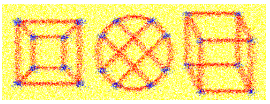
Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 24, no. 3, pp. 215–245 (2020)
DOI: 10.7155/jgaa.00528

Parameterized Complexity of Safe Set

Rémy Belmonte[1] *Tesshu Hanaka*[2] *Ioannis Katsikarelis*[3]
Michael Lampis[3] *Hirotaka Ono*[4] *Yota Otachi*[4]

[1]The University of Electro-Communications, Chofu, Tokyo, Japan
[2]Chuo University, Bunkyo-ku, Tokyo, Japan
[3]Université Paris-Dauphine, PSL University, CNRS,
LAMSADE, 75016 Paris, France
[4]Nagoya University, Nagoya, Japan

Abstract

In this paper we study the problem of finding a small safe set S in a graph G, i.e., a non-empty set of vertices such that no connected component of $G[S]$ is adjacent to a larger component in $G - S$. We enhance our understanding of the problem from the viewpoint of parameterized complexity by showing that (1) the problem is W[2]-hard when parameterized by the pathwidth pw and cannot be solved in time $n^{o(\mathsf{pw})}$ unless the ETH is false, (2) it admits no polynomial kernel parameterized by the vertex cover number vc unless PH $= \Sigma_3^{\mathrm{p}}$, but (3) it is fixed-parameter tractable (FPT) when parameterized by the neighborhood diversity nd, and (4) it can be solved in time $n^{f(\mathsf{cw})}$ for some double exponential function f where cw is the clique-width. We also present (5) a faster fixed-parameter algorithm when parameterized by the solution size.

Submitted:	Reviewed:	Revised:	Accepted:
June 2019	December 2019	February 2019	April 2019
	Final:	Published:	
	April 2019	April 2019	
Article type:		Communicated by:	
Regular Paper		A. Wolff	

Partially supported by JSPS and MAEDI under the Japan-France Integrated Action Program (SAKURA) Project GRAPA 38593YJ, and by JSPS/MEXT KAKENHI Grant Numbers JP24106004, JP17H01698, JP18K11157, JP18K11168, JP18K11169, JP18H04091, 18H06469.
A preliminary version of this paper appeared in the proceedings of the 11th International Conference on Algorithms and Complexity (CIAC 2019), Lecture Notes in Computer Science 11485 (2019) 38–49, doi:10.1007/978-3-030-17402-6_4.

E-mail addresses: remy.belmonte@uec.ac.jp (Rémy Belmonte) hanaka.91t@g.chuo-u.ac.jp (Tesshu Hanaka) ioannis.katsikarelis@dauphine.fr (Ioannis Katsikarelis) michail.lampis@dauphine.fr (Michael Lampis) ono@nagoya-u.jp (Hirotaka Ono) otachi@nagoya-u.jp (Yota Otachi)

1 Introduction

Let G be a graph. We denote the vertex set of G by $V(G)$ and the edge set of G by $E(G)$. For a subset S of $V(G)$, we denote by $G[S]$ the subgraph of G induced by S, and by $G - S$ the subgraph induced by $V(G) \setminus S$. If $G[S]$ is connected, we also say that S is connected. A vertex set $C \subseteq V(G)$ is a *component* of G if C is an inclusion-wise maximal connected set. Two vertex sets $A, B \subseteq V(G)$ are *adjacent* if there is an edge $\{a, b\} \in E(G)$ such that $a \in A$ and $b \in B$. Now, a non-empty vertex set $S \subseteq V(G)$ of G is a *safe set* if no connected component C of $G[S]$ has an adjacent connected component D of $G - S$ with $|C| < |D|$. A safe set S of G is a *connected safe set* if $G[S]$ is connected.

The *safe number* $\mathsf{s}(G)$ of G is the size of a minimum safe set of G, and the *connected safe number* $\mathsf{cs}(G)$ of G is the size of a minimum connected safe set of G. It is known that $\mathsf{s}(G) \leq \mathsf{cs}(G) \leq 2 \cdot \mathsf{s}(G) - 1$ [20].

Fujita, MacGillivray, and Sakuma [20] introduced the concept of safe sets motivated by a facility location problem that can be used to design a safe evacuation plan. Subsequently, Bapat et al. [2] observed that a safe set can control the consensus of the underlying network with a majority of each part of the subnetwork induced by a component in the safe set and an adjacent component in the complement, thus the minimum size of a safe set can be used as a vulnerability measure of the network. That is, contrary to its name, having a small safe set could be unsafe for a network. Both the combinatorial and algorithmic aspects of the safe set problem have already been extensively studied [1, 14, 18, 19].

In this paper, we study the problems of finding a small safe set and a small connected safe set mainly from the parameterized-complexity point of view. We show a number of both tractability and intractability results, that highlight the difference in complexity that this parameter exhibits compared to similarly defined vulnerability parameters.

Our results

Our main results are the following (see also Figure 1).

1. Both problems are W[2]-hard parameterized by the pathwidth pw of the input graph and cannot be solved in time $n^{o(\mathsf{pw})}$ unless the Exponential Time Hypothesis (ETH) fails, where n is the number of vertices of the input graph.
2. They do not admit kernels of polynomial size when parameterized by the vertex cover number of the input graph unless $\mathrm{PH} = \Sigma_3^{\mathrm{p}}$.
3. Both problems are fixed-parameter tractable (FPT) when parameterized by the neighborhood diversity.
4. Both problems can be solved in XP time when parameterized by the clique-width.[1]

[1] By XP time, we mean $O(f(k) \cdot n^{g(k)})$ time, where n is the input size, k is the parameter, and f and g are some computable functions.

5. Both problems can be solved in $k^{O(k)} \cdot n^{O(1)}$ time when parameterized by the solution size k.

The W[2]-hardness parameterized by the pathwidth complements the known fixed-parameter tractability when parameterized by the solution size [1], since for every graph the size of the solution is at least half of the graph's pathwidth (see Section 2). The $n^{o(\mathsf{pw})}$-time lower bound is tight since there is an $n^{O(\mathsf{tw})}$-time algorithm [1], where tw is the treewidth. The second result also implies that there is no polynomial kernel parameterized by the solution size, as the vertex cover number is an upper bound on the size of the solution. The third result marks the first fixed-parameter algorithm (FPT algorithm, for short) by a parameter that is incomparable to the solution size. The fourth result implies XP-time solvability for all the parameters mentioned in this paper and extends the result for treewidth by Águeda et al. [1]. The fifth result improves the known algorithm [1] that uses Courcelle's theorem.

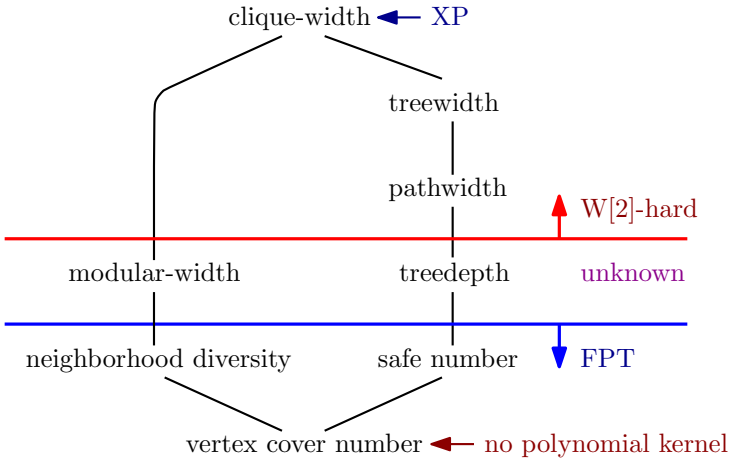

Figure 1: Graph parameters and an overview of the parameterized complexity landscape for SS and CSS. Connections between two parameters imply the existence of a function in the one above (being in this sense more general) that lowerbounds the one below.

Previous work

In the first paper on this topic, Fujita et al. [20] showed that the problems are NP-complete in general. Their hardness proof implies that the safe number and connected safe number are hard to approximate within a factor of 1.3606 [1]. They also showed that a minimum connected safe set in a tree can be found in linear time.

Bapat et al. [2] considered the problems on vertex-weighted graphs, where the problems are naturally generalized. They showed that these are weakly NP-complete even for weighted stars (thus for all parameters generalizing vertex

cover number and graph classes like interval and split graphs). On the other hand, they showed that the problems can be solved in $O(n^3)$ time for weighted paths. Ehard and Rautenbach [14] presented a PTAS for the problem of computing the connected safe number of a weighted tree. Fujita et al. [19] showed among other results that the problems can be solved in linear time for weighted cycles.

Águeda et al. [1] studied the original unweighted versions of the two problems. They presented an XP-time algorithm for graphs of bounded treewidth and showed that the problems can be solved in polynomial time for interval graphs, while they are NP-complete for split and bipartite planar graphs of maximum degree at most 7. Observing that the treewidth of a graph is bounded by a function of its safe number, they also showed that the problems are FPT parameterized by the solution size.

2 Definitions and Preliminaries

The problems studied in this paper are formalized as follows:

Safe Set (SS)
Input: A graph G and an integer k.
Question: Is there a safe set $S \subseteq V(G)$ of size at most k?

Connected Safe Set (CSS)
Input: A graph G and an integer k.
Question: Is there a connected safe set $S \subseteq V(G)$ of size at most k?

In our positive results, we also study the optimization versions of both problems.

We assume that the reader is familiar with the concepts relating to fixed-parameter tractability. See the book by Cygan et al. [9] (and references therein) for the definitions of relevant notions in parameterized complexity theory. We let $N[v]$ denote the *closed neighborhood* of vertex v, i.e., the set containing v and all vertices adjacent to it. Furthermore, for a positive integer k, we denote the set $\{1, \ldots, k\}$ by $[k]$.

Graph parameters

We recall relationships among some graph parameters used in this paper, and give a map of the parameters with the results. The graph parameters we explicitly use in this paper are vertex cover number vc, pathwidth pw, neighborhood diversity nd, and clique-width cw. They are situated in the hierarchy of well-studied graph parameters, along with safe number s, as depicted in Figure 1. (The parameters are defined in the respective sections.)

The treewidth $\mathsf{tw}(G)$, pathwidth $\mathsf{pw}(G)$, treedepth $\mathsf{td}(G)$, and vertex cover number $\mathsf{vc}(G)$ of a graph G can be defined as the minimum of the maximum clique-size (-1 for $\mathsf{tw}(G)$, $\mathsf{pw}(G)$, and $\mathsf{vc}(G)$) among all supergraphs of G that are of type chordal, interval, trivially perfect and threshold, respectively. This gives us $\mathsf{tw}(G) \leq \mathsf{pw}(G) \leq \mathsf{td}(G) - 1 \leq \mathsf{vc}(G)$ for every graph G. One can easily

see that $\mathsf{s}(G) \leq \mathsf{vc}(G)$ and $\mathsf{td}(G) \leq 2\mathsf{s}(G)$. (See also the discussion by Águeda et al. [1].) This justifies the hierarchical relationships among them in Figure 1.

Although modular-width $\mathsf{mw}(G)$ and neighborhood diversity $\mathsf{nd}(G)$ are incomparable to most of the parameters mentioned above, all these parameters are generalized by clique-width, while the vertex cover number is their specialization.

More formally, a *path decomposition* of a graph G is a sequence $(X_1, \ldots, X_r)$ of subsets (called *bags*) of $V(G)$ such that:

1. $\bigcup_{1 \leq i \leq r} X_i = V(G)$,
2. for each $e \in E(G)$, there exists X_i such that $e \subseteq X_i$, and
3. each $v \in V(G)$ appears in consecutive sets in the sequence.

The *width* of a path decomposition $(X_1, \ldots, X_r)$ is $\max_{1 \leq i \leq r} |X_i|$. The *path-width* $\mathsf{pw}(G)$ of G is the minimum width over all path decompositions of G.

Next, the clique-width of a graph measures the simplicity of the graph as the number of labels required to construct the graph [8]. In a vertex-labeled graph, an *i-vertex* is a vertex of label i . A *c-expression* is a rooted binary tree such that:

- each leaf has label $\circ_i$ for some $i \in [c]$,
- each non-leaf node with two children has label $\cup$, and
- each non-leaf node with exactly one child has label $\rho_{i,j}$ or $\eta_{i,j}$ $(i, j \in [c], i \neq j)$.

