second important measure relates to the number of non-isomorphic "half"-ordered patterns we can obtain from a bipartite pattern, that is, how many distinct objects we find by ordering one partition. A useful tool to concretise this notion is the following function:

Definition 5 (Signature)**.** *Let* H *be a bipartite pattern with bipartition* (X, Y) *and let* $\mathbb{Z}$ *be an ordering of* $Z \in \{X, Y\}$. *Then the* signature $\sigma_{\mathbb{Z}}(H)$ *is defined as the multiset*

$$\sigma_{\mathbb{Z}}(H) \coloneqq \{\{\iota_{\mathbb{Z}}(N(u)) \mid u \in (X \cup Y) \setminus Z\}\}.$$

For orderings $\mathbb{Z}, \mathbb{Z}' \in \pi(Z)$ *we define the equivalence relation*

$$\mathbb{Z} \sim_H \mathbb{Z}' \iff \sigma_{\mathbb{Z}}(H) = \sigma_{\mathbb{Z}'}(H).$$

Definition 6 (Half-ordering asymmetry)**.** *Given a bipartite pattern* H *with bipartition* (X, Y) *and a partite set* $Z \in \{X, Y\}$, *we define the* half-ordering asymmetry $\mathrm{hoa}(H, Z)$ *as the number of equivalence classes under the* $\sim_H$ *relation*

$$\mathrm{hoa}(H, Z) \coloneqq |\pi(Z)/\sim_H|.$$

We further define the half-ordering asymmetry of H *as*

$$\mathrm{hoa}(H) \coloneqq \max\{\mathrm{hoa}(H, X), \mathrm{hoa}(H, Y)\}.$$

Alternatively, $\mathrm{hoa}(H, Z) \coloneqq |\{\sigma_{\mathbb{Z}}(H) \mid \mathbb{Z} \in \pi(Z)\}|$.

3 A general pattern-finding algorithm

We first describe a general-purpose algorithm for finding patterns in degenerate graphs. Afterwards, we will describe more specialised algorithms using similar ideas to find specific patterns.

The crucial primitive is to query neighbourhood patterns on a small set $S \subseteq V(G)$. Suppose we fix such a set S. Then, for every subset $X \subseteq S$, we would like to know how many vertices v satisfy $N(v) \cap S = X$. This is exactly what Theorem 2 provides. The reason this is useful is that a bipartite pattern H can be viewed as prescribing, for each vertex on one side, which subset of the other side it should be adjacent to. Thus, once we guess one side of the pattern, Theorem 2 lets us test whether the required neighbourhood traces exist on the other side, and even count how many choices there are.

Theorem 1. *Let* G *be a* d*-degenerate graph and let* H *be a bipartite pattern with bipartition* (X, Y) *where* $|X| \geqslant |Y|$. *Then after a preprocessing time of* $O(|X|^{\mathrm{lc}(H)}|H|! + d2^d n)$, *we can count how often* H *appears in* G *in time* $O\big(n^{\mathrm{lc}(H)}(4d\,\mathrm{lc}(H))^{|X|} d|X|^3\,\mathrm{hoa}(H)\big)$.

Throughout this theorem, counting appearances of H means counting ordered pairs (X', Y') of disjoint subsets of $V(G)$ with $|X'| = |X|$ and $|Y'| = |Y|$ for which there exists a bijection $\varphi\colon V(H) \to X' \cup Y'$ satisfying $\varphi(X) = X'$, $\varphi(Y) = Y'$, all black-edge adjacency constraints, and all red-edge non-adjacency constraints. Different bijections witnessing the same ordered pair (X', Y') are not counted separately.

The main ingredient of our algorithm is the following data structure on $\mathbb{G}$, $Q_S : 2^S \to \mathbb{N}$:

Theorem 2. *Let $\mathbb{G}$ be a d-degenerate ordering of a graph G on n vertices. After a preprocessing time of $O(d2^d n)$, we can, for any given $S \subseteq V(G)$, compute a subset dictionary Q_S in time $O(|S|2^{|S|} + d|S|^2)$ which for any $X \subseteq S \subseteq V(G)$ answers the query*

$$Q_S[X] \coloneqq \big|\{v \in G \mid S \cap N(v) = X\}\big|$$

in time $O(|X|)$.

The first part to the above theorem is to compute a subset dictionary that can, given any subset of vertices $X \subseteq V$, output in linear time in $|X|$ the number of vertices v such that $X \subseteq N^-(v)$, i.e., how many vertices have X contained in their left-neighbourhood? Clearly the query needs to read X, so the running time needs to be $\Omega(|X|)$. We show that it can be done, after a global preprocessing, in time $O(|X|)$.

Lemma 1. *Let $\mathbb{G}$ be a d-degenerate ordering of a graph G. Then in time $O(d2^d n)$ we can compute a subset dictionary R over $V(G)$ which for any $X \subseteq V(G)$ answers the query*

$$R[X] \coloneqq \big|\{v \in G \mid X \subseteq N^-(v)\}\big|$$

in time $O(|X|)$.

Proof: Given $\mathbb{G}$ as input, we compute R as follows:

Initialize R as an empty trie storing integers
for $u \in \mathbb{G}$ **do**
  for $X \subseteq N^-(u)$ **do**
    $R[X] \leftarrow R[X] + 1$ `// Non-existing keys are treated as zero`
return R

Note that every update of the data structure with key X takes time $O(|X|)$, since $|X| \leqslant d$ it follows that the total initialisation time is bounded by $O(d2^d n)$. □

Lemma 2. *Let $\mathbb{G}$ be a d-degenerate ordering of a graph G and let $S \subseteq V(G)$. Provided the subset dictionary R of Lemma 1, we can construct in time $O(|S|2^{|S|} + d|S|^2)$ a subset dictionary Q_S over S which for $X \subseteq S$ answers the query*

$$Q_S[X] \coloneqq \big|\{v \in G \mid S \cap N(v) = X\}\big|$$

in time $O(|X|)$.

Proof: The data structure is built in two steps (i and ii below). First, we count for every small set X how many vertices have X contained in their left-neighbourhood. Second, we recover the number of vertices whose neighbourhood inside a query set S is *exactly* X by an inclusion–exclusion

step, implemented via fast Möbius inversion on the subset lattice of S: (i) We first construct an auxiliary subset dictionary $\hat{Q}$ which for $X \subseteq S$ answers the query

$$\hat{Q}_S[X] \coloneqq \big|\{v \in G \mid S \cap N^-(v) = X\}\big|$$

in time $O(|X|)$. We first prove the following claim which implies that $\hat{Q}_S$ is the (upwards) Möbius inversion of R over S and hence can be computed in time $O(|S|2^{|S|})$ using Yates' algorithm [24, 18, 2].

Claim 1. $\big|\{v \in G \mid S \cap N^-(v) = X\}\big| = \sum_{X \subseteq Y \subseteq S} (-1)^{|Y \setminus X|} R[Y]$.

Proof: First consider $v \not\geqslant_{\mathbb{G}} X$, meaning that v does *not come after* X in $\mathbb{G}$. Then X cannot be contained in $N^-(v)$ and therefore v does not contribute to the left-hand side. Note that v is not counted by $R[Y]$ for any $Y \supseteq X$, therefore v does not contribute to the right-hand side.

Consider therefore $v \geqslant_{\mathbb{G}} X$. First, assume that $S \cap N^-(v) = X$ and therefore v contributes to the left-hand side. Then v is counted on the right-hand side exactly once by the term $R[X]$ which has a positive sign.

Consider now v with $S \cap N^-(v) \neq X$. If $X \not\subseteq N^-(v)$, then v does not contribute to the left-hand side and it is not counted by any term $R[Y]$, $Y \supseteq X$ on the right-hand side. We are therefore left with vertices v where $I \coloneqq S \cap N^-(v)$ satisfies $X \subset I$. Note that I is counted by every term $R[Y]$ with $X \subseteq Y \subseteq I$. Since

$$\sum_{X \subseteq Y \subseteq I} (-1)^{|Y \setminus X|} = \sum_{0 \leqslant k \leqslant |I \setminus X|} (-1)^k \binom{|I \setminus X|}{k} = 0$$

we conclude that these counts of v cancel out and contribute a sum-total of zero to the right-hand side. This covers all cases and we conclude that the claim holds. □

(ii) It remains to be shown how the query $Q_S[X]$ can be computed using $\hat{Q}_S[X]$. To this end, consider a vertex $v \in G$ where $S \cap N(v) \neq S \cap N^-(v)$ as these contribute to $\hat{Q}_S[X]$ but must not be counted by $Q_S[X]$. Note that any such vertex must be contained in $N^-(S)$ since v has at least one right-neighbour in S. Accordingly, we apply the following correction to $\hat{Q}_S[X]$:

Let $Q_S = \hat{Q}_S$
for $u \in N^-(S)$ **do**
  $Q_S[N^-(u) \cap S] \leftarrow Q_S[N^-(u) \cap S] - 1$
  $Q_S[N(u) \cap S] \leftarrow Q_S[N(u) \cap S] + 1$

Considering that we can compute $A \cap B$ in time $\min\{|A|, |B|\}$ by going through all elements of the smaller and querying the larger, this correction takes time $O(d|S|^2)$. □

We are now ready to describe the pattern-counting algorithm.

Proof of Theorem 1: The problem is trivial for $|X| = 1$ since then the pattern is either a single edge or anti-edge. Thus assume $|X| \geqslant 2$ in the following, in particular for the running time calculations.

We first compute the left-covering number $\mathrm{lc}(H)$ by simply brute-forcing all orderings of H in time $O(|H|! \cdot \max\{|X|, |Y|\}^{\mathrm{lc}(H)}) = O(|H|!|X|^{\mathrm{lc}(H)})$. At the same time, whenever we find that a specific ordering $\mathbb{H}$ has a minimal left-covering of X, then we add the signature $\sigma_{\mathbb{X}}(H)$ with $\mathbb{X} \coloneqq \mathbb{H}[X]$ to a collection $\mathcal{X}$. Similarly, if we find that a minimal left-covering in $\mathbb{H}$ covers Y we add the signature $\sigma_{\mathbb{Y}}(H)$ with $\mathbb{Y} \coloneqq \mathbb{H}[Y]$ to a collection $\mathcal{Y}$. We will later use that $|\mathcal{X}| \leqslant \mathrm{hoa}(H, X)$ and $|\mathcal{Y}| \leqslant \mathrm{hoa}(H, Y)$.

We now compute a d-degenerate ordering $\mathbb{G}$ of G in time $O(dn)$ with ordered neighbourhoods, and compute the data structure R as per Lemma 1 in time $O(d2^d n)$. If we want to compute the number of times H appears in G, we further need to initialise a subset dictionary K.

We now iterate through all subsets $C \subseteq V(G)$ of size $\mathrm{lc}(H)$ and for each such set we iterate through all subsets $Z \subseteq N^-_{\mathbb{G}}(C) \cup C$ of size $|X|$ or $|Y|$, in total this takes time

$$O(n^{\mathrm{lc}(H)}((d+1)\,\mathrm{lc}(H))^{|X|}).$$

We describe the remainder of the algorithm for a set $X = Z$ of size $|X|$, the procedure for a set Y works analogously. Let $\mathbb{X}$ be the ordering of X in $\mathbb{G}$.

