Breaking a Graph into Connected Components with Small Dominating Sets

Matthias Bentert
University of Bergen, Norway

Michael R. Fellows
University of Bergen, Norway
Lebanese American University, Beirut, Lebanon

Petr A. Golovach
University of Bergen, Norway

Frances A. Rosamond
University of Bergen, Norway
Lebanese American University, Beirut, Lebanon

Saket Saurabh
The Institute of Mathematical Sciences, Chennai, India

Abstract

We study Dominated Cluster Deletion. Therein, we are given an undirected graph $G = (V, E)$ and integers k and d and the task is to find a set of at most k vertices such that removing these vertices results in a graph in which each connected component has a dominating set of size at most d. We also consider the special case where d is a constant. We show an almost complete tetrachotomy in terms of para-NP-hardness, containment in XP, containment in FPT, and admitting a polynomial kernel with respect to parameterizations that are a combination of k, d, c, and Δ, where c and Δ are the degeneracy and the maximum degree of the input graph, respectively. As a main contribution, we show that the problem can be solved in $f(k, d) \cdot n^{O(d)}$ time, that is, the problem is FPT when parameterized by k when d is a constant. This answers an open problem asked in a recent Dagstuhl seminar (23331). For the special case $d = 1$, we provide an algorithm with running time $2^{\mathcal{O}(k \log k)}nm$. Furthermore, we show that even for $d = 1$, the problem does not admit a polynomial kernel with respect to $k + c$.

2012 ACM Subject Classification Theory of computation → Parameterized complexity and exact algorithms; Mathematics of computing → Graph algorithms

Keywords and phrases Parameterized Algorithms, Recursive Understanding, Polynomial Kernels, Degeneracy

Digital Object Identifier 10.4230/LIPIcs.MFCS.2024.24

Funding *Matthias Bentert*: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).
Michael R. Fellows: Supported by the Alexander von Humboldt Foundation under the Humboldt Renewed Research Stay in Germany and the Return to Germany Program for Humboldt Research Prizewinner Fellows.
Petr A. Golovach: Supported by the Research Council of Norway via the project BWCA (grant no. 314528).
Frances A. Rosamond: Supported by the Western Sydney University and the Alexander von Humboldt Foundation under the Humboldt Renewed Research Stay in Germany for Humboldt Research Prizewinner Fellows.
Saket Saurabh: Supported by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).

1 Introduction

Cluster Vertex Deletion is an important problem in theoretical computer science [3, 22]. Therein, one is given a graph and the task is to delete as few vertices as possible such that the resulting graph is a *cluster graph*, that is, a graph in which each connected component is a clique. However in many real-world applications, it is often too restrictive to require that each connected component is a clique. For this reason, many generalizations (but also special cases) of Cluster Vertex Deletion have been studied. Examples include variants where each connected component in the resulting graph is an s-club (a graph of diameter at most s) [12, 19] or an s-plex (a graph in which each vertex has degree at least $n - s$) [15, 23]. The special case, where each clique in the solution graph has to be of size one is Vertex Cover, one of the most studied problems in the entire field of theoretical computer science [1, 16].

Different problems of this clustering type were explored in the recent Dagstuhl seminar on "Recent Trends in Graph Decomposition" [2]. Starting from the observation that each connected component of the resulting graph in Cluster Vertex Deletion contains a dominating set of size 1, the participants introduced a problem that explores a generalization in which the resulting connected components (clusters) satisfy other "simple integrity requirements" (see Ajwani et al. [2]). The notion they came up with was the following.

Dominated Cluster Deletion

Input:	A graph G and positive integers k and d.
Question:	Is there a set S of at most k vertices such that the domination number of each connected component of $G - S$ is at most d?

In this work, we consider both Dominated Cluster Deletion as well as the special case where d is a constant. We refer to the latter problem as d-Dominated Cluster Deletion. A natural question to ask is whether Dominated Cluster Deletion is fixed parameter tracatble (FPT) when parameterized by $k+d$? That is, can the problem be solved in $f(k,d) \cdot n^{\mathcal{O}(1)}$, where f is a function that depends only on k and d. Unfortunately, even for $k = 0$, the problem corresponds to the Dominating Set, and hence it is unlikely to be FPT, as Dominating Set is W[2]-complete parameterized by solution size [10]. This led the participants of the Dagstuhl seminar to ask for a parameterized tractability result, where d is allowed to play an XP-role in the exponent of the polynomial. In particular, they ask the following question, which we answer in the affirmative.

> Can Dominated Cluster Deletion be solved in $f(k,d) \cdot n^{\mathcal{O}(d)}$ time?