Each node in a c-expression represents a vertex-labeled graph as follows:

- a $\circ_i$-node represents a single-vertex graph with one i-vertex;
- a $\cup$-node represents the disjoint union of the labeled graphs represented by its children;
- a $\rho_{i,j}$-node represents the labeled graph obtained from the one represented by its child by replacing the labels of all i-vertices with j;
- an $\eta_{i,j}$-node represents the labeled graph obtained from the one represented by its child by adding all possible edges between the i-vertices and the j-vertices.

A c-expression gives the graph represented by its root. The *clique-width* $\mathsf{cw}(G)$ of G is the minimum integer c such that some c-expression represents a graph isomorphic to G. It is known that for any constant c, one can compute a $(2^{c+1} - 1)$-expression of a graph of clique-width c in $O(n^3)$ time [22, 32, 33].

A c-expression of a graph is *irredundant* if for each edge $\{u, v\}$, there is exactly one node $\eta_{i,j}$ that adds the edge between u and v. A c-expression of a graph can be transformed into an irredundant one with $O(n)$ nodes in linear time [8]. In what follows, we assume that c-expressions are irredundant.

Some observations

Here we list some simple observations that are not directly related to our main results yet may still be of interest.

Observation 2.1 *Given a graph G, the safe number and the connected safe number can be approximated in polynomial time within a factor of $\mathsf{s}(G)+1$.*

Proof: If a graph is not connected, we can apply the following algorithm for each component and output a found set of the minimum size. Hence we assume that G is connected in what follows.

We try all values $s \in \{1, \ldots, |V(G)|\}$ as the hypothetical value of the safe number $\mathsf{s}(G)$. (One may binary-search the value s to have a polynomial speed-up.) We start with an arbitrary connected set S such that $|S| = s+1$. Assume that $G-S$ has a component C of size more than s. Let $C' \subseteq C$ be a connected set of size $s+1$ that has a neighbor in S. We put all the vertices of C' into S. We repeat this process until every component of $G-S$ has size at most s.

Observe that the resulting set S is a safe set since $G[S]$ is connected and $G-S$ has no component of size larger than $|S| \geq s$. Observe that each time we added some vertices to S (even at the initialization of S), we added a connected set X of size $s+1$. Thus, for any safe set S of size s, it holds that $S \cap X \neq \emptyset$. This means that we add at most $s+1$ vertices to include each vertex in an optimal solution. Therefore the output is a connected safe set of size at most $\mathsf{s}(G) \cdot (\mathsf{s}(G)+1)$ provided that the guess s is correct. □

Recall that SS and CSS are NP-complete for graphs of bounded maximum degree [1], and do not admit a polynomial kernel even for connected graphs when parameterized only by the solution size (Corollary 4.2). The following observation says that a large connected graph of small maximum degree does not contain a small safe set. This implies that when parameterized by both the solution size and the maximum degree $\Delta(G)$, the problems admit FPT algorithms for general graphs and polynomial kernels for connected graphs.

Observation 2.2 *For every connected graph G, $|V(G)| \leq \mathsf{s}(G) + \mathsf{s}(G)^2 \cdot \Delta(G)$.*

Proof: If S is a safe set, then there are at most $|S| \cdot \Delta(G)$ components in $G-S$ and each of them has size at most $|S|$. □

A related observation is that both SS and CSS admit kernels of order roughly $k^{O(k)}$. Here we need the (already mentioned) fact that $\mathsf{td}(G) \leq 2\mathsf{s}(G)$. The treedepth $\mathsf{td}(G)$ of a graph G is defined recursively as follows:

- if G has one vertex only, then $\mathsf{td}(G) = 1$;
- if G has the components $C_1, \ldots, C_q$ with $q \geq 2$, then
$$\mathsf{td}(G) = \max_{1 \leq i \leq q} \mathsf{td}(G[C_i]);$$
- otherwise, $\mathsf{td}(G) = 1 + \min_{v \in V(G)} \mathsf{td}(G) - \{v\}$.

Observation 2.3 $\mathsf{td}(G) \leq 2\mathsf{s}(G)$ *for every graph* G.

Proof: Let S be a safe set of G with size $k := \mathsf{s}(G)$. Observe that $\mathsf{td}(G) \leq k + \mathsf{td}(G - S)$. Now since S is a safe set, each component of $G - S$ has size at most k. A graph of k vertices has treedepth at most k, and thus $\mathsf{td}(G - S) \leq k$. □

Observation 2.4 *There exists a polynomial-time algorithm which, given a connected instance of SS, or CSS, produces an equivalent instance with at most* $\mathsf{s}(G) + (2\mathsf{s}(G))^{2\mathsf{s}(G)}$ *vertices.*

Proof: Let $k := \mathsf{s}(G)$. We first observe that if a vertex u has degree at least $2k$ then any safe set (connected or not) must contain u, because otherwise u together with the (at least k) of its neighbors which are not in the solution form a component of size $k+1$ or more. Therefore, if G has $k+1$ or more vertices of degree at least $2k$ we immediately produce a trivial NO instance. Let H be the set of vertices of degree at least $2k$.

If we have $|H| \leq k$ we need to show that $|V(G) \setminus H| \leq (2k)^{2k}$ in any YES instance. However, in a YES instance, $\mathsf{td}(G) \leq 2k$ by Observation 2.3. This implies that $\mathsf{td}(G - H) \leq 2k$. Furthermore, the maximum degree of $G - H$ is at most $2k$. It is now an easy observation that a graph with treedepth $2k$ and maximum degree $2k$ cannot have more than $(2k)^{2k}$ vertices. So, if $V(G) \setminus H$ is larger than that we reject, otherwise we have the promised bound. □

Observation 2.4 is interesting as a demonstration of the algorithmic difficulties posed by the fact that our problems are not closed under taking induced subgraphs (unlike most graph parameters). Since we succeed in bounding the number of vertices of high degree, one may be tempted to believe that Observation 2.2 would then allow us to obtain a polynomial kernel (as in the remainder of the graph $\Delta = O(k)$). This does not work, as a safe set of the original graph does not necessarily remain a safe set of the graph induced by low-degree vertices. As a result, we are forced to argue indirectly through a more well-behaved parameter (treedepth). The results of Section 4 indicate that this is most likely to be inevitable, as the problem parameterized by the solution size does not admit a polynomial kernel unless $\mathrm{PH} = \Sigma_3^{\mathrm{p}}$.

Similar graph parameters

While the safe number is a recently-introduced parameter, it bears some similarities with other graph parameters that are somewhat older and relatively well-studied. These are the vertex integrity and the ℓ-component order connectivity. Here we discuss some known results on these parameters and compare them to those on the safe number.

The *vertex integrity* $\mathsf{vi}(G)$ [3] of a graph G is the minimum integer k such that there is a vertex set $S \subseteq V(G)$ such that $|S|$ plus the maximum size of the components of $G - S$ is at most k. Fujita and Furuya [18] showed that $2\sqrt{\mathsf{s}(G) - 2} + 1 \leq \mathsf{vi}(G) \leq 2\mathsf{s}(G)$ for every connected graph G (except stars)

and that the bounds are tight. From this relation, one might speculate whether the concept of vertex integrity is essentially equivalent to that of safe number. This is not in fact true as their complexity differs for some cases:

- *Interval graphs*: For unweighted interval graphs, both the safe number and the vertex integrity can be computed in polynomial time [1, 24]. However, for weighted interval graphs, determining the safe number is NP-hard as mentioned above, but it is still polynomial-time solvable for the vertex integrity [34].

- *Split graphs*: For unweighted split graphs, the vertex integrity is trivially determined [29], while the safe number is NP-hard to determine [1]. Determining the vertex integrity is NP-hard for weighted split graphs [12].

- *Parameterized complexity*: The vertex integrity problem parameterized by the solution size k admits a kernel of k^3 vertices even for the weighted version [12]. On the other hand, our (unweighted) problem does not admit any polynomial kernel parameterized by the solutions size unless $\mathrm{PH} = \Sigma_3^{\mathrm{p}}$ (see Corollary 4.2).

The *ℓ-component order connectivity* of a graph is the minimum integer k such that there is a vertex set $S \subseteq V(G)$ with $|S| \leq k$ and each component of $G - S$ has size at most ℓ. Such an S is an *ℓ-size separator*. Because of the second parameter ℓ, this parameter cannot be directly compared to the safe number. We summarize their similarities and differences:

- *Graph classes*: For weighted interval graphs [4, 12] and weighted circular-arc graphs [4] the problem can be solved in polynomial time even if ℓ is part of the input.

- *Parameterized complexity*: When parameterized only by ℓ or the solution size k, the problem is W[1]-hard even for (unweighted) split graphs [12]. There is a $2^{O(k \log \ell)} \cdot n$-time algorithm, which is tight in the sense that there is no $2^{o(k \log \ell)} \cdot n^{O(1)}$-time algorithm unless the ETH fails [12]. When parameterized by both k and ℓ, the problem is fixed-parameter tractable [12] and admits a kernel of $O(k\ell)$ vertices [36]. (See also the previous result [25] when ℓ is considered as a fixed constant, not a parameter.) The algorithm by Xiao [36] was later generalized for a related problem on directed graphs [21]. When parameterized by both the treewidth tw and the solution size k, the problem is W[1]-hard, and the ETH implies that there is no $f(\mathsf{tw} + k) \cdot n^{o((\mathsf{tw}+k)/\log(\mathsf{tw}+k))}$-time algorithm [6].

- *Approximation*: An $(\ell + 1)$-approximation is obtained by a greedy algorithm [4]. There is a $2^{O(\ell)}n + n^{O(1)}$-time approximation algorithm of a factor $O(\log \ell)$, but there is no approximation algorithm that runs in time polynomial both in n and ℓ and with an approximation factor $n^{(1/\log\log n)^c}$ unless the ETH fails [27].

There is another graph parameter, called the *fracture number*, introduced by Dvořák et al. [13] to provide a way to tackle integer linear programs with small "backdoors". Given a bipartite graph G, its fracture number is the minimum k such that there is a vertex set S of size at most k while each component of $G - S$ has size at most k. Among other results, Dvořák et al. [13] showed that the problem is NP-hard in general, but can be solved in time $O((k+1)^k \cdot m)$, and there is also an approximation algorithm with ratio $k+1$.

Note that the property of having a constant vertex integrity or a constant ℓ-component order connectivity is minor closed. This is not true for the safe number and the connected safe number. For example, let C_8' be the graph obtained from C_8, the cycle of eight vertices, by adding a vertex v adjacent to a diagonal pair of distance 4 in C_8. Since $N[v]$ is a connected safe set, $\mathsf{s}(C_6) \leq \mathsf{cs}(C_6) \leq 3$. On the other hand, it can be shown that $\mathsf{s}(C_8) = \mathsf{cs}(C_8) = 4$.

3 W[2]-hardness parameterized by the pathwidth

In this section we show that Safe Set is W[2]-hard parameterized by the pathwidth, via a reduction from Dominating Set.

A set $K \subseteq V(G)$ is a *dominating set* of G if each vertex in G is either included in K or has a neighbor in K; that is, $N[v] \cap K \neq \emptyset$ for each $v \in V(G)$. Dominating Set asks to find a dominating set of size at most k and is known to be W[2]-complete when parameterized by the solution size k [11].

Given an instance $[G, k]$ of Dominating Set, we will construct an instance $[G', \mathsf{pw}(G')]$ of Safe Set parameterized by the pathwidth. Let $V(G) = \{v_1, \ldots, v_n\}$ and let $k' = 1 + kn + \sum_{v \in V(G)} k(\delta(v) + 1)$ be the target size of the safe set in the new instance, where $\delta(v)$ is the degree of v in G.

Before proceeding, let us give a high-level description of some of the key ideas of our reduction (an overview is given in Figure 3). First, we note that our new instance will include a universal vertex. This simplifies things, as such a vertex must be included in any safe set (of reasonable size) and ensures that the safe set is connected. The problem then becomes: can we select $k' - 1$ additional vertices so that their deletion disconnects the graph into components of size at most k'.

The main part of our construction consists of a collection of k cycles of length n^2. By attaching an appropriate number of leaves to every n-th vertex of such a cycle we can ensure that any safe set that uses exactly n vertices from each cycle must space them evenly, that is, if it selects the i-th vertex of the cycle, it also selects the $(n+i)$-th vertex, the $(2n+i)$-th vertex, etc. As a result, we expect the solution to invest kn vertices in the cycles, in a way that encodes k choices from $[n]$.