To verify that X can be completed into a pattern H in G, we compute the data structure Q_X in time $O(|X|2^{|X|} + d|X|^2)$ as per Theorem 2. To check whether H exists in G, we iterate through all signatures $\sigma \in \mathcal{X}$ and test whether $Q_X[\mathbb{X}[A]] > 0$ for all index sets $A \in \sigma$, this takes time $O(|\mathcal{X}||X||Y|)$, in total the verification step for X takes time

$$O\big((|X|2^{|X|} + d|X|^2) \cdot |\mathcal{X}||X||Y|\big) = O\big(d|X|^3 2^{|X|}\,\mathrm{hoa}(H)\big)$$

where we used that $|X| \geqslant |Y|$ and $|X| \geqslant 2$. This bound also holds for checking Y since $|\mathcal{X}| + |\mathcal{Y}| \leqslant 2\,\mathrm{hoa}(H)$. Finally, if we exhaust all orderings of H without finding the pattern, we report that it does not exist in G.

To *count* in how many ways X can be extended into the pattern H in G, we compute

$$c_{H,X} \coloneqq \sum_{\sigma \in \mathcal{X}} \prod_{A^{(k)} \in \sigma} \binom{Q_X[\mathbb{X}[A]]}{k},$$

where k denotes the multiplicity of A in the multiset σ. Note, however, that we have to take care not to double-count the contribution of X to the overall count as we might encounter the set X multiple times. To that end, we record the intermediate result by setting $K[X] \coloneqq c_{H,X}$ and we forgo the above computation if X exists already as a key in K. The computation of $c_{H,X}$ and this additional book keeping takes time $O(|X| + |\mathcal{X}||X||Y|)$, in total we arrive at the same running time as the decision variant, $O\big(d|X|^3 2^{|X|}\,\mathrm{hoa}(H)\big)$. After exhausting all orderings of H we report back the number of times H appears in G as the sum of all entries of K.

The total running time of either variant of the algorithm is, as claimed,

$$\begin{aligned} &O\Big(|X|^{\mathrm{lc}(H)}|H|! + d2^d n + dn + n^{\mathrm{lc}(H)}\big((d+1)\,\mathrm{lc}(H)\big)^{|X|} \cdot d|X|^3 2^{|X|}\,\mathrm{hoa}(H)\Big) \\ &= O\Big(|X|^{\mathrm{lc}(H)}|H|! + d2^d n + n^{\mathrm{lc}(H)}(4d\,\mathrm{lc}(H))^{|X|} d|X|^3\,\mathrm{hoa}(H)\Big). \end{aligned}$$

□

4 Concrete applications

4.1 Finding bicliques and co-matchings

We note that $\mathrm{lc}(K_{t,t}) = 1$ and $\mathrm{hoa}(K_{t,t}) = 1$, therefore the application of Theorem 1 gives the following:

Corollary 1. *Let G be a d-degenerate graph on n vertices. Then we can compute the number of biclique patterns $K_{s,t}$ $(s \geqslant t)$ in G in time $O\big(s \cdot (2s)! + d2^d n + n(4d)^s ds^3\big)$.*

Let $\overline{M}_t$ be a co-matching on $2t$ vertices. We will assume in the following that the partite sets of $\overline{M}_t$ are $X \coloneqq (x_1, \ldots, x_t)$ and $Y \coloneqq (y_1, \ldots, y_t)$ so that the edges $x_i y_i$ for $i \in [t]$ are forbidden.

Lemma 3. $\mathrm{lc}(\overline{M}_t) = 2$ *and* $\mathrm{hoa}(\overline{M}_t) = 1$.

Proof: Let $\bar{\mathbb{M}}_t$ be an ordering of $\overline{M}_t$ and let z be the last vertex in that order. Then $N^-(z)$ covers all vertices of one partite set except one vertex z'. Thus $\{z, z'\}$ is a left-cover of $\bar{\mathbb{M}}_t$.

To determine the half-ordering asymmetry, note that for *every* ordering $\mathbb{Z}$ of $Z \in \{X, Y\}$ the signature $\sigma_{\mathbb{Z}}(\overline{M}_t)$ is simply the set $\binom{[t]}{t-1}$, so the total number of signatures is one. □

Corollary 2. *Let G be a d-degenerate graph. Then we can compute the number of co-matching patterns $\overline{M}_t$ in time $O\big(t^2(2t)! + d2^d n + n^2(8d)^t dt^3\big)$.*

4.2 Finding shattered sets

A direct application of Theorem 1 to locate a shattered pattern U_t is unsatisfactory as the running time will include a factor of n^t since $\mathrm{lc}(U_t) = t$. By the following observation, we can bound t by the degeneracy of the graph, but we can greatly improve the running time by further adjusting the algorithm.

Observation 1. *Let G be a d-degenerate graph. Then* $\mathbf{vc}(G) \leqslant d + 1$.

Proof: Assume $S \subseteq V(G)$ is shattered by $W \subseteq V(G)$, with $|S| = \mathbf{vc}(G)$. Let $W' \subseteq W$ be those witnesses that have $|S| - 1$ neighbours in S. Then $G[W' \cup S]$ induces a graph of minimum degree $|S| - 1$ and we must have that $|S| - 1 \leqslant d$ and accordingly $\mathbf{vc}(G) = |S| \leqslant d + 1$. □

The core observation that allows further improvements is that many orderings of U_{d+1} have degeneracy *larger* than d and can therefore not appear in a d-degenerate graph. In particular, the ordering in which all witnesses of U_{d+1} appear before the shattered set has degeneracy 2^{d+1} and can therefore be ruled out. We refine this idea further in the following lemma.

Lemma 4. *Let $\mathbb{G}$ be a d-degenerate ordering of a graph G. Let G contain the shattered pattern U_t and let $\mathbb{U}_t \coloneqq \mathbb{G}[U_t]$ be its ordering. Then* $\mathrm{lc}(\mathbb{U}_t) \leqslant \lceil \log d + 1 \rceil$. *Specifically, we either have that $t \leqslant \lceil \log d + 1 \rceil$ or that $\mathbb{U}_t$ can be covered by $\lceil \log d + 1 \rceil$ witness vertices.*

Proof: Let $S = (s_1, \ldots, s_t)$ and $W = (w_1, \ldots, w_{2^t})$ be the vertices of U_t in G and let the indices of the variables reflect the ordering of the corresponding vertices in $\mathbb{U}_t$.

Partition the set S into $p \coloneqq \lceil \log d + 1 \rceil$ sets $S_1, \ldots, S_p$ such that each set has size at least $\lfloor t/p \rfloor$ and at most $\lceil t/p \rceil$. For each set S_i define the set of "apex"-witnesses $A_i \coloneqq \{w \in W \mid N(w) \supseteq S_i\}$. Note that, for all $i \in [p]$,

$$|A_i| = 2^{|S \setminus S_i|} \geqslant 2^{t - \lceil t/p \rceil} \geqslant 2^{\lceil t \frac{p-1}{p} \rceil - 1}.$$

We call a set A_i *good* if $\max_{\mathbb{G}} A_i > \max_{\mathbb{G}} S_i$, that is, at least one apex vertex from A_i can be found to the right of S_i. We now distinguish two cases:

Case 1. All A_i, $i \in [p]$, are good.
It follows that $\mathbb{U}_t$ can be left-covered by taking one vertex from each A_i, $i \in [p]$. We conclude that $\mathrm{lc}(\mathbb{U}_t) \leqslant p = \lceil \log d + 1 \rceil$.

Case 2. Some A_i, $i \in [p]$, is not good.

Let $u = \max_{\mathbb{G}} S_i$ be the last vertex in S_i, note that $A_i \leqslant_{\mathbb{G}} u$ and accordingly $A_i \subseteq N^-(u)$. But then we must have that $|A_i| \leqslant d$ and accordingly that

$$2^{\lceil t\frac{p-1}{p}\rceil} \leqslant d \iff \lceil t\frac{p-1}{p}\rceil \leqslant \log d \implies t\frac{p-1}{p} \leqslant \log d \iff t \leqslant \frac{p}{p-1}\log d$$
$$\iff t \leqslant \frac{\lceil \log d + 1\rceil}{\lceil \log d + 1\rceil - 1}\log d = \frac{\log d}{\lceil \log d\rceil}\lceil \log d + 1\rceil \leqslant \lceil \log d + 1\rceil.$$

We therefore find that $\mathrm{lc}(\mathbb{U}_t) \leqslant |S| \leqslant \lceil \log d + 1\rceil$. □

Theorem 3. *Let G be a d-degenerate graph on n vertices. Then we can determine the VC-dimension of its neighbourhood set system $\mathcal{F}(G)$ in time $O(n^{\lceil \log d+1\rceil} d^{d+2}(2d\log d)^{d+1})$.*

Proof: We first compute an ordering $\mathbb{G}$ of G with degeneracy d in time $O(dn)$ with sorted neighbourhoods. Let $p := \lceil \log d + 1\rceil$ in the following.

Let $\mathbb{U}_t = (S, W)$ be a shattered set of size $t \leqslant d+1$ in $\mathbb{G}$. By Lemma 4 we then have that $\mathrm{lc}(\mathbb{U}_t) \leqslant p$. Therefore to locate the set S we first guess up to p vertices and then exhaustively search through their (closed) left-neighbourhoods in time

$$\binom{n}{p}\binom{dp}{t} \leqslant \left(\frac{en}{p}\right)^p \left(\frac{edp}{t}\right)^t = O\left(n^p (d\log d)^{d+1}\right) = O\left(n^{\lceil \log d+1\rceil}(d\log d)^{d+1}\right).$$

Now that we can locate S we apply Theorem 2 in order to verify that S is indeed shattered: For each candidate set S from the previous step, we compute a subset dictionary Q_S in time $O(|S|2^{|S|} + d|S|^2) = O(d2^d)$ and then check whether $Q_S[X] > 0$ for each $X \subseteq S$. This latter step takes time $O(|S|2^{|S|})$ and is therefore subsumed by the construction time of Q_S. We conclude that the algorithm runs in total time

$$O(d\cdot n) + O(d2^d n) + O\left(n^{\lceil \log d+1\rceil}(d\log d)^{d+1}\cdot d2^d\right) = O\left(n^{\lceil \log d+1\rceil} d^{d+2}(2d\log d)^{d+1}\right)$$

as claimed. □

We note that the exponent of $\lceil \log d + 1\rceil$ in the running time is essentially tight:

Theorem 4. *Graph VC-dimension parameterized by the degeneracy d of the input graph cannot be solved in time $f(d)\cdot n^{o(\log d)}$ unless all problems in* SNP *can be solved in subexponential time.*

Proof: We adapt the W[1]-hardness reduction from k-Clique to VC-dimension by Downey, Evans, and Fellows [8] and combine it with the result by Chen *et al.* [5, 4] which states that k-Clique cannot be solved in time $f(k)n^{o(k)}$ unless all problems in SNP admit subexponential-time algorithms.