Our Results. We study Dominated Cluster Deletion and d-Dominated Cluster Deletion and show an almost complete tetrachotomy in terms of para-NP-hardness, containment in XP, containment in FPT, and admitting a polynomial kernel with respect to parameterizations that are a combination of k, d, c, and Δ, where c and Δ are the degeneracy and the maximum degree of the input graph, respectively. As a main contribution and using the framework of *recursive understanding*, we show that the problem can be solved in $f(k,d) \cdot n^{O(d)}$ time, that is, the problem is FPT when parameterized by k when d is a constant. For the special case $d = 1$, we provide an algorithm with running time $2^{\mathcal{O}(k \log k)}nm$. Furthermore, we show that even for $d = 1$, the problem does not admit a polynomial kernel with respect to $k + c$ using OR-cross-compositions. Our results are summarized in Figure 1.

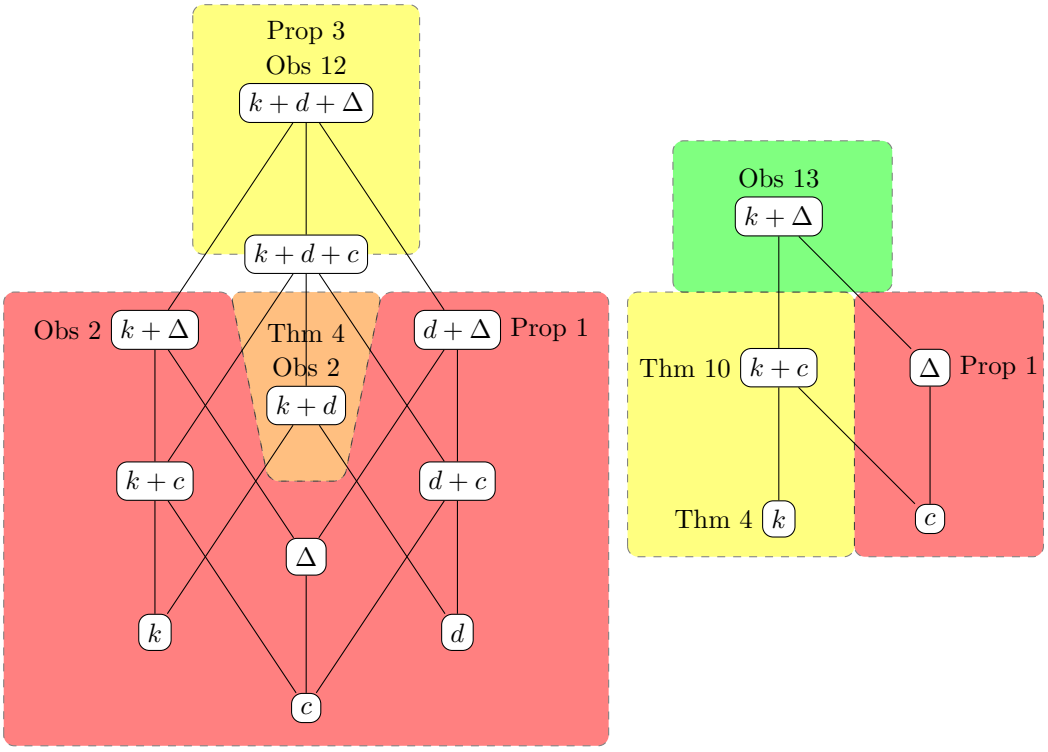

Figure 1 Overview over our results. Results for Dominated Cluster Deletion are shown on the left and results for d-Dominated Cluster Deletion are shown on the right. Therein, k and d are the input values and c and Δ are the degeneracy and the maximum degree of the input graph, respectively. An edge between two parameters α and β where β is above α indicates that the value of β in any instance will always be larger than the value of α in that instance. Any hardness result for β immediately implies the same hardness result for α and any positive result for α immediately implies the same positive result for β. The red boxes indicate parameters that lead to para-NP-hardness (NP-hardness for constant parameter values), an orange box indicates that the problem is in XP and W[1]-hard, a yellow box indicates that the parameterized problem is fixed-parameter tractable, but does not admit a polynomial kernel, and a green box indicated that the problem admits a polynomial kernel. We remark that all negative results apply even if the input graph is connected with the exception of $k+d+\Delta$, which does not admit a polynomial kernel in general but does so when the input graph is connected. We do not know whether Dominated Cluster Deletion parameterized by $k+d+c$ is fixed-parameter tractable or W[1]-hard.

