Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 30, no. 1, pp. 385–419 (2026)
DOI: 10.7155/jgaa.v30i1.3100

The Parameter Report: An Orientation Guide for Data-Driven Parameterization

Christian Komusiewicz[1] *Nils Morawietz*[1] *Frank Sommer*[1] *Luca Pascal Staus*[1]

[1]Institute of Computer Science, Friedrich Schiller University Jena, Germany

Submitted: October 2025	Accepted: July 2026	Published: August 2026
Article type: Regular paper	Communicated by: Alexander Wolff	

Abstract. A strength of parameterized algorithmics is that each problem can be parameterized by an essentially inexhaustible set of parameters. Usually, the choice of the considered parameter is informed by the theoretical relations between parameters with the general goal of achieving FPT-algorithms for smaller and smaller parameters. However, the FPT-algorithms for smaller parameters usually have higher running times and it is unclear whether the decrease in the parameter value or the increase in the running time bound dominates in real-world data. This question cannot be answered from purely theoretical considerations and any answer requires knowledge on typical parameter values.

To provide a data-driven guideline for parameterized complexity studies of graph problems, we present the first comprehensive comparison of parameter values for a set of benchmark graphs originating from real-world applications. Our study covers degree-related parameters, such as maximum degree or degeneracy, neighborhood-based parameters such as neighborhood diversity and modular-width, modulator-based parameters such as vertex cover number and feedback vertex set number, and the treewidth of the graphs.

Our results may help assess the significance of FPT-running time bounds on the solvability of real-world instances. For example, the vertex cover number vc of n-vertex graphs is often only slightly below $n/2$. Thus, a running time bound of $\mathcal{O}(2^{\mathsf{vc}})$ is only slightly better than a running time bound of $\mathcal{O}(1.4^n)$. In contrast, the treewidth tw is almost always below $n/3$ and often close to $n/10$, making a running time of $\mathcal{O}(2^{\mathsf{tw}})$ much more practical on real-world instances.

We make our implementation and full experimental data openly available[1]. In particular, this provides the first implementations for several graph parameters such as 4-path vertex cover number and vertex integrity.

E-mail addresses: c.komusiewicz@uni-jena.de (Christian Komusiewicz) nils.morawietz@uni-jena.de (Nils Morawietz) frank.sommer@uni-jena.de (Frank Sommer) luca.staus@uni-jena.de (Luca Pascal Staus)

[1]The source code of all solvers, all graph data and our experimental results are all publicly available at `https://www.fmi.uni-jena.de/en/19723/parameter-report`.

1 Introduction

In the quest to cope with computational intractability, parameterized algorithmics offers an approach for designing exact algorithms with practically useful running time guarantees. The aim is to design FPT-algorithms, that is, algorithms with running time $f(k) \cdot \mathrm{poly}(n)$ where k is a problem-specific parameter and n denotes the overall input size. Now, if k is small on real-world instances and f does not grow too quickly, then these algorithms have a practically feasible running time. The classic example for this approach is the NP-hard VERTEX COVER problem where the input is a graph G and a number k and the task is to decide whether G contains a set of at most k vertices that cover all edges. VERTEX COVER can be easily seen to be solvable in $2^k \cdot \mathrm{poly}(n)$ time by a simple branching algorithm. After achieving such a first FPT-algorithm for a problem at hand, the daily trade of researchers in parameterized algorithms is now to chip away from the rock of intractability by providing better and better FPT-algorithms [16].

There are two pathways to achieve this: The first is to improve FPT-algorithms for the parameter k at hand which means first and foremost to improve the superpolynomial running time part $f(k)$. For VERTEX COVER this pathway has been thoroughly explored, with the current best FPT-algorithm having a running time of $\mathcal{O}(1.26^k + n)$ [31]. The second pathway is to consider smaller and smaller parameters: After developing an FPT-algorithm for some parameter k, one may try to find an FPT-algorithm for some parameter k' which is never larger than k. For VERTEX COVER, such a parameter is for example the treewidth tw of G for which an algorithm with running time $\mathcal{O}(2^{\mathsf{tw}} \cdot n)$ is known.[2] From a purely theoretical standpoint, the latter algorithm is usually deemed preferable: For VERTEX COVER, the parameterization by the solution size bound k is essentially a parameterization by the vertex cover number of G, denoted vc. This number is the size of a smallest vertex cover of G. Due to the known parameter relation $\mathsf{vc} \geq \mathsf{tw}$, any FPT-algorithm for the treewidth trivially implies an FPT-algorithm for the vertex cover number vc. This line of research in parameterized algorithms is commonly referred to as multivariate algorithmics [8, 19, 20, 40, 44] or structural parameterization [7, 25, 32, 39]. There are at least two potential pitfalls with this approach: First, at least for graph problems, the space of known parameterizations is huge and ever-growing. Hence, it is infeasible to thoroughly study all parameterizations and essentially every choice of a collection of parameters in an algorithmic study is in some sense arbitrary. Second, the improvement in the parameter dimension often comes at the cost of an increased running time. In the VERTEX COVER example, the running time bound for treewidth is larger than for the vertex cover number. Hence, it is not clear whether the theoretical improvement in the parameter dimension carries over to better algorithms in practice or whether the running times actually get worse.

To avoid both pitfalls, we propose a data-driven approach, where the actual values of parameters in real-world instances guide the navigation through parameter space: When going from a parameterization k to a parameterization k', we may then choose the parameterization which has smaller values in practice so any obtained FPT-algorithms will be more relevant in practice. Moreover, when knowing the typical relation between the values of two parameters in real-world instances, we may better compare known running time bounds.

Our Contribution. To make a first step towards this data-driven approach, we present a study on the values of 21 popular graph parameters on a benchmark set of 144 graphs from real-world applications. The 21 parameters are as follows:

[2] Here, one assumes that one is given a tree decomposition of width tw.

- Degree-related: maximum degree (Δ), h-index, degeneracy (d), 2-core, and 3-core, c-closure (c), and weak-c-closure (γ).
- Neighborhood-based: neighborhood diversity (nd), Dilworth number (∇), modular-width (mw), and splitwidth (sw).
- Modulator-based: vertex cover number (vc), 1-bounded-degree deletion number (1-bdd), 2-bounded-degree deletion number (2-bdd), 4-path vertex cover number (4-pvc), cluster vertex deletion number (cvd), distance to cograph (dco), vertex-integrity (vi), and feedback vertex set number (fvs).
- Treewidth (tw) and treedepth (tdp).

We present various ways to compare the parameter values in an algorithmic setting and describe some scenarios how the parameter values may be used to guide the exploration of parameter space. Our results indicate, for example, that the running time bound of $\mathcal{O}(2^{\mathsf{tw}} \cdot n)$ for the Vertex Cover problem is better than the $\mathcal{O}(1.26^{\mathsf{vc}}+n)$-time bound. We make the source code of the implementation of the algorithms for computing the considered graph parameters and all the experimental data fully available[3] to facilitate the computation of parameter values by the parameterized algorithmics community. Since the exact computation of some parameters is hard, we split our evaluation for these parameters in two parts: First, we consider those 59 instances where all parameters can be computed exactly. For the remaining 85 instances, we resort to heuristically computed parameter values whenever the exact algorithms failed.

Moreover, we analyze the parameter values that can be obtained via two approaches for fusing neighborhood diversity nd [48] and modular-width mw [4, 33, 42] with essentially any other parameter k which gives a parameter that is simultaneously upper-bounded by nd and k and mw and k, respectively.

This work is structured as follows. In Section 2 we introduce some graph notation and define all graph parameters under consideration. In Section 3 we present the evaluation for the exact parameter values. In Section 4 we present the evaluation for the heuristically computed parameter values. Some details concerning the algorithms for computing the parameters are given in Section B.

2 Preliminaries

2.1 Graph Notation

For $x \in \mathbb{N}$ by $[x]$ we denote the set $\{1, 2, \dots, x\}$. We use standard graph notation according to Diestel [13]. An *undirected graph* is a tuple $G \coloneqq (V, E)$ where V is the set of *vertices* and $E \subseteq \{\{u, v\} \subseteq V : u \neq v\}$ is the set of *edges*. We also use $V(G)$ and $E(G)$ to denote the vertices and edges of G, respectively. We let $n \coloneqq |V(G)|$ and $m \coloneqq |E(G)|$. For a vertex v of G, we denote by $N_G(v) \coloneqq \{w \in V(G) : \{v, w\} \in E(G)\}$ the *open neighborhood* of v. Let $X, Z \subseteq V(G)$. By $N_G(X) \coloneqq (\cup_{v \in X} N_G(v)) \setminus X$ we denote the *open neighborhood of* X and by $N_G[X] \coloneqq N_G(X) \cup X$ we denote the *closed neighborhood of* X. By $E_G(X, Z) \coloneqq \{\{x, z\} \in E(G) : x \in X \text{ and } z \in Z\}$ we denote the set of edges *between* X and Z. Moreover, we use $E_G(X)$ as a shorthand for $E_G(X, X)$. We omit the subscript if G is clear from context. By $G[X] \coloneqq (X, E_G(X))$ we denote the *subgraph of* G *induced by* X. A graph $G' \coloneqq (V', E')$ is a *subgraph* of G if $V' \subseteq V(G)$ and $E' \subseteq E(G[V'])$. We let $G - X \coloneqq G[V \setminus X]$ denote the subgraph obtained by *removing* the vertices of X from G.

[3] For several parameters, for example for the treewidth, we use the currently best openly available solvers.

A sequence $P \coloneqq (v_1, \ldots, v_p)$ of vertices is a *path* in G if $\{v_i, v_{i+1}\} \in E(G)$ for each $i \in [p-1]$ and no vertex appears more than once. Its *length* is $p-1$. For a path $P \coloneqq (v_1, \ldots, v_p)$ in G, we denote by $V(P) \coloneqq \{v_i : i \in [p]\}$ the vertices of P and by $E(P) \coloneqq \{\{v_i, v_{i+1}\} : i \in [p-1]\}$. Furthermore, a path P in G is an *induced path* if $E(G[V(P)]) = E(P)$. Two vertices u and v are *connected* if there exists a path P such that $u, v \in V(P)$. A *connected component* of G is an inclusion maximal induced subgraph of G where any two vertices are connected to each other. Let G be a graph and let $X \subseteq V(G)$. By $\mathsf{conn}(X)$ we denote the *vertex connectivity* of the graph $G[X]$, that is, $\mathsf{conn}(X)$ is the smallest number r, such that for some $Z \subseteq X$ of size r, the graph $G[X \setminus Z]$ has more than one connected component. A vertex set X is a *clique* if each two distinct vertices in X are adjacent. A vertex v is *simplicial* if $N(v)$ is a clique.

2.2 Graph Parameters

2.2.1 Degree-Related

Let $v \in V(G)$. We denote the *degree* of v by $\deg_G(v) \coloneqq |N(v)|$. The *maximum degree* and *minimum degree* of G are $\Delta(G) \coloneqq \max_{v \in V(G)} \deg_G(v)$ and $\delta(G) \coloneqq \min_{v \in V(G)} \deg_G(v)$, respectively. The *degeneracy* of G is $d(G) \coloneqq \max_{S \subseteq V(G)} \delta_{G[S]}$ [51]. The *h-index* of a graph G, denoted $h(G)$, is the largest integer h such that G has at least h vertices of degree at least h [18]. The *closure number* $\mathrm{cl}_G(v)$ of a vertex v is $\max_{u \in V(G) \setminus N[v]} |N(v) \cap N(u)|$. We say that G is *c-closed* if $\mathrm{cl}_G(v) < c$ for each vertex $v \in V(G)$. Furthermore, the *closure number* of a graph G, denoted $\mathrm{cl}(G)$, is the smallest integer c such that G is c-closed [22]. We say that G is weakly γ-closed if every induced subgraph G' of G has a vertex $v \in V(G')$ such that $\mathrm{cl}_{G'}(v) < \gamma$. The *weak closure number* of a graph G, denoted $\gamma(G)$ is the smallest integer γ such that G is weakly γ-closed [22].

The k-core of G is the maximal induced subgraph H of G such that $\delta(H) \geq k$ [56]. For $k \leq d$, the k-core is nonempty and for $k > d$, the k-core has size 0. We consider the cases $k = 2$ and $k = 3$ and let 2-core and 3-core denote the number of vertices of the corresponding cores.

2.2.2 Neighborhood-Based Parameters

Here, we define the considered parameters which aim at quantifying the complexity of the family of different neighborhoods in a graph.

Two vertices $u, w \in V(G)$ have the *same type* if $N(u) \setminus \{w\} = N(w) \setminus \{u\}$. Vertices of the same type are also called *twins*. The *neighborhood diversity* $\mathsf{nd}(G)$ of G is the smallest number ℓ such that there is a partition $(V_1, \ldots, V_\ell)$ of $V(G)$, where in each subset V_i, all vertices have the same type [47].

The *Dilworth number* $\nabla(G)$ of G is the size of a largest set of vertices D such that for each two vertices u and w in D we have $N(u) \not\subseteq N(w)$ and $N(w) \not\subseteq N(u)$ [14, 64].

Modular-width. A *modular decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{T}, \beta)$ consisting of a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{A})$ and a function β that maps each node $x \in \mathcal{V}$ to a graph $\beta(x)$. If x is a leaf of $\mathcal{T}$, then $\beta(x)$ contains a single vertex of V and for each vertex $v \in V$, there is exactly one leaf ℓ of $\mathcal{T}$ such that the graph $\beta(\ell)$ consists only of v. If x is not a leaf node, then the vertex set of $\beta(x)$ is exactly the set of child nodes of x in $\mathcal{T}$. Moreover, let V_x denote the set of vertices of V contained in leaf nodes of the subtree rooted in x. Formally, V_x is defined as $V(\beta(\ell))$ for leaf nodes ℓ and recursively defined as $\bigcup_{y \in V(\beta(x))} V_y$ for each nonleaf node x. Moreover, we define $G_x = (V_x, E_x) \coloneqq G[V_x]$. A modular decomposition has the property that for each nonleaf node x and any pair of distinct nodes $y \in V(\beta(x))$ and $z \in V(\beta(x))$, y and z are adjacent in $\beta(x)$

if and only if every vertex in V_y is adjacent in G to every vertex in V_z, similarly, y and z are not adjacent if and only if there is no edge in G between a vertex in V_y and a vertex in V_z. Hence, it is impossible that there are vertex pairs $(v_1, w_1) \in V_y \times V_z$ and $(v_2, w_2) \in V_y \times V_z$ such that v_1 is adjacent to w_1 and v_2 is not adjacent to w_2.

We call $\beta(x)$ the *quotient graph* of x. The *width of a modular decomposition* is the size of a largest vertex set of any quotient graph and the *modular-width* of a graph G, denoted by $\mathsf{mw}(G)$, is the minimal width of any modular decomposition of G [24].