Given an instance (H, k) for k-Clique, we construct a graph G as follows. We first create k copies $V_1, \ldots, V_k$ of $V(H)$. For $v \in V(H)$, let us denote its copies by $v^{(1)}, \ldots, v^{(k)}$ with $v^{(i)} \in V_i$ for $i \in [k]$. We now add the following vertices and edges:

- A single isolated vertex w_0,
- a vertex set W_1 which contains one pendant vertex for each $v^{(i)}$, $v \in V(H)$ and $i \in [k]$,
- a vertex set W_2 which for each edge $uv \in E(H)$ contains $\binom{k}{2}$ vertices w_{uv}^{ij}, $i, j \in [k]$, each of which has $u^{(i)}$ and $v^{(j)}$ as its only neighbours, and

- a vertex set A which for each index set $I \subseteq [k]$ with $|I| \geqslant 3$, contains a vertex a_I which is connected to all vertices in V_i for each $i \in I$.

Note that the graph is bipartite with partite sets $\mathcal{V} \coloneqq V_1 \cup \cdots \cup V_k$ and $\mathcal{W} \coloneqq W_1 \cup W_2 \cup A$.

Let us first show that if H contains a clique of size k then G contains a shattered set of size k. Let $u_1, \ldots, u_k$ be distinct vertices that form a complete graph in H. We claim that then the set $S \coloneqq \{u_1^{(1)}, \ldots, u_k^{(k)}\}$ is shattered in G. First, note that for every subset $X \subset S$, $|X| \geqslant 3$, there exists a witness vertex $a \in A$ such that $N(a) \cap S = X$. For the empty set we have the witness w_0, for every singleton subset $\{u\} \subseteq S$ we have that the pendant vertex $p \in N(u) \cap W_1$ witnesses $\{u\}$. Therefore, only subsets of size exactly two need to be witnesses to shatter S. Consider $\{u_i^{(i)}, u_j^{(j)}\} \subseteq S$ for $i \neq j$. Since $u_i u_j \in E(H)$, the vertex $w_{u_i u_j}^{ij}$ exists in W_2 and its neighbourhood in S is exactly $\{u_i^{(i)}, u_j^{(j)}\}$. We conclude that all subsets of size two in S are witnessed as well and therefore S is shattered.

In the other direction, assume that G contains a shattered set (S, W) of size k. Without loss of generality, assume that $k \geqslant 3$.

Claim 2. *$S \subseteq \mathcal{V}$ and $W \subseteq \mathcal{W}$.*

Proof: Since G is bipartite we either have that $S \subseteq \mathcal{V}$ and $W \subseteq \mathcal{W}$ or that $S \subseteq \mathcal{W}$ and $W \subseteq \mathcal{V}$. Let us now show that the latter is impossible.

Since $k \geqslant 3$ we have that every vertex in S has degree at least four. Accordingly, W cannot contain vertices from W_1 or W_2, which leaves us with $W \subseteq A$. However, all vertices in V_i, $i \in [k]$, have the exact same neighbours in A. Therefore only k subsets of A are witnessed by vertices in $\mathcal{V}$ and therefore the largest shattered set in A has size at most $\log k$. We conclude that S cannot be contained in A and the claim holds. □

We now claim that $|S \cap V_i| = 1$ for all $i \in [k]$. Assume otherwise, so let $u^{(i)}, v^{(i)} \in S$ for some $i \in [k]$. But then the set $\{u^i, v^i\}$ cannot be witnessed: not by a vertex from W_1, since it only contains vertices with one neighbour, not by a vertex from W_2, since these vertices each have at most one neighbour in each set V_i, and not by a vertex from A since we need all $2^k - \binom{k}{2} - k - 1$ vertices of A to witness subsets of S of size at least three.

Therefore S intersects each V_i in exactly one vertex. Since S is shattered, every subset $\{u^{(i)}, v^{(j)}\}$, $i \neq j$, is shattered. By the same logic as above, this can only be due to a witness $w_{uv}^{ij} \in W_2$ and therefore $uv \in E(H)$. We conclude that indeed $u_1, \ldots, u_k$ induce a complete graph in H, as claimed.

Finally, we need to determine the degeneracy of G. Consider the following elimination sequence: We first delete all of $\{w_0\} \cup W_1 \cup W_2$, all of which have degree at most two. Note now that all vertices in $\mathcal{V}$ have at most $|A| < 2^k$ neighbours in A, so we delete $\mathcal{V}$ and then A. In total, the maximum degree we encountered in this deletion sequence is $< 2^k$.

Assume we could solve Graph VC-Dimension in time $f(d)n^{o(\log d)}$. In the above reduction the degeneracy of the constructed graph is $d < 2^k$, thus this running time for Graph VC-Dimension would imply a running time of

$$f(d) \cdot n^{o(\log d)} = f(2^k) \cdot n^{o(\log 2^k)} = f(2^k) \cdot n^{o(k)}$$

for k-Clique. We conclude that VC-dimension parameterized by the degeneracy of the input graph cannot be solved in time $f(d)n^{o(\log d)}$ unless all problems in SNP can be solved in subexponential time. □

We note that Lemma 4 allows us to approximate the VC-dimension of degenerate graphs.

Theorem 5. *Let G be a d-degenerate graph on n vertices. Then for any $0 < \varepsilon \leqslant 1$ we can approximate the VC-dimension of G in time $O(d2^d(2n)^{\lceil \varepsilon(1+\log d)\rceil})$ within a factor of ε.*

Proof: We first compute a d-degenerate ordering $\mathbb{G}$ of G in time $O(dn)$ with sorted neighbourhoods. Let $U_t = (S, W)$ be the largest shattered set in G and let $\mathbb{U}_t$ be its ordering in $\mathbb{G}$. We further prepare the use of Theorem 2 by computing the necessary data structure in time $O(d2^d n)$.

Let $c \coloneqq \lceil \varepsilon(1 + \log d)\rceil$. The algorithm now iterates over all $C \subseteq V(G)$ of size c and searches the left-neighbourhood $L \coloneqq N^-[C]$ for a shattered set by first computing a subset dictionary Q_L in time $O(d2^d)$ and then finding the largest shattered subset $S \subseteq L$ by brute-force in time $O(|L|2^{|L|}) = O(cd2^{cd})$.

We claim that this simple algorithm computes the claimed approximation of the VC-dimension. By Lemma 4 we either have that $t \leqslant \log d + 1$ or that $\mathbb{U}_t$ can be left-covered by $\log d + 1$ witness vertices. In the first case, our algorithm will trivially locate an ε-fraction of a maximal solution since it tests every set of size c. In the second case, the shattered set S of $\mathbb{U}_t$ is covered by the left-neighbourhood of witness vertices $w_1, \ldots, w_p \in W$ for $p \coloneqq \log d + 1$. Then by simple averaging, there exist c witnesses W' such that $|N^-[W'] \cap S| \geqslant c|S|/p = ct/(\log d + 1)$. Since the above algorithm will find the shattered set $N^-[W'] \cap S$ when inspecting the left-neighbourhood of W, we conclude that it will output at least a value of $ct/(\log d + 1)$. In either case the approximation factor is $\frac{c}{1+\log d} \geqslant \varepsilon$, as claimed. □

We would like to highlight the special case of $c = 1$ of the above theorem as it provides us with a linear-time approximation of the VC-dimension, which is probably a good starting point for practical applications:

Corollary 3. *Let G be a d-degenerate graph on n vertices. Then we can approximate the VC-dimension of G in time $O(d2^d n)$ within a factor of $\frac{1}{1+\log d}$.*

4.3 Approximating the ladder and semi-ladder index

We now give a factor-2 approximation algorithm for the ladder index in d-degenerate graphs. Note that the algorithm does not need to know t beforehand; If the largest ladder in G has size t, then the algorithm outputs a ladder of size at least $\lfloor t/2 \rfloor$. Before we proceed, we note that degenerate graphs cannot contain arbitrarily long ladders:

Observation 2. *If G is d-degenerate then G cannot contain a ladder of length $2d + 2$.*

Proof: Note that a ladder of length t contains a complete bipartite graph $K_{\lfloor t/2 \rfloor, \lfloor t/2 \rfloor}$, i.e. a subgraph of minimum degree $\lfloor t/2 \rfloor$. Therefore $t < 2d + 2$. □

Again we find that a direct application of Theorem 1 to ladder patterns does not yield a satisfying running time since $\mathrm{lc}(L_t) \approx t/2$. However, we can always left-cover a large portion of a ladder with only one vertex:

Observation 3. *Let (A, B) induce a ladder of length t in G. For every ordering $\mathbb{G}$ of G there exists a vertex $u \in A \cup B$ such that $|N^-(u) \cap (A \cup B)| \geqslant \lfloor t/2 \rfloor$.*

Proof: Let $A' \coloneqq (a_i)_{i \geqslant t/2}$ and $B' \coloneqq (b_i)_{i \leqslant t/2}$, then $G[A' \cup B']$ contains a biclique with partite sets A', B'. Let $u \in A' \cup B'$ be the largest vertex according to $<_{\mathbb{G}}$, then $N^-(u) \cap (A' \cup B')$ is either all of A' or all of B'. In either case the claim holds. □

Theorem 6. *Let G be a d-degenerate graph on n vertices and let t be its ladder-index. Then we can in time $O(d^2 8^d \cdot n)$ compute a ladder of size at least $\lfloor t/2 \rfloor$ in G.*

Proof: Note that the algorithm does not know t in advance; The parameter t is only used in the analysis to measure the quality of the returned ladder against an optimum solution. We start by computing a degeneracy ordering $\mathbb{G}$ of G and initialize the data structure R as per Lemma 1 in time $O(2^d n)$.

Let (A, B) induce a ladder of maximum size t in G, by Observation 2 we have that $t \leqslant 2d+1$. By Observation 3, there exists a vertex $u \in A \cup B$ such that $N^-(u)$ contains either $A' \coloneqq (a_i)_{i \geqslant t/2}$ or $B' \coloneqq (b_i)_{i \geqslant t/2}$. Without loss of generality assume $A' \subseteq N^-(u)$ and let $k \coloneqq |A'|$. We guess u in $O(n)$ time and $A' \subseteq N^-(u)$ in time $O(2^d)$. To verify that A' can be completed into a ladder, we compute the data structure $Q_{A'}$ in time $O(k2^k + dk)$ using Lemma 2.

Finally, we verify that there exists a sequence of subsets $A'_1 \subset A'_2 \subset \ldots \subset A'_k = A'$ where $Q_{A'}[A'_i] > 0$ for all $i \in [k]$; as each lookup in $Q_{A'}$ has cost equal to the size of the query set this will take time proportional to $\sum_{i=0}^{k} i\binom{k}{i} = k2^{k-1}$ in the worst case (where we have to query all subsets of A' before finding the sequence). Since $k \coloneqq \lfloor t/2 \rfloor$, the total running time of this algorithm is

$$O(2^d n) + O\Big(2^d n \cdot (k2^k + dk) \cdot k2^{k-1}\Big) = O\Big(2^d k 2^k (k2^k + dk) \cdot n\Big).$$

We can simplify this expression further by using that $k \leqslant d$ which leads us to the claimed running time of $O(d^2 8^d \cdot n)$ □

It is clear that the proof of Theorem 6 applies to semi-ladders as well, since semi-ladders of length t contain the same biclique, while the remaining verification conditions are weaker.