We must now check that these choices form a dominating set of the original graph G. For each vertex of G we construct a gadget and connect this gadget to a different length-n section of the cycles. This type of connection ensures that

the construction will in the end have small pathwidth, as the different gadgets are only connected through the highly-structured "choice" part that consists of the k cycles. We then construct a gadget for each vertex v_i that can be broken down into small enough components by deleting $k(\delta(v_i)+1)$ vertices, if and only if we have already selected from the cycles a vertex corresponding to a member of $N[v]$ in the original graph.

Domination gadget: Before we go on to describe in detail the full construction, we describe a *domination gadget* $\hat{D}_i$. This gadget refers to the vertex $v_i \in V(G)$ and its purpose is to model the domination of v_i in G and determine the member of $N[v_i]$ that belongs to the dominating set. We construct $\hat{D}_i$ as follows, while Figure 2 provides an illustration:

- We make a *central* vertex z^i. We attach to this vertex $k' - k(\delta(v_i) + 1)$ leaves. Call this set of leaves W^i.
- We make k independent sets $X_1^i, \ldots, X_k^i$ of size $|N[v_i]|$. For each $j \in [1, k]$ we associate each $x \in X_j^i$ with a distinct member of $N[v_i]$. We attach to each vertex x of each $X_j^i, j \in [1, k]$ an independent set of $k' - 1$ vertices. Call this independent set Q_x.
- We then make another k independent sets $Y_1^i, \ldots, Y_k^i$ of size $|N[v_i]|$. For each $j \in [1, k]$ we construct a perfect matching from X_j^i to Y_j^i. We then connect z to all vertices of Y_j^i for all $j \in [1, k]$.

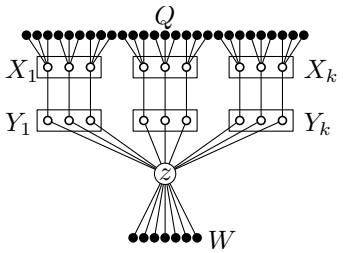

Figure 2: Our domination gadget $\hat{D}_i$. Superscripts i are omitted.

The intuition behind this construction is the following: we will connect the vertices of X_j^i to the j-th selection cycle, linking each vertex with the element of $N[v_i]$ it represents. In order to construct a safe set, we need to select at least one vertex $y \in Y_j^i$ for some $j \in [1, k]$, otherwise the component containing z will be too large. This gives us the opportunity to *not* place the neighbor $x \in X_j^i$ of y in the safe set. This can only happen, however, if the neighbor of x in the main part is also in the safe set, i.e., if our selection dominates v_i.

Construction: Graph G' is constructed as follows:

- We first make n copies of $V(G)$ and serially connect them in a long cycle, conceptually divided into n *blocks*:

$$v_1, v_2, \ldots, v_n | v_{n+1}, \ldots, v_{2n} | v_{2n+1}, \ldots | \ldots, v_{n^2} | v_1.$$

  Each block corresponds to one vertex of $V(G)$.

- We make k copies $V^1, \ldots, V^k$ of this cycle, where $V^i = \{v^i_1, \ldots, v^i_{n^2}\}, \forall i \in [1, k]$ and refer to each as a *line*. Each such line will correspond to one vertex of a dominating set in G.

- We add a set B^i_j of $k'-n+1$ neighbors to each vertex $v^i_{(j-1)n+1}, \forall j \in [1, n]$, i.e., the first vertex of every block of every line. We refer to these sets collectively as the *guards*.

- Then, for each *column* of blocks (i.e., the i-th block of all k lines for $i \in [1, n]$), we make a domination gadget $\hat{D}_i$ that refers to vertex $v_i \in V(G)$. As described above, the gadget contains k copies $X^i_1, \ldots, X^i_k$ of $N[v_i] \subseteq V(G)$.

- For $i \in [1, n]$ and $j \in [1, k]$, we add an edge between each vertex in X^i_j and its corresponding vertex in the i-th block of V^j, i.e., for the given i-th column, we connect a vertex from the j-th line (V^j) to the vertex from the j-th copy of $N[v_i]$ (X^i_j) if they correspond to the same original vertex from $V(G)$.

- We add a *universal vertex* u and connect it to every other vertex in the graph. This concludes our construction (see also Figure 3).

Lemma 3.1 *If G has a dominating set of size k, then G' has a safe set of size $k' = 1 + kn + \sum_{v \in V(G)} k(\delta(v) + 1)$.*

Proof: Given a dominating set K in G of size $|K| = k$, we will construct a safe set S in G' of size $|S| = k' = 1 + kn + \sum_{v \in V(G)} k(\delta(v) + 1)$. First, we include in S the universal vertex u. We then arbitrarily order the vertices of K and will include in S the n copies (one from each block) of each vertex of K from the line that matches this ordering, i.e., if the i-th vertex of K is v_j, then set S will include all vertices $v^i_j, v^i_{j+n}, \ldots, v^i_{j+n^2-n}$ for each $i \in [1, k]$.

Then, for each column and domination gadget $\hat{D}_i$, we consider the closed neighborhood $N[v_i]$ of vertex v_i in G: as K is a dominating set, $K \cap N[v_i] \neq \emptyset$, i.e., there must be at least one vertex x from $N[v_i]$ included in K. If this is the j-th vertex in K according to our ordering, we then include all vertices from sets $X^i_1, \ldots, X^i_k$ in S, *except* for the copy of x in X^i_j and we also include in S the vertex y from set Y^i_j that is paired with x (and is adjacent to it). Note also that set S must already include a vertex corresponding to x from line V^j. Thus $k(\delta(v_i) + 1)$ vertices are selected from $\hat{D}_i$, one from the sets Y and all but one from the sets X, while these selections are complementary.

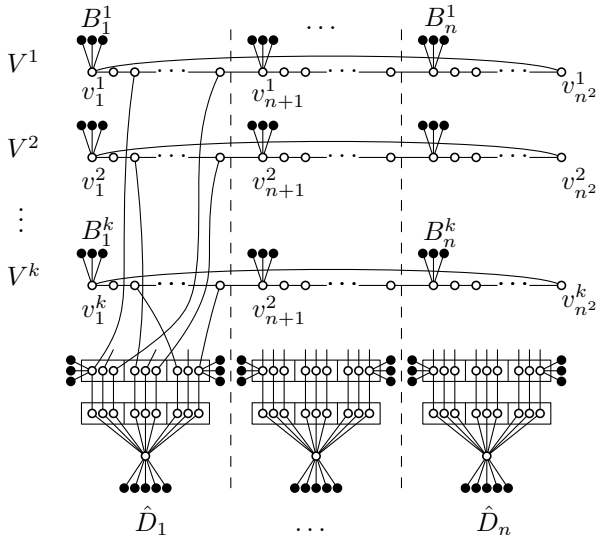

Figure 3: A simplified picture of our construction. Note sets Q are only shown for two vertices per gadget, exemplary connections between corresponding vertices in sets X and lines V are only shown for the first gadget, while the universal vertex u is omitted.

Repeating this process for all gadgets $\hat{D}_i$ (i.e., all columns) completes set S and we claim it is a safe set for G'. Observe that the size of S is now $|S| = k' = 1 + kn + \sum_{v \in V(G)} k(\delta(v) + 1)$.

Since S includes vertex u that is connected to all other vertices, then set S is connected and for it to be safe, the size of any *remaining* connected component in $G' - S$ must be no larger than $k' = 1 + kn + \sum_{v \in V(G)} k(\delta(v) + 1)$. First, from every line V^i, the set S periodically includes every n-th vertex (all corresponding to some vertex of the dominating set solution K) and if some $v_l^i \in V^i$ with a neighbor in a dominating gadget $\hat{D}_j$ is not included in S, then the neighbor in the gadget must be included in S instead. Thus the size of any remaining component in V^i is at most $|B_j^i| + n - 1 = k' - n + 1 + n - 1 = k' = |S|$. Within a domination gadget $\hat{D}_i$, there is only one vertex x from sets X_j^i that does not belong in S, while both its neighbors from V^j and Y_j^i are in S. The size of any remaining component that includes x is equal to $|Q_x| + 1 = k' = |S|$. Finally, the size of any remaining component that contains the central vertex z^i is equal to $1 + |W^i| + k(\delta(v_i) + 1) - 1 = 1 + k' - k(\delta(v_i) + 1) + k(\delta(v_i) + 1) - 1 = k' = |S|$, due to set S containing only one vertex from sets Y_j^i (that is adjacent to the only vertex from X_j^i that is not in S). As there is no remaining component in $G' - S$ that is larger than S, our solution S is safe. □

Lemma 3.2 *If G' has a safe set of size $k' = 1 + kn + \sum_{v \in V(G)} k(\delta(v) + 1)$, then G has a dominating set of size k.*

Proof: Given a safe set S of size k' in G', we will construct a dominating set K of size k in G. First, we may assume without loss of generality that set S includes the universal vertex u, as it being adjacent to every other vertex would imply that otherwise the size of S would have to be at least $|V(G')|/2$.

We now make an easy observation that we may assume (without loss of generality) that the safe set we are given does not contain any leaves: if the safe set contains a leaf u and does not contain its neighbor, we can exchange u with its neighbor; if the safe set contains u and its neighbor, we can exchange u with some arbitrary non-leaf vertex, and this does not affect the feasibility of the solution thanks to the universal vertex.

Next, due to the guard vertices B^i_j connected to every first vertex of every block of every line, any safe set of size k' must include at least one vertex per block per line and these must be periodical so as to allow only sub-paths of length n between consecutive selections: the size of every B^i_j being $k' - n + 1$, if there is such a sub-path between any pair of selections that is longer than n, then there would be a remaining component that is larger than k'. Thus set S must include at least $kn + 1$ vertices outside the domination gadgets $\hat{D}_i$.

Next consider a gadget $\hat{D}_i$ that corresponds to vertex $v_i \in V(G)$. The size of the set W^i being $k' - k(\delta(v_i) + 1)$ implies that there must be at least one vertex in S from the closed neighborhood of central vertex z^i, as $|N[z^i]| = |W^i| + 1 + k(\delta(v_i) + 1) = k' + 1$. Furthermore, for any vertex x in some X^i_j, set S must include either vertex x itself, or at least two vertices from the closed neighborhood of x as the size of each set Q_x is $k' - 1$ and $|N[x]| = k' + 2$. This means set S must include at least $k(\delta(v_i) + 1)$ vertices from each gadget $\hat{D}_i$ (completing the budget for S) and furthermore, that it is only possible for S to not include some vertex $x \in X^i_j$ if both its neighbor $y \in Y^i_j$ (its pair) and its neighbor v^j_l in the line V^j (the pair's corresponding vertex) are included in S.

Having thus identified the necessary structure of S, we construct our dominating set K in G by considering the selections for S from every line V^i: as explained above, these selections must be periodical for each line (every n-th vertex) and for each $i \in [1, k]$ we include in K the vertex from $V(G)$ that corresponds to every copy from every block that is included in S. Note that these need not necessarily be distinct, i.e., the selections from a pair of lines might correspond to the same original vertex from G, meaning K may be of size less than k, yet if S is a safe set of size k' in G', then the resulting set K must be a dominating set in G.

Consider each column i of blocks and each domination gadget $\hat{D}_i$. As explained above, the size of $S \cap \hat{D}_i$ is $k(\delta(v_i) + 1)$, with at least one of these belonging in the closed neighborhood of central vertex z^i, while for each $x \in X^i_j$ it must be either $x \in S$, or $y \in S$ and $v^j_l \in S$, where y is the pair of x in Y^i_j and v^j_l is the pair's corresponding vertex in the line V^j. The budget for $\hat{D}_i$ implies there must be at least one x not included in S, whose pair y must be included instead (for the remaining component containing central vertex z^i to be of size $\leq k'$) and thus, the other neighbor of x in line V^j must also have been included in S, i.e., set S must already include every copy of v^j_l from every block of line V^j.

This implies that for the vertex $v_i \in V(G)$ in question (i.e., corresponding to column i and gadget $\hat{D}_i$), at least one vertex from its closed neighborhood $N[v_i]$ will be included in K and vertex v_i will thus be dominated by K in G. As this must hold for all columns i/gadgets $\hat{D}_i$ and vertices $v_i \in V(G)$, set K will be a dominating set in G. □

Lemma 3.3 *The pathwidth* $\mathsf{pw}(G')$ *of* G' *is at most* $2k+4$.