Splitwidth. A *split* of a graph $G = (V, E)$ is a partition (V_1, V_2) of V with $|V_1| \geq 2$ and $|V_2| \geq 2$ such that all vertices in V_1 with at least one neighbor in V_2 have the same neighborhood in V_2. In other words, there are sets $V_1' \subseteq V_1$ and $V_2' \subseteq V_2$ such that $N(v) \cap V_2 = V_2'$ for each $v \in V_1'$ and $N(w) \cap V_2 = \emptyset$ for each $w \in V_1 \setminus V_1'$. If there is no split for G, we call G *prime*. Let (V_1, V_2) be a split of G. A *simple decomposition of G with respect to* (V_1, V_2) consists of two graphs G_1 and G_2 where $G_i = (W_i, E_i)$ for $i \in \{1, 2\}$ such that $W_i = V_i \cup \{x\}$ for some vertex x which is not contained in V, $G_i[V_i] = G[V_i]$ and x is adjacent to exactly the vertices of V_i' in G_i. The vertex x is called a *marker* vertex. Conversely, two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ with $V_1 \cap V_2 = \{x\}$ can be *composed* into a graph $G \coloneqq (V, E)$ as follows: The graph G is the union of G_1 and G_2 without the marker vertex x plus the edges between each neighbor of x in G_1 and each neighbor of x in G_2. Formally, $V \coloneqq (V_1 \cup V_2) \setminus \{x\}$ and $E \coloneqq E_{G_1}(V_1 \setminus \{x\}) \cup E_{G_2}(V_2 \setminus \{x\}) \cup \{\{v_1, v_2\} : v_1 \in N_{G_1}(x), v_2 \in N_{G_2}(x)\}$. Note that $(V_1 \setminus \{x\}, V_2 \setminus \{x\})$ is a split for G and G_1 and G_2 are a simple decomposition of G with respect to $(V_1 \setminus \{x\}, V_2 \setminus \{x\})$. A *split decomposition* $(\mathcal{T}, \beta)$ of a graph G consists of an undirected tree $\mathcal{T} = (\mathcal{V}, \mathcal{E})$ and a function β that maps each of the nodes x of $\mathcal{V}$ to a prime graph $\beta(x)$ such that

- $|V(\beta(x)) \cap V(\beta(y))| = 1$ if x and y are adjacent in $\mathcal{T}$,
- $V(\beta(x)) \cap V(\beta(y)) = \emptyset$ if x and y are nonadjacent in $\mathcal{T}$, and
- G is equivalent to the graph obtained from recursively composing the graphs of all adjacent node pairs in $\mathcal{T}$.

The *width of a split decomposition* is the size of the largest vertex set of any prime graph and the *splitwidth*, denoted by $\mathsf{sw}(G)$, of a graph G is the minimal width of any split decomposition of G [11].

2.2.3 Modulator-Based Parameters

A *graph property* Π is a collection of graphs. Let G be a graph and let Π be a graph property. We call a vertex set $S \subseteq V(G)$ a *modulator of G to* Π if $G - S$ is contained in Π. In the following, we define several *modulator parameters* we analyze in this work, that is, graph parameters that are defined as the size of a smallest modulator to some fixed graph property Π.

Vertex cover number. The *vertex cover number* of a graph G, denoted by $\mathsf{vc}(G)$, is the size of a smallest modulator of G to the graph property that consists of all edgeless graphs.

Bounded-degree deletion. Let $r > 0$ be an integer. The *r-bounded-degree deletion number* of a graph G, denoted by r-$\mathsf{bdd}(G)$, is the size of a smallest modulator of G to the graph property that consists of all graphs of maximum degree at most r [3, 25, 41].

d-path vertex cover number. Let $d > 0$ be an integer. The *d-path vertex cover number* of a graph G, denoted by $d\text{-}\mathsf{pvc}(G)$, is the size of a smallest modulator of G to the graph property that consists of all graphs that contain no path of length d [6].

Feedback vertex set number. The *feedback vertex set number* of a graph G, denoted by $\mathsf{fvs}(G)$, is the size of a smallest modulator of G to the graph property that consists of all acyclic graphs [1, 36, 50].

Cluster vertex deletion number. The *cluster vertex deletion number* of a graph G, denoted by $\mathsf{cvd}(G)$, is the size of a smallest modulator of G to the cluster graphs which are the graphs where each connected component is a clique [15, 34].

Distance to cograph. The *distance to cograph* of a graph G, denoted by $\mathsf{dco}(G)$, is the size of a smallest modulator of G to the cographs which are defined as the graphs that do not contain a path on four vertices (P_4) as induced subgraph or, equivalently, the graphs with modular-width 2 [9, 52].

Vertex integrity. For a graph H, let $\mathsf{cc}(H)$ denote the number of vertices in a largest connected component of H. The *vertex integrity* of G is defined as $\mathsf{vi}(G) \coloneqq \min_{X \subseteq V(G)}(|X| + \mathsf{cc}(G - X))$ [2, 17, 26]. Any set X with $\mathsf{vi}(G) = |X| + \mathsf{cc}(G - X)$ is called a *vi-set* of G.

2.2.4 Treewidth and Treedepth

Finally, we give the definition for treewidth and the related parameter treedepth.

Treewidth. A *tree decomposition* of a graph $G = (V, E)$ is a pair $(\mathcal{T}, \beta)$ consisting of a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{A})$ and a function $\beta \colon \mathcal{V} \to 2^V$ such that

1. for each vertex v of V, there is at least one node $x \in \mathcal{V}$ with $v \in \beta(x)$,
2. for each edge $\{u, v\}$ of E, there is at least one node $x \in \mathcal{V}$ such that $\beta(x)$ contains u and v, and
3. for each vertex $v \in V$, the subgraph $\mathcal{T}[\mathcal{V}_v]$ is connected, where $\mathcal{V}_v \coloneqq \{x \in \mathcal{V} : v \in \beta(x)\}$.

We call $\beta(x)$ the *bag* of x. The *width of a tree decomposition* is the size of the largest bag minus one and the *treewidth* of a graph G, denoted by $\mathsf{tw}(G)$, is the minimal width of any tree decomposition of G [30, 54].

Treedepth. A *treedepth decomposition* of G is a rooted tree T using the same vertex set $V(G)$ such that for each edge $\{u, w\} \in E(G)$, either u is an ancestor of w or w is an ancestor of u in T. The *depth* of T is the maximum number of vertices on any root-to-leaf path. The *treedepth* $\mathsf{tdp}(G)$ of G is the minimum depth among all treedepth decompositions [53].

3 Exact Parameter Values

3.1 Experimental Setup

Our experiments were performed on a server with two Intel(R) Xeon(R) Gold 6526Y CPUs with 2.80 GHz, 16 cores, and 1024 GB RAM. Each individual experiment was performed on one core and was allowed to use up to 28 GB RAM and run for up to 24 hours. Our Python code was executed using Python 3.13.5. To solve the ILPs we used Gurobi 12.0.1 [28]. To compile and run the existing solvers that were implemented in Java we used OpenJDK 21.0.8. To compile the existing solvers that were implemented in C or C++ we used CMake 3.28.3 and GCC 13.3.0.

The source code of all solvers, all graph data, and our experimental results are all publicly available at

`https://www.fmi.uni-jena.de/en/19723/parameter-report`.

More precisely, the source code of all solvers is available at

`https://git.uni-jena.de/algo-engineering/param-report`

while the graph data and our experimental results are all available at

`https://git.uni-jena.de/algo-engineering/data/graph-repo`.

For archival purposes, the data set and code corresponding to the current version of this work are also available via Zenodo [43].

3.2 Data Set

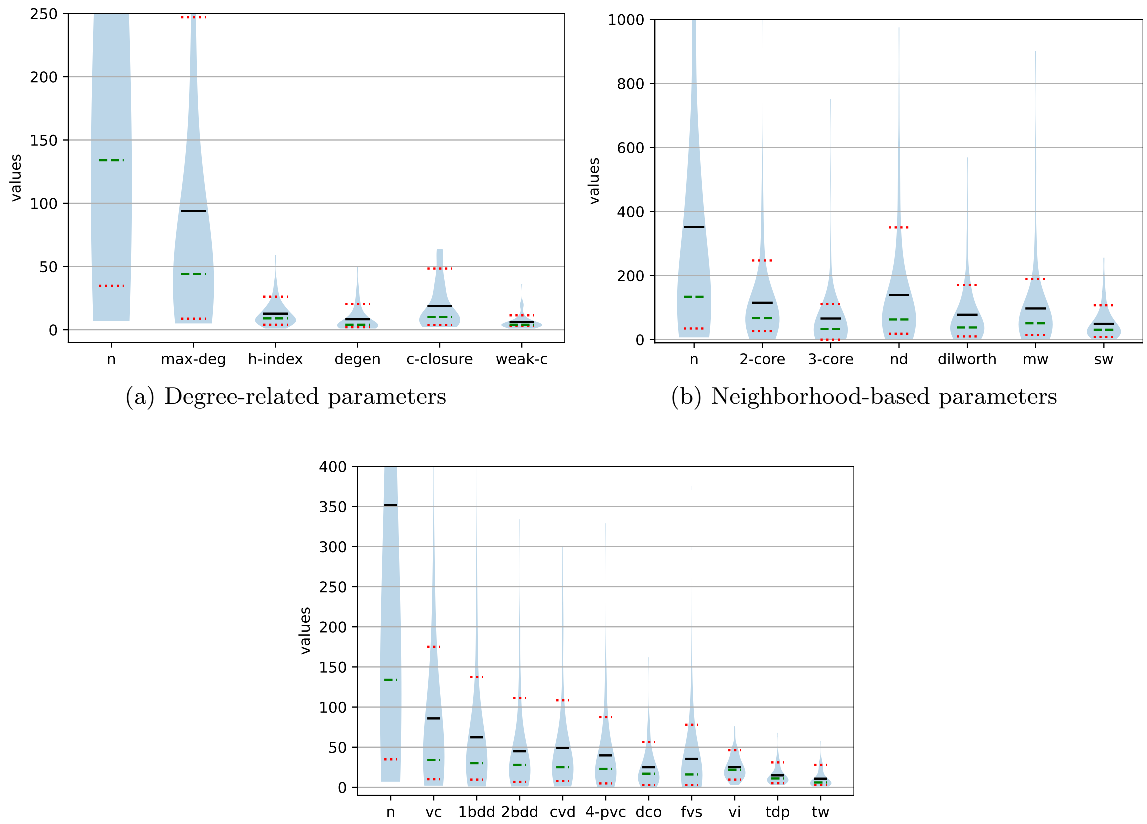
We collected real-world graphs from various applications and repositories such as Konect [46], Matrix Market [5], Network Repository [55], or SNAP [49] with a focus on obtaining graphs of moderate size that are indeed undirected (and not just underlying graphs of directed graphs). The graphs arise, for example, in social network analysis, bioinformatics, scientific computing, or route planning. For most of the graphs, a detailed description of the content is given in the above-mentioned data repository. Overall, we collected 144 graphs with 20 of them being weighted graphs (where we ignored the weights) and 13 being ego networks where we took the largest connected component of the graph induced by the open neighborhood of some vertex in a larger graph.[4]

We partitioned these graphs into two categories as follows: For 59 graphs we were able to compute all parameter values exactly; we refer to them as the *easy* graphs. The remaining 85 graphs for which at least one parameter value could not be computed exactly are referred to as the *hard* graphs. A complete list of the easy and hard graphs can be found in Section A in Table 8 and Table 9, respectively.

3.3 Single Parameters

Absolute Parameter Values. Arguably the most important aspect that we would like to consider is the values of the parameters in our study. Some statistics for these values and also for the relation between these parameters and n are presented in Figure 1 and Table 1.

[4]In social network analysis, the term ego network refers to the graph in the neighborhood of a vertex and it is a very common analysis step to work on ego networks. We chose the largest connected component because most parameters can be calculated independently on connected components, so the largest connected component is more indicative of instance difficulty than the whole graph.



(a) Degree-related parameters

(b) Neighborhood-based parameters

(c) Modulator- and tree-based parameters

Figure 1: Distribution of the absolute parameter values on the 59 easy graphs. The thick lines indicate the 10th percentile (dotted red), the average (solid black), the median (dashed green), and the 90th percentile (dotted red), respectively. The shape of the violin visualizes the value distribution.

We can make the following observations. First, the maximum degree Δ has reasonable parameter values in the median, but there are some instances with prohibitively high values. All the degree-based parameters that are upper-bounded by Δ have among the best statistics of all considered parameters. Unsurprisingly, degeneracy and weak-γ-closure have very low parameter values for most instances.

The 2-core is rather large for most of the instances in terms of absolute values and in terms of the relation to n. The 3-core is often substantially smaller than the 2-core so it is motivated to consider this parameter whenever we have positive results for parameterization by the 2-core size.

For the neighborhood-based parameters, the values are rather large when compared to the other parameter groups. The neighborhood diversity nd is the largest of these parameters and to provide a running time improvement, an FPT-algorithm for nd would need to have essentially the same bound as a known algorithm for n. In comparison, the Dilworth number ∇ and the splitwidth sw are more motivated from the perspective of parameter values. The statistics of the parameter values for splitwidth sw are even better than those for the Dilworth number ∇; modular-width is better

Table 1: Average, median, and 90th percentile of k and k/n for all parameters k.

k	avg. k	median k	90th p. k	avg. k/n	median k/n	90th p. k/n
n	351.7	134.0	1 058.2	1.00	1.00	1.00
Δ	93.9	44.0	247.0	0.39	0.36	0.93
h-index	12.8	9.0	26.2	0.13	0.08	0.32
d	8.3	4.0	20.4	0.09	0.04	0.25
c	18.7	10.0	48.4	0.18	0.06	0.49
γ	6.1	4.0	11.4	0.07	0.03	0.20
2-core	115.3	67.0	247.2	0.60	0.74	1.00
3-core	66.0	33.0	110.8	0.44	0.51	1.00
nd	139.4	63.0	350.4	0.61	0.71	1.00
∇	77.9	38.0	170.6	0.37	0.33	0.71
mw	97.4	51.0	189.6	0.51	0.56	1.00
sw	49.4	31.0	107.2	0.40	0.17	0.99
vc	85.8	34.0	175.2	0.39	0.43	0.73
1-bdd	62.3	30.0	137.6	0.31	0.32	0.59
2-bdd	44.9	28.0	111.4	0.24	0.21	0.52
cvd	48.9	25.0	108.4	0.25	0.24	0.45
4-pvc	39.7	23.0	87.4	0.24	0.23	0.54
dco	25.0	17.0	56.6	0.16	0.11	0.34
fvs	35.5	16.0	78.0	0.22	0.19	0.55
vi	25.1	22.0	46.2	0.23	0.15	0.53
tdp	14.8	11.0	31.0	0.17	0.09	0.44
tw	10.8	6.0	28.0	0.13	0.06	0.31

than neighborhood diversity nd but worse than splitwidth sw and Dilworth number ∇.

The modulator-based parameters are typically smaller than the neighborhood-based parameters. As maybe expected, the vertex cover number has rather large values. In comparison, 1-bdd is considerably smaller, and cvd and 2-bdd are mostly the same and even smaller than 1-bdd. In turn, the other parameters in that group, dco and 4-pvc, have substantially smaller values than 2-bdd and cvd. Thus, from a data-driven standpoint these parameters should receive more consideration. Parameterization by vc should be considered only as a first step and gives a substantial speed-up only if the running time bound for vc is roughly the same as for n. Maybe surprisingly, the vertex cover has roughly similar values to the Dilworth number and is overall somewhat smaller than the maximum degree.

The feedback vertex set number fvs is of course substantially larger than tw on some instances, but it also regularly takes on small values. Thus, one may consider developing FPT-algorithms for fvs even when the problem is FPT for treewidth, for example because such algorithms may avoid dynamic programming, which is typical for treewidth algorithms and has several drawbacks including exponential space requirements and a tendency that the worst-case running time is often met in practice.