Corollary 4. *Let G be a d-degenerate graph on n vertices, and let t be its semi-ladder index. Then in time $O(d^2 8^d \cdot n)$ one can compute a semi-ladder of size at least $\lfloor t/2 \rfloor$ in G.*

4.4 Computing the c-closure

The final complexity measure we implement is a recently introduced complexity measure from social network theory, the c-closure [15] of a graph G. The c-closure fits naturally into our algorithmic framework: A graph has c-closure at least $c+1$ if and only if it contains a (not necessarily induced) copy of $K_{2,c}$ in which the two vertices on the side of size 2 are non-adjacent. Thus, from the perspective of our algorithms, computing the c-closure is closely related to searching for biclique-like patterns, except that one additionally has to enforce a non-edge between the two distinguished vertices.

The c-closure is defined as

$$\mathsf{cc}(G) = \max_{uv \notin E} |N(u) \cap N(v)| + 1,$$

i.e., the largest common neighbourhood of two non-adjacent vertices *plus one*. The "plus one" part stems from the contrapositive view in social network theory where we frame it as "any pair of vertices with $\mathsf{cc}(G)$ common neighbours are themselves neighbours". For us, it will be more convenient with the definition

$$\kappa(G) = \max_{uv \notin E} |N(u) \cap N(v)|,$$

which is the value our algorithm computes, after which we can return $\mathsf{cc}(G) = \kappa(G)+1$. The c-closure is a generalization of the *triadic closure*: A graph is triadic closed if the common neighbourhood of two non-adjacent vertices is empty. In other words, cluster graphs are 1-closed. The best known running time for computing the c-closure of a graph is $O(cn^2 + m^{3/2})$ [19]. We proceed to give

two algorithms, running in time $O(nd^3 + n^2 d)$, and in linear time $O(n \cdot d(d^2 + 2^d))$. These two algorithms have the interesting property that if the c-closure is at least the degeneracy—which we find to happen often in real-world (non-bipartite) networks—they both run in FPT-in-P linear time $O(nd^3)$.

Let R be as in Lemma 1, except we only perform the update step for $X = \{u, v\}$, i.e. X consists of exactly two vertices. We call this the parametric version of the data structure, R_2:

Initialize R_2 as an empty trie storing integers
for $x \in \mathbb{G}$ **do**
  for $X = \{u, v\} \in N^-(x)$ **do**
    $R_2[X] \leftarrow R_2[X] + 1$ `// Non-existing keys are treated as zero`
return R_2

It is evident that the running time of the above algorithm is $O(n \cdot d^2)$, and subsequently, we can query R_2 for the value of $|N^+(u) \cap N^+(v)|$ for any pair u and v. We note here that if we need to look at more than pairs of neighbours, e.g. t many neighbours, we can use R_t.

Let us describe how to compute the c-closure of $\mathbb{G}$, and let $C(u, v) = N(u) \cap N(v)$ denote the *common neighbourhood* of u and v. We want to compute, for every pair $uv \notin E$, the values of $|C(u, v)| = |N(u) \cap N(v)|$. We can skip all pairs u, v with $|C(u, v)| = 0$, hence we can safely assume that there is some vertex $x \in C(u, v)$, and we now let x denote the *rightmost* vertex in $C(u, v)$. Consider the ordering of u, v, and x in $\mathbb{G}$, where u comes before v. We distinguish three cases depending on where x appears relative to u and v. Recall that our algorithm computes (and returns) the value $\kappa(G) = \mathtt{cc}(G) - 1$.

Case 1: $u < v < x$. We determine the size of $C(u, v)$ by counting the part of $C(u, v)$ that comes before v (ℓ) and the part that comes after v (r):

$\kappa \leftarrow 0$
for $x \in \mathbb{G}$ **do**
  for $u, v \in N^-(x)$, *with* $uv \notin E$ **do**
    $r \leftarrow R_2[\{u, v\}]$
    $\ell \leftarrow |N^-(v) \cap N(u)|$
    $\kappa \leftarrow \max\{\kappa, r + \ell\}$
return κ

The algorithm runs in time $O(nd^3)$, which also dominates the computation of R_2. Hence the total running time for the initialization and checking Case 1 is $O(nd^3)$. Note that the complexity of checking $uv \notin E$ can be done while computing $|N^-(v) \cap N(u)|$: If $u \in N^-(v)$, we do not update κ. The same observation holds for the following cases as well.

Case 2: $u < x < v$. Since x is the rightmost vertex of $C(u, v)$, this means that $C(u, v) \subseteq N^-(v)$, and we simply run the following algorithm:

$\kappa \leftarrow 0$
for $v \in \mathbb{G}$ **do**
  for $x \in N^-(v)$ **do**
    for $u \in N^-(x)$ *with* $uv \notin E$ **do**
      $\kappa \leftarrow \max\{\kappa, |N^-(v) \cap N(u)|\}$
return κ

The running time is again $O(n \cdot d^3)$. This leaves us with the final case.

Case 3: $x < u < v$. This step is different from the two others, in that the running time is $O(n^2 \cdot d)$ and the only part of computing the c-closure which takes quadratic time. However, computing this case is only necessary when $\kappa \leqslant d$, something that rarely happens, according to our experiments:

$\kappa \leftarrow 0$
for $v \in \mathbb{G}$ **do**
  for $u \in \mathbb{G}$ *with* $uv \notin E$ **do**
    $\kappa \leftarrow \max\{\kappa,\ |N^-(u) \cap N^-(v)|\}$
return κ

We only enter Case 3 when $\kappa \leqslant d$, i.e., the c-closure is *smaller* than the degeneracy. The reason is that the assignment

$$\kappa \leftarrow \max\left\{\kappa,\ \left|N^-(u) \cap N^-(v)\right|\right\}$$

can only update κ to a size at most d (since $|N^-(\bullet)| \leqslant d$).

It is also clear that this can take effect only when the κ computed *up until this point* is less than d. Our experiments show that in practice, only very specific real-world networks have $\kappa \leqslant d$ (more than 95% of the networks in our benchmark had $\kappa > d$, which also shows that a small degeneracy is more realistic than a small c-closure, the exceptions being bipartite graphs).

Theorem 7. *There is an algorithm computing the c-closure of a d-degenerate n-vertex graph in time $O(nd^3 + n^2 d)$. When $d \leqslant \kappa$, the algorithm runs in time $O(nd^3)$.*

Proof: The correctness is evident from the three cases above. Let $\mathbb{G}$ be any ordered graph, and u and v be a pair (with $u <_{\mathbb{G}} v$) which maximises $|C(u,v)|$, with $uv \notin E(\mathbb{G})$. Let x be the rightmost (according to $\mathbb{G}$) vertex in $C(u,v)$. Then u, v, and x appear in one of the three orders corresponding to the three cases above, hence we will consider $|C(u,v)|$. □

Theorem 8. *There is an algorithm computing the c-closure of a d-degenerate n-vertex graph in time $O(n \cdot d(d^2 + d2^d))$. When $d \leqslant \kappa$, the algorithm runs in time $O(nd^3)$.*

Proof: The two cases Case 1 and Case 2 are as in Theorem 7; They are both performed in time $O(nd^3)$. However, we replace Case 3 above with the following procedure using the data structure R from Lemma 1 with one additional piece of information: a boolean flag $F[X]$ that tells us whether there are two vertices u and v with $uv \notin E$ such that $X \subseteq N^-(u) \cap N^-(v)$.

Initialize R as an empty trie storing integers
Initialize Y as an empty trie storing a list of vertices
Initialize F as an empty trie storing booleans
for $v \in \mathbb{G}$ **do**
  for $X \subseteq N^-(v)$ **do**
    $R[X] \leftarrow R[X] + 1$
    if $F[X] =$ `false` **then**
      for $u \in Y[X]$ **do**
        if $u \notin N^-(v)$ **then**
          $F[X] \leftarrow$ `true`
          $Y[X] \leftarrow \{u\}$ `// Y[X] ← {v, u} after this loop`
          break
      $Y[X] \leftarrow Y[X] \cup \{v\}$
return R with only keys X such that $F[X]$ is `true`

While the flag for X is false, we maintain a set $Y[X]$ such that $u \in Y[X]$ have $X \subseteq N^-(u)$. Now, when we increase $R[X]$ for v and the flag is false, we go through $N^-(v) \cap Y[X]$ and see if v is adjacent to all of $Y[X]$. If it is, we add v to $Y[X]$, otherwise we set the flag to true, and store a witness in $Y[X]$. Notice that $Y[X]$ will be a clique unless $F[X]$ is set to true, and that $|Y[X]| + |X| \leqslant d$ since $Y[X] \cup X \subseteq N^-(v)$. After construction of R with our flags, we find the largest possible X in R with a true flag:

$$\kappa(G) = \max_{X \in R} \{|X| \mid F[X] = \texttt{true}\}.$$

□

5 Implementation and experiments

Based on the above theoretical ideas, we implemented all algorithms in Rust[5] to compute the VC-dimension, find the largest biclique, co-matchings (within an additive error of 1, since our implementation uses only the left-neighbourhood of a single vertex) and ladder (within a factor 2). The last three algorithms all simply check the left-neighbourhood for the respective structure. Aside from optimisations of the involved data structures we will not describe these algorithms in further detail.

We observe that for practical purposes the data structure R can be computed progressively: if we know that our algorithm currently only needs to compute Q_S from R (as per Lemma 2) with $|S| = k$ ($k \leqslant d$), then it is enough to only count sets of size $\leqslant k$ in R. We can achieve this in time $O(\binom{d}{k}n)$, which is far preferable to using $O(2^d n)$ time to insert all left-neighbourhood subsets into R. If k remains much smaller than d, this improves our running time and space consumption substantially.

The second important optimisation regards subset dictionaries. While tries are useful in our theoretical analysis, in practice we opted to use bitsets for the data structures Q_S, as their universe S can be assumed to be small. Bitsets also allow for a very concise and fast implementation of the fast Möbius inversion, which needs to happen very frequently inside the hot loop of the search algorithms.

The algorithm to compute the VC-dimension includes a few simple optimisations that vastly improved its performance. Note that if we are currently searching for a shattered set of size k, then a candidate vertex for a shattered set of size k must have at least $\binom{k-1}{i-1}$ neighbours of degree at least i, for $1 \leqslant i \leqslant k-1$. Our algorithm recomputes the set of remaining candidates each time it finds a larger shattered set. The (progressive) computation of the data structure R can then also be restricted to only those left-neighbourhood subsets which only contain candidate vertices.

Accordingly, the algorithm performs well if it finds large shattered sets fast. To that end, it first only looks at k-subsets of left-neighbourhoods of single vertices. Once that search is exhausted, it considers left-neighbourhoods of pairs, then triplets, *etc.* up to $\lceil \log d + 1 \rceil$ vertices (as per Lemma 4). As this search is very expensive once we need to consider the joint left-neighbourhood of several vertices, the algorithm estimates the work needed and compares it against simply brute-forcing all k-subsets of the remaining candidates. Since the number of candidates shrinks quite quickly in practice, the algorithm usually concludes with such a final exhaustive search.