Proof: We consider the graph G'' obtained from G' by deleting the universal vertex as well as the edges connecting v_1^j to $v_{n^2}^j$ for each $j \in [1,k]$. We will show that this graph has pathwidth at most $k+3$. This will imply the claim as we can obtain a path decomposition of G' by adding to all bags the universal vertex and the vertices v_1^j for all $j \in [1,k]$.

Consider now the i-th block of the construction, consisting of the gadget $\hat{D}_i$, the vertices $\{v_{(i-1)n+1}^j, \ldots, v_{in}^j\}$ and B_i^j for $j \in [1,k]$, and additionally the vertex v_{in+1}^j, if $i < n$. We show that the graph induced by these vertices admits a path decomposition of width $k+3$ with the additional property that the first bag of the decomposition is $\{v_{(i-1)n+1}^j : 1 \leq j \leq k\}$, and the last bag of the decomposition is $\{v_{in+1}^j : 1 \leq j \leq k\}$, if $i < n$. If we prove this claim we are done, since then we can take such a decomposition for each block and identify the last bag of the decomposition for the i-th block with the first bag of the decomposition for the $(i+1)$-th block to obtain a decomposition for G'' of the same width.

Let us now consider the i-th block. We will make use of the following standard pathwidth fact: for any graph H, if H' is the graph obtained by deleting all leaves of H, then $\mathsf{pw}(H) \leq \mathsf{pw}(H') + 1$. This can easily be seen by first considering a path decomposition of H' and then, for each leaf u of H, finding a bag of the original decomposition that contains the neighbor of u and inserting immediately after it a copy of the same bag with u added. Furthermore, we can do this without changing the first and the last bags in the decomposition. We use this fact to bound the pathwidth of the i-th block as follows: we add the central vertex z to all bags, and therefore may delete it from the graph. Now, removing all leaves from the block eliminates all B, Y, Q, and W vertices. Removing again all leaves from this graph eliminates X. As a result, the new graph is a collection of k disjoint paths, for which we can easily construct the required decomposition. □

Theorem 3.4 Safe Set *and* Connected Safe Set *are W[2]-hard parameterized by the pathwidth of the input graph. Furthermore, both problems cannot be solved in time* $n^{o(\mathsf{pw})}$ *unless the ETH is false.*

Proof: Given an instance $[G,k]$ of Dominating Set, we use the above construction to create an instance $[G', \mathsf{pw}(G')]$ of Safe Set parameterized by the pathwidth. Lemmas 3.1 and 3.2 show the correctness of our reduction, while Lemma 3.3 provides the bound on the pathwidth of the constructed graph,

showing that our new parameter $\mathsf{pw}(G')$ is linearly bounded by the original parameter k. The running time bound follows from the fact that, under the ETH, Dominating Set does not admit an $n^{o(k)}$ algorithm.

The results for CSS follow by the connectivity of the safe set in Lemmas 3.1 and 3.2. □

4 No polynomial kernel parameterized by the vertex cover number

A set $X \subseteq V(G)$ is a *vertex cover* of G if each edge $e \in E(G)$ has at least one endpoint in X. The minimum size of a vertex cover in G is the *vertex cover number* of G, denoted by $\mathsf{vc}(G)$. Parameterized by the vertex cover number vc, both problems are FPT (see Figure 1) and in this section we show the following kernelization hardness of SS and CSS.

Theorem 4.1 Safe Set *and* Connected Safe Set *parameterized by the vertex cover number do not admit polynomial kernels even for connected graphs unless* $\mathrm{PH} = \Sigma_3^{\mathrm{p}}$.

Since $\mathsf{cs}(G)/2 \leq \mathsf{s}(G) \leq \mathsf{vc}(G)$ for every graph G [1], the above theorem implies that SS and CSS parameterized by the natural parameters do not admit a polynomial kernel.

Corollary 4.2 *SS and CSS parameterized by the solution size do not admit polynomial kernels even for connected graphs unless* $\mathrm{PH} = \Sigma_3^{\mathrm{p}}$.

Let P and Q be parameterized problems. A polynomial-time computable function $f\colon \Sigma^* \times N \to \Sigma^* \times N$ is a *polynomial parameter transformation* from P to Q if there is a polynomial p such that for all $(x,k) \in \Sigma^* \times N$, it holds that $(x,k) \in P$ if and only if $(x',k') = f(x,k) \in Q$ and $k' \leq p(k)$. If such a function exits, then P is *polynomial-parameter reducible* to Q.

Proposition 4.3 ([5]) *Let P and Q be parameterized problems, and P' and Q' be unparameterized versions of P and Q, respectively. Suppose P' is NP-hard, Q' is in NP, and P is polynomial-parameter reducible to Q. If Q has a polynomial kernel, then P also has a polynomial kernel.*

To prove Theorem 4.1, we present a polynomial-parameter transformation from the well-known Red-Blue Dominating Set problem (RBDS) to SS (and CSS) parameterized by the vertex cover number.

Red-Blue Dominating Set (RBDS)
Input: A bigraph $G = (R, B; E)$ and an integer k.
Question: Is there $D \subseteq B$ of size at most k that dominates all vertices in R?

RBDS becomes trivial when $k \geq |R|$ and thus we assume that $k < |R|$ in what follows. It is known that RBDS parameterized simultaneously by k and

$|R|$ does not admit a polynomial kernel unless PH $= \Sigma_3^{\mathrm{p}}$ [10]. Since RBDS is NP-hard and SS and CSS are in NP, it suffices to present a polynomial-parameter transformation from RBDS parameterized by $k + |R|$ to SS and CSS parameterized by the vertex cover number.

From an instance $[G, k]$ of RBDS with $G = (R, B; E)$, we construct an instance $[H, s]$ of SS as follows (see Figure 4). Let $s = k + |R| + 1$. We add a vertex u to G and make it adjacent to all vertices in B. We then attach $2s$ pendant vertices to each vertex in R and to u. Finally, for each $r \in R$, we make a star $K_{1,s-1}$ and add an edge between r and the center of the star. We call the resultant graph H. Observe that $\mathsf{vc}(H) \leq 2|R| + 1$ since $\{u\} \cup R \cup C$ is a vertex cover of H, where C is the set of centers of stars attached to R. This reduction is a polynomial-parameter transformation from RBDS parameterized by $k + |R|$ to SS parameterized by the vertex cover number.

If $D \subseteq V(G)$ is a solution of RBDS of size k, then $S := \{u\} \cup R \cup D$ is a connected safe set of size s. To see this, recall that B is an independent set and $N_H(B) = \{u\} \cup R$. Thus each component in $H - S$ is either an isolated vertex in $D \setminus B$, or a star $K_{1,s-1}$ with s vertices.

Assume that (H, s) is a yes instance of SS and let S be a safe set of H with $|S| \leq s$. Observe that $\{u\} \cup R \subseteq S$ since u and all vertices in R have degree at least $2s$. Since $|S \setminus (\{u\} \cup R)| \leq k < |R|$, S cannot intersect all stars attached to the vertices in R. Hence $H - S$ has a component of size at least $|V(K_{1,s-1})| = s$. This implies that S is connected. Since R is an independent set and each path from u to a vertex in R passes through B, each vertex in R has to have a neighbor in $B \cap S$. Thus $B \cap S$ dominates R. Since $B \cap S \subseteq S \setminus (\{u\} \cup R)$, it has size at most k. Therefore, $[G, k]$ is a YES instance of RBDS. This completes the proof of Theorem 4.1.

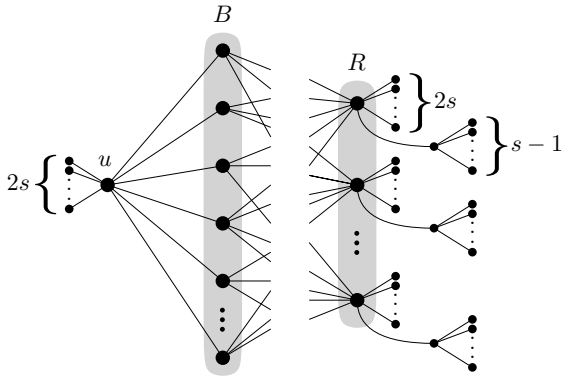

Figure 4: The graph H in our reduction from RBDS to SS for the proof of Theorem 4.1.

5 FPT algorithm parameterized by the neighborhood diversity

In this section, we present FPT algorithms for SS and CSS parameterized by the neighborhood diversity. That is, we prove the following theorem.

Theorem 5.1 *Both* Safe Set *and* Connected Safe Set *are fixed-parameter tractable when parameterized by the neighborhood diversity.*

In a graph G, two vertices $u, v \in V(G)$ are *twins* if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. The *neighborhood diversity* $\mathsf{nd}(G)$ of G is the minimum integer k such that $V(G)$ can be partitioned into k sets $T_1, \ldots, T_k$ of pairwise twin vertices. It is known that such a minimum partition can be found in linear time using fast modular decomposition algorithms [30, 35]. It is also known that $\mathsf{nd}(G) \leq 2^{\mathsf{vc}(G)} + \mathsf{vc}(G)$ for every graph G [26].

Let G be a connected graph such that $\mathsf{nd}(G) = k$. Let $T_1, \ldots, T_k$ be the partition of $V(G)$ into sets of pairwise twin vertices. Note that each T_i is either a clique or an independent set by definition. We assume that $k \geq 2$ since otherwise the problem becomes trivial. Since each T_i is a twin set, the sizes of intersections $|S \cap T_i|$ completely characterize the sizes of the components in $G[S]$ and $G - S$, and the adjacency among them.

Let $S \subseteq V(G)$ and $s_i = |S \cap T_i|$ for $i \in [k]$. We partition $[k]$ into $I_{\mathbf{f}}$, $I_{\mathbf{p}}$, and $I_\emptyset$ as follows:

$$i \in \begin{cases} I_\emptyset & \text{if } s_i = 0, \\ I_{\mathbf{p}} & \text{if } 1 \leq s_i \leq |T_i| - 1, \\ I_{\mathbf{f}} & \text{otherwise } (s_i = |T_i|). \end{cases} \tag{1}$$

For $i, i' \notin I_\emptyset$ (not necessarily distinct), twin sets T_i and $T_{i'}$ are *reachable* in S if either

- $T_i = T_{i'}$ and T_i is a clique, or
- there is a sequence $i_0, \ldots, i_\ell$ of indices such that $\ell \geq 1$, $i_0 = i$, $i_\ell = i'$, $i_j \notin I_\emptyset$ for all j, and T_{i_j} and $T_{i_{j+1}}$ are adjacent for $0 \leq j < \ell$.

Lemma 5.2 *If $i \notin I_\emptyset$ and T_i is not reachable from T_i itself in S, then each vertex in $T_i \cap S$ induces a component of size* 1.

Proof: The assumption implies that T_i is an independent set and $i' \in I_\emptyset$ for each $T_{i'}$ adjacent to T_i. Therefore, each vertex in $T_i \cap S$ has no other vertex in S that belongs to the same component of $G[S]$. □

Lemma 5.3 *Two vertices $u, v \in S$ are in the same component of $G[S]$ if and only if $u \in T_i$ and $v \in T_{i'}$ for some i and i', and T_i is reachable from $T_{i'}$ in S.*

Proof: If $u, v \in T_i$ for some i where T_i is a clique, then $\{u, v\} \in E(G)$ and thus they are in the same component of $G[S]$. Assume that $u \in T_i$ and $v \in T_{i'}$, and there is a sequence $i_0, \ldots, i_\ell$ with $\ell \geq 1$ that shows the reachability. We show that u, v are in the same component of $G[S]$ by induction on ℓ. If $\ell = 1$, then T_i and $T_{i'}$ are adjacent and we are done. Let $\ell > 1$. Since $i_{\ell-1} \notin I_\emptyset$, there is a vertex $w \in T_{\ell-1} \cap S$. By the induction hypothesis, u and w are in the same component of $G[S]$. Furthermore, since $T_{\ell-1}$ and $T_\ell = T_{i'}$ are adjacent, there is an edge between w and v. Hence u and v are in the same component of $G[S]$.

Now assume that $u \in T_i$ and $v \in T_{i'}$ are in the same connected connected of $G[S]$. Then there is a shortest u–v path $(p_0, \ldots, p_q)$ in $G[S]$. If $T_i = T_{i'}$ and T_i is a clique, then $p_0, p_q \in T_i$. Assume that this is not the case. Let T_{i_j} be the twin set including p_j. Since the path is shortest, $i_j \neq i_{j+1}$ holds and T_j is adjacent to T_{j+1} for each $j < q$. Also, because of p_j, we have $i_j \notin I_\emptyset$ for each j. Therefore, T_i and $T_{i'}$ are reachable in S. □

By Lemmas 5.2 and 5.3, each component of $G[S]$ is either a single vertex, or the intersection of S and the union of a maximal family of pairwise reachable twin sets in S. Observe that the maximal families of pairwise reachable twin sets in S is determined only by the set $I_\emptyset$. Also, if R is a maximal family of pairwise reachable twin sets in S, then the corresponding component of $G[S]$ has size $\sum_{T_i \in R} |T_i \cap S|$.