Klam-Values. A classic idea for computing the range of feasible parameter values for an FPT-algorithm is Klam-Values, proposed by Downey and Fellows [16]. For a given FPT-algorithm with running time $f(k) \cdot \text{poly}(n)$, this is the largest value of k such that $f(k) \le 10^{20}$. Surely, this is a simplification of the actual border of tractability: the choice of 10^{20} may seem arbitrary, the polynomial running time factor is ignored, and we assume that the actual running times are close to the proven worst-case running time bound. Nevertheless, this value serves as a rough estimate of the range of tractable parameters.

For a variety of typical running times $f(k)$, we count how many of the 59 easy instances would be solvable for the observed parameter values. The results are shown in Table 2. One can see for example, that a slightly subexponential running time $2^{k/\log k}$ would give feasible algorithms for almost all instance-parameter combinations. For a running time of 2^k, still at least half of the instances are solvable for the majority of the parameters. For a running time of k^k, one needs to

Table 2: Number of benchmark instances with k being below the Klam-Values for different running time bounds $f(k)$. The total number of instances is 59.

parameter	$2^{k/\log(k)}$	$\sqrt{2}^k$	2^k	4^k	$k!$	k^k	2^{k^2}	$2^{(2^k)}$
Δ	59	47	38	27	19	14	6	4
h-index	59	59	59	56	51	44	27	19
d	59	59	59	57	53	50	39	39
c	59	59	59	48	41	37	24	19
γ	59	59	59	58	58	54	48	46
2-core	58	49	29	11	4	4	2	1
3-core	58	54	42	31	25	21	17	15
nd	57	42	30	13	7	5	3	3
∇	59	49	39	25	15	14	3	3
mw	58	49	36	20	13	8	4	3
sw	59	54	45	31	20	15	8	5
vc	58	48	39	29	17	13	5	4
1-bdd	59	53	42	33	21	17	6	5
2-bdd	59	55	48	37	28	19	9	6
cvd	59	54	48	36	26	18	7	5
4-pvc	59	56	49	39	27	21	11	9
dco	59	58	54	44	36	29	16	11
fvs	59	57	49	43	37	31	20	15
vi	59	59	58	45	29	19	6	4
tdp	59	59	58	55	48	41	21	10
tw	59	59	59	57	49	47	36	32

consider either the smaller degree-based parameters or tw to solve half of the instances. Maybe somewhat surprisingly, more than half of the instances can be solved when the parameter is the treewidth and the running time is doubly exponential.

The table also reveals a danger when comparing two parameters based on the relation to n as done above: for splitwidth, the average value of sw/n is 0.40, whereas for the Dilworth number ∇ the average value is 0.37. The situation is similar for the median and 90th percentile of k/n. Given these values, one would prefer parameterization by the Dilworth number. However, Table 2 shows that splitwidth will lead to more solved instances for every single one of the considered running time bounds. This is due to the fact that statistics on the distribution of k/n may be influenced by many instances where k and n are so large that they are clearly outside of the realm of tractable instances. In other words, smaller values of k/n may indicate that we get improved running times, but this improvement is possibly made for instances that are too hard to be solved exactly by FPT-algorithms for the parameter k.

For the Vertex Cover problem, this table can also be used to decide whether the running time bound of $\mathcal{O}(1.26^{\mathsf{vc}}+n)$ or the bound of $\mathcal{O}(2^{\mathsf{tw}}\cdot n)$ is better on this data: We see that 2^{tw} is below 10^{20} for all instances while this is barely not the case for $2^{\mathsf{vc}/\log \mathsf{vc}}$ which is smaller than 1.26^{vc} for the observed parameter values.[5] Hence, under the stated assumptions, the algorithm for treewidth would be preferable.

Annotating Parameter Hierarchies. A common strategy for exploring the parameterization space is to use diagrams that represent parameter relations as a guideline [38, 57, 62][6]. In this diagram, parameters with large values are located in the upper part, whereas parameters with small values are located in the lower part. An edge between two parameters represents a relation between them. Roughly speaking this means that the lower parameter is guaranteed to be smaller than the higher parameter. The concrete definition of what it means to be smaller may differ from

[5] The advantage of tw persists also when directly comparing the precise running time bounds including the dependence on n: the bound $2^{\mathsf{tw}}\cdot n$ is below 10^{20} for all 59 instances while $1.26^{\mathsf{vc}}+n$ is below 10^{20} for 53 instances.

[6] See also the websites https://manyu.pro/assets/parameter-hierarchy.html and https://vaclavblazej.github.io/parameters/.

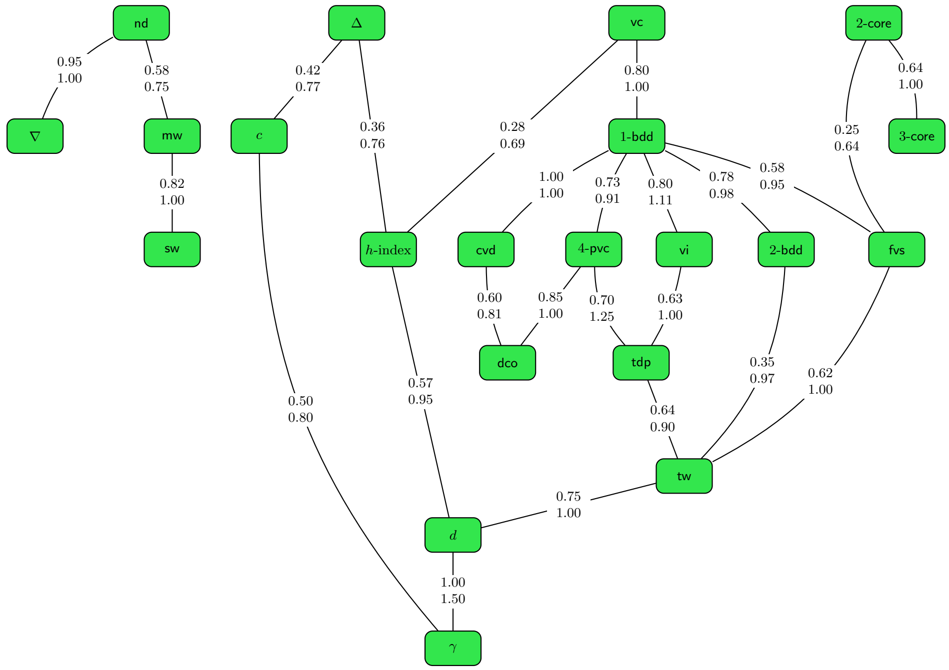


Figure 2: An annotated parameter hierarchy. For an edge from parameter k above to parameter ℓ below, the upper label of an edge is the median value of ℓ/k, the lower label of an edge is the 90th percentile value of ℓ/k. Note that the 90th percentile of vi is larger than that of 1-bdd since in the former we additionally add the component size.

diagram to diagram. Here, we consider a very strict definition, where an edge from a parameter k to a parameter ℓ below means that in every graph G, we have $k(G) \geq \ell(G) + \mathcal{O}(1)$. The additive constant usually only concerns offsets that are created by design choices in the parameter definition. For example, for the parameters degeneracy d and weak closure γ, we have $d \leq \gamma + 1$ but $d \leq \gamma$ is not always true.

When pondering which parameters to consider for a particular problem, the process is often to start with large parameters in the hierarchy and then to proceed downwards as long as one still obtains FPT-algorithms. There may be, however, multiple options for proceeding downwards, as can be seen in Figure 2. The choice of the next parameterization may also benefit from a data-driven approach: We should try to get FPT-results for the parameter whose values give the biggest improvement relative to the values of the current parameter, for which we have shown an FPT-algorithm. To provide an estimate of this improvement, we propose to annotate the parameter hierarchy by statistics on the relation between the parameters. The annotations for the considered parameters are shown in Figure 2. Note that for the weak closure γ and the degeneracy d, the value of the 90th percentile of γ/d is larger than 1. This is due to the above-mentioned additive constant in the bound $d \leq \gamma + 1$: 10 of the instances have degeneracy 2 and weak-closure 3.

An example scenario, where the annotations may be helpful is as follows. Assume that we have shown an FPT-algorithm for 1-bdd. Then, according to Figure 2, one should consider fvs or 4-pvc next since in our set of graphs they provide the biggest improvement. In contrast, cvd is not such a good choice since for half of the instances we essentially have cvd(G) = 1-bdd(G).

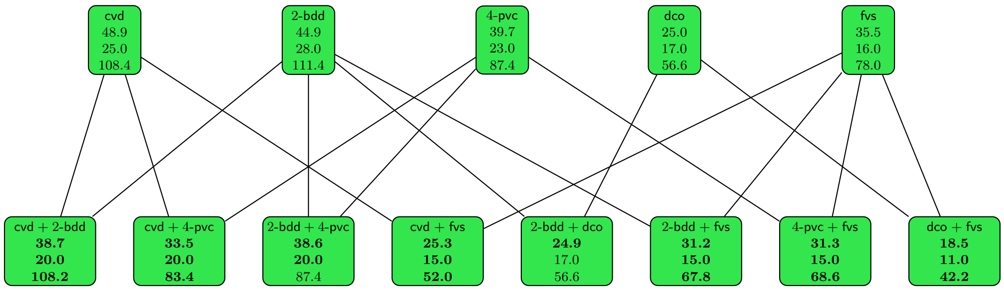


Figure 3: An annotated parameter hierarchy for the pairwise minimum of cvd, 2-bdd, 4-pvc, dco, and fvs. For a parameter k the three values in its box are the average, median, and 90th percentile, respectively. A value for a combined parameter is bold if it is smaller than the minimum of the respective values for the two individual parameters.

Table 3: Parameter combination of tw with an unrelated parameter k. In the table we report statistics on the maximum of tw and k.

	avg.		median		90th p.	
k	max(k, tw)	k	max(k, tw)	k	max(k, tw)	k
tw	10.8	10.8	6.0	6.0	28.0	28.0
Δ	93.9	93.9	44.0	44.0	247.0	247.0
h-index	13.2	12.8	9.0	9.0	28.0	26.2
c	19.1	18.7	10.0	10.0	48.4	48.4
2-core	115.3	115.3	67.0	67.0	247.2	247.2
3-core	66.5	66.0	33.0	33.0	110.8	110.8
∇	79.0	77.9	38.0	38.0	170.6	170.6
cvd	49.7	48.9	28.0	25.0	108.4	108.4

3.4 Multiple Parameters

We also consider the influence of combining noncomparable parameters in different ways.

Best-of-Parameterization. A first implicit type of combination is to take the minimum of two parameters. The scenario for this combination is that whenever we have two different FPT-algorithms, one for the parameter k and another one for the parameter ℓ, we could run for each instance the one whose parameter value is smaller. In other words, we have an FPT-algorithm for the minimum of the two parameters. Now in the algorithm design process, we would like to find parameter combinations where this minimum parameter is particularly worthwhile.

To give an example of such combinations, we consider the pairwise minimum of cvd, dco, 2-bdd, 4-pvc, and fvs. Figure 3 shows the results. For example, the minimum combination of cvd with any of the three unrelated parameters in the diagram is notably smaller than either single parameter. Hence, the respective parameters complement each other nicely, with cvd being small on some instances where the other parameters are large. In contrast, combining 2-bdd and dco only gives a small improvement on average, not on the median or 90th percentile. So either both parameters take on similar values in both instances, or fvs is in most cases better than 2-bdd. If we instead combine dco with fvs, then we obtain the overall best parameter combination.

Combined Parameters. A common phenomenon in parameterized complexity is that if a problem is W-hard with respect to some parameter, then adding a further parameter leads to an FPT-algorithm. Here, we consider the combination of treewidth tw with one unrelated parameter k.

For each such parameter, we compute $\max(\mathsf{tw}(G), k(G))$ on all graphs. The results are shown in Table 3. One can see that the value of $\max(k, \mathsf{tw})$ is almost identical to k. This implies that although tw is unrelated to parameters such as Δ or ∇, in practice tw is usually smaller than these parameters. Additionally, when designing an FPT-algorithm for one of these parameters, one can utilize the additional structure of a tree decomposition without significantly increasing the combined parameter value.

3.5 Parameterizations Based on Quotient Graphs

As observed in the analysis of the single parameters, the neighborhood-based parameterizations neighborhood diversity and modular-width are often not small enough to lead to practical FPT running times. Still, for many problems two vertices with similar neighborhoods are, informally speaking, redundant and we would like to have parameters that are small whenever we have many such redundancies. We propose a systematic way to define such parameterizations by basing them on the quotient graphs of the modular decomposition or on the *twin-quotient graph*.

Twin-Quotient Graph-Based Parameters. Recall that in the definition of neighborhood diversity, two vertices u and v are twins if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. The twin relation is an equivalence relation. For a vertex u, let $[u]$ denote the equivalence class (also called *twin class*) of u.

Definition 1. *The* twin-quotient graph *of a graph G is the graph G_{nd} with vertex set $V(G_{\mathsf{nd}}) \coloneqq \{[u] : u \in V(G)\}$ and $E(G_{\mathsf{nd}}) \coloneqq \{\{[u],[v]\} : [u] \neq [v] \wedge \{u,v\} \in E(G)\}$.*

Informally, the twin-quotient graph has one vertex of every twin class and two such vertices t and t' are adjacent if the members of t are adjacent to the members of t'. Note that the twin-quotient graph of G is an induced subgraph of G. We use the notation G_{nd} to highlight that the neighborhood diversity is essentially the number of vertices of that graph. Now for any parameter k we define a corresponding parameter k_{nd} which is defined just as k but on the twin-quotient graph. This is formalized as follows.

Definition 2. *Let $k : \mathcal{G} \to \mathbb{N}$ be a parameter. Then, the parameter $k_{\mathsf{nd}} : \mathcal{G} \to \mathbb{N}$ is defined via $k_{\mathsf{nd}}(G) \coloneqq k(G_{\mathsf{nd}})$.*

In general, it is not guaranteed that $k_{\mathsf{nd}}(G) \leq k(G)$ holds for every graph. Most common parameters and all parameters considered in this work, however, behave monotonically with respect to taking induced subgraphs. More precisely, they do not increase whenever we delete a vertex from G. For such parameters, $k_{\mathsf{nd}}(G) \leq k(G)$ holds since G_{nd} is an induced subgraph of G. This parameterization has been considered for example by Lampis [48] who studied $\mathsf{tw}_{\mathsf{nd}}$ for vertex coloring.

Modular-Width-Based Parameters. For the modular-width, the parameter definition is more involved. Recall that the modular-width is defined over modular decompositions. These are pairs $(\mathcal{T}, \beta)$ consisting of a rooted tree $\mathcal{T} = (\mathcal{V}, \mathcal{A})$ and a function β that maps each node $x \in \mathcal{V}$ to its quotient graph $\beta(x)$. The modular-width is the maximum number of vertices of any $\beta(x)$. The idea now is simply to consider any parameter k instead of the number of vertices. Formally, this leads to the following.