[5]Source code available under github.com/microgravitas/mantis-shrimp with the graph library available at github.com/microgravitas/graphbench. The graph library uses a `u32` (int) for representing vertices, with the left neighbours stored in a vector `Vec<u32>`, and the right neighbours in an `FxHashMap` from the `Fx Hash` library.

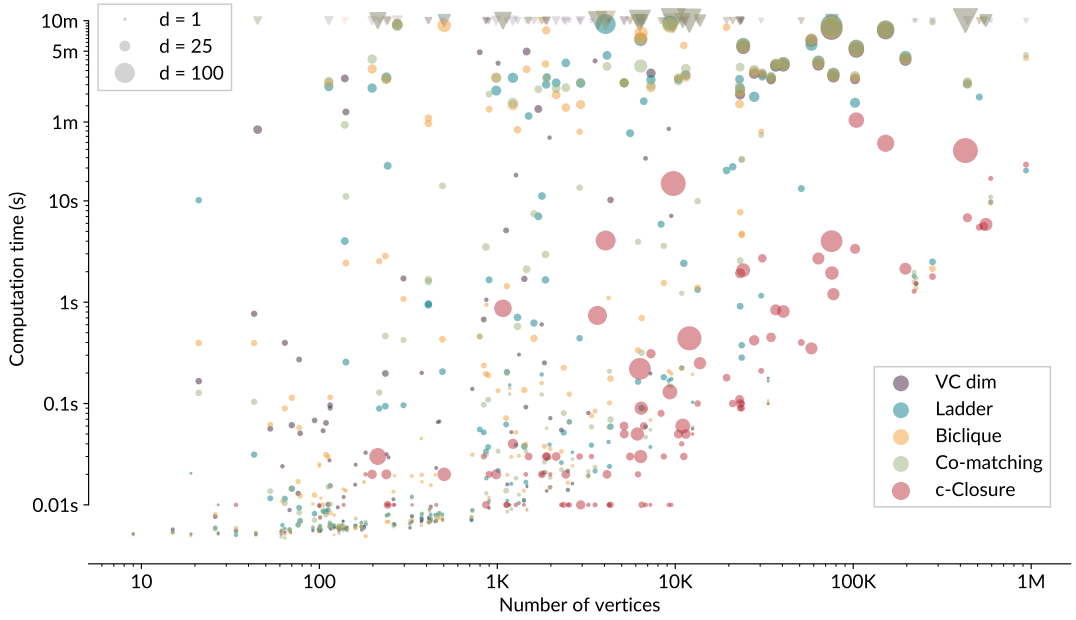


Figure 1: Running times of all five algorithms on a collection of 206 networks. The size of the circles indicates the degeneracy of the networks, triangles indicate that program timed out on the network after 10 minutes.

5.1 Results

We implemented all five algorithms in Rust and tested them on a diverse collection of 206 networks[6], using a PC with an AMD Ryzen 3 2200G CPU and 24 GB RAM. The primary goal of our experiments was to verify that the data structures and algorithms in this paper could be of practical use, therefore we ran each algorithm only once per network[7] and timed out after 10 minutes.

Of all the five measures, computing the VC-dimension is, unsurprisingly, the most computationally challenging and the program timed out or ran out of memory for networks larger than a few tens of thousands of nodes or of degeneracy higher than 24. The broad summary of the results is as follows (Figure 1 visualises these results in more detail):

Statistics	Completed	Max size (n)	Max degeneracy
VC-dimension	126	33266 (`BioGrid-Chemicals`)	24 (`wafa-eies`)
Biclique	176	935591 (`teams`)	191 (`BioGrid-All`)
Co-matching	179	935591 (`teams`)	255 (`dogster_friendships`)
Ladder index	187	935591 (`teams`)	191 (`BioGrid-All`)
c-Closure	206 (all)	935591 (`teams`)	255 (`dogster_friendships`)

We are also interested in typical values of the VC-dimension of networks and how it compares to the degeneracy. This topic deserves a deeper investigation, but we can report some preliminary

[6]Our network corpus with statistics is available at github.com/microgravitas/network-corpus.
[7]The variance in running times is on the orders of seconds.

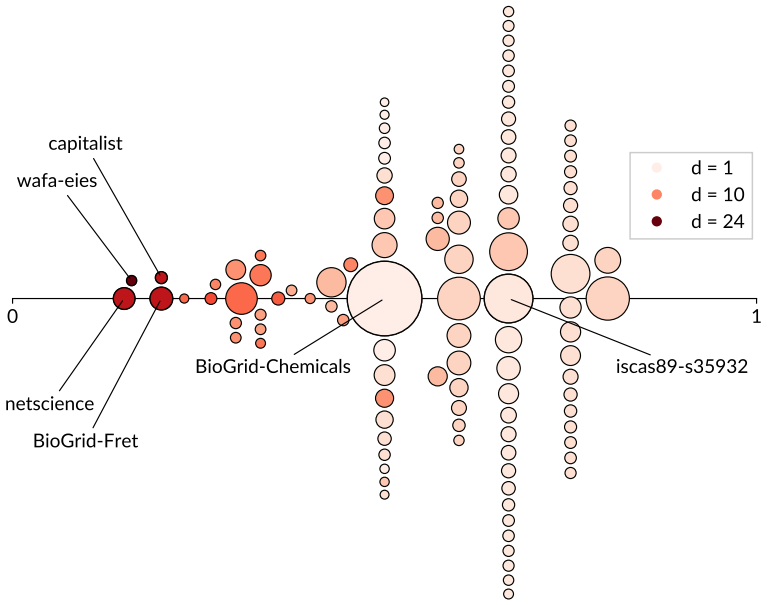


Figure 2: VC-dimension of networks normalized by their degeneracy + 1. Networks whose VC-dimension matches the upper bound given by the degeneracy would be positioned to the very right at the 1 mark, networks whose VC-dimension is significantly smaller tend to the left. The colour indicates the degeneracy with a lighter colour meaning smaller and darker meaning larger degeneracy. We observe that denser networks tend to the left (VC-dimension much smaller than the theoretical upper bound) and sparser networks to the right (VC-dimension closer to the theoretical upper bound). The size of the circles represents network sizes and there is no clear trend regarding this measure.

results here for those networks where our program terminated before the timeout. In Figure 2 we normalised the VC-dimension by the degeneracy-plus-one, so values close to one indicate that the VC-dimension is on the order of the degeneracy while values close to zero indicate that it is much smaller than the degeneracy. We see a clear tendency that networks with larger degeneracy tend towards zero, which we interpret as the VC-dimension "growing slower" than the degeneracy in typical networks.

6 Conclusion

On the theoretical side, we outlined a general bipartite pattern-finding and -counting algorithm in degenerate graphs. Its running time crucially depends on two complexity measures of patterns, namely the left-covering number and the half-ordering asymmetry. These general algorithms can be further improved for specific patterns, which we exemplify for shattered set, ladder, co-matching, and biclique patterns. Our results also include improved running times when the input graphs are of bounded degeneracy.

On the experimental side, we demonstrate that this style of algorithm is feasible and practical for computation on real-world networks, which often exhibit low degeneracy. The experiments also suggest that the VC-dimension of networks tends to be a very small parameter, which makes it an interesting target for the development of fast algorithms that exploit low VC-dimension.

Our algorithm for computing the c-closure of a graph performs quite well, spending less than a second on 90% of the networks, and solves all but one in less than a minute. A related complexity measure for social networks is the *weak c-closure*, which we do not know how to solve efficiently on degenerate graphs. We leave this as an open problem.

References

[1] Suman K. Bera, Noujan Pashanasangi, and C. Seshadhri. Linear time subgraph counting, graph degeneracy, and the chasm at size six. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151, pages 38:1–38:20. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. doi:10.4230/LIPIcs.ITCS.2020.38.

[2] Andreas Björklund, Thore Husfeldt, and Mikko Koivisto. Set Partitioning via Inclusion-Exclusion. *SIAM Journal on Computing*, 39(2):546–563, 2009. doi:10.1137/070683933.

[3] Marco Bressan and Marc Roth. Exact and Approximate Pattern Counting in Degenerate Graphs: New Algorithms, Hardness Results, and Complexity Dichotomies. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 276–285, 2022. doi:10.1109/FOCS52979.2021.00036.

[4] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. On the computational hardness based on linear FPT-reductions. *Journal of Combinatorial Optimization*, 11(2):231–247, 2006. doi:10.1007/s10878-006-7137-6.

[5] Jianer Chen, Xiuzhen Huang, Iyad A. Kanj, and Ge Xia. Strong computational lower bounds via parameterized complexity. *Journal of Computer and System Sciences*, 72(8):1346–1367, 2006. doi:10.1016/j.jcss.2006.04.007.

[6] Norishige Chiba and Takao Nishizeki. Arboricity and Subgraph Listing Algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985. doi:10.1137/0214017.

[7] Erik D. Demaine, Felix Reidl, Peter Rossmanith, Fernando Sánchez Villaamil, Somnath Sikdar, and Blair D. Sullivan. Structural sparsity of complex networks: Bounded expansion in random models and real-world graphs. *Journal of Computer and System Sciences*, 105:199–241, 2019. doi:10.1016/j.jcss.2019.05.004.

[8] Rodney G. Downey, Patricia A. Evans, and Michael R. Fellows. Parameterized learning complexity. In *Proceedings of the sixth annual conference on Computational learning theory*, pages 51–57, 1993.

[9] Pål Grønås Drange, Patrick Greaves, Irene Muzi, and Felix Reidl. Computing complexity measures of degenerate graphs. In *18th International Symposium on Parameterized and Exact Computation (IPEC 2023)*, volume 285 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 14:1–14:21, Dagstuhl, Germany, 2023. Schloss Dagstuhl — Leibniz-Zentrum für Informatik. doi:10.4230/LIPIcs.IPEC.2023.14.

[10] Robin I. M. Dunbar. The social brain hypothesis. *Evolutionary Anthropology: Issues, News, and Reviews*, 6(5):178–190, 1998. doi:10.1002/(SICI)1520-6505(1998)6:5<178::AID-EVAN5>3.0.CO;2-8.

[11] Eduard Eiben, Robert Ganian, Thekla Hamm, Lars Jaffke, and O-joung Kwon. A Unifying Framework for Characterizing and Computing Width Measures. In *13th Innovations in Theoretical Computer Science Conference (ITCS 2022)*, volume 215, pages 63:1–63:23. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ITCS.2022.63.

[12] Kord Eickmeyer, Archontia C. Giannopoulou, Stephan Kreutzer, O-joung Kwon, Michał Pilipczuk, Roman Rabinovich, and Sebastian Siebertz. Neighborhood Complexity and Kernelization for Nowhere Dense Classes of Graphs. In *44th International Colloquium on Automata, Languages, and Programming (ICALP 2017)*, volume 80, pages 63:1–63:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.ICALP.2017.63.

[13] David Eppstein, Maarten Löffler, and Darren Strash. Listing all maximal cliques in large sparse real-world graphs. *ACM Journal of Experimental Algorithmics*, 18:3.1:3.1–3.1:3.21, 2013. doi:10.1145/2543629.