Now, just by interchanging the roles of S and $V(G) \setminus S$ in Lemmas 5.2 and 5.3, we can show the following counterparts that imply that the maximal families of pairwise reachable twin sets in $V(G) \setminus S$ are determined only by the set $I_{\mathbf{f}}$, while the size of the component of $G - S$ corresponding to a maximal family R of pairwise reachable twin sets in $V(G) \setminus S$ is $\sum_{T_i \in R} |T_i \setminus S|$.

Lemma 5.4 *If $i \notin I_{\mathbf{f}}$ and T_i is not reachable to T_i itself in $V(G) \setminus S$, then each vertex in $T_i \setminus S$ induces a component of size* 1.

Lemma 5.5 *Two vertices $u, v \in V(G) \setminus S$ are in the same component of $G - S$ if and only if $u \in T_i$ and $v \in T_{i'}$ for some i and i', and T_i is reachable from $T_{i'}$ in $V(G) \setminus S$.*

ILP formulation

Now we reduce the problem to at most 3^k instances of integer linear programming with a bounded number of variables. We first divide $[k]$ into the subsets $I_\emptyset$, $I_{\mathbf{p}}$, $I_{\mathbf{f}}$ in Eq. (1). There are 3^k candidates for such a partition.

For each $i \in [k]$, we use a variable x_i to represent the size of $T_i \cap S$. To find a minimum safe set satisfying $I_\emptyset$, $I_{\mathbf{p}}$, $I_{\mathbf{f}}$, we set the objective function to be $\sum_{i \in [k]} x_i$ and minimize it subject to the following linear constraints. The first set of constraints is to make S consistent with the guess of $I_\emptyset$, $I_{\mathbf{p}}$, $I_{\mathbf{f}}$:

$$\begin{aligned} x_i &= 0 && \text{for } i \in I_\emptyset, \\ 1 \leq x_i &\leq |T_i| - 1 && \text{for } i \in I_{\mathbf{p}}, \\ x_i &= |T_i| && \text{for } i \in I_{\mathbf{f}}. \end{aligned}$$

As discussed above, the set of sizes x_i completely characterizes the structure of components in $G[S]$ and $G - S$. In particular, we can decide whether $G[S]$ is connected or not at this point. We reject the disconnected case if we are looking for a connected safe set.

Let $\mathcal{C}$ and $\mathcal{D}$ be the sets of maximal families of pairwise reachable twin sets in S and $V(G) \setminus S$, respectively. Note that the twin sets that satisfy the conditions of Lemma 5.2 (Lemma 5.4) are not included in any member of $\mathcal{C}$ ($\mathcal{D}$, respectively).

For each $C_j \in \mathcal{C}$, we use a variable y_j to represent the size of the corresponding component of $G[S]$. Also, for each $D_h \in \mathcal{D}$, we use a variable z_h to represent the size of the corresponding component of $G - S$. This can be stated as follows:

$$y_j = \sum_{T_i \in C_j} x_i \quad \text{for } C_j \in \mathcal{C},$$

$$z_h = \sum_{T_i \in D_h} |T_i| - x_i \quad \text{for } D_h \in \mathcal{D}.$$

We say that $C_j \in \mathcal{C}$ and $D_h \in \mathcal{D}$ are *touching* if there are $T \in C_j$ and $T' \in D_h$ that are adjacent, or the same. We can see that C_j and D_h are touching if and only if the corresponding components are adjacent via an edge from T and T' or an edge completely in $T = T'$. We add the following constraint to guarantee the safeness of S:

$$y_j \geq z_h \quad \text{for each pair of touching } C_j \in \mathcal{C} \text{ and } D_h \in \mathcal{D}.$$

Now we have to deal with the singleton components of $G[S]$ (we can ignore the singleton components of $G - S$ because the components of $G[S]$ adjacent to them have size at least 1). Let T_i be a twin set that satisfies the conditions of Lemma 5.2. That is, $T_i \cap S \neq \emptyset$, T_i is an independent set, and no twin set adjacent to T_i has a non-empty intersection with S. Hence a component of $G - S$ is adjacent to the singleton components in $T_i \cap S$, if and only if the corresponding family $D_h \in \mathcal{D}$ includes a twin set adjacent to T_i. We say that such D_h is *adjacent to* T_i. Therefore, we add the following constraint:

$$z_h \leq 1 \quad \text{for each } D_h \in \mathcal{D} \text{ adjacent to } T_i \text{ satisfying Lemma 5.2.}$$

Solving the ILP

Lenstra [28] showed that the feasibility of an ILP formula can be decided in FPT time when parameterized by the number of variables. The running time was later improved by Frank and Tardos [16] and by Kannan [23]. Among other generalizations and extensions of this result, we need a version by Fellows et al. [15], which is for the following optimization version.

p-Opt-ILP
Input: A matrix $A \in \mathbb{Z}^{m \times p}$, and vectors $b \in \mathbb{Z}^m$ and $c \in \mathbb{Z}^p$.
Question: Find a vector $x \in \mathbb{Z}^p$ that minimizes $c^\top x$ and satisfies that $Ax \geq b$.

Using the algorithms for the original feasibility problem as a black box, they showed the following:

Theorem 5.6 ([15]) *p-Opt-ILP can be solved using $O(p^{2.5p+o(p)} \cdot L \cdot \log(MN))$ arithmetic operations and space polynomial in L, where L is the number of bits in the input, N is the maximum absolute values any variable can take, and M is an upper bound on the absolute value of the minimum taken by the objective function.*

In the formulation for SS and CSS, we have at most $O(k)$ variables: x_i for $i \in [k]$, y_j for $C_j \in \mathcal{C}$, and z_h for $D_h \in \mathcal{D}$. Observe that the elements of $\mathcal{C}$ (and of $\mathcal{D}$ as well) are pairwise disjoint. We have only $O(k^2)$ constraints and the variables and coefficients can have values at most $|V(G)|$. Therefore, Theorem 5.1 holds.

6 XP-time algorithm parameterized by the clique-width

This section presents an XP-time algorithm for SS and CSS parameterized by the clique-width. The algorithm runs in time $O(g(c) \cdot n^{f(c)})$, where c is the clique-width. It is known that for any constant c, one can compute a $(2^{c+1}-1)$-expression of a graph of clique-width c in $O(n^3)$ time [22, 32, 33]. We omit $g(c)$ in the running time and focus on the exponent $f(c)$.

Theorem 6.1 *Given an n-vertex graph G and an irredundant c-expression of G, the values of $\mathsf{s}(G)$ and $\mathsf{cs}(G)$, along with their corresponding sets can be computed in $O(n^{28 \cdot 2^c+1})$ time.*

Corollary 6.2 *Given an n-vertex graph G, the values of $\mathsf{s}(G)$ and $\mathsf{cs}(G)$, along with their corresponding sets can be computed in time $n^{O(f(\mathsf{cw}(G)))}$, where $f(c) = 2^{2^{c+1}}$.*

For each node t in the c-expression T of G, let G_t be the vertex-labeled graph represented by t. We denote by V_t the vertex set of G_t. For each i, we denote the set of i-vertices in G_t by V_t^i. For sets $S \subseteq V_t$ and $L \subseteq [c]$, we denote by $\mathcal{C}(S, L)$ and $\mathcal{D}(S, L)$ the set of components of $G_t[S]$ and $G_t - S$, respectively, that include exactly the labels in L.

For each node t in T, we construct a table $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d}) \in \{\mathsf{true}, \mathsf{false}\}$ with indices $\mathtt{n}, \overline{\mathtt{n}} \colon 2^{[c]} \to \{0, \ldots, n\}$, $\mathtt{s} \colon 2^{[c]} \to \{0, \ldots, n\} \cup \{\infty\}$, $\mathtt{l} \colon 2^{[c]} \to \{0, \ldots, n\} \cup \{-\infty\}$, $\mathtt{s}^{\mathtt{a}} \colon 2^{[c]} \times 2^{[c]} \to \{0, \ldots, n\} \cup \{\infty\}$, $\mathtt{l}^{\mathtt{a}} \colon 2^{[c]} \times 2^{[c]} \to \{0, \ldots, n\} \cup \{-\infty\}$, and $\mathtt{d} \colon 2^{[c]} \times 2^{[c]} \to \{-n, \ldots, n\} \cup \{\infty\}$. We set $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d}) = \mathsf{true}$ if and only if there exists a set $S \subseteq V_t$ such that, for all $L \subseteq [c]$:[2]

[2] We assume that min and max return ∞ and $-\infty$, respectively, when applied to the empty set.

- $\mathtt{n} = \sum_{C \in \mathcal{C}(S,L)} |C|$, $\overline{\mathtt{n}} = \sum_{D \in \mathcal{D}(S,L)} |D|$,
- $\mathtt{s} = \min_{C \in \mathcal{C}(S,L)} |C|$, $\mathtt{l} = \max_{D \in \mathcal{D}(S,L)} |D|$,

and for all $L_1, L_2 \subseteq [c]$:

- $\mathtt{s}^{\mathtt{a}} = \min\{|C| : C \in \mathcal{C}(S, L_1), D \in \mathcal{D}(S, L_2), C \text{ and } D \text{ are adjacent}\}$,
- $\mathtt{l}^{\mathtt{a}} = \max\{|D| : C \in \mathcal{C}(S, L_1), D \in \mathcal{D}(S, L_2), C \text{ and } D \text{ are adjacent}\}$,
- $\mathtt{d} = \min\{|C| - |D| : C \in \mathcal{C}(S, L_1), D \in \mathcal{D}(S, L_2), C \text{ and } D \text{ are adjacent}\}$.

Let r be the root node of T. Observe that $\mathtt{s}(G)$ is the minimum integer s such that there exist $\mathtt{n}$, $\overline{\mathtt{n}}$, $\mathtt{s}$, $\mathtt{l}$, $\mathtt{s}^{\mathtt{a}}$, $\mathtt{l}^{\mathtt{a}}$, and $\mathtt{d}$ satisfying that $\mathtt{dp}_r(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d}) = \mathsf{true}$, $s = \sum_{L \subseteq [c]} \mathtt{n}$, and $\mathtt{d}(L_1, L_2) \geq 0$ for all $L_1, L_2 \subseteq [c]$. For $\mathtt{cs}(G)$, we additionally ask that $\mathtt{n}(L)$ is nonzero with exactly one $L \subseteq [c]$.

We can compute in a bottom-up manner all entries $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d})$. Note that there are $O(n^{10 \cdot 2^c + 1})$ such entries. Hence, to prove Theorem 6.1, it suffices to show that each entry can be computed in time $O(n^{18 \cdot 2^c})$ assuming that the entries for the children of t are already computed. This is indeed the case for a $\cup$-node, while for ρ- and η-nodes $O(n^{10 \cdot 2^c})$ will suffice.