Definition 3. *Let $k : \mathcal{G} \to \mathbb{N}$ be a parameter. Moreover, for a given graph G and a modular decomposition $(\mathcal{T}, \beta)$ of G, we define $k_{\mathsf{mw}}((\mathcal{T}, \beta)) \coloneqq \max_{x \in V(\mathcal{T})} k(\beta(x))$. Then, the parameter k_{mw} :*

Table 4: Values of the parameterization based on twin-quotient graphs, k_{nd}, and modular decompositions, k_{mw}, relative to the original parameter k.

	avg.		median		90th p.	
	k_{nd}/k	k_{mw}/k	k_{nd}/k	k_{mw}/k	k_{nd}/k	k_{mw}/k
n	0.61	0.51	0.71	0.56	1.00	1.00
Δ	0.60	0.59	0.71	0.71	1.00	1.00
h-index	0.82	0.80	0.88	0.88	1.00	1.00
d	0.88	0.87	1.00	1.00	1.00	1.00
c	0.73	0.72	0.87	0.87	1.00	1.00
γ	0.96	0.96	1.00	1.00	1.00	1.00
2-core	0.69	0.62	0.71	0.64	1.00	1.00
3-core	0.56	0.53	0.70	0.58	1.00	1.00
nd	0.90	0.75	1.00	0.94	1.00	1.00
∇	0.89	0.75	1.00	0.96	1.00	1.00
mw	1.00	1.00	1.00	1.00	1.00	1.00
sw	0.99	0.98	1.00	1.00	1.00	1.00
vc	0.83	0.69	0.99	0.91	1.00	1.00
1-bdd	0.74	0.66	0.83	0.75	1.00	1.00
2-bdd	0.71	0.64	0.79	0.75	1.00	1.00
cvd	0.81	0.70	0.93	0.91	1.00	1.00
4-pvc	0.86	0.74	1.00	0.98	1.00	1.00
dco	0.94	0.81	1.00	1.00	1.00	1.00
fvs	0.77	0.70	0.88	0.86	1.00	1.00
vi	0.85	0.81	0.92	0.89	1.00	1.00
tdp	0.93	0.92	1.00	1.00	1.00	1.00
tw	0.91	0.90	1.00	1.00	1.00	1.00

$\mathcal{G} \to \mathbb{N}$ is defined as $k_{\mathsf{mw}}(G) \coloneqq k_{\mathsf{mw}}((\mathcal{T}_G, \beta_G))$, where $(\mathcal{T}_G, \beta_G)$ denotes a modular decomposition of G that minimizes $k_{\mathsf{mw}}((\mathcal{T}_G, \beta_G))$.

Such a modular decomposition-based parameter was first considered by Bodlaender and Jansen [4], who studied $\mathsf{tw}_{\mathsf{md}}$ and also more recently by Hegerfeld and Kratsch [33]. Another application was given in the context of parameterized local search, where an FPT-algorithm for Δ was improved to one for Δ_{mw} [42].

Note that with these definitions, we have $n_{\mathsf{nd}}(G) = \mathsf{nd}(G)$ and $n_{\mathsf{mw}}(G) = \mathsf{mw}(G)$. Since we consider parameters k that are never larger than n, each parameter $k_{\mathsf{nd}}(G)$ is simultaneously upper-bounded by k *and* by $\mathsf{nd}(G)$. Moreover, as the twin-quotient graph of G is a supergraph of every prime graph in every modular decomposition of G, we further obtain that $k_{\mathsf{mw}} \leq k_{\mathsf{nd}}$.

We now describe how we find the right modular decomposition of a graph, that is, the modular decomposition that defines the parameter k_{mw}, efficiently.

Lemma 1. *Let $k : \mathcal{G} \to \mathbb{N}$ be a parameter for which the value does not increase by taking induced subgraphs. Then, for each graph G, we can compute a modular decomposition $(\mathcal{T}_G, \beta_G)$ of G in linear time, such that $k_{\mathsf{mw}}((\mathcal{T}_G, \beta_G)) = k_{\mathsf{mw}}(G)$.*

Proof: We can compute in linear time [61] a modular decomposition $(\mathcal{T}, \beta)$ of G for which each quotient graph $\beta(x)$ on at least two vertices is *prime*, that is, the modular width of each quotient graph $\beta(x)$ equals the number of vertices of $\beta(x)$. Let $(\mathcal{T}', \beta')$ be any other modular decomposition of G, then for each $x \in V(\mathcal{T})$, there is a $y \in V(\mathcal{T}')$, such that $\beta(x)$ is isomorphic to an induced subgraph of $\beta'(y)$. In particular for $x^* = \arg\min_{x \in V(\mathcal{T})} k(\beta(x))$, there is some $y \in V(\mathcal{T}')$ where $\beta(x^*)$ is isomorphic to an induced subgraph of $\beta'(y)$. As parameter k does not increase by taking induced subgraphs, this implies that $k(\beta(x^*)) \leq k(\beta'(y))$, which implies that $k_{\mathsf{mw}}((\mathcal{T}, \beta)) \leq k_{\mathsf{mw}}((\mathcal{T}', \beta'))$. Hence, $k_{\mathsf{mw}}((\mathcal{T}, \beta)) = k_{\mathsf{mw}}(G)$. □

Parameter Values. Table 4 shows the values of these two variants of each parameter relative to the corresponding parameter of the original graph. Overall we can see the biggest improvement

Table 5: Average, median, and 90th percentile value of the percentage increase from the exact values to the heuristic results on the set of 59 instances where all of these values could be calculated exactly.

k	avg. k	median k	90th p. k
vc	0.9	0.0	3.1
1-bdd	1.8	0.0	7.2
2-bdd	4.3	2.2	13.1
cvd	3.4	0.0	12.7
4-pvc	11.3	1.5	29.5
dco	5.1	4.5	14.4
fvs	1.3	0.0	4.8
vi	5.7	0.0	20.2
tdp	0.0	0.0	0.0
tw	0.0	0.0	0.0

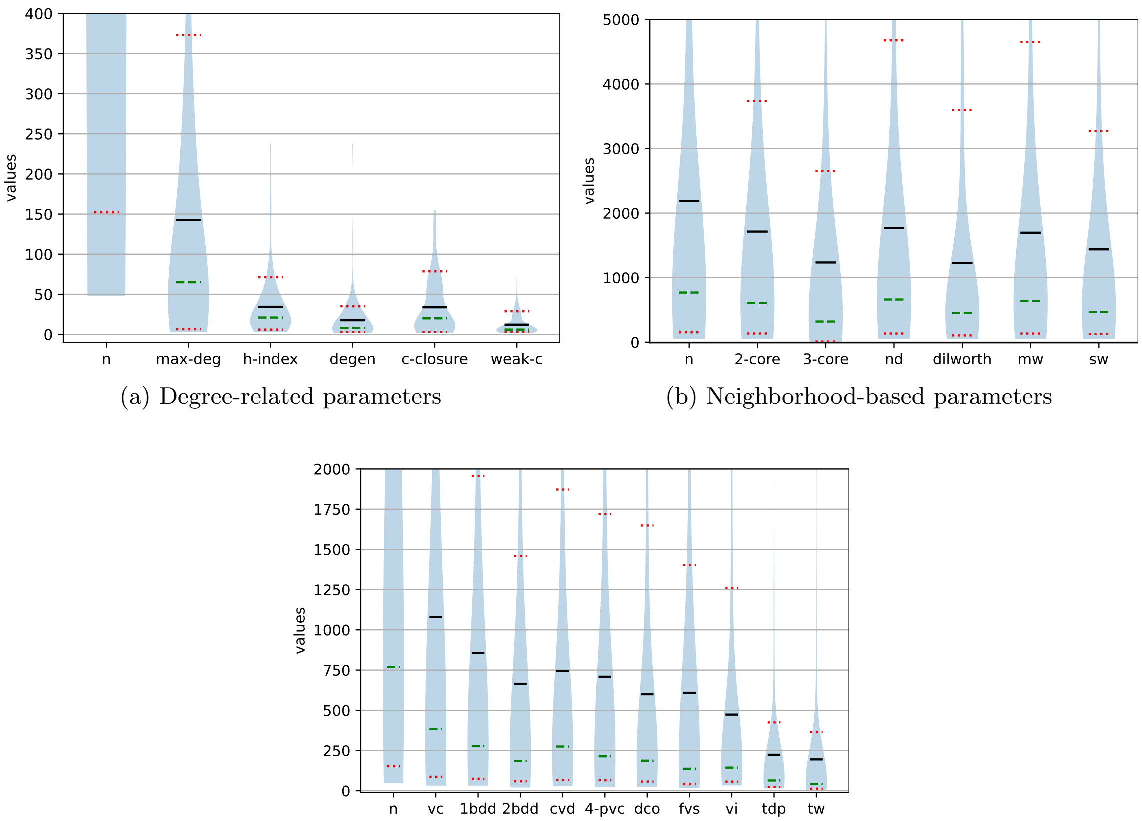
for the maximum degree Δ which suggests that high-degree vertices are often adjacent to many vertices with similar neighborhoods. Other parameters with sizable improvements are the closure number c, the 2-core and 3-core, the 1- and 2-bounded-degree deletion numbers, and the feedback vertex set number fvs: Here the median values decrease by up to 40% while the median values for most other parameters decrease by less than 10%. Hence, for these parameters it seems worthwhile to lift tractability results from the standard parameters to their quotient-graph-based counterparts.

4 Heuristic Parameters

To obtain an overview for a larger set of instances, we resorted to heuristics when the exact algorithms failed to compute the parameter. Using this, we obtained results on the 85 remaining graphs which, as expected, are mostly among the larger instances in our benchmark set. The parameters for which we used heuristics are vc, 1-bdd, 2-bdd, cvd, 4-pvc, dco, fvs, vi, tdp, and tw. For tw we use the winning solver from the 2017 PACE challenge [59], for tdp we use the second-place solver from the 2020 PACE challenge [58]. These are anytime solvers that continue trying to find better solutions until they receive a stop signal. We set a timeout for 30 minutes on each instance. For the remaining parameter computations we use simple greedy algorithms. To see how good the heuristics are, we calculated the percentage increase from the exact values to the heuristic results on the set of 59 graphs where we were able to calculate all parameters exactly. These results are listed in Table 5; overall, the quality is good enough to allow for consideration of these heuristic parameter values.

4.1 Absolute Parameter Values

The distribution of the parameter values is illustrated in Figure 4. In the degree-related parameter group, the overall picture is roughly the same as for the easy instance set. All parameters except the maximum degree take on reasonably small values for a large majority of the instances. The values of the maximum degree are also small for roughly half of the instances. The relative differences between n and the parameter values are now larger, which indicates that this parameter group grows sublinearly with n. In the neighborhood-based parameter group, the values are quite large, and the relative differences between n and the parameter values are smaller than for the easy instances. Intuitively, this means that the structure captured by these parameters is less pronounced in larger graphs. The two smallest parameters are Dilworth number and splitwidth, with the Dilworth number now being slightly smaller than splitwidth; both parameter values are usually too high to



(a) Degree-related parameters

(b) Neighborhood-based parameters

(c) Modulator- and tree-based parameters

Figure 4: Distribution of the absolute parameter values on the 85 hard graphs. The thick lines indicate the 10th percentile (dotted red), the average (solid black), the median (dashed green), and the 90th percentile (dotted red), respectively. The shape of the violin visualizes the value distribution.

be useful, however. Finally, in the modulator-based parameter group, the values are larger than in the degree-based group and smaller than in the neighborhood-based group. The values of the parameters 2-bdd, cvd, 4-pvc, dco, and fvs are roughly similar. The treewidth and treedepth are considerably smaller and their distribution is now clearly distinguishable from the other parameters. Moreover, the median values of both treewidth and treedepth are below 50 which shows that they are also practically very useful.

4.2 Klam-Values

To get a clearer view of the relation between growth of f and the number of feasible instances for each parameter, we again make use of Klam-Values. Table 6 shows the results. As suggested by the parameter value distributions, degree-based parameterizations and treewidth and treedepth are the most promising. For treedepth and treewidth, already slightly superexponential running time bounds such as k^k are too high for a large majority of the instances. To guarantee acceptable

Table 6: Number of benchmark instances with k being below the Klam-Values for different running time bounds $f(k)$. The total number of instances is 85.

parameter	$2^{k/\log(k)}$	$\sqrt{2}^k$	2^k	4^k	$k!$	k^k	2^{k^2}	$2^{(2^k)}$
Δ	83	57	43	32	28	21	12	9
h-index	85	81	76	58	43	38	15	10
d	85	84	82	73	61	58	46	41
c	85	83	71	54	46	40	27	23
γ	85	85	84	78	67	64	49	43
2-core	43	8	2	0	0	0	0	0
3-core	55	20	12	11	11	9	9	9
nd	41	9	2	0	0	0	0	0
∇	45	14	4	0	0	0	0	0
mw	42	9	2	0	0	0	0	0
sw	46	10	2	0	0	0	0	0
vc	59	19	6	2	0	0	0	0
1-bdd	63	27	6	3	0	0	0	0
2-bdd	66	33	11	4	1	0	0	0
cvd	64	30	7	3	0	0	0	0
4-pvc	66	30	11	3	0	0	0	0
dco	66	34	13	4	0	0	0	0
fvs	67	41	18	7	1	0	0	0
vi	70	40	12	2	0	0	0	0
tdp	77	59	44	15	5	1	0	0
tw	78	65	53	34	24	16	3	0

running times for modulator-based parameters such as the vertex cover number, one essentially can only afford single-exponential running times α^k with a small base α.

4.3 Parameterizations Based on Quotient Graphs

Finally, we consider the potential for improvements that can be obtained by using the parameters that are defined on twin-quotient graphs or on the quotient graphs of the modular decomposition. The results are shown in Table 7. As could be expected from the large values of neighborhood diversity nd and modular-width mw on the hard instances, the improvement of k_{nd} and k_{mw} over k is usually only very small and virtually nonexistent for half of the instances. For some parameter combinations, it seems even that $k_{\mathsf{nd}} > k$ but this is only because the parameter values are computed heuristically on some instances.

5 Conclusion

We evaluated 21 popular graph parameters on 144 real-world graphs. On one hand, some parameters such as neighborhood diversity nd or modular-width mw are surprisingly large for most graphs. Thus, in order to improve upon $f(n)$-time algorithms with the help of these parameters, one essentially needs to design algorithms with running time $f(\mathsf{nd})$ or $f(\mathsf{mw})$. Of course studying these parameters is still well motivated if, for example, the resulting algorithms use interesting techniques.

On the other hand, treewidth tw, treedepth tdp and degree-based parameters that are smaller than the maximum degree Δ such as h-index or degeneracy d are quite small on most of the graphs. Hence, from a data-driven perspective the parameterized algorithmics community should focus on developing efficient FPT-algorithms for these parameters. The modulator-based parameters sit somewhere in between the large and small parameters. The best ones of these parameters, such as distance to cograph and feedback vertex set, take on promising values on some instances but these are much fewer than for tw, tdp, and the degree-based category.