[14] Grzegorz Fabiański, Michał Pilipczuk, Sebastian Siebertz, and Szymon Toruńczyk. Progressive algorithms for domination and independence. In *36th International Symposium on Theoretical Aspects of Computer Science (STACS 2019)*, volume 126, pages 27:1–27:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.STACS.2019.27.

[15] Jacob Fox, Tim Roughgarden, C. Seshadhri, Fan Wei, and Nicole Wein. Finding cliques in social networks: A new distribution-free model. *SIAM Journal on Computing*, 49(2):448–464, 2020. doi:10.1137/18M1210459.

[16] Archontia C. Giannopoulou, George B. Mertzios, and Rolf Niedermeier. Polynomial fixed-parameter algorithms: A case study for longest path on interval graphs. *Theoretical Computer Science*, 689:67–95, 2017. doi:10.1016/j.tcs.2017.05.017.

[17] Frank Kammer. A linear-time kernelization for the rooted k -leaf outbranching problem. *Discrete Applied Mathematics*, 193:126–138, 2015. doi:10.1016/j.dam.2015.04.028.

[18] Donald E. Knuth. *Art of computer programming, volume 2: Seminumerical algorithms*. Addison–Wesley Professional, 2014.

[19] Tomohiro Koana and André Nichterlein. Detecting and enumerating small induced subgraphs in c-closed graphs. *Discrete Applied Mathematics*, 302:198–207, 2021. doi:10.1016/j.dam.2021.06.019.

[20] David W. Matula and Leland L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *Journal of the ACM*, 30(3):417–427, 1983. doi:10.1145/2402.322385.

[21] Christos H. Papadimitriou and Mihalis Yannakakis. On limited nondeterminism and the complexity of the VC dimension. *Journal of Computer and System Sciences*, 53(2):161–170, 1996.

[22] Anatol Rapoport. Spread of information through a population with socio-structural bias: I. Assumption of transitivity. *The bulletin of mathematical biophysics*, 15(4):523–533, 1953. doi:10.1007/BF02476440.

[23] Robert Sedgewick and Kevin Wayne. *Algorithms, 4th edition*. Addison–Wesley, 2011.

[24] Frank Yates. The design and analysis of factorial experiments. *Imperial Bureau of Soil Science*, 1937.

A Complete results

For space reasons, some of the network names are abbreviated below.

						Running-time					Statistics				
Network	n	m	$\lceil \bar{d} \rceil$	deg	Δ	VC-dim	biclique	crown	ladder	c-c	VC-dim	biclique	crown	ladder	c-c
AS-oregon-1	11 174	23 409	5	17	2389	—	342.08	175.43	2.42	0.03	[5, 18]	12	[13, 14]	[17, 35]	75
AS-oregon-2	11 461	32 730	6	31	2432	—	167.57	201.43	174.27	0.05	[6, 32]	[7, 31]	[7, 32]	[7, 63]	118
BG-AC-Luminesc..	1840	2312	3	6	376	0.25	0.04	0.03	0.02	0.00	4	5	7	[6, 13]	38
BG-AC-Ms	40 495	321 887	16	58	2217	217.75	224.87	227.38	227.77	0.81	[4, 59]	[4, 58]	[4, 59]	[4, 117]	843
BG-AC-Rna	13 765	42 815	7	54	3572	—	—	—	—	0.25	[4, 55]	[5, 54]	[5, 55]	[5, 109]	1630
BG-AC-Western	21 028	64 046	7	17	535	—	—	—	21.70	0.10	[5, 18]	[7, 17]	[9, 18]	[17, 35]	93
BG-All	75 550	1 316 843	35	191	3620	496.99	514.30	519.60	540.54	4.00	[2, 192]	[2, 191]	[2, 192]	[2, 383]	1792
BG-ATC	10 417	47 916	10	26	1341	—	163.89	152.58	159.18	0.05	[6, 27]	[8, 26]	[8, 27]	[8, 53]	385
BG-Biochemical..	8620	17 746	5	11	427	—	1.55	3.59	0.18	0.03	[5, 12]	8	[7, 8]	[11, 23]	182
BG-Bos-Taurus	454	424	2	3	27	0.02	0.01	0.01	0.01	0.01	2	3	[3, 4]	[3, 7]	7
BG-C.-Elegans	6394	23 646	8	64	522	—	399.05	214.18	390.57	0.03	[4, 65]	[6, 64]	[5, 65]	[6, 129]	82
BG-C.-Albicans..	1121	1609	3	9	427	5.11	0.07	0.01	0.06	0.01	3	6	10	[9, 19]	64
BG-Canis-Famil..	143	125	2	2	90	0.01	0.01	0.01	0.01	0.00	2	2	3	[2, 5]	3
BG-Chemicals	33 266	28 093	2	1	413	0.18	0.10	0.11	0.17	0.10	1	[0, 1]	2	[0, 3]	2
BG-Co-Crystal-..	2291	2021	2	5	92	0.05	0.03	0.02	0.03	0.01	3	3	6	[3, 7]	6
BG-Co-Fraction..	11 017	56 354	11	83	187	—	—	—	—	0.06	[4, 84]	[5, 83]	[5, 84]	[5, 167]	84
BG-Co-Localiza..	3543	4452	3	6	63	240.56	0.08	0.02	0.02	0.01	3	5	7	[6, 13]	25
BG-Co-Purifica..	4326	5970	3	12	1972	10.21	0.18	0.07	0.06	0.01	4	6	13	[12, 25]	71
BG-Cricetulus-..	69	57	2	1	30	0.01	0.01	0.01	0.01	0.00	1	[0, 1]	2	[0, 3]	2
BG-Danio-Rerio	261	266	2	3	61	0.01	0.01	0.01	0.01	0.01	2	2	4	[3, 7]	4
BG-D.-Discoide..	27	20	2	1	4	0.01	0.01	0.01	0.01	0.01	1	[0, 1]	2	[0, 3]	2
BG-D.-Growth-D..	1447	2193	3	5	213	0.09	0.04	0.03	0.02	0.02	4	4	6	[5, 11]	70
BG-D.-Lethality	1776	2289	3	4	392	0.60	0.16	0.12	0.19	0.02	3	4	[4, 5]	[4, 9]	33
BG-D.-Rescue	3380	6444	4	7	75	—	0.08	0.06	0.04	0.03	[4, 8]	6	[7, 8]	[7, 15]	35
BG-D.-M'gaster	9330	60 556	13	83	303	—	538.71	553.50	566.57	0.13	[4, 84]	[5, 83]	[5, 84]	[5, 167]	115
BG-E.-Nidulans..	64	62	2	2	44	0.01	0.01	0.00	0.01	0.00	2	2	3	[2, 5]	3
BG-E.-Coli-K12Mg	1273	1889	3	5	58	17.95	0.05	0.02	0.04	0.01	3	4	6	[5, 11]	13
BG-E.-Coli-K12W	4063	181 620	90	159	1187	—	—	—	556.39	4.06	[3, 160]	[3, 159]	[3, 160]	[3, 319]	442
BG-Far-Western	1199	1089	2	3	60	0.05	0.03	0.01	0.02	0.00	3	3	4	[3, 7]	11
BG-Fret	1700	2395	3	19	51	80.71	—	126.45	7.04	0.00	4	[11, 19]	[17, 18]	[19, 39]	38
BG-Gallus-Gallus	413	436	3	4	110	0.01	0.01	0.01	0.01	0.00	3	3	[4, 5]	[4, 9]	22
BG-Glycine-Max	44	39	2	2	13	0.01	0.01	0.01	0.01	0.00	2	2	3	[2, 5]	3
BG-Hepatitus-C..	136	134	2	1	133	0.01	0.01	0.01	0.01	0.00	1	[0, 1]	2	[0, 3]	2
BG-Homo-Sapiens	24 093	369 767	31	71	2882	329.05	338.13	343.83	346.77	2.07	[3, 72]	[3, 71]	[3, 72]	[3, 143]	802
BG-HSV-1	178	208	3	3	40	0.01	0.02	0.01	0.01	0.00	3	3	4	[3, 7]	11
BG-HSV-4	323	326	2	2	154	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	6
BG-HSV-5	121	107	2	1	27	0.01	0.01	0.01	0.01	0.00	1	[0, 1]	2	[0, 3]	2
BG-HSV-8	716	691	2	3	119	0.01	0.01	0.01	0.01	0.00	2	3	[3, 4]	[3, 7]	6
BG-HIV-1	1138	1319	3	3	324	0.02	0.02	0.01	0.01	0.00	3	3	4	[3, 7]	43
BG-HIV-2	19	15	2	1	6	0.01	0.00	0.01	0.02	0.00	1	[0, 1]	2	[0, 3]	2
BG-HPV-16	173	186	3	2	93	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	18
Cannes2013	438 089	835 892	4	27	15 169	—	144.70	149.74	142.22	6.81	[5, 28]	[6, 27]	[6, 28]	[6, 55]	3168