Lemma 6.3 *For a leaf node t with label $\circ_i$, $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d})$ can be computed in $O(1)$ time.*

Proof: Observe that $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d}) = \mathsf{true}$ if and only if:

- $\mathtt{n}(L) = \overline{\mathtt{n}}(L) = 0$, $\mathtt{s}(L) = \infty$, and $\mathtt{l}(L) = -\infty$ for all $L \neq \{i\}$,
- $\mathtt{s}^{\mathtt{a}}(L_1, L_2) = \infty$, $\mathtt{l}^{\mathtt{a}}(L_1, L_2) = -\infty$, and $\mathtt{d}(L_1, L_2) = \infty$ for all $L_1, L_2 \subseteq [c]$,

and either:

- $\mathtt{n}(\{i\}) = 0$, $\overline{\mathtt{n}}(\{i\}) = 1$, $\mathtt{s}(\{i\}) = \infty$, $\mathtt{l}(\{i\}) = 1$, or
- $\mathtt{n}(\{i\}) = 1$, $\overline{\mathtt{n}}(\{i\}) = 0$, $\mathtt{s}(\{i\}) = 1$, $\mathtt{l}(\{i\}) = -\infty$,

where the first case corresponds to $S = \emptyset$ and the second one to $S = V_t^i = V_t$. These conditions can be checked in $O(1)$ time. □

Lemma 6.4 *For a $\cup$-node t, $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d})$ can be computed in $O(n^{18 \cdot 2^c})$ time.*

Proof: Let t_1 and t_2 be the children of t in T. Now, $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d}) = \mathsf{true}$ if and only if there exist tuples $(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}_1^{\mathtt{a}}, \mathtt{l}_1^{\mathtt{a}}, \mathtt{d}_1)$ and $(\mathtt{n}_2, \overline{\mathtt{n}}_2, \mathtt{s}_2, \mathtt{l}_2, \mathtt{s}_2^{\mathtt{a}}, \mathtt{l}_2^{\mathtt{a}}, \mathtt{d}_2)$ such that:

- $\mathtt{dp}_{t_1}(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}_1^{\mathtt{a}}, \mathtt{l}_1^{\mathtt{a}}, \mathtt{d}_1) = \mathsf{true}$,
- $\mathtt{dp}_{t_2}(\mathtt{n}_2, \overline{\mathtt{n}}_2, \mathtt{s}_2, \mathtt{l}_2, \mathtt{s}_2^{\mathtt{a}}, \mathtt{l}_2^{\mathtt{a}}, \mathtt{d}_2) = \mathsf{true}$,
- $\mathtt{n}(L) = \mathtt{n}_1(L) + \mathtt{n}_2(L)$ for all $L \subseteq [c]$,

- $\overline{\mathtt{n}}(L) = \overline{\mathtt{n}}_1(L) + \overline{\mathtt{n}}_2(L)$ for all $L \subseteq [c]$,
- $\mathtt{s}(L) = \min\{\mathtt{s}_1(L), \mathtt{s}_2(L)\}$ for all $L \subseteq [c]$,
- $\mathtt{l}(L) = \max\{\mathtt{l}_1(L), \mathtt{l}_2(L)\}$ for all $L \subseteq [c]$,
- $\mathtt{s}^{\mathtt{a}}(L_1, L_2) = \min\{\mathtt{s}^{\mathtt{a}}_1(L_1, L_2), \mathtt{s}^{\mathtt{a}}_2(L_1, L_2)\}$ for all $L_1, L_2 \subseteq [c]$,
- $\mathtt{l}^{\mathtt{a}}(L_1, L_2) = \max\{\mathtt{l}^{\mathtt{a}}_1(L_1, L_2), \mathtt{l}^{\mathtt{a}}_2(L_1, L_2)\}$ for all $L_1, L_2 \subseteq [c]$, and
- $\mathtt{d}(L_1, L_2) = \min\{\mathtt{d}_1(L_1, L_2), \mathtt{d}_2(L_1, L_2)\}$ for all $L_1, L_2 \subseteq [c]$.

There are at most n^{2^c} possible pairs for $(\mathtt{n}_1, \mathtt{n}_2)$ as $\mathtt{n}_2$ is uniquely determined by $\mathtt{n}_1$. Similarly, there are at most n^{2^c} possible pairs for $(\overline{\mathtt{n}}_1, \overline{\mathtt{n}}_2)$. There are at most $n^{2 \cdot 2^c}$ candidates each for $(\mathtt{s}_1, \mathtt{s}_2)$ and $(\mathtt{l}_1, \mathtt{l}_2)$, and at most $n^{4 \cdot 2^c}$ candidates each for $(\mathtt{s}^{\mathtt{a}}_1, \mathtt{s}^{\mathtt{a}}_2)$, $(\mathtt{l}^{\mathtt{a}}_1, \mathtt{l}^{\mathtt{a}}_2)$, and $(\mathtt{d}_1, \mathtt{d}_2)$. In total, there are at most $n^{18 \cdot 2^c}$ candidates for the tuples $(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}^{\mathtt{a}}_1, \mathtt{l}^{\mathtt{a}}_1, \mathtt{d}_1)$ and $(\mathtt{n}_2, \overline{\mathtt{n}}_2, \mathtt{s}_2, \mathtt{l}_2, \mathtt{s}^{\mathtt{a}}_2, \mathtt{l}^{\mathtt{a}}_2, \mathtt{d}_2)$. Each candidate can be checked in $O(1)$ time, and thus the lemma holds. □

Lemma 6.5 *For a $\rho_{i,j}$-node t, $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d})$ can be computed in time $O(n^{10 \cdot 2^c})$.*

Proof: Let t_1 be the child of t in T. Observe that a component with label set L in G_t has label set either L, L^{+i}, or L^{+i-j} in G_{t_1}, where $L^{+i} = L \cup \{i\}$ and $L^{+i-j} = L \cup \{i\} \setminus \{j\}$. Thus $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d}) = \mathsf{true}$ if and only if there exists a tuple $(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}^{\mathtt{a}}_1, \mathtt{l}^{\mathtt{a}}_1, \mathtt{d}_1)$ such that $\mathtt{dp}_{t_1}(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}^{\mathtt{a}}_1, \mathtt{l}^{\mathtt{a}}_1, \mathtt{d}_1) = \mathsf{true}$, where for all $L \subseteq [c]$:

- if $i \in L$, then $\mathtt{n}(L) = \overline{\mathtt{n}}(L) = 0$, $\mathtt{s}(L) = \infty$, and $\mathtt{l}(L) = -\infty$;
- if $i, j \notin L$, then $\mathtt{n}(L) = \mathtt{n}_1(L)$, $\overline{\mathtt{n}}(L) = \overline{\mathtt{n}}_1(L)$, $\mathtt{s}(L) = \mathtt{s}_1(L)$, and $\mathtt{l}(L) = \mathtt{l}_1(L)$;
- if $i \notin L$ and $j \in L$, then:
  - $\mathtt{n}(L) = \mathtt{n}_1(L) + \mathtt{n}_1(L^{+i}) + \mathtt{n}_1(L^{+i-j})$,
  - $\overline{\mathtt{n}}(L) = \overline{\mathtt{n}}_1(L) + \overline{\mathtt{n}}_1(L^{+i}) + \overline{\mathtt{n}}_1(L^{+i-j})$,
  - $\mathtt{s}(L) = \min\{\mathtt{s}_1(L), \mathtt{s}_1(L^{+i}), \mathtt{s}_1(L^{+i-j})\}$, and
  - $\mathtt{l}(L) = \max\{\mathtt{l}_1(L), \mathtt{l}_1(L^{+i}), \mathtt{l}_1(L^{+i-j})\}$;

and for all $L_1, L_2 \subseteq [c]$:

- if $i \in L_1 \cup L_2$, then $\mathtt{s}^{\mathtt{a}}(L_1, L_2) = \infty$, $\mathtt{l}^{\mathtt{a}}(L_1, L_2) = -\infty$, and $\mathtt{d}(L_1, L_2) = \infty$;
- if $i, j \notin L_1 \cup L_2$, then $\mathtt{s}^{\mathtt{a}}(L_1, L_2) = \mathtt{s}^{\mathtt{a}}_1(L_1, L_2)$, $\mathtt{l}^{\mathtt{a}}(L_1, L_2) = \mathtt{l}^{\mathtt{a}}_1(L_1, L_2)$, and $\mathtt{d}(L_1, L_2) = \mathtt{d}_1(L_1, L_2)$;

- if $i \notin L_1 \cup L_2$ and $j \in L_1 \cup L_2$, then

$$\begin{aligned}
\mathtt{s}^{\mathtt{a}}(L_1, L_2) &= \min\{\mathtt{s}^{\mathtt{a}}(L_1', L_2') : L_1' \in \mathcal{L}_1, L_2' \in \mathcal{L}_2\},\\
\mathtt{l}^{\mathtt{a}}(L_1, L_2) &= \max\{\mathtt{l}^{\mathtt{a}}(L_1', L_2') : L_1' \in \mathcal{L}_1, L_2' \in \mathcal{L}_2\},\\
\mathtt{d}(L_1, L_2) &= \min\{\mathtt{d}(L_1', L_2') : L_1' \in \mathcal{L}_1, L_2' \in \mathcal{L}_2\},
\end{aligned}$$

  where, for $h \in \{1, 2\}$, $\mathcal{L}_h = \{L_h\}$ if $j \notin L_h$, and $\mathcal{L}_h = \{L_h, L_h^{+i}, L_h^{+i-j}\}$ if $j \in L_h$.

The claimed running time follows from the fact that there are $O(n^{10 \cdot 2^c})$ candidates for $(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}_1^{\mathtt{a}}, \mathtt{l}_1^{\mathtt{a}}, \mathtt{d}_1)$ and that each candidate can be checked in $O(1)$ time. □

Lemma 6.6 *For an $\eta_{i,j}$-node t, $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d})$ can be computed in time $O(n^{10 \cdot 2^c})$.*

Proof: Let t_1 be the child of t in T. If there is a set $S \subseteq V(G_t)$ corresponding to the tuple $\tau = (\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d})$, then $V_t^h \cap S$ is non-empty if and only if $\mathtt{n}(\tau, h) := \sum_{C \in \mathcal{C}(S,L),\ h \in L \subseteq [c]} |C| > 0$. Also, $V_t^h - S$ is non-empty if and only if $\overline{\mathtt{n}}(\tau, h) := \sum_{D \in \mathcal{D}(S,L),\ h \in L \subseteq [c]} |D| > 0$. Given a tuple, these conditions can be checked in linear time. Hence we assume that we know which cases apply to the tuple.

Let $\mathcal{L}_{i,j} = \{L \subseteq [c] : L \cap \{i, j\} \neq \emptyset, \mathtt{n}(L) > 0\}$ and $\overline{\mathcal{L}}_{i,j} = \{L \subseteq [c] : L \cap \{i, j\} \neq \emptyset, \overline{\mathtt{n}}(L) > 0\}$. By slightly abusing the notation, we denote by $\bigcup \mathcal{L}_{i,j}$ the union $\bigcup_{L \in \mathcal{L}_{i,j}} L$, and by $\bigcup \overline{\mathcal{L}}_{i,j}$ the union $\bigcup_{L \in \overline{\mathcal{L}}_{i,j}} L$. If $V_t^i \cap S \neq \emptyset$ and $V_t^j \cap S \neq \emptyset$, then all components in $G_{t_1}[S]$ containing an i-vertex or a j-vertex will be merged into one component with the color set $\bigcup \mathcal{L}_{i,j}$ in $G_t[S]$. Otherwise, at most one color class i or j contains vertices of S. Hence no components of $G_{t_1}[S]$ will be merged in $G_t[S]$, while the edges added between V_t^i and V_t^j make each component in $G_t[S]$ with i-vertices (j-vertices), if any, adjacent to each component in $G_t - S$ with j-vertices (i-vertices, respectively). The analogous observations hold also for the components in $G_{t_1} - S$ and $G_t - S$.

The following claims are almost direct consequences of the discussion above.

Claim 6.7 *If $\mathtt{n}(L) > 0$, then one of the following cases hold:*

- *$i, j \notin L$;*
- *$|\{i, j\} \cap L| = 1$ and $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) = 0$; or*
- *$i, j \in L = \bigcup \mathcal{L}_{i,j}$ and $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) > 0$.*

Proof: If none of the cases above holds, then we have either

1. $\{i, j\} \cap L \neq \emptyset$, $L \neq \bigcup \mathcal{L}_{i,j}$, and $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) > 0$; or
2. $i, j \in L$ and $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) = 0$.

In the first case, $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) > 0$ implies that there is a unique component with color class L such that $L \cap \{i, j\} \neq \emptyset$. Furthermore, we know that $L = \bigcup \mathcal{L}_{i,j}$. In the second case, $\mathtt{n}(\tau, i) = 0$ or $\mathtt{n}(\tau, j) = 0$ holds, and thus no component can contain both i-vertices and j-vertices. □

We can prove the next claim in the same way.

Claim 6.8 *If $\overline{\mathtt{n}}(L) > 0$, then one of the following cases hold:*

- $i, j \notin L$*;*
- $|\{i, j\} \cap L| = 1$ *and* $\overline{\mathtt{n}}(\tau, i) \cdot \overline{\mathtt{n}}(\tau, j) = 0$*; or*
- $i, j \in L = \bigcup \overline{\mathcal{L}}_{i,j}$ *and* $\overline{\mathtt{n}}(\tau, i) \cdot \overline{\mathtt{n}}(\tau, j) > 0$.

In the following, we only explain the case where each color set satisfies the condition in Claim 6.7 or Claim 6.8 (depending on which function we are talking about), since otherwise the values of functions can be trivially determined.