In addition, we considered combinations of treewidth with other parameters and showed that the

Table 7: Values of the parameterization based on twin-quotient graphs, k_{nd}, and modular decompositions, k_{mw}, relative to the original parameter k.

	avg.		median		90th p.	
	k_{nd}/k	k_{mw}/k	k_{nd}/k	k_{mw}/k	k_{nd}/k	k_{mw}/k
n	0.88	0.86	0.96	0.95	1.00	1.00
Δ	0.91	0.91	1.00	1.00	1.00	1.00
h-index	0.96	0.96	1.00	1.00	1.00	1.00
d	0.95	0.95	1.00	1.00	1.00	1.00
c	0.95	0.95	1.00	1.00	1.00	1.00
γ	0.99	0.98	1.00	1.00	1.00	1.00
2-core	0.93	0.92	0.99	0.98	1.00	1.00
3-core	0.84	0.83	0.99	0.98	1.00	1.00
nd	0.98	0.97	1.00	1.00	1.00	1.00
∇	0.98	0.96	1.00	1.00	1.00	1.00
mw	1.00	1.00	1.00	1.00	1.00	1.00
sw	1.00	1.00	1.00	1.00	1.00	1.00
vc	0.94	0.92	0.99	0.99	1.00	1.00
1-bdd	0.94	0.93	0.97	0.99	1.00	1.00
2-bdd	0.94	0.93	0.99	0.99	1.00	1.00
cvd	0.97	0.95	1.00	0.99	1.00	1.00
4-pvc	0.96	0.95	1.00	1.00	1.01	1.00
dco	1.00	0.98	1.00	1.00	1.01	1.00
fvs	0.94	0.92	0.99	1.00	1.02	1.00
vi	0.96	0.97	0.98	0.99	1.01	1.00
tdp	0.99	0.99	1.00	1.00	1.00	1.00
tw	0.99	0.99	1.00	1.00	1.03	1.01

combination of h-index and treewidth is, from the data perspective, a good parameter combination that could be considered for problems that are hard when parameterized by treewidth alone. We also considered approaches of combining modular-width and neighborhood diversity with other parameters k by computing k on the modular decompositions and twin-quotient graphs, respectively. For the easier graphs, these two parameters gave some modest improvements over k. In contrast, for the harder graphs the parameter values essentially stayed the same.

In summary, our study shows that for our group of hard graphs (which are still modestly large compared to the massive graphs arising in many applications), the considered parameterizations are not fully satisfactory, and that there is still a need for other approaches to capture real-world structure. In other words, adding further parameters is a clear goal for future parameter reports. In particular, it would be interesting to extend the neighborhood-based category by smaller parameters such as twinwidth which had been excluded since current solvers for these parameters were not capable of solving a sufficient number of instances.

Another avenue for future work is to extend the data set on which we perform the study. Considering even larger graphs is possible but in our opinion less interesting at the moment, as we already have some modestly large graphs that have quite large values for all considered parameters. Instead, the goal will be to add further graphs of the considered sizes so that the number and diversity of graphs have increased sufficiently to distinguish the application background. For example, we may ask whether the treewidth of social networks is smaller or larger than the treewidth of infrastructure networks. With our current data set the single categories are, however, too small to provide a trustworthy answer to such questions.

One may also consider more sophisticated descriptive statistics for comparing parameters. For example, the currently considered median and percentile values of the relation k/n ignore whether an instance with $k/n = 0.1$ was large or small. In a similar direction, one could think of more sophisticated models for estimating the tractable parameter value range other than the Klam-Values.

Finally, it would be interesting to perform similar studies for other classes of inputs such as directed graphs, strings, or Boolean formulas.

Acknowledgements

Frank Sommer was supported by the Alexander von Humboldt Foundation. Luca Pascal Staus was supported by the Carl Zeiss Foundation, Germany, within the project "Interactive Inference". We thank Michael Rohleder (Friedrich Schiller University Jena) for helping us with the implementation of the greedy algorithms for the heuristic parameter computations. We also thank the reviewers and the editor from the Journal of Graph Algorithms and Applications (JGAA) for their valuable and detailed feedback and suggestions.

References

[1] V. Bafna, P. Berman, and T. Fujito. A 2-approximation algorithm for the undirected feedback vertex set problem. *SIAM Journal on Discrete Mathematics*, 12(3):289–297, 1999. doi:10.1137/S0895480196305124.

[2] C. A. Barefoot, R. Entringer, and H. Swart. Vulnerability in graphs–a comparative survey. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 1(38):13–22, 1987. URL: https://combinatorialpress.com/jcmcc-articles/volume-001/vulnerability-in-graphs-a-comparative-survey/.

[3] N. Betzler, R. Bredereck, R. Niedermeier, and J. Uhlmann. On bounded-degree vertex deletion parameterized by treewidth. *Discrete Applied Mathematics*, 160(1-2):53–60, 2012. URL: https://doi.org/10.1016/j.dam.2011.08.013, doi:10.1016/J.DAM.2011.08.013.

[4] H. L. Bodlaender and K. Jansen. On the complexity of the maximum cut problem. *Nordic Journal of Computing*, 7(1):14–31, 2000. URL: https://dl.acm.org/doi/abs/10.5555/640044.640046.

[5] R. F. Boisvert, R. Pozo, K. Remington, R. F. Barrett, and J. J. Dongarra. *Matrix Market: a web resource for test matrix collections*, pages 125–137. Springer US, Boston, MA, 1997. doi:10.1007/978-1-5041-2940-4_9.

[6] B. Bresar, F. Kardos, J. Katrenic, and G. Semanisin. Minimum k-path vertex cover. *Discrete Applied Mathematics*, 159(12):1189–1195, 2011. URL: https://doi.org/10.1016/j.dam.2011.04.008, doi:10.1016/J.DAM.2011.04.008.

[7] H. Bruhn, M. Chopin, F. Joos, and O. Schaudt. Structural parameterizations for boxicity. *Algorithmica*, 74(4):1453–1472, 2016. URL: https://doi.org/10.1007/s00453-015-0011-0, doi:10.1007/S00453-015-0011-0.

[8] L. Bulteau, F. Hüffner, C. Komusiewicz, and R. Niedermeier. Multivariate algorithmics for NP-hard string problems. *Bulletin of the EATCS*, 114, 2014. URL: https://bulletin.eatcs.org/index.php/beatcs/article/view/310.

[9] L. Cai. Parameterized complexity of vertex colouring. *Discrete Applied Mathematics*, 127(3):415–429, 2003. URL: https://doi.org/10.1016/S0166-218X(02)00242-1.

[10] P. Charbit, F. de Montgolfier, and M. Raffinot. Linear time split decomposition revisited. *SIAM Journal on Discrete Mathematics*, 26(2):499–514, 2012. doi:10.1137/10080052X.

[11] W. H. Cunningham. Decomposition of directed graphs. *SIAM Journal on Algebraic Discrete Methods*, 3(2):214–228, 1982. URL: https://doi.org/10.1137/0603021.

[12] H. Dell, T. Husfeldt, B. M. P. Jansen, P. Kaski, C. Komusiewicz, and F. A. Rosamond. The first parameterized algorithms and computational experiments challenge. In *Proceedings of the 11th International Symposium on Parameterized and Exact Computation (IPEC '16)*, volume 63 of *LIPIcs*, pages 30:1–30:9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016. URL: https://doi.org/10.4230/LIPIcs.IPEC.2016.30, doi:10.4230/LIPICS.IPEC.2016.30.

[13] R. Diestel. *Graph Theory, 5th Edition*, volume 173 of *Graduate Texts in Mathematics*. Springer, 2016. URL: https://doi.org/10.1007/978-3-662-70107-2.

[14] R. P. Dilworth. A decomposition theorem for partially ordered sets. *Annals of Mathematics*, 51(1):161–166, 1950. URL: https://doi.org/10.1007/978-0-8176-4842-8_10.

[15] M. Doucha and J. Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS '12)*, volume 7464 of *Lecture Notes in Computer Science*, pages 348–359. Springer, 2012. doi:10.1007/978-3-642-32589-2_32.

[16] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Monographs in Computer Science. Springer, 1999. doi:10.1007/978-1-4612-0515-9.

[17] P. G. Drange, M. S. Dregi, and P. van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. URL: https://doi.org/10.1007/s00453-016-0127-x.

[18] D. Eppstein and E. S. Spiro. The h-index of a graph and its application to dynamic subgraph statistics. *Journal of Graph Algorithms and Applications*, 16(2):543–567, 2012. URL: https://doi.org/10.7155/jgaa.00273.

[19] M. R. Fellows, B. M. P. Jansen, and F. A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *European Journal of Combinatorics*, 34(3):541–566, 2013. URL: https://doi.org/10.1016/j.ejc.2012.04.008.

[20] M. R. Fellows, D. Lokshtanov, N. Misra, M. Mnich, F. A. Rosamond, and S. Saurabh. The complexity ecology of parameters: An illustration using bounded max leaf number. *Theory of Computing Systems*, 45(4):822–848, 2009. URL: https://doi.org/10.1007/s00224-009-9167-9.

[21] S. Felsner, V. Raghavan, and J. P. Spinrad. Recognition algorithms for orders of small width and graphs of small dilworth number. *Order*, 20(4):351–364, 2003. URL: https://doi.org/10.1023/B:ORDE.0000034609.99940.fb, doi:10.1023/B:ORDE.0000034609.99940.FB.

[22] J. Fox, T. Roughgarden, C. Seshadhri, F. Wei, and N. Wein. Finding cliques in social networks: A new distribution-free model. *SIAM Journal on Computing*, 49(2):448–464, 2020. URL: https://doi.org/10.1137/18M1210459.

[23] D. R. Fulkerson. Note on dilworth's decomposition theorem for partially ordered sets. *Proceedings of the American Mathematical Society*, 7(4):701–702, 1956. URL: https://doi.org/10.1016/S0021-9800(67)80079-6.

[24] J. Gajarský, M. Lampis, and S. Ordyniak. Parameterized algorithms for modular-width. In *Proceedings of the 8th International Symposium on Parameterized and Exact Computation (IPEC '13)*, volume 8246 of *Lecture Notes in Computer Science*, pages 163–176. Springer, 2013. doi:10.1007/978-3-319-03898-8_15.

[25] R. Ganian, F. Klute, and S. Ordyniak. On structural parameterizations of the bounded-degree vertex deletion problem. *Algorithmica*, 83(1):297–336, 2021. URL: https://doi.org/10.1007/s00453-020-00758-8, doi:10.1007/S00453-020-00758-8.

[26] T. Gima, T. Hanaka, Y. Kobayashi, R. Murai, H. Ono, and Y. Otachi. Structural parameterizations of vertex integrity. In *Proceedings of the 18th International Conference and Workshops on Algorithms and Computation (WALCOM '24)*, volume 14549 of *Lecture Notes in Computer Science*, pages 406–420. Springer, 2024. URL: https://doi.org/10.1007/978-981-97-0566-5_29.

[27] W. Goddard and H. C. Swart. Integrity in graphs: bounds and basics. *Journal of Combinatorial Mathematics and Combinatorial Computing*, 7:139–151, 1990. URL: https://combinatorialpress.com/jcmcc-articles/volume-007/integrity-in-graphs-bounds-and-basics/.

[28] Gurobi Optimization, LLC. Gurobi Optimizer Reference Manual, 2023. URL: https://www.gurobi.com.

[29] A. Hagberg, P. J. Swart, and D. Schult. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference*, 2008. URL: https://doi.org/10.25080/TCWV9851.

[30] R. Halin. S-functions for graphs. *Journal of Geometry*, 8:171–186, 1976. URL: https://doi.org/10.1007/BF01917434.

[31] D. G. Harris and N. S. Narayanaswamy. A faster algorithm for vertex cover parameterized by solution size. In *Proceedings of the 41st International Symposium on Theoretical Aspects of Computer Science (STACS '24)*, volume 289 of *LIPIcs*, pages 40:1–40:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024. URL: https://doi.org/10.4230/LIPIcs.STACS.2024.40.

[32] S. Hartung, C. Komusiewicz, A. Nichterlein, and O. Suchý. On structural parameterizations for the 2-club problem. *Discrete Applied Mathematics*, 185:79–92, 2015. URL: https://doi.org/10.1016/j.dam.2014.11.026, doi:10.1016/J.DAM.2014.11.026.

[33] F. Hegerfeld and S. Kratsch. Tight algorithms for connectivity problems parameterized by modular-treewidth. In *Proceedings of the 49th International Workshop on Graph-Theoretic Concepts in Computer Science (WG '23)*, volume 14093 of *Lecture Notes in Computer Science*, pages 388–402. Springer, 2023.

[34] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010. URL: https://doi.org/10.1007/s00224-008-9150-x, doi:10.1007/S00224-008-9150-X.

[35] Y. Iwata. Linear-time kernelization for feedback vertex set. In *Proceedings of the 44th International Colloquium on Automata, Languages, and Programming (ICALP '17)*, volume 80

of *LIPIcs*, pages 68:1–68:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. URL: https://doi.org/10.4230/LIPIcs.ICALP.2017.68.

[36] Y. Iwata and Y. Kobayashi. Improved analysis of highest-degree branching for feedback vertex set. *Algorithmica*, 83(8):2503–2520, 2021. URL: https://doi.org/10.1007/s00453-021-00815-w, doi:10.1007/S00453-021-00815-W.

[37] Y. Iwata, M. Wahlström, and Y. Yoshida. Half-integrality, LP-branching, and FPT algorithms. *SIAM Journal on Computing*, 45(4):1377–1411, 2016.

[38] B. M. P. Jansen. *The power of data reduction: Kernels for fundamental graph problems*. PhD thesis, Utrecht University, 2013.

[39] B. M. P. Jansen and S. Kratsch. On polynomial kernels for structural parameterizations of odd cycle transversal. In *Proceedings of the 6th International Symposium on Parameterized and Exact Computation (IPEC '11)*, volume 7112 of *Lecture Notes in Computer Science*, pages 132–144. Springer, 2011. doi:10.1007/978-3-642-28050-4_11.

[40] C. Komusiewicz. Multivariate algorithmics for finding cohesive subnetworks. *Algorithms*, 9(1):21, 2016. URL: https://doi.org/10.3390/a9010021, doi:10.3390/A9010021.

[41] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theoretical Computer Science*, 410(38-40):3640–3654, 2009. URL: https://doi.org/10.1016/j.tcs.2009.04.021, doi:10.1016/J.TCS.2009.04.021.

[42] C. Komusiewicz and N. Morawietz. Parameterized local search for vertex cover: When only the search radius is crucial. In *Proceedings of the 17th International Symposium on Parameterized and Exact Computation (IPEC '22)*, volume 249 of *LIPIcs*, pages 20:1–20:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: https://doi.org/10.48550/arXiv.2605.07941.

[43] C. Komusiewicz, N. Morawietz, F. Sommer, and L. P. Staus. Code, data and experimental results for "The parameter report: An orientation guide for data-driven parameterization", 2025. Zenodo, https://doi.org/10.5281/zenodo.19882369.

[44] C. Komusiewicz and R. Niedermeier. New races in parameterized algorithmics. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS '12)*, volume 7464 of *Lecture Notes in Computer Science*, pages 19–30. Springer, 2012. URL: https://doi.org/10.1007/978-3-642-32589-2_2.

[45] L. Kowalik, M. Mucha, W. Nadara, M. Pilipczuk, M. Sorge, and P. Wygocki. The PACE 2020 parameterized algorithms and computational experiments challenge: Treedepth. In *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC '20)*, volume 180 of *LIPIcs*, pages 37:1–37:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL: https://doi.org/10.4230/LIPIcs.IPEC.2020.37, doi:10.4230/LIPICS.IPEC.2020.37.

[46] J. Kunegis. KONECT: the Koblenz network collection. In *Proceedings of the 22nd International World Wide Web Conference (WWW '13)*, pages 1343–1350. International World Wide Web Conferences Steering Committee / ACM, 2013.