Continued on next page

						Running-time					Statistics				
Network	n	m	$\lceil \bar{d} \rceil$	deg	Δ	VC-dim	biclique	crown	ladder	c-c	VC-dim	biclique	crown	ladder	c-c
CoW-interstate	182	319	4	4	25	0.03	0.00	0.01	0.01	0.02	3	4	[3, 4]	[4, 9]	13
DNC-emails	1866	4384	5	17	402	—	224.12	2.95	1.66	0.00	[5, 18]	10	[16, 17]	[17, 35]	74
EU-email-core	986	16 064	33	34	345	—	164.82	163.41	122.48	0.02	[5, 35]	[6, 34]	[6, 35]	[6, 69]	162
JDK-dependency	6434	53 658	17	65	5923	—	453.02	—	—	0.09	[4, 66]	[5, 65]	[5, 66]	[5, 131]	707
JUNG-javax	6120	50 290	17	65	5655	—	—	—	—	0.05	[4, 66]	[5, 65]	[5, 66]	[5, 131]	619
NYClimateMarch..	102 378	327 080	7	34	14 687	158.65	165.32	170.79	92.84	3.36	[5, 35]	[5, 34]	[5, 35]	[4, 69]	1037
NZ-legal	2141	15 739	15	25	429	—	110.96	128.55	146.23	0.03	[6, 26]	[7, 25]	[7, 26]	[7, 51]	129
Noord-terror-loc	127	190	3	3	18	0.01	0.01	0.01	0.01	0.00	3	3	[3, 4]	[3, 7]	7
Noord-terror-org	129	181	3	3	21	0.01	0.01	0.01	0.01	0.00	3	3	[3, 4]	[3, 7]	11
Noord-terror-rel	70	251	8	11	28	0.06	0.11	0.01	0.01	0.00	4	6	12	[11, 23]	11
ODLIS	2900	16 377	12	12	592	—	48.04	13.49	0.44	0.03	[5, 13]	6	[9, 10]	[12, 25]	86
Opsahl-forum	899	7036	16	14	128	—	80.31	113.04	1.66	0.02	[5, 15]	5	[6, 7]	[14, 29]	38
Opsahl-socnet	1899	13 838	15	20	255	—	—	—	166.27	0.03	[6, 21]	[7, 20]	[8, 21]	[20, 41]	112
StackOverflow-..	115	245	5	6	16	0.09	0.01	0.01	0.01	0.00	3	4	7	[6, 13]	7
Y2H-union	1966	2705	3	4	89	42.11	0.01	0.03	0.04	0.00	3	4	5	[4, 9]	30
Yeast	2361	7182	7	10	66	—	0.23	0.08	0.05	0.01	[4, 11]	7	[9, 10]	[10, 21]	22
actor-movies	511 463	1 470 404	6	14	646	—	—	—	105.76	5.49	[5, 15]	[7, 14]	[6, 15]	[14, 29]	217
advogato	5155	39 285	16	25	803	—	145.81	146.46	146.33	0.06	[5, 26]	[6, 25]	[6, 26]	[6, 51]	216
airlines	235	1297	11	13	130	0.20	2.85	0.46	0.09	0.00	5	8	[11, 12]	[13, 27]	43
american-revol..	141	160	3	3	59	0.01	0.01	0.01	0.01	0.00	3	3	[2, 3]	[3, 7]	10
as-22july06	22 963	48 436	5	25	2390	—	90.18	137.08	158.60	0.11	[6, 26]	[8, 25]	[10, 26]	[10, 51]	218
as20000102	6474	12 572	4	12	1458	—	0.70	0.32	0.09	0.01	[5, 13]	9	[10, 11]	[12, 25]	43
autobahn	374	478	3	2	5	0.01	0.05	0.01	0.01	0.00	2	2	3	[2, 5]	3
bahamas	219 856	246 291	3	6	14 902	—	1.59	1.97	1.80	1.28	[3, 7]	6	[3, 4]	[6, 13]	58
bergen	53	272	11	9	32	0.06	0.06	0.01	0.01	0.00	4	6	[8, 9]	[9, 19]	13
bitcoin-otc-ne..	1606	3259	5	16	227	—	0.44	7.48	0.62	0.00	[4, 17]	16	[5, 6]	[16, 33]	39
bitcoin-otc-po..	5573	18 591	7	20	788	—	—	—	46.50	0.03	[6, 21]	[8, 20]	[10, 21]	[20, 41]	92
medulla	1781	8911	10	18	927	—	—	—	11.17	0.02	[5, 19]	[8, 18]	[8, 19]	[18, 37]	42
bn-mouse-retin..	1076	90 811	169	121	744	—	—	—	—	0.87	[3, 122]	[3, 121]	[3, 122]	[3, 243]	403
boards-gender-1m	4134	19 993	10	25	88	—	—	212.44	272.75	0.02	[4, 26]	[13, 25]	26	[25, 51]	20
boards-gender-2m	4220	5598	3	4	45	—	1.10	0.06	0.04	0.01	[3, 5]	4	[3, 4]	[4, 9]	21
ca-CondMat	23 133	93 439	9	25	279	—	—	208.73	—	0.10	[4, 26]	[14, 25]	26	[18, 51]	27
ca-HepPh	12 006	118 489	20	238	491	—	—	—	—	0.44	[3, 239]	[3, 238]	[3, 239]	[3, 477]	90
capitalist	139	1071	16	19	91	161.57	—	56.29	4.01	0.00	4	[12, 19]	[17, 18]	[19, 39]	35
celegans	297	2148	15	10	134	1.72	1.08	0.42	0.10	0.00	5	7	[8, 9]	[10, 21]	41
chess	7301	55 899	16	29	181	182.00	130.97	138.03	157.14	0.31	[6, 30]	[6, 29]	[6, 30]	[6, 59]	64
chicago	1467	1298	2	1	12	0.02	0.01	0.01	0.02	0.00	1	[0, 1]	2	[0, 3]	2
cit-HepPh	34 546	420 877	25	30	846	156.96	163.15	169.08	164.14	0.45	[4, 31]	[4, 30]	[4, 31]	[4, 61]	300
cit-HepTh	27 769	352 285	26	37	2468	180.07	189.90	194.29	106.40	0.42	[4, 38]	[4, 37]	[4, 38]	[3, 75]	1670
codeminer	724	1015	3	4	55	0.14	0.03	0.01	0.01	0.00	3	4	[4, 5]	[4, 9]	7
columbia-mobil..	863	4147	10	9	228	—	0.20	0.03	0.03	0.00	[4, 10]	6	10	[9, 19]	12
columbia-social	863	7724	18	18	545	—	—	3.51	—	0.01	[4, 19]	[12, 18]	[18, 19]	[14, 37]	54
cora-citation	23 166	89 157	8	13	377	—	7.72	2.57	0.91	0.09	[5, 14]	9	[10, 11]	[13, 27]	70
countries	592 414	624 402	3	6	110 602	—	9.49	10.88	9.75	16.68	[4, 7]	5	[4, 5]	[6, 13]	11 048
cpan-authors	839	2112	5	9	327	0.68	0.24	0.10	0.05	0.03	5	6	[8, 9]	[9, 19]	52
digg	30 398	86 312	6	9	285	—	48.32	44.61	1.16	0.21	[4, 10]	4	[5, 6]	[9, 19]	20
diseasome	1419	2738	4	11	84	1.70	0.14	0.04	0.04	0.00	4	6	12	[11, 23]	9
dogster-friend..	426 820	8 546 581	40	255	46 505	—	—	—	—	31.34	[1, 256]	[0, 255]	[1, 256]	[0, 511]	32 804

Continued on next page

Network	n	m	$\lceil \bar{d} \rceil$	deg	Δ	Running-time					Statistics				
						VC-dim	biclique	crown	ladder	c-c	VC-dim	biclique	crown	ladder	c-c
dolphins	62	159	6	4	12	0.02	0.01	0.01	0.01	0.00	3	3	5	[4, 9]	5
ecoli-transcript	423	578	3	3	74	0.01	0.02	0.01	0.01	0.00	3	3	[3, 4]	[3, 7]	11
edinburgh	23 132	297 094	26	34	1062	112.09	119.67	123.49	128.27	1.93	[3, 35]	[3, 34]	[3, 35]	[3, 69]	205
email-Enron	36 692	183 831	10	43	1383	213.38	219.83	223.53	219.47	0.84	[4, 44]	[4, 43]	[4, 44]	[4, 87]	187
euroroad	1174	1417	3	2	10	0.01	0.14	0.12	0.15	0.00	2	2	3	[2, 5]	4
eva-corporate	7253	6711	2	3	552	0.20	0.07	0.06	0.04	0.01	3	3	4	[3, 7]	7
exnet-water	1893	2416	3	2	10	0.01	0.02	0.06	0.03	0.00	2	2	3	[2, 5]	3
facebook-links	63 731	817 090	26	52	1098	221.29	229.98	233.75	237.25	2.70	[3, 53]	[3, 52]	[3, 53]	[3, 105]	233
foldoc	13 356	91 471	14	12	728	—	1.39	9.93	1.33	0.10	[5, 13]	12	[9, 10]	[12, 25]	63
foodweb-caribb..	492	3313	14	13	196	—	0.43	14.08	0.21	0.00	[4, 14]	12	[7, 8]	[13, 27]	155
foodweb-otago	141	832	12	14	45	75.41	2.43	11.06	0.26	0.00	4	12	[7, 8]	[14, 29]	36
football	115	613	11	8	12	0.10	0.12	0.01	0.02	0.00	4	4	9	[8, 17]	10
google+	23 628	39 194	4	12	2761	—	4.60	25.37	0.38	0.10	[6, 13]	9	[7, 8]	[12, 25]	490
gowalla	196 591	950 327	10	51	14 730	246.26	254.67	256.89	263.96	2.14	[3, 52]	[3, 51]	[3, 52]	[3, 103]	1132
haggle	274	2124	16	39	101	—	551.34	533.96	550.92	0.00	[4, 40]	[8, 39]	[8, 40]	[8, 79]	41
hex	331	930	6	3	6	0.01	0.02	0.01	0.01	0.00	3	2	[3, 4]	[3, 7]	3
hypertext-2009	113	2196	39	28	98	—	146.97	150.88	134.22	0.00	[5, 29]	[9, 28]	[9, 29]	[9, 57]	54
ia-email-univ	1133	5451	10	11	71	—	1.44	0.20	0.20	0.01	[4, 12]	6	12	[11, 23]	19
ia-infect-dublin	410	2765	14	17	50	—	65.59	1.60	0.94	0.00	[4, 18]	9	[16, 17]	[17, 35]	23
ia-reality	6809	7680	3	5	261	26.27	0.09	0.05	0.06	0.01	4	4	[5, 6]	[5, 11]	19
infectious	410	2765	14	17	50	—	58.09	1.67	0.96	0.00	[4, 18]	9	[16, 17]	[17, 35]	23
iscas89-s1196	377	537	3	2	16	0.01	0.02	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	3
iscas89-s1238	416	625	3	2	18	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	3
iscas89-s13207	2492	3406	3	4	37	0.01	0.02	0.02	0.02	0.01	4	4	[4, 5]	[4, 9]	31
iscas89-s1423	423	554	3	2	17	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	5
iscas89-s1488	463	779	4	3	53	0.07	0.01	0.01	0.01	0.00	3	3	[3, 4]	[3, 7]	11
iscas89-s1494	473	796	4	3	56	0.07	0.01	0.01	0.01	0.00	3	3	[3, 4]	[3, 7]	11
iscas89-s15850	3247	4004	3	4	25	0.08	0.02	0.02	0.02	0.01	3	4	[3, 4]	[4, 9]	16
iscas89-s208.1	61	67	3	2	8	0.01	0.01	0.00	0.01	0.00	2	2	[2, 3]	[2, 5]	3
iscas89-s27	9	8	2	1	3	0.01	0.01	0.01	0.01	0.00	1	[0, 1]	2	[0, 3]	2
iscas89-s298	92	131	3	2	11	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	6
iscas89-s344	100	122	3	2	9	0.01	0.02	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	5
iscas89-s349	102	127	3	2	9	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	5
iscas89-s35932	12 515	15 961	3	2	1440	0.06	0.19	0.06	0.10	0.05	2	[0, 1]	3	[2, 5]	2
iscas89-s382	116	168	3	2	18	0.01	0.02	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	5
iscas89-s38417	9500	10 635	3	4	39	7.11	0.20	0.15	0.17	0.01	4	2	[4, 5]	[4, 9]	12
iscas89-s38584	9193	12 573	3	4	54	51.37	0.21	0.15	0.17	0.01	3	4	[3, 4]	[4, 9]	13
iscas89-s386	114	200	4	3	23	0.03	0.01	0.01	0.01	0.00	2	3	[2, 3]	[3, 7]	13
iscas89-s400	121	182	3	2	19	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	5
iscas89-s420.1	129	145	3	2	9	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	3
iscas89-s444	134	206	4	2	19	0.01	0.01	0.01	0.01	0.00	2	2	3	[2, 5]	5
iscas89-s510	172	251	3	2	12	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	3
iscas89-s526	160	270	4	3	12	0.02	0.01	0.01	0.01	0.00	3	3	[2, 3]	[3, 7]	9
iscas89-s526n	159	268	4	3	12	0.02	0.01	0.01	0.01	0.00	3	3	[3, 4]	[3, 7]	9
iscas89-s5378	1411	1639	3	3	10	0.01	0.01	0.01	0.01	0.01	3	2	[3, 4]	[3, 7]	5
iscas89-s641	100	144	3	3	12	0.01	0.03	0.01	0.01	0.01	3	2	[2, 3]	[3, 7]	7
iscas89-s713	137	180	3	3	12	0.01	0.01	0.01	0.01	0.01	3	2	[2, 3]	[3, 7]	7
iscas89-s820	239	480	4	3	48	0.03	0.01	0.01	0.01	0.01	3	3	[3, 4]	[3, 7]	16