Now we can see that $\mathtt{dp}_t(\mathtt{n}, \overline{\mathtt{n}}, \mathtt{s}, \mathtt{l}, \mathtt{s}^{\mathtt{a}}, \mathtt{l}^{\mathtt{a}}, \mathtt{d}) = \mathtt{true}$ if and only if there exists a tuple $(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}_1^{\mathtt{a}}, \mathtt{l}_1^{\mathtt{a}}, \mathtt{d}_1)$ such that $\mathtt{dp}_{t_1}(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}_1^{\mathtt{a}}, \mathtt{l}_1^{\mathtt{a}}, \mathtt{d}_1) = \mathtt{true}$, where for all $L \subseteq [c]$, the following holds:

- if $i, j \notin L$, then $\mathtt{n}(L) = \mathtt{n}_1(L)$, $\overline{\mathtt{n}}(L) = \overline{\mathtt{n}}_1(L)$, $\mathtt{s}(L) = \mathtt{s}_1(L)$, and $\mathtt{l}(L) = \mathtt{l}_1(L)$;
- if $|\{i, j\} \cap L| = 1$ and $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) = 0$, then $\mathtt{n}(L) = \mathtt{n}_1(L)$ and $\mathtt{s}(L) = \mathtt{s}_1(L)$;
- if $|\{i, j\} \cap L| = 1$ and $\overline{\mathtt{n}}(\tau, i) \cdot \overline{\mathtt{n}}(\tau, j) = 0$, then $\overline{\mathtt{n}}(L) = \overline{\mathtt{n}}_1(L)$ and $\mathtt{l}(L) = \mathtt{l}_1(L)$;
- if $i, j \in L = \bigcup \mathcal{L}_{i,j}$ and $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) > 0$, then $\mathtt{n}(L) = \sum_{Q \in \mathcal{L}_{i,j}} \mathtt{n}_1(Q)$ and $\mathtt{s}(L) = \mathtt{n}(L)$;
- if $i, j \in L = \bigcup \overline{\mathcal{L}}_{i,j}$ and $\overline{\mathtt{n}}(\tau, i) \cdot \overline{\mathtt{n}}(\tau, j) > 0$, then $\overline{\mathtt{n}}(L) = \sum_{Q \in \overline{\mathcal{L}}_{i,j}} \overline{\mathtt{n}}_1(Q)$ and $\mathtt{l}(L) = \overline{\mathtt{n}}(L)$;

and for all $L_1, L_2 \subseteq [c]$, the following holds:

- **(no related merge / no new edge)** if either
  - $i, j \notin L_1 \cup L_2$
  - $i, j \notin L_1$, $|\{i, j\} \cap L_2| = 1$, and $\overline{\mathtt{n}}(\tau, i) \cdot \overline{\mathtt{n}}(\tau, j) = 0$, or
  - $i, j \notin L_2$, $|\{i, j\} \cap L_1| = 1$, and $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) = 0$,

  then $\mathtt{s}^{\mathtt{a}}(L_1, L_2) = \mathtt{s}_1^{\mathtt{a}}(L_1, L_2)$, $\mathtt{l}^{\mathtt{a}}(L_1, L_2) = \mathtt{l}_1^{\mathtt{a}}(L_1, L_2)$, and $\mathtt{d}(L_1, L_2) = \mathtt{d}_1(L_1, L_2)$;
- **(related but unmerged color sets)** if $|\{i, j\} \cap L_1| = |\{i, j\} \cap L_2| = 1$, $\mathtt{n}(\tau, i) \cdot \mathtt{n}(\tau, j) = 0$, $\overline{\mathtt{n}}(\tau, i) \cdot \overline{\mathtt{n}}(\tau, j) = 0$, then

- if $\{i,j\} \cap L_1 = \{i,j\} \cap L_2$, then $\mathtt{s}^{\mathtt{a}}(L_1,L_2) = \mathtt{s}^{\mathtt{a}}_1(L_1,L_2)$, $\mathtt{l}^{\mathtt{a}}(L_1,L_2) = \mathtt{l}^{\mathtt{a}}_1(L_1,L_2)$, and $\mathtt{d}(L_1,L_2) = \mathtt{d}_1(L_1,L_2)$;
- if $\{i,j\} \cap L_1 \neq \{i,j\} \cap L_2$, then $\mathtt{s}^{\mathtt{a}}(L_1,L_2) = \mathtt{s}(L_1)$, $\mathtt{l}^{\mathtt{a}}(L_1,L_2) = \mathtt{l}(L_2)$, and $\mathtt{d}(L_1,L_2) = \mathtt{s}(L_1) - \mathtt{l}(L_2)$;

- **(merged color set L_1)** if $i,j \in L_1 = \bigcup \mathcal{L}_{i,j}$ and $\mathtt{n}(\tau,i) \cdot \mathtt{n}(\tau,j) > 0$, then
  - if $i,j \notin L_2$, then

$$\mathtt{s}^{\mathtt{a}}(L_1,L_2) = \begin{cases} \infty & \text{if } \min_{L_1' \in \mathcal{L}_{i,j}} \mathtt{s}^{\mathtt{a}}_1(L_1',L_2) = \infty, \\ \mathtt{n}(L_1) & \text{otherwise,} \end{cases}$$

    $\mathtt{l}^{\mathtt{a}}(L_1,L_2) = \max_{L_1' \in \mathcal{L}_{i,j}} \mathtt{l}^{\mathtt{a}}_1(L_1',L_2)$, and $\mathtt{d}(L_1,L_2) = \mathtt{s}^{\mathtt{a}}(L_1,L_2) - \mathtt{l}^{\mathtt{a}}(L_1,L_2)$;
  - if either
    - $|\{i,j\} \cap L_2| = 1$, $\overline{\mathtt{n}}(\tau,i) \cdot \overline{\mathtt{n}}(\tau,j) = 0$, and $\overline{\mathtt{n}}(L_2) \neq 0$, or
    - $i,j \in L_2 = \bigcup \overline{\mathcal{L}}_{i,j}$ and $\overline{\mathtt{n}}(\tau,i) \cdot \overline{\mathtt{n}}(\tau,j) > 0$,

    then it holds that $\mathtt{s}^{\mathtt{a}}(L_1,L_2) = \mathtt{n}(L_1)$, $\mathtt{l}^{\mathtt{a}}(L_1,L_2) = \mathtt{l}(L_2)$, and $\mathtt{d}(L_1,L_2) = \mathtt{s}^{\mathtt{a}}(L_1,L_2) - \mathtt{l}^{\mathtt{a}}(L_1,L_2)$;

- **(merged color set L_2)** if $i,j \in L_2 = \bigcup \overline{\mathcal{L}}_{i,j}$ and $\overline{\mathtt{n}}(\tau,i) \cdot \overline{\mathtt{n}}(\tau,j) > 0$, then
  - if $i,j \notin L_1$, then $\mathtt{s}^{\mathtt{a}}(L_1,L_2) = \min_{L_2' \in \overline{\mathcal{L}}_{i,j}} \mathtt{s}^{\mathtt{a}}_1(L_1,L_2')$,

$$\mathtt{l}^{\mathtt{a}}(L_1,L_2) = \begin{cases} -\infty & \text{if } \max_{L_2' \in \overline{\mathcal{L}}_{i,j}} \mathtt{l}^{\mathtt{a}}_1(L_1,L_2') = -\infty, \\ \overline{\mathtt{n}}(L_2) & \text{otherwise,} \end{cases}$$

    $\mathtt{d}(L_1,L_2) = \mathtt{s}^{\mathtt{a}}(L_1,L_2) - \mathtt{l}^{\mathtt{a}}(L_1,L_2)$;
  - if $|\{i,j\} \cap L_1| = 1$, $\mathtt{n}(\tau,i) \cdot \mathtt{n}(\tau,j) = 0$, and $\mathtt{n}(L_1) \neq 0$, then $\mathtt{s}^{\mathtt{a}}(L_1,L_2) = \mathtt{s}(L_1)$, $\mathtt{l}^{\mathtt{a}}(L_1,L_2) = \overline{\mathtt{n}}(L_2)$, and $\mathtt{d}(L_1,L_2) = \mathtt{s}^{\mathtt{a}}(L_1,L_2) - \mathtt{l}^{\mathtt{a}}(L_1,L_2)$;

The number of candidate tuples $(\mathtt{n}_1, \overline{\mathtt{n}}_1, \mathtt{s}_1, \mathtt{l}_1, \mathtt{s}^{\mathtt{a}}_1, \mathtt{l}^{\mathtt{a}}_1, \mathtt{d}_1)$ is $O(n^{10 \cdot 2^c})$ and each of them can be checked in $O(1)$ time. □

7 Faster algorithms parameterized by the solution size

We know that both SS and CSS admit FPT algorithms [1] when parameterized by the solution size. The algorithms by Águeda et al. [1] use Courcelle's theorem [7]. It is known that such algorithms may have non-elementary dependency on the parameter [17]. The natural question would be whether they admit $O^*(k^k)$-time[3] algorithms as is the case for vertex integrity [12].

[3]The $O^*(\cdot)$ notation omits terms bounded by a polynomial in the input size.

We answer this question with the following theorems. The first step of our algorithm for SS is a branching procedure to first guess the correct number of components (k choices) and then guess their sizes (at most k^k choices). We complete our solutions (ensuring they are connected) by constructing and solving appropriate Steiner Tree sub-instances. With a simple modification our algorithm also works for CSS.

Theorem 7.1 Safe Set *can be solved in* $O^*(2^k k^{3k})$ *time, where* k *is the solution size.*

Proof: As before, we assume that the input graph is connected, otherwise we solve the problem on each component and take the minimum. Suppose that the input graph G has a safe set $O \subseteq V(G)$ of size k. Since $G[O]$ is not necessarily connected, suppose that $G[O]$ has ℓ components, for $\ell \leq k$, $O_1, \ldots, O_\ell$. The first step of our algorithm is to guess the value of ℓ (k choices) and then guess the ℓ sizes of the components of $G[O]$ (at most $k^\ell \leq k^k$ choices). For all $i \in [\ell]$ we denote $k_i := |O_i|$. In the remainder we assume that the algorithm guessed these values correctly, as we will repeat it for all possible values.

The first phase of our algorithm is a branching process. We maintain ℓ sets of vertices $S_1, \ldots, S_\ell$ with the intuitive meaning that $S_i \subseteq O_i$ for all $i \in [\ell]$. We denote $S := \cup_{i=1}^{\ell} S_i$. Initially $S := \emptyset$. If at any point we have $|S_i| > k_i$ for some i, the algorithm rejects this branch.

We will say that a vertex $u \in V(G) \setminus S$ is *problematic* if it fulfills one of the following two properties: (i) the component of $G - S$ that contains u has size at least $k + 1$, or (ii) there exists an $i \in [\ell]$ such that $N(u) \cap S_i \neq \emptyset$ and the component of $G - S$ that contains u has size at least $k_i + 1$. It is not hard to see from this description that finding a problematic vertex can be done in polynomial time.

The main branching step of our algorithm is now the following: we check if there exists a problematic vertex u. If it does, then we find a set of vertices C such that $u \in C$, C is connected in $G - S$, and C fulfills the following properties: (i) if the component of u in $G - S$ has size at least $k + 1$, then $|C| = k + 1$ or, (ii) if the component of u in $G - S$ has size at least $k_i + 1$ for some $i \in [\ell]$ with $N(u) \cap S_i \neq \emptyset$, then $|C| = k_i + 1$. It is not hard to see that this can be done in polynomial time. Now we produce $|C|\ell \leq (k + 1)k$ branches: for each vertex $w \in |C|$ we consider the case where we place w in S_i for $i \in [\ell]$. We then call the same algorithm recursively.

To see the correctness of this branching procedure, we observe that if the branching is correct thus far, that is, if $S_i \subseteq O_i$ and $k_i = |O_i|$ for all i, then $O \cap C \neq \emptyset$. In case (i) this is clear as we have a connected set of size $k + 1$. In case (ii) this is a consequence of the fact that C is adjacent to S_i (therefore, to O_i), but has size strictly larger than the size we guessed (correctly) for O_i.

The above search-tree process produces $O(k^{2k})$ branches, since we have at most $(k+1)k$ choices in each branch, and the branching depth is at most k (since in each branch we add a vertex to S). The branching procedure terminates either because $|S_i| > k_i$ for some i (in which case we reject), or because there are no

more problematic vertices in the graph. Let us now explain how to complete the solution in this case.