[47] M. Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. URL: https://doi.org/10.1007/s00453-011-9554-x, doi:10.1007/S00453-011-9554-X.

[48] M. Lampis. Finer tight bounds for coloring on clique-width. *SIAM Journal on Discrete Mathematics*, 34(3):1538–1558, 2020. URL: https://doi.org/10.1137/19M128032.

[49] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[50] J. Li and J. Nederlof. Detecting feedback vertex sets of size k in $O^{\star}(2.7k)$ time. *ACM Transactions on Algorithms*, 18(4):34:1–34:26, 2022. doi:10.1145/3504027.

[51] D. R. Lick and A. T. White. k-Degenerate graphs. *Canadian Journal of Mathematics*, 22(5):1082–1096, 1970. URL: https://doi.org/10.4153/CJM-1970-125-1.

[52] J. Nastos and Y. Gao. Bounded search tree algorithms for parametrized cograph deletion: Efficient branching rules by exploiting structures of special graph classes. *Discrete Mathematics, Algorithms and Applications*, 4(01):1250008, 2012. URL: https://doi.org/10.1142/S1793830912500085.

[53] J. Nesetril and P. O. de Mendez. Tree-depth, subgraph coloring and homomorphism bounds. *European Journal of Combinatorics*, 27(6):1022–1041, 2006. URL: https://doi.org/10.1016/j.ejc.2005.01.010, doi:10.1016/J.EJC.2005.01.010.

[54] N. Robertson and P. D. Seymour. Graph minors. II. Algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309–322, 1986. doi:10.1016/0196-6774(86)90023-4.

[55] R. A. Rossi and N. K. Ahmed. The network data repository with interactive graph analytics and visualization. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI '15)*, pages 4292–4293. AAAI Press, 2015. URL: https://doi.org/10.1609/aaai.v29i1.9277, doi:10.1609/AAAI.V29I1.9277.

[56] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983. URL: https://doi.org/10.1016/0378-8733(83)90028-X.

[57] M. Sorge and M. Weller. The graph parameter hierarchy, 2019. https://manyu.pro/assets/parameter-hierarchy.pdf.

[58] B. Strasser. PACE solver description: Tree depth with flowcutter. In Y. Cao and M. Pilipczuk, editors, *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC '20)*, volume 180 of *LIPIcs*, pages 32:1–32:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

[59] H. Tamaki. Positive-instance driven dynamic programming for treewidth. *Journal of Combinatorial Optimization*, 37(4):1283–1311, 2019. URL: https://doi.org/10.1007/s10878-018-0353-z.

[60] H. Tamaki. A contraction-recursive algorithm for treewidth. In *Proceedings of the 18th International Symposium on Parameterized and Exact Computation (IPEC '23)*, volume 285 of *LIPIcs*, pages 34:1–34:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023. URL: https://doi.org/10.4230/LIPIcs.IPEC.2023.34.

[61] M. Tedder, D. G. Corneil, M. Habib, and C. Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming (ICALP '08)*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-70575-8_52.

[62] D. L. Tran. *Expanding the Graph Parameter Hierarchy*. Bachelor's thesis, TU Berlin, 2022.

[63] J. Trimble. PACE solver description: Bute-plus: A bottom-up exact solver for treedepth. In *Proceedings of the 15th International Symposium on Parameterized and Exact Computation (IPEC '20)*, volume 180 of *LIPIcs*, pages 34:1–34:4. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020. URL: https://doi.org/10.4230/LIPIcs.IPEC.2020.34, doi:10.4230/LIPICS.IPEC.2020.34.

[64] R. van Bevern, A. M. Kirilin, D. A. Skachkov, P. V. Smirnov, and O. Y. Tsidulko. Serial and parallel kernelization of multiple hitting set parameterized by the Dilworth number, implemented on the GPU. *Journal of Computer and System Sciences*, 139:103479, 2024. URL: https://doi.org/10.1016/j.jcss.2023.103479.

A Instance Properties

Table 8: Sizes and some additional parameter values for all 59 *easy* graphs for which all parameters could be calculated. A complete overview is available in the online repository.

name	n	m	Δ	vc	tw
nd_7_10	7	10	6	3	2
florentine_families	15	20	6	8	3
zebra	27	111	14	20	12
davis_southern_women	32	89	14	14	8
montreal	33	78	18	12	6
karate-club	34	78	17	14	5
insecta-ant-colony4-day35	35	501	33	31	28
ceo_club	40	95	21	15	8
elite	44	99	12	20	9
macaque_neural	47	313	27	33	16
USA_Mixed_2016	48	103	8	30	6
bcspwr02	49	59	6	22	3
contiguous-usa	49	107	8	30	6
eco-stmarks	54	350	48	36	22
ENZYMES_g103	59	115	9	33	5
dolphins	62	159	12	34	10
terrorists_911	62	152	22	29	7
eco-everglades	69	880	64	42	31
mammalia-voles-bhp-trapping-49	78	119	9	44	6
world-trade	80	875	77	50	22
baseball	84	84	44	12	2
tols90	90	1 449	89	18	18
bn-macaque-rhesus_brain_2	91	582	87	11	11
bn-macaque-rhesus_cerebral-cortex	91	1 401	87	29	28
GD99_c	105	120	5	47	3
centrality-literature	118	613	66	59	24
PACE_2021_jena_3	121	1 191	50	96	36
PACE_2021_jena_4	123	433	18	91	11
PACE_2021_jena_1	125	1 319	58	96	31
PACE_2021_jena_2	134	2 877	85	112	58
revolution	141	160	59	5	3
PACE_2021_jena_6	173	1 039	35	126	29
world_Mixed_2016	173	328	14	91	6
interactome_pdz	212	242	21	56	6
PACE_2021_strings_1	236	1 045	23	155	14
hyves_n	278	302	28	77	4
PACE_2021_colors_2	293	573	24	168	13
oryza_sativa_japonica	327	341	234	25	3
marvel_partnerships	350	346	12	144	4
oryctolagus_cuniculus	350	333	81	66	4
as_skitter_n	375	439	155	10	4
gallus_gallus	457	480	111	101	5
human_papillomavirus_16	484	511	185	6	2
hepatitus_C_Virus	493	491	490	2	1
human_herpesvirus_4	505	543	204	20	4
nd_511_1830	511	1 830	90	255	4
diseasome	516	1 188	50	285	10
canis_familiaris	529	538	406	30	3
danio_rerio	581	602	244	87	3
bos_taurus	605	585	81	164	4
human_Herpesvirus_8	904	914	140	65	4
middle-east_coronavirus	987	1 051	540	24	4
PhD	1 025	1 043	46	259	3
candida_albicans_SC5314	1 191	1 708	427	159	11
board_directors	1 217	1 130	26	204	3
moreno_crime	1 380	1 476	25	451	6
human_Immunodeficiency_Virus_1	1 446	1 732	477	10	7
netscience	1 461	2 742	34	899	19
severe_acute_coronavirus	1 546	1 906	259	30	16

B Algorithms

In this appendix we briefly explain what algorithms we use to compute the various parameters and how they are implemented. The algorithms for most of the simple parameters that can be computed in polynomial time are directly implemented in Python with the help of the NetworkX package [29]. These algorithms are described in Section B.1. For some of the parameters that are NP-hard to compute, we used existing solvers written in Java, C, or C++. We will list these solvers in Section B.2. Finally, to compute the remaining parameters, we use ILP-formulations that are then solved by the ILP-solver Gurobi [28]. These will be explained in Section B.3.

B.1 Polynomial-time Computable Parameters

To compute the parameters n, m, the maximum degree Δ, the degeneracy d, and the k-core size we use existing implementations that are part of the NetworkX package [29]. For the parameters h-index, closure number, and neighborhood diversity we implemented simple and straightforward algorithms that directly follow from the definition of these parameters. To compute the Dilworth number, we use the known reduction of the problem to a bipartite maximum matching instance [23, 21]. This instance is solved by a method provided by NetworkX.

To compute the weak closure number, we first fix some number γ. We then continuously remove any vertex v from G that does not share at least γ neighbors with any nonneighbor of v. A graph G is weakly γ-closed if and only if the graph is empty after this process [22]. Hence, we can compute the weak closure number of G by performing a binary search for γ. We use the minimum of $d+1$ and the closure number c of G as an initial upper bound for γ.

B.1.1 Splitwidth

Our implementation for computing the splitwidth is based on an algorithm developed by Cunningham [11] for directed graphs that runs in $\mathcal{O}(n^3)$ time and is trivially adaptable to undirected graphs. Even though there are linear-time algorithms to compute the splitwidth of an undirected graph [10], we decided to implement the comparably slower algorithm by Cunningham [11] based on the simplicity of the algorithm.

Intuitively, Cunningham's algorithm [11] works as follows: For a given graph G, the algorithm tries to find a simple decomposition of G, if one exists. If no such simple decomposition exists, that is, if G is prime, the algorithm terminates. Otherwise, if there is a simple decomposition (G_1, G_2) of G, the algorithm performs the above procedure for both G_1 and G_2 recursively. To determine, whether there is a simple decomposition (G_1, G_2) of G, the algorithm iterates over all edges e of G and checks whether there is a split (V_1, V_2) of G that is "crossed" by e. Here, we say that an edge e *crosses* a split (V_1, V_2) of G if each of the parts of the split contains exactly one endpoint of e. Checking whether there is a split (V_1, V_2) of G that is crossed by e (and if so, finding one) can be done efficiently [11].

To speed up our implementation, we prevent the algorithm from checking whether an edge e crosses a split in any subgraph, if we already checked that e does not cross any split of the supergraph. This is based on the fact that, if an edge e of G crosses no split of G, then for each simple decomposition (G_1, G_2) of G, e does not cross any split of G_1 and e does not cross any split of G_2. Below, we give a proof of this fact.

Lemma 2. *Let G be a graph, let (G_1, G_2) be a simple decomposition of G, and let $\{u, v\} \in E(G) \cap E(G_1)$. If there is a split (W_1, W_2) of G_1 with $u \in W_1$ and $v \in W_2$, then there is a split of G that is crossed by $\{u, v\}$.*

Proof: Let x be the unique vertex of $V(G_1) \setminus V(G)$. Without loss of generality assume that $x \in W_1$. We define a partition (V_1', V_2') of $V(G)$ with $u \in V_1'$ and $v \in V_2'$ as follows: We set $V_1' := (W_1 \cup V(G_2)) \setminus \{x\}$ and $V_2' := V(G) \setminus V_1' = W_2$. In the following, we show that (V_1', V_2') is a split of G. Since (W_1, W_2) is a split of G_1, $|W_1| \geq 2$ and $|W_2| \geq 2$. Hence, V_2' has size at least two. Moreover, since (G_1, G_2) is a simple decomposition of G, $V(G_2)$ contains at least three vertices. Consequently, V_1' has size at least two. Hence, to show that (V_1', V_2') is a split of G, it remains to show that each vertex of $Y \coloneqq \{w \in V_1' : \exists w' \in V_2'. \{w, w'\} \in E(G)\}$ has the same neighborhood in $V_2' = W_2$. Let y and z be distinct vertices of Y. We show that $N(y) \cap V_2' = N(z) \cap V_2'$. To this end, we distinguish three cases about the location of y and z.

Case 1: Both y and z are contained in $V(G_1)$**.** Then, the statement holds by the fact that (W_1, W_2) is a split of G_1 with $\{y, z\} \subseteq V_1'$ and $V_2' = W_2$.

Case 2: Both y and z are contained in $V(G_2)$**.** Then, the statement holds by the fact that W_2 contains no vertex of $V(G_2)$ and each vertex of $V(G_2)$ with at least one neighbor in $V(G_1) \setminus \{x\}$ has the same neighborhood in $V(G_1) \setminus \{x\}$. The latter holds, since (G_1, G_2) is a simple decomposition of G.

Case 3: Without loss of generality y is contained in $V(G_1)$ and z is contained in $V(G_2)$**.** Since (G_1, G_2) is a simple decomposition of G, z has the same neighborhood in $V(G_1)$ as x. Moreover, since z is contained in Y, z has at least one neighbor in W_2. Hence, x and z have the same neighborhood in $W_2 = V_2'$ as y since (W_1, W_2) is a split of G_1.

Hence, each vertex of Y has the same neighborhood in W_2. This implies that, (V_1', V_2') is a split of G. □

B.2 Existing Solvers

To compute the parameters modular-width, feedback vertex set number, treedepth, and treewidth we use existing solvers. For modular-width we use a solver by Mizutani who implemented a linear-time algorithm by Tedder et al. [61]. The original code can be found at https://github.com/mogproject/modular-decomposition. For feedback vertex set we use the winning solver from the 2016 PACE challenge [12] by Iwata and Imanishi. Their solver first computes a linear-time kernel [35] and then uses an FPT branch-and-bound algorithm [37]. The original code can be found at https://github.com/wata-orz/fvs. For treedepth we use an updated version of the winning solver from the 2020 PACE challenge [45] by Trimble [63]. The original code can be found at https://github.com/jamestrimble/bute. Finally, for treewidth we use a solver by Tamaki [60]. The original code can be found at https://github.com/twalgor/RTW.

B.3 ILP-Formulations for the Remaining Parameters

We use standard ILP-formulations of the respective optimization problems to compute the vertex subset parameters from Section 2.2.3 with the exception of feedback vertex set for which we use the above-mentioned solver by Iwata and Imanishi. To solve these formulations we then use Gurobi [28]. In this section we will explain these ILP-formulations.

Generally, the ILP-formulations for all these problems work similarly, since each of these problems involves finding a minimum-size modulator S to some graph property Π: For each vertex $v \in V(G)$, we introduce a binary variable x_v. The variable x_v should be set to 1 by the ILP if and only if v is contained in the sought minimum-size modulator S.

We now just need to add constraints to ensure that $G - S$ fulfills property Π. In the following, we describe how these constraints look for the individual parameters.

- For the vertex cover number, we add for each $\{u, v\} \in E(G)$ the constraint $x_u + x_v \geq 1$.
- For the r-bounded-degree deletion number, we need to ensure that for each vertex $v \in V(G)$ we include v in S or we include at least $\deg_G(v) - r$ neighbors of v in S. To this end, we add

for each vertex $v \in V(G)$ the constraint

$$\deg_G(v) - r \leq \deg_G(v) \cdot x_v + \sum_{n \in N_G(v)} x_n.$$

- For the cluster vertex deletion number, we add the constraint $1 \leq x_u + x_v + x_w$ for each induced path (u, v, w) in G.
- For distance to cograph, we add the constraint $1 \leq \sum_{v \in V(P)} x_v$ for every induced path P of length 3.
- For 4-path vertex cover we add the constraint $1 \leq \sum_{v \in V(P)} x_v$ for every (not necessarily induced) path P of length 3.

Additionally, for cluster vertex deletion and 4-path vertex cover we used some straightforward reduction rules and added some additional constraints to the ILP for small-degree vertices to slightly speed up the computation of these parameters.

Note that for 4-path vertex cover, we might add $\Theta(n^4)$ constraints to the ILP. To avoid this, we add most of the constraints in a lazy fashion. That is, initially we only add a small subset of the constraints. Afterwards, we let Gurobi find an optimal solution S to the currently added constraints. If S is not a modulator to the respective graph property Π, then at least one (currently not added) constraint is violated. In this case, we add a small number of new constraints to the ILP that are currently violated. This procedure is repeated until, eventually, a minimum-size modulator is found.