Continued on next page

						Running-time					Statistics				
Network	n	m	$\lceil \bar{d} \rceil$	deg	Δ	VC-dim	biclique	crown	ladder	c-c	VC-dim	biclique	crown	ladder	c-c
iscas89-s832	245	498	5	3	49	0.03	0.01	0.01	0.01	0.01	3	3	[3, 4]	[3, 7]	19
iscas89-s838.1	265	301	3	2	12	0.01	0.02	0.01	0.01	0.01	2	2	[2, 3]	[2, 5]	3
iscas89-s9234	1985	2370	3	4	18	0.07	0.04	0.02	0.01	0.02	3	4	[3, 4]	[4, 9]	10
iscas89-s953	332	454	3	2	12	0.01	0.01	0.01	0.01	0.01	2	2	[2, 3]	[2, 5]	4
jazz	198	2742	28	29	100	—	200.16	250.91	129.73	0.02	[5, 30]	[12, 29]	[13, 30]	[11, 59]	42
karate	34	78	5	4	17	0.01	0.01	0.01	0.01	0.01	3	3	5	[4, 9]	7
lederberg	8324	41 532	10	15	1103	—	—	—	5.88	0.08	[5, 16]	[7, 15]	[9, 16]	[15, 31]	175
lesmiserables	77	254	7	9	36	0.27	0.06	0.01	0.01	0.00	3	6	10	[9, 19]	9
link-pedigree	898	1125	3	2	14	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	14
linux	30 834	213 217	14	23	9338	178.38	179.08	184.87	169.29	2.71	[7, 24]	[7, 23]	[7, 24]	[7, 47]	4836
livemocha	104 103	2 193 083	43	92	2980	308.64	318.47	325.14	326.65	62.6	[2, 93]	[2, 92]	[2, 93]	[2, 185]	1129
loc-br.kite-e	58 228	214 078	8	52	1134	381.48	385.52	393.61	346.98	0.35	[5, 53]	[5, 52]	[5, 53]	[5, 105]	184
location	225 486	293 697	3	5	12 189	—	1.38	1.73	1.53	1.52	[4, 6]	5	[4, 5]	[5, 11]	854
marvel	19 428	96 662	10	18	1625	—	516.95	—	19.96	0.18	[6, 19]	15	[8, 19]	[18, 37]	745
mg-casino	109	326	6	9	94	0.06	0.03	0.01	0.01	0.00	3	5	10	[9, 19]	8
mg-forrestgump	94	271	6	8	89	0.07	0.01	0.01	0.01	0.00	3	4	9	[8, 17]	4
mg-godfatherII	78	219	6	8	34	0.05	0.01	0.01	0.01	0.00	3	5	9	[8, 17]	9
mg-watchmen	76	201	6	7	33	0.01	0.02	0.01	0.01	0.00	3	4	8	[7, 15]	6
minnesota	2642	3303	3	2	5	0.02	0.02	0.03	0.03	0.00	2	2	3	[2, 5]	3
moreno-health	2539	10 455	9	7	27	—	0.12	0.07	0.07	0.01	[4, 8]	5	[7, 8]	[7, 15]	17
mousebrain	213	16 089	152	111	205	—	—	—	—	0.03	[3, 112]	[3, 111]	[3, 112]	[3, 223]	179
movielens-1m	9746	1 000 209	206	255	3428	—	—	—	—	14.8	[2, 256]	[2, 255]	[2, 256]	[2, 511]	2356
movies	101	192	4	3	19	0.01	0.01	0.01	0.01	0.00	3	2	[3, 4]	[3, 7]	6
muenchen-bahn	447	578	3	2	13	0.01	0.02	0.01	0.01	0.00	2	[0, 1]	[2, 3]	[2, 5]	2
munin	1324	1397	3	3	66	0.30	0.03	0.01	0.01	0.00	2	3	[2, 3]	[3, 7]	29
netscience	1461	2742	4	19	34	297.79	198.83	2.11	—	0.00	3	10	20	[12, 39]	6
offshore	278 877	505 965	4	13	37 336	—	2.14	583.20	2.50	1.79	[5, 14]	13	[9, 10]	[13, 27]	5698
openflights	2939	15 677	11	28	242	—	89.14	147.54	144.73	0.01	[5, 29]	[7, 28]	[7, 29]	[7, 57]	111
p2p-Gnutella04	10 876	39 994	8	7	103	—	0.22	1.22	0.22	0.04	[4, 8]	7	[5, 6]	[7, 15]	29
panama	556 686	702 437	3	62	7015	—	—	—	—	5.86	[4, 63]	[6, 62]	[6, 63]	[6, 125]	1948
paradise	542 102	794 545	3	23	35 359	—	—	—	—	5.62	[5, 24]	[22, 23]	[6, 24]	[23, 47]	4242
photoviz-dynamic	376	610	4	4	29	0.20	0.02	0.01	0.01	0.00	3	3	[3, 4]	[4, 9]	13
pigs	492	592	3	2	39	0.01	0.01	0.01	0.01	0.00	2	2	[2, 3]	[2, 5]	7
polblogs	1224	16 715	28	36	351	—	87.99	92.62	163.87	0.04	[5, 37]	[5, 36]	[5, 37]	[6, 73]	130
polbooks	105	441	9	6	25	0.05	0.01	0.01	0.01	0.00	4	5	[6, 7]	[6, 13]	16
pollination-ca..	1500	15 255	21	18	157	—	—	—	68.80	0.03	[5, 19]	[6, 18]	[6, 19]	[18, 37]	67
pollination-da..	797	2933	8	9	124	292.96	0.46	0.46	0.06	0.00	5	6	[6, 7]	[9, 19]	45
pollination-te..	68	129	4	4	17	0.01	0.01	0.01	0.01	0.00	3	4	[3, 4]	[4, 9]	11
ratbrain	503	23 030	92	67	497	—	538.67	—	—	0.02	[4, 68]	[5, 67]	[5, 68]	[5, 135]	127
reactome	6327	147 547	47	191	855	—	—	—	—	0.22	[3, 192]	[3, 191]	[3, 192]	[3, 383]	446
residence-hall	217	1839	17	11	56	—	2.54	0.13	0.09	0.00	[4, 12]	6	[10, 11]	[11, 23]	18
rhesusbrain	242	3054	26	19	111	—	—	—	22.19	0.01	[5, 20]	[9, 19]	[11, 20]	[19, 39]	38
roget-thesaurus	1010	3648	8	6	28	228.14	0.13	0.03	0.02	0.00	4	3	[6, 7]	[6, 13]	8
slashdot	51 083	117 378	5	14	2915	—	—	—	13.23	0.4	[5, 15]	[5, 14]	[6, 15]	[14, 29]	65
soc-Epinions1	75 879	405 740	11	67	3044	—	—	—	—	1.94	[3, 68]	[3, 67]	[3, 68]	[3, 135]	551
soc-Slashdot0811	77 360	469 180	13	54	2539	169.08	174.42	179.09	179.02	1.20	[3, 55]	[3, 54]	[3, 55]	[3, 109]	618
soc-advogato	5167	39 432	16	25	807	—	145.63	146.80	145.69	0.05	[5, 26]	[6, 25]	[6, 26]	[6, 51]	218
soc-gplus	23 628	39 194	4	12	2761	—	4.74	25.84	0.28	0.09	[6, 13]	9	[7, 8]	[12, 25]	490

Continued on next page

Network	n	m	$\lceil \bar{d} \rceil$	deg	Δ	Running-time VC-dim	biclique	crown	ladder	c-c	Statistics VC-dim	biclique	crown	ladder	c-c
soc-hamsterster	2426	16 630	14	24	273	—	83.26	131.04	229.61	0.02	[5, 25]	[11, 24]	[23, 25]	[24, 49]	77
soc-wiki-Vote	889	2914	7	9	102	—	0.19	0.11	0.04	0.00	[4, 10]	5	[7, 8]	[9, 19]	18
sd-school-2	238	5539	47	33	88	—	159.31	159.79	165.56	0.02	[5, 34]	[6, 33]	[7, 34]	[7, 67]	54
teams	935 591	1 366 466	3	9	2671	—	256.73	275.42	19.94	22.8	[6, 10]	6	[6, 7]	[9, 19]	351
train-bombing	64	243	8	10	29	0.40	0.09	0.01	0.01	0.00	3	5	11	[10, 21]	9
tv-tropes	152 093	3 232 134	43	115	12 400	476.72	489.41	494.33	496.33	37.0	[2, 116]	[2, 115]	[2, 116]	[2, 231]	2655
twittercrawl	3656	154 824	85	143	1084	—	—	—	—	0.74	[3, 144]	[3, 143]	[3, 144]	[3, 287]	507
unicode-langua..	868	1255	3	4	141	1.06	0.94	0.01	0.02	0.00	3	4	[3, 4]	[4, 9]	19
wafa-ceos	26	93	8	5	22	0.01	0.01	0.01	0.01	0.00	3	3	[5, 6]	[5, 11]	7
wafa-eies	45	652	29	24	44	50.46	—	—	—	0.00	4	[14, 24]	[22, 25]	[13, 49]	31
wafa-hightech	21	159	16	12	20	0.17	0.40	0.13	10.16	0.00	3	7	[10, 11]	[8, 17]	17
wafa-padgett	15	27	4	3	8	0.01	0.01	0.01	0.01	0.00	2	2	[3, 4]	[3, 7]	4
web-EPA	4271	8909	5	6	175	—	0.10	0.21	0.02	0.01	[3, 7]	5	[4, 5]	[6, 13]	64
web-california	6175	15 969	6	11	199	—	0.33	3.95	0.16	0.02	[4, 12]	11	[10, 11]	[11, 23]	69
web-google	1299	2773	5	17	59	—	50.40	0.58	0.71	0.00	[3, 18]	9	18	[17, 35]	18
wikipedia-norm	1881	15 372	17	22	455	—	482.15	161.62	140.63	0.03	[6, 23]	[10, 22]	[11, 23]	[11, 45]	138
win95pts	99	112	3	2	9	0.01	0.01	0.01	0.01	0.00	2	2	3	[2, 5]	4
windsurfers	43	336	16	11	31	0.77	0.39	0.10	0.03	0.00	4	6	[9, 10]	[11, 23]	17
word-adjacencies	112	425	8	6	49	0.01	0.03	0.01	0.01	0.00	4	4	[5, 6]	[6, 13]	14
zewail	6651	54 182	17	18	331	—	—	—	96.19	0.06	[5, 19]	[9, 18]	[10, 19]	[18, 37]	99