At this point we have ℓ sets $S_1, \ldots, S_\ell$ with the property that for all $i \in [\ell]$, every component of $G - S$ that is adjacent to S_i has size at most k_i. However, we do not have a feasible solution yet because $G[S_i]$ is not necessarily connected for all i. We therefore construct ℓ instances of Steiner Tree: for each $i \in [\ell]$, we construct an instance where the set of terminals that must be connected is S_i and we solve Steiner Tree on the graph $G - (\cup_{j \neq i} S_j)$ (in other words, for each set S_i we find a Steiner Tree that connects S_i without using any of the vertices of $S \setminus S_i$). We execute the algorithm of Nederlof [31], which runs in time $O^*(2^t)$, where t is the number of terminals. The algorithm returns ℓ sets $S'_1, S'_2, \ldots, S'_\ell$, such that for all $i \in [\ell]$ we have $S_i \subseteq S'_i$ and $G[S'_i]$ is connected. If for some i we have $|S'_i| > k_i$ or there is no solution (because deleting $S \setminus S_i$ disconnects S_i) we reject this branch. Otherwise, for each i such that $|S'_i| < k$, we augment S'_i by adding to it arbitrary neighbors so that it remains connected and we have in the end $|S'_i| = k_i$. We return $S' := \cup_{i=1}^{\ell} S'_i$ as our solution.

To see that the above algorithm is correct, we first argue that if we return a solution S', then S' clearly has size k, so we only need to explain why S' is a safe set. Suppose for contradiction that S' is not a safe set, therefore there exists a component of $G[S']$ which is adjacent to a component C of $G - S'$ of larger size. In other words, there exists an $i \in [\ell]$ such that $G[S'_i]$ has a neighbor in a component C of $G - S'$ with $|C| > |S'_i| = k_i$. Let S_i be the set of terminals on which we ran the i-th Steiner Tree procedure, and S the set of vertices we had when the branching procedure stopped. We will show that the graph contained a problematic vertex (with respect to S_i), and therefore the branching procedure could not have stopped. Take a vertex $u_1 \in C$ that has a neighbor in S'_i and consider a shortest path in $G - S$ from u_1 to a vertex that has a neighbor in S_i. Such a path exists, because in $G - (S \setminus S_i)$ all vertices of S_i are in the same component (otherwise there would be no Steiner Tree connecting them), which is the same component that contains all vertices of S'_i, and u_1 has a neighbor in S'_i. Let u_2 be the last vertex of this path from u_1 to S_1. Then u_2 is problematic: we have $N(u_2) \cap S_i \neq \emptyset$, and the component that u_2 belongs to in $G - S$ is at least as large as C.

For the other direction, if O exists and we have guessed the k_i values correctly and $S_i \subseteq O_i$ for all i, then the Steiner Tree instances will all return a solution that we accept, as O_i itself is a valid solution of the proper size. Therefore, if we reject in this phase it implies that no solution exists with the guessed properties.

Finally, for the running time, we repeat the algorithm $O^*(k^k)$ times (for each value of ℓ, and values of k_i), each repetition has a branching step with $O^*(k^{2k})$ leaves, and in each leaf we run ℓ times the Steiner Tree algorithm, each with at most k terminals, therefore taking at most $O^*(2^k)$. Thus, the total running time is at most $O^*(2^k k^{3k})$. □

When we set $\ell = 1$, the algorithm above will find a connected safe set of size at most k (if one exists). In that case, we have a single execution of the branching algorithm with the search tree size $O(k^k)$ in which we execute the

$O^*(2^k)$-time Steiner Tree algorithm for each leaf. Thus we have the following corollary.

Corollary 7.2 Connected Safe Set *can be solved in* $O^*(2^k k^k)$ *time, where* k *is the size of the solution.*

8 Conclusion

We studied the parameterized complexity of Safe Set and Connected Safe Set and showed the results summarized in Figure 1. As the figure shows, we presented complexity borders of the problems parameterized by the pathwidth, the safe number, and the neighborhood diversity of the input graph. To further sharpen our results, it would be interesting to investigate the unsettled cases parameterized by the treedepth and by the modular-width. One obstacle we realized when tackling the modular-width case was that if we take a similar approach as in Section 5, some of the constraints would not be linear.

Although the concept of safe set was originally introduced as a facility location problem [20], it would be also important as a vulnerability measure of networks [2]. We compared the safe number to other vulnerability measures (vertex integrity and ℓ-component order connectivity) and showed that the safe number behaves quite differently from a parametrized-complexity point of view. It would be also interesting to investigate how different the safe number is in the study of network vulnerability.

References

[1] Raquel Águeda, Nathann Cohen, Shinya Fujita, Sylvain Legay, Yannis Manoussakis, Yasuko Matsui, Leandro Montero, Reza Naserasr, Hirotaka Ono, Yota Otachi, Tadashi Sakuma, Zsolt Tuza, and Renyu Xu. Safe sets in graphs: Graph classes and structural parameters. *J. Comb. Optim.*, 36(4):1221–1242, 2018. doi:10.1007/s10878-017-0205-2.

[2] Ravindra B. Bapat, Shinya Fujita, Sylvain Legay, Yannis Manoussakis, Yasuko Matsui, Tadashi Sakuma, and Zsolt Tuza. Safe sets, network majority on weighted trees. *Networks*, 71:81–92, 2018. doi:10.1002/net.21794.

[3] Curtis A. Barefoot, Roger C. Entringer, and Henda C. Swart. Vulnerability in graphs — a comparative survey. *J. Combin. Math. Combin. Comput.*, 1:13–22, 1987.

[4] Walid Ben-Ameur, Mohamed-Ahmed Mohamed-Sidi, and José Neto. The k-separator problem: polyhedra, complexity and approximation results. *J. Comb. Optim.*, 29(1):276–307, 2015. doi:10.1007/s10878-014-9753-x.

[5] Hans L. Bodlaender, Rodney G. Downey, Michael R. Fellows, and Danny Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009. doi:10.1016/j.jcss.2009.04.001.

[6] Édouard Bonnet, Nick Brettell, O-joung Kwon, and Dániel Marx. Generalized feedback vertex set problems on bounded-treewidth graphs: Chordality is the key to single-exponential parameterized algorithms. In *IPEC 2017*, volume 89 of *LIPIcs*, pages 7:1–7:13, 2017. doi:10.4230/LIPIcs.IPEC.2017.7.

[7] Bruno Courcelle. The monadic second-order logic of graphs III: tree-decompositions, minor and complexity issues. *Theor. Inform. Appl.*, 26:257–286, 1992. doi:10.1051/ita/1992260302571.

[8] Bruno Courcelle and Stephan Olariu. Upper bounds to the clique width of graphs. *Discrete Appl. Math.*, 101:77–114, 2000. doi:10.1016/S0166-218X(99)00184-5.

[9] Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015. doi:10.1007/978-3-319-21275-3.

[10] Michael Dom, Daniel Lokshtanov, and Saket Saurabh. Kernelization lower bounds through colors and ids. *ACM Trans. Algorithms*, 11(2):13:1–13:20, 2014. doi:10.1145/2650261.

[11] Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM J. Comput.*, 24(4):873–921, 1995. doi:10.1137/S0097539792228228.

[12] Pål Grønås Drange, Markus S. Dregi, and Pim van 't Hof. On the computational complexity of vertex integrity and component order connectivity. *Algorithmica*, 76(4):1181–1202, 2016. doi:10.1007/s00453-016-0127-x.

[13] Pavel Dvořák, Eduard Eiben, Robert Ganian, Dušan Knop, and Sebastian Ordyniak. Solving integer linear programs with a small number of global variables and constraints. In *IJCAI 2017*, pages 607–613, 2017. doi:10.24963/ijcai.2017/85.

[14] Stefan Ehard and Dieter Rautenbach. Approximating connected safe sets in weighted trees. *Discrete Appl. Math.* To appear. doi:10.1016/j.dam.2019.11.017.

[15] Michael R. Fellows, Daniel Lokshtanov, Neeldhara Misra, Frances A. Rosamond, and Saket Saurabh. Graph layout problems parameterized by vertex cover. In *ISAAC 2008*, volume 5369 of *Lecture Notes in Computer Science*, pages 294–305, 2008. doi:10.1007/978-3-540-92182-0_28.

[16] András Frank and Éva Tardos. An application of simultaneous diophantine approximation in combinatorial optimization. *Combinatorica*, 7:49–65, 1987. doi:10.1007/BF02579200.

[17] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. *Ann. Pure Appl. Logic*, 130(1-3):3–31, 2004. doi:10.1016/j.apal.2004.01.007.

[18] Shinya Fujita and Michitaka Furuya. Safe number and integrity of graphs. *Discrete Appl. Math.*, 247:398–406, 2018. doi:10.1016/j.dam.2018.03.074.

[19] Shinya Fujita, Tommy Jensen, Boram Park, and Tadashi Sakuma. On the weighted safe set problem on paths and cycles. *J. Comb. Optim.*, 37(2):685–701, 2019. doi:10.1007/s10878-018-0316-4.

[20] Shinya Fujita, Gary MacGillivray, and Tadashi Sakuma. Safe set problem on graphs. *Discrete Appl. Math.*, 215:106–111, 2016. doi:10.1016/j.dam.2016.07.020.

[21] Alexander Göke, Dániel Marx, and Matthias Mnich. Parameterized algorithms for generalizations of directed feedback vertex set. In *CIAC 2019*, volume 11485 of *Lecture Notes in Computer Science*, pages 249–261, 2019. doi:10.1007/978-3-030-17402-6_21.

[22] Petr Hlinený and Sang-il Oum. Finding branch-decompositions and rank-decompositions. *SIAM J. Comput.*, 38(3):1012–1032, 2008. doi:10.1137/070685920.

[23] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Math. Oper. Res.*, 12:415–440, 1987. doi:10.1287/moor.12.3.415.

[24] Dieter Kratsch, Ton Kloks, and Haiko Müller. Measuring the vulnerability for classes of intersection graphs. *Discrete Appl. Math.*, 77(3):259–270, 1997. doi:10.1016/S0166-218X(96)00133-3.

[25] Mithilesh Kumar and Daniel Lokshtanov. A $2\ell k$ kernel for ℓ-component order connectivity. In *IPEC 2016*, volume 63 of *LIPIcs*, pages 20:1–20:14, 2016. doi:10.4230/LIPIcs.IPEC.2016.20.

[26] Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012. doi:10.1007/s00453-011-9554-x.

[27] Euiwoong Lee. Partitioning a graph into small pieces with applications to path transversal. *Math. Program.*, 177(1-2):1–19, 2019. doi:10.1007/s10107-018-1255-7.

[28] Hendrik W. Lenstra Jr. Integer programming with a fixed number of variables. *Math. Oper. Res.*, 8:538–548, 1983. doi:10.1287/moor.8.4.538.

[29] Yinkui Li, Shenggui Zhang, and Qilong Zhang. Vulnerability parameters of split graphs. *Int. J. Comput. Math.*, 85(1):19–23, 2008. doi:10.1080/00207160701365721.

[30] Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Math.*, 201(1-3):189–241, 1999. doi:10.1016/S0012-365X(98)00319-7.

[31] Jesper Nederlof. Fast polynomial-space algorithms using inclusion-exclusion. *Algorithmica*, 65(4):868–884, 2013. doi:10.1007/s00453-012-9630-x.

[32] Sang-il Oum. Approximating rank-width and clique-width quickly. *ACM Trans. Algorithms*, 5, 2008. doi:10.1145/1435375.1435385.

[33] Sang-il Oum and Paul D. Seymour. Approximating clique-width and branch-width. *J. Comb. Theory, Ser. B*, 96:514–528, 2006. doi:10.1016/j.jctb.2005.10.006.

[34] Sibabrata Ray, Rajgopal Kannan, Danyang Zhang, and Hong Jiang. The weighted integrity problem is polynomial for interval graphs. *Ars Combin.*, 79:77–95, 2006.

[35] Marc Tedder, Derek G. Corneil, Michel Habib, and Christophe Paul. Simpler linear-time modular decomposition via recursive factorizing permutations. In *ICALP 2008 (1)*, volume 5125 of *Lecture Notes in Computer Science*, pages 634–645, 2008. doi:10.1007/978-3-540-70575-8_52.

[36] Mingyu Xiao. Linear kernels for separating a graph into components of bounded size. *J. Comput. Syst. Sci.*, 88:260–270, 2017. doi:10.1016/j.jcss.2017.04.004.