B.3.1 Vertex Integrity

We compute the vertex integrity via ILPs for a related parameter, the *r-component order connectivity number* of a graph G. This parameter, denoted by $r\text{-}\mathsf{coc}(G)$, is the size of a smallest modulator of G to the graph property that consists of all graphs G' with $\mathsf{cc}(G') \leq r$ where $\mathsf{cc}(G')$ is the number of vertices in the largest connected component of G' [17]. Recall that $\mathsf{vi}(G) \coloneqq \min\{|X| + \mathsf{cc}(G - X) : X \subseteq V(G)\}$ [2, 17, 26]. An equivalent way to define the vertex integrity via $r\text{-}\mathsf{coc}(G)$ is $\mathsf{vi}(G) \coloneqq \min\{r + r\text{-}\mathsf{coc}(G) : r \in \mathbb{N}\}$. Hence, to compute $\mathsf{vi}(G)$, we simply compute $r\text{-}\mathsf{coc}(G)$ for all values of $r \in [ub]$ in increasing order where ub is an upper bound for $\mathsf{vi}(G)$. As initial upper bound we set $ub \coloneqq \mathsf{cc}(G)$ [27].

To compute $r\text{-}\mathsf{coc}(G)$, we use the above-described general ILP approach and add the constraint $1 \leq \sum_{v \in C} x_v$ for every connected subgraph C of G with exactly $r + 1$ vertices. Since the number of constraints becomes prohibitively large already for intermediate values of r, we add the constraints in a lazy fashion as described above for the 4-path vertex cover number.

After computing $r\text{-}\mathsf{coc}(G)$ for some r, we can update ub by setting $ub := \min(ub, r + r\text{-}\mathsf{coc}(G))$. Once we have $r = ub$, we know that ub must be equal to $\mathsf{vi}(G)$. We can also use $ub - 1 - r$ as an upper bound for the objective value of the ILPs. Since we only care about improving ub until it is equal to $\mathsf{vi}(G)$ it is fine if this leads to some of the ILP formulations being infeasible. We also use $h(G) + 1 - r$ as a lower bound for the objective value of the ILPs [27]. We denote this algorithm as `vi basic`.

Our experiments show that `vi basic` is not fast enough to compete with the solvers for the other parameters. Because of this, we developed several improvements for our implementation in the form of an initial graph reduction or additional constraints for the ILP formulations. We will use this subsection to explain and show the correctness of these improvements.

The first two improvements let us identify some vertices that are either always part of a minimum-size modulator for $r\text{-}\mathsf{coc}(G)$ or never. We can use these to reduce the number of variables we add to each ILP instance.

Lemma 3 (Folklore)**.** *Let G be a graph and let r be some integer. Any vertex $v \in V(G)$ that is contained in a connected component C with $|C| \leq r$ is never part of a minimum-size modulator for $r\text{-}\mathsf{coc}(G)$.*

For Lemma 4, recall that a vi-set of a graph G is any vertex set X with $\mathsf{vi}(G) = |X| + \mathsf{cc}(G - X)$.

Lemma 4. *Let G be a graph and let C be a connected subset of $V(G)$ with $|N[C]| > \mathsf{vi}(G)$. Then each vi-set of G contains at least one vertex of $N[C]$.*

Proof: Assume towards a contradiction that there is a vi-set X of G with $X \cap C = \emptyset$. Note that since C is connected and since $X \cap C = \emptyset$, all vertices in C must be in the same connected component C' of $G - X$. Now, any vertex in $N[C]$ is either in C' or in X which implies $|X| + |C'| \geq |N[C]| > \mathsf{vi}(G)$. But since X is a vi-set of G, $\mathsf{vi}(G) = |X| + \mathsf{cc}(G - X) \geq |X| + |C'|$; a contradiction. □

The special case of Lemma 4 for $|C| = 1$ was originally shown by Drange et al. [17, Theorem 6]. We can use this special case to identify vertices that are part of any vi-set. Since we do not know the exact value of $\mathsf{vi}(G)$ while the algorithm is running, we instead use the current best upper bound. When we would add constraints for a connected vertex set D of size at most $r + 1$ to the ILP, we can keep track of the size of smaller connected subsets $C \subsetneq D$. If the size of the closed neighborhood of such a vertex set C exceeds the current upper bound for $\mathsf{vi}(G)$, we add the constraint only for the vertex set C to the ILP. Due to Lemma 4, this is correct.

Next, we use a result by Gima et al. [26, Lemma 2.2]. For any vertex set $S \subseteq V(G)$, a vertex $v \in S$ is *redundant* if at most one connected component of $G - S$ contains neighbors of v. Gima et al. [26] showed that there always exists a vi-set that contains no redundant vertices. The correctness of this statement follows from the observation that if v is a redundant vertex of S, then removing v from S increases the size of $\mathsf{cc}(G - S)$ by at most one, since all neighbors of v are in at most one connected component of $G - S$. In fact, since we know that S is a vi-set, the size of $\mathsf{cc}(G - S)$ must increase by exactly one. Lemma 5 states some properties that directly follow from this result.

Lemma 5. *Let G be a graph and let S be a vi-set that contains no redundant vertex. Then,*

1. *there is no vertex $v \in V(G)$ with $N[v] \subseteq S$,*
2. *the vertex set S contains no simplicial vertex, and*
3. *for each pair of vertices $v_1, v_2 \in V(G)$ with $N(v_1) \setminus S \subseteq N[v_2]$, we have $v_1 \in S \Rightarrow v_2 \in S$.*

Proof: We prove the statement by contradiction. To this end, we show that S contains a redundant vertex, if at least one of the three properties is not fulfilled.

Item 1: Let v be a vertex with $N[v] \subseteq S$. In this case the neighborhood of v in $G - S$ is empty which implies that by definition, v is a redundant vertex.

Item 2: Let v be a simplicial vertex in S. Since the neighborhood of S in G is a clique, the neighbors of v that remain in $G - S$ are still a clique. Hence, v is a redundant vertex.

Item 3: Let $v_1, v_2 \in V(G)$ be a pair of vertices fulfilling (i) $N(v_1) \setminus S \subseteq N[v_2]$, (ii) $v_1 \in S$, and (iii) $v_2 \notin S$. Then each neighbor of v_1 in $G - S$ is also a neighbor of v_2 in $G - S$. Thus, the entire neighborhood of v_1 in $G - S$ is part of the same connected component. Consequently, v_1 is a redundant vertex. □

It would be difficult to add constraints to the ILP enforcing the property that the solution set is not allowed to contain redundant vertices. Instead, we add constraints that enforce the three properties of Lemma 5. For Item 1, for every vertex $v \in V(G)$ we add the constraint

$$\deg(v) \geq \sum_{w \in N[v]} x_w.$$

We can further use Item 2 to reduce the number of variables in the ILP by not including simplicial vertices. Finally, for Item 3, we do the following: For each two distinct vertices $v_1, v_2 \in V(G)$, let $R \coloneqq N(v_1) \setminus N[v_2]$. Item 3 states that if the vi-set S contains at least all vertices of $R \cup \{v_1\}$, then S should also contain vertex v_2. This can be expressed by the constraint

$$x_{v_1} \leq x_{v_2} + |R| - \sum_{v \in R} x_v.$$

We could theoretically add this constraint for every pair of vertices. However, we decided to only add the constraint for pairs for which R has size at most three.

Our final improvement is based on the following observation. Let r be some number and let v be a cut vertex in G, that is, a vertex that if removed from G increases the number of connected components by at least one. If removing v from G creates a new connected component C with $|C| \leq r$ then it is always better to include v in a modulator for $r\text{-}\mathsf{coc}(G)$ instead of including any vertex in C. To generalize this idea, we use the following definition. To this end, recall that for a vertex set C, $\mathsf{conn}(C)$ denotes the smallest integer k for which there is a vertex set $F \subseteq C$ of size k, such that $G[C] - F$ is not connected.

Definition 4. *For a graph G and an integer r, a family $\mathcal{C}$ of pairs of vertex sets is called an* r-cut decomposition *of G, if each pair $(C, \mathsf{cut}(C)) \in \mathcal{C}$ fulfills the following properties:*

a) C has size at most r,

b) $\mathsf{cut}(C) \subseteq N(C)$,

c) $\mathsf{conn}(C \cup \mathsf{cut}(C)) \geq |\mathsf{cut}(C)|$, and

d) for each two pairs $(C_1, \mathsf{cut}(C_1)), (C_2, \mathsf{cut}(C_2)) \in \mathcal{C}$, we have $C_1 \cap \mathsf{cut}(C_2) = \emptyset$ or $C_2 \subseteq C_1$.

One possible r-cut decomposition consists of the pairs $(\{v\}, D)$ for some fixed vertex $v \in V(G)$ combined with all sets $D = \mathsf{cut}(\{v\})$ which fulfill the requirements of Definition 4. However, in most graphs this will only cover a small part of the graph. Ideally we would like to find an r-cut decomposition, where each vertex of the graph is part of some set C or some set $\mathsf{cut}(C)$. For each pair $(C, \mathsf{cut}(C)) \in \mathcal{C}$, we ideally also want that $N(C) \setminus \mathsf{cut}(C)$ is as small as possible. The reason for this is the following lemma, which describes how an r-cut decomposition can be exploited in the ILP formulation.

Lemma 6. *Let $\mathcal{C}$ be an r-cut decomposition of G. There is a minimum-size modulator S for $r\text{-}\mathsf{coc}(G)$ such that for each pair $(C, \mathsf{cut}(C)) \in \mathcal{C}$, the following conditions hold.*

1. $N(C) \setminus S \subseteq \mathsf{cut}(C) \Rightarrow C \cap S = \emptyset$

2. $(N(C) \setminus S \subseteq \mathsf{cut}(C) \wedge |C| = r) \Rightarrow \mathsf{cut}(C) \subseteq S$

Proof: Let S be a minimum-size modulator for r-$\mathsf{coc}(G)$.

We first show that Item 2 directly follows from Item 1. For this let $(C, \mathsf{cut}(C)) \in \mathcal{C}$ be a pair. If $|C| < r$ or $N(C) \setminus S \not\subseteq \mathsf{cut}(C)$ then Item 2 trivially holds. This means we can now assume that the left side of Item 2 is true. This also means that the left side of Item 1 must be true, which implies that the right side of Item 1 is true. If C is a connected vertex set in $G - S$, then we must have $N(C) \subseteq S$ and in particular $\mathsf{cut}(C) \subseteq S$ since we know that S is a modulator for r-$\mathsf{coc}(G)$ and $|C| = r$. If C is not connected in $G - S$, then we know that $C \cup \mathsf{cut}(C)$ is connected in G with $\mathsf{conn}(C \cup \mathsf{cut}(C)) \geq |\mathsf{cut}(C)|$ (see Property c)). This implies that $\mathsf{cut}(C) \subseteq S$ since at least that many vertices must be removed in order to disconnect C in $G - S$. Hence, Item 2 follows from Item 1 as long as S is a modulator for r-$\mathsf{coc}(G)$.

If Item 1 holds for each pair $(C, \mathsf{cut}(C)) \in \mathcal{C}$ with respect to S, the statement follows. Hence, assume that this is not the case and let $(C, \mathsf{cut}(C))$ be a pair of $\mathcal{C}$ that does not fulfill Item 1 with respect to S.

Let S' be a vertex set obtained from S by firstly removing all vertices of C and secondly adding $\min(|\mathsf{cut}(C)|, |C \cap S|)$ vertices of $\mathsf{cut}(C) \setminus S$ if $N(C) \setminus S \subseteq \mathsf{cut}(C)$. In the following, we show that S' is a minimum-size modulator for r-$\mathsf{coc}(G)$ such that strictly more pairs of $\mathcal{C}$ fulfill Item 1. First, we show that S' is a minimum-size modulator for r-$\mathsf{coc}(G)$. By construction, $|S'| \leq |S|$. Hence, we only have to show that S' is a modulator for r-$\mathsf{coc}(G)$.

Without loss of generality, we assume that $N(C) \setminus S \subseteq \mathsf{cut}(C)$ holds since this is the only case where $S \neq S'$. If $|(C \cup \mathsf{cut}(C)) \cap S| \geq |\mathsf{cut}(C)|$ all vertices of $\mathsf{cut}(C)$ are contained in S' and all vertices of C are removed from S, that is, $S' \cap C = \emptyset$. Observe that $|S'| \leq |S|$ in this case and $\mathsf{cc}(G - S) \leq r$ still holds since $|C| \leq r$. Also, notice if we have $|C| = r$ the condition $|(C \cup \mathsf{cut}(C)) \cap S| \geq |\mathsf{cut}(C)|$ must always be true because $\mathsf{conn}(C \cup \mathsf{cut}(C)) \geq |\mathsf{cut}(C)|$. Hence, we have $\mathsf{cut}(C) \subseteq S$.

Otherwise, if $|(C \cup \mathsf{cut}(C)) \cap S| < |\mathsf{cut}(C)|$, we know that the remaining vertices in $(C \cup \mathsf{cut}(C)) \setminus S$ are all part of the same connected component in $G - S$. Hence, adding any $|C \cap S|$ vertices from $\mathsf{cut}(C) \setminus S$ to S and then removing all vertices in C from S preserves the property $\mathsf{cc}(G - S) \leq r$ and $|S'| = |S|$.

By definition of S', $(C, \mathsf{cut}(C))$ trivially fulfills Item 1 with respect to S'. Next, we show that for each pair $(D, \mathsf{cut}(D)) \in \mathcal{C}$, $(D, \mathsf{cut}(D))$ fulfills the right side of Item 1 with respect to S' if $(D, \mathsf{cut}(D))$ fulfills the right side of Item 1 with respect to S. This then implies that the above-described procedure can be recursively applied and eventually constructs a minimum-size modulator S^* for r-$\mathsf{coc}(G)$ such that each pair of $\mathcal{C}$ fulfills Item 1 (and thus also Item 1) with respect to S^*, since with each application, at least one additional pair of $\mathcal{C}$ fulfills Item 1 with respect to the newly constructed modulator.

Let $(D, \mathsf{cut}(D)) \in \mathcal{C}$, such that $(D, \mathsf{cut}(D))$ fulfills the right side of Item 1 with respect to S. We show that $(D, \mathsf{cut}(D))$ fulfills Item 1 with respect to S'. Assume towards a contradiction that this is not the case. We distinguish three cases.

Case 1: $C \subseteq D$. Recall that $(C, \mathsf{cut}(C))$ does not fulfill the right side of Item 1 with respect to S. Hence, $(D, \mathsf{cut}(D))$ does not fulfill the right side of Item 1 with respect to S, since $C \subseteq D$; a contradiction to the choice of $(D, \mathsf{cut}(D))$.

Case 2: $D \subsetneq C$. Since S' contains no vertex of C, S' contains no vertex of D. This implies that $(D, \mathsf{cut}(D))$ fulfills the right side of Item 2 with respect to S'; a contradiction.

Case 3: $D \not\subseteq C$ and $C \not\subseteq D$. In this case, $C \cap \mathsf{cut}(D) = \emptyset$ and $D \cap \mathsf{cut}(C) = \emptyset$ hold (see Property d)). Hence, we never add vertices in D to S, which implies that $(D, \mathsf{cut}(D))$ fulfills the right side of Item 2 with respect to S', since $(D, \mathsf{cut}(D))$ fulfills the right side of Item 2 with respect to S; a contradiction to the choice of $(D, \mathsf{cut}(D))$. $\square$

To compute an r-cut decomposition we do the following: Let $\mathcal{D}$, $\mathcal{S}$, and $\mathcal{Q}$ be families of subsets of $V(G)$. Initially, $\mathcal{D}$ and $\mathcal{S}$ both contain the vertex set of every connected component in G, and $\mathcal{Q}$ is empty. We now choose some subset $S \in \mathcal{S}$ such that $G[S]$ is not a clique and find a set of vertices Q that is a minimum vertex cut of $G[S]$. We then remove S from $\mathcal{S}$, add the connected components of $G[S \setminus Q]$ to $\mathcal{D}$ and $\mathcal{S}$, and add Q to $\mathcal{Q}$. We repeat this procedure until all subsets in $\mathcal{S}$ induce a clique in G. Finally, for each two vertex sets $D \in \mathcal{D}, Q \in \mathcal{Q}$, we add (D, Q) to $\mathcal{C}$ if (i) $|D| \leq r$, (ii) $Q \subseteq N_G(D)$, and (iii) $\mathsf{conn}(D \cup Q) \geq |Q|$. A composition created in that way is called an *r-min-cut decomposition*. Next, we verify that an *r-min-cut decomposition* is indeed an r-cut decomposition.

Lemma 7. *An r-min-cut decomposition is an r-cut decomposition.*

Proof: The Properties a), b) and c) of Definition 4 are trivially correct based on the three conditions under which we add a pair (D, Q) to the construction of the r-min-cut decomposition $\mathcal{C}$.

In the following, we show Property d) of Definition 4. That is, we have to show that $C_1 \cap \mathsf{cut}(C_2) = \emptyset$ or $C_2 \subseteq C_1$ for any two distinct vertex pairs $(C_1, \mathsf{cut}(C_1))$ and $(C_2, \mathsf{cut}(C_2))$ in $\mathcal{C}$. To this end, we consider an auxiliary rooted forest using $\mathcal{D}$ and $\mathcal{Q}$.[7] Each node N in any tree of the forest corresponds to a vertex set. More precisely, the nodes of this forest are exactly the vertex sets from $\mathcal{D} \cup \mathcal{Q}$. For each connected component X of G, the forest contains a rooted tree with root C. Furthermore, the children of a node N are the connected components of $G[N] - Q$ and the minimum vertex cut $Q \in \mathcal{Q}$ of $G[N]$. Observe that if some node P is an ancestor of some other node N then we have $N \subseteq P$ and if they are unrelated we must have $N \cap P = \emptyset$.

Let $(C_1, \mathsf{cut}(C_1))$ and $(C_2, \mathsf{cut}(C_2))$ be distinct vertex pairs of $\mathcal{C}$. We now show that $C_1 \cap \mathsf{cut}(C_2) = \emptyset$ or $C_2 \subseteq C_1$. Assume towards a contradiction that this is not the case, that is, $C_1 \cap \mathsf{cut}(C_2) \neq \emptyset$ and $C_2 \not\subseteq C_1$. By construction of the r-min-cut decomposition $\mathcal{C}$, $\mathsf{cut}(C_2)$ is contained in $\mathcal{Q}$ and thus has no children in its unique tree. Thus, C_1 is an ancestor of $\mathsf{cut}(C_2)$. Since $\mathsf{cut}(C_2) \subseteq N_G(C_2)$, C_2 was added to $\mathcal{D}$ at the earliest when $\mathsf{cut}(C_2)$ was first discovered as a minimum vertex cut. This implies that C_2 is contained in a subtree which is rooted at some sibling of $\mathsf{cut}(C_2)$. Consequently, $C_2 \subseteq C_1$; a contradiction.

This then implies that an r-min-cut decomposition is an r-cut decomposition. $\square$

Note that the computations of the families $\mathcal{D}$ and $\mathcal{Q}$ are independent of the choice of r. Consequently, we have to compute these families only once and can reuse them for each value of r. For each concrete value of r, where we want to compute an r-cut decomposition, we can then just select the sets from $\mathcal{D}$ of size at most r. To actually use an r-cut decomposition as a speed-up for our implementation, we add the following constraint representing Item 1 from Lemma 6 to the ILP for each pair $(C, \mathsf{cut}(C)) \in \mathcal{C}$:

$$n \cdot \left(|N(C) \setminus \mathsf{cut}(C)| - \sum_{v \in N(C) \setminus \mathsf{cut}(C)} x_v \right) \geq \sum_{v \in C} x_v.$$

If $|C| = r$ we also add the following constraint representing Item 2:

$$n \cdot \left(|N(C) \setminus \mathsf{cut}(C)| - \sum_{v \in N(C) \setminus \mathsf{cut}(C)} x_v \right) \geq |\mathsf{cut}(C)| - \sum_{v \in \mathsf{cut}(C)} x_v.$$

[7] This forest is not computed by our algorithm and just serves the purpose to prove the desired statement.

Additionally, if $N(C) \setminus \mathsf{cut}(C) = \emptyset$, then we do not need to add these constraints and can instead use Items 1 and 2 during the initial graph reduction.

Finally, to ensure that we can use both our main speed-up strategies simultaneously, we need to verify that there exists a vi-set that not just fulfills the properties of Lemma 6 but also does not contain redundant vertices. In the following, we show that this is the case.

Theorem 1. *Let G be a graph. Moreover, for each $r \in [|V(G)|]$, let $\mathcal{C}_r$ be an r-cut decomposition. There exists a vi-set S that fulfills the following conditions:*

1. *S contains no redundant vertices.*
2. *There exists an integer r, such that (i) S is a minimum-size modulator for r-$\mathsf{coc}(G)$ and (ii) S together with the r-cut decomposition $\mathcal{C}_r$ fulfill Items 1 and 2 from Lemma 6.*

Proof: Let r be the largest integer such that there exists a vi-set S, such that S is a minimum-size modulator for r-$\mathsf{coc}(G)$. Due to the definition of vertex integrity and Lemma 6, we can assume without loss of generality that $\mathcal{C}_r$ fulfills Items 1 and 2 from Lemma 6 with respect to S.

If S contains no redundant vertex, we are done. Hence, let us assume that there is a redundant vertex v in S. Consider the vertex set $S' := S \setminus \{v\}$. Since v is a redundant vertex, there is at most one connected component C_v in $G - S$ where v has neighbors in. Moreover, since S is a modulator for r-$\mathsf{coc}(G)$, C_v has size at most r. Hence, S' is a modulator for $(r+1)$-$\mathsf{coc}(G)$, since no component of $G - S'$ has size more than $r+1$. This implies that $|S'| + \mathsf{cc}(G - S') \leq |S| + \mathsf{cc}(G - S) = \mathsf{vi}(G)$. Recall that by definition of vi, $\mathsf{vi}(G) \leq |S'| + \mathsf{cc}(G - S')$. Hence, $|S'| + \mathsf{cc}(G - S') = \mathsf{vi}(G)$, which implies that S' is (i) a vi-set and (ii) a minimum-size modulator for $r + 1$-$\mathsf{coc}(G)$. □

C Running Time Comparison

Finally, we briefly discuss the time needed to compute the parameters. Table 10 shows the average, median, and 90th percentile running times for the considered parameter on the 59 instances where all parameters could be computed. Unsurprisingly, the running time for the polynomial-time computable parameters is negligible with the exception of the splitwidth and the Dilworth number which are still easily computable in the given time frame. Among the parameters that are NP-hard to compute, the vertex cover number can be computed quite fast compared to the other parameters of which the currently hardest parameters seem to be distance to cograph, treewidth, treedepth, and vertex integrity.

In Figure 5 we compare these parameters on all 144 graphs. We also included vertex cover as a contrast since it is the easiest of the parameters that are NP-hard to compute. The plot shows for each time t on how many graphs each parameter could be calculated in at most t seconds. We can see that it is a lot more difficult to compute dco, tw, tdp, and vi compared to vc where almost all instances could be solved in less than one second. However, even the dco, tw algorithms are quite fast compared to the algorithms for tdp and vi. This shows that it is important to develop faster algorithms for computing these parameters.

In Figure 5 we also compare the two different algorithms for vi described in Section B.3.1. Recall that the algorithm denoted as `vi basic` uses a simple ILP-formulation while the algorithm denoted as `vi` is an improved version with additional reduction rules and ILP constraints. We see that the improved version of the algorithm is significantly faster than the basic version, which could only solve 42 instances. Moreover, each instance solved by `vi basic` within the time limit is solved by `vi` in less than 800 seconds.

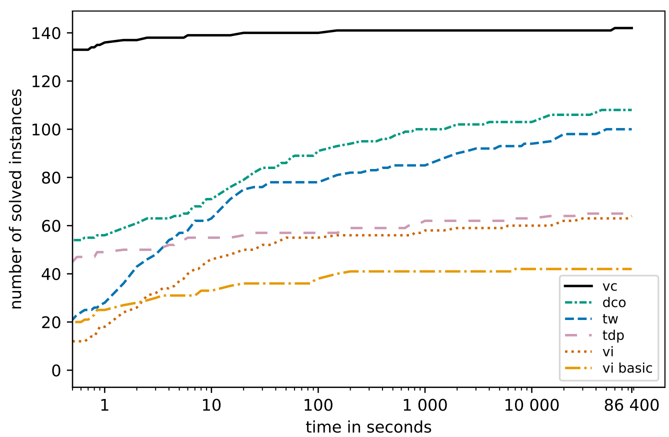


Figure 5: Running time comparison between the parameters vc, dco, tw, tdp and the basic and improved versions of the vi algorithm. For each parameter and each time t the figure shows on how many of the 144 instances that parameter could be computed in at most t seconds.

Table 9: Sizes and some additional parameter values for all 85 *hard* graphs for which at least one parameter could not be calculated. A complete overview is available in the online repository.

name	n	m	Δ	vc	tw
flying	48	284	22	36	20
ar_m8x8	64	112	4	32	8
moreno_highschool	70	274	19	48	13
polbooks	105	441	25	62	13
adjnoun_adjacency	112	425	49	59	28
infect-hyper	113	2 196	98	90	73
football	115	613	12	94	34
flower_4_1	129	372	8	80	36
email-enron-only	143	623	42	86	21
digg_n	166	1 375	70	86	41
rel5	172	646	44	33	15
PACE_2021_jena_5	184	1 367	53	136	23
arenas-jazz	198	2 742	100	158	51
scotland	228	358	13	89	14
can_229	229	774	12	162	21
saylr1	238	445	4	119	14
dwt_245	245	608	12	147	11
PACE_2021_strings_2	248	2 451	38	209	23
ar_m16x16	256	480	4	128	16
econ-wm1	258	2 389	106	115	57
c260	260	390	3	140	20
as_internet_topology_n	266	6 300	223	110	66
PACE_2021_colors_1	330	2 256	43	260	33
usair97	332	2 126	139	149	33
maier_facebook_friends	348	1 988	63	218	26
poisson2D	367	1 025	8	248	20
blogcatalog_n	372	2 627	188	152	71
BNU1_0025884_1_DTI_DS00446	425	4 817	95	314	67
celegans_metabolic	453	2 025	237	249	32
n3c6-b2	455	1 313	16	91	67
youtube_n	479	2 428	80	245	107
power-494-bus	494	586	9	216	7
us_patents_n	502	1 189	24	234	49
NL_Mixed_2016	514	1 618	26	341	17
wikiconflicts_n	586	5 704	297	228	66
dwt_607	607	2 262	13	405	32
social_location_gowalla_n	654	2 104	212	190	52
nos6	675	1 290	4	337	15
ppi_interolog_human_n	677	1 611	231	131	30
wordnet_n	692	1 711	517	383	27
can_715	715	2 975	104	507	41
internet_top_pop_kdl	754	895	7	369	7
socfb-Caltech36	769	16 656	248	536	325
BNU1_0025913_1_DTI_DS00833	780	9 336	113	587	116
reptilia-tortoise-network-fi	787	1 197	17	396	13
copenhagen_fb_friends	800	6 418	101	531	249
cpan_perl_module_users	839	2 112	327	116	34
kegg_metabolic_aae	926	2 334	188	441	34
flixster_n	941	7 745	150	600	158
flickr_n	955	9 448	207	510	228
socfb-Reed98	962	18 812	313	687	380
bn-mouse-kasthuri_graph_v4	1 029	1 559	123	171	35
arenas-email	1 133	5 451	71	594	197
euroroad	1 174	1 417	10	571	13
plasmodium_falciparum_3D7	1 229	2 448	51	450	119
escherichia_coli_K12_MG1655	1 258	1 875	58	452	48
jagmesh9	1 349	3 876	6	899	21
socfb-Simmons81	1 518	32 988	300	1 106	576
xenopus_leavis	1 596	2 054	640	212	12
collins_yeast	1 622	9 070	127	1 004	67
bcspwr09	1 723	2 394	14	775	10
biological_pathways_kegg	1 729	1 924	18	782	12
ppi-yeast	1 846	2 203	56	626	43
plant_pol_robertson	1 884	15 255	300	411	273
blckhole	2 121	6 370	19	1 423	75
nd_2453_47659	2 453	47 659	444	1 429	117
socfb-Trinity100	2 613	111 996	404	2 155	1 456
dwt_2680	2 680	11 173	18	1 977	43
sstmodel	2 730	9 702	17	1 911	37
route_views	3 015	5 156	590	566	39
lshp3466	3 466	10 215	6	2 310	72
california	4 271	8 909	175	892	235
nyc_restaurant_checkin	4 936	13 472	88	2 010	853
opsahl-powergrid	4 941	6 594	19	2 203	17
grqc	5 241	14 484	81	2 783	174
c6000	6 000	9 000	3	3 176	135
rattus_norvegicus	6 147	10 387	1 305	847	100
nyc_restaurant_tips	6 410	9 472	196	2 491	340
erdos992	6 927	11 850	507	474	148
ppi-dmela	7 393	25 569	190	2 630	1 037
us_roads_DC	9 559	14 841	6	5 068	34
hepth	9 875	25 973	65	4 981	725
hepph	12 006	118 489	491	7 012	1 230
astroph	18 771	198 050	504	12 012	3 329
condmat	23 133	93 439	279	13 521	2 004

Table 10: Average, median, and 90th percentile running times (in ms) for computing the parameters.

k	avg. time	median time	90th p. time
Δ	0.0	0.0	0.0
h-index	0.0	0.0	0.0
d	0.4	0.0	1.0
c	1.6	0.0	4.0
nd	3.6	0.0	9.2
2-core	2.0	1.0	4.0
3-core	1.5	1.0	4.0
mw	2.8	2.0	5.0
γ	43.3	9.0	122.2
vc	11.8	9.0	22.4
2-bdd	145.6	9.0	192.6
1-bdd	193.5	12.0	211.2
sw	122.3	17.0	284.0
∇	46.2	20.0	146.2
cvd	641.6	22.0	712.4
tdp	154 108.7	43.0	16 446.2
fvs	126.3	103.0	161.8
dco	11 738.8	125.0	9 082.4
4-pvc	925.4	385.0	1 599.8
tw	3 402.6	1 284.0	10 962.0
vi	2 529 051.8	2 183.0	189 970.0