2 Preliminaries and Basic Observations

We consider only undirected simple graphs and use the standard graph-theoretic terminology (see, e.g., [9]). Throughout the paper, we use n and m to denote the number of vertices and edges of G, respectively. For a set of vertices $X \subseteq V(G)$, $G[X]$ denotes the subgraph of G induced by the vertices in X. We define $G - X = G[V(G) \setminus X]$ and we write $G - v$ instead of $G - \{v\}$ for a single vertex. For a vertex v, $N_G(v)$ denotes the *open neighborhood* of v, that is, the set of vertices adjacent to v, and $N_G[v] = \{v\} \cup N_G(v)$ is the *closed neighborhood*. For $X \subseteq V(G)$, we define $N_G(X) = \left(\bigcup_{v \in X} N_G(v)\right) \setminus X$ and $N_G[X] = \bigcup_{v \in X} N_G[v]$. For a vertex v, $d_G(v) = |N_G(v)|$ denotes the *degree* of v. We write $P = v_0 v_1 \dots v_\ell$ to denote the path P with vertices $v_0, v_1, \dots, v_\ell$ and edges $\{v_{i-1}, v_i\}$ for $i \in \{1, \dots, \ell\}$. The vertices v_0 and v_ℓ are the *end-vertices* of P. The *length* of P is the number of edges of P. For a path P

with end-vertices u and v, we say that P is a (u,v)-*path*. A (u,v)-path of minimum length is a *shortest* (u,v)-path. We also say that P is a shortest path if P is a shortest (u,v)-path for the end-vertices u and v of P. A graph G is *connected* if for every two vertices u and v, G contains a path whose end-vertices are u and v. A *connected component* (or simply a *component*) is an inclusion maximal induced connected subgraph of G. The *diameter* of a connected graph G is the maximum length of a shortest (u,v)-path where the maximum is taken over all pairs of vertices u and v. A vertex u *dominates* $v \in V(G)$ if $v \in N_G[u]$. A set of vertices $D \subseteq V(G)$ is a *dominating set* if every vertex $v \in V(G)$ is dominated by some vertex of D, that is, $N_G[D] = V(G)$. The minimum size of a dominating set is called the *domination number* of G and is denoted by $\gamma(G)$. For two distinct vertices u and v of a graph G, a set $S \subseteq V(G)$ is a (u,v)-*separator* if $G - S$ has no (u,v)-path. A pair (A,B), where $A, B \subseteq V(G)$ and $A \cup B = V(G)$, is called a *separation* of G if there is no edge $\{u,v\}$ with $u \in A \setminus B$ and $v \in B \setminus A$. In other words, $A \cap B$ is a (u,v)-separator for every $u \in A \setminus B$ and $v \in B \setminus A$. We consider only separations with $A \setminus B \neq \emptyset$ and $B \setminus A \neq \emptyset$. The *order* of (A,B) is $|A \cap B|$.

We refer to the books of Downey and Fellows [11] and Cygan et al. [7] for a detailed introduction to parameterized complexity theory. Formally, a *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ^* is a set of strings over a finite alphabet Σ. This means that an input of a parameterized problem is a pair (x,k), where x is a string over Σ and $k \in \mathbb{N}$ is a *parameter*. A parameterized problem is *fixed-parameter tractable* (or FPT) if it can be solved in $f(k) \cdot |x|^{\mathcal{O}(1)}$ time for some computable function f. We also say that a parameterized problem belongs to the class XP if it can be solved in $|x|^{f(k)}$ time for some computable function f. The complexity class FPT contains all fixed-parameter tractable parameterized problems.

A *kernelization algorithm* (or *kernel*) for a parameterized problem is a polynomial-time algorithm that maps each instance (x,k) of a parameterized problem L to an instance (x',k') of the same problem such that (i) $(x,k) \in L$ if and only if $(x',k') \in L$, and (ii) $|x'|+k' \leq f(k)$ for some computable function f. A kernel is *polynomial* if f is polynomial. While it can be shown that every decidable parameterized problem is FPT if and only if it admits a kernel, it is unlikely that every problem in FPT has a polynomial kernel. In particular, the standard *cross-composition* technique [4] can be used to show that a parameterized problem has no polynomial kernel unless NP $\subseteq$ coNP/poly.

In the conclusion of the section, we show a couple of simple observations. We start by showing that Dominated Cluster Deletion is NP-complete. Notice that the property that each connected component of a graph has a dominating set of size at most d is not hereditary and we cannot use the general result of Lewis and Yannakakis [18] about deletion problems.

▶ **Proposition 1.** *For any constant $d \geq 1$, d-Dominated Cluster Deletion is* NP*-complete in graphs of maximum degree* 4*.*

Proof. We reduce from Vertex Cover in graphs of maximum degree 3, which is known to be NP-hard [14]. We remind that a set S of vertices in a graph G is a *vertex cover* if for each edge $\{u,v\}$ in G, $u \in S$ or $v \in S$. Then, the task of Vertex Cover is the following. Given a graph G and a positive integer k, decide whether G has a vertex cover of size at most k.

Let (G,k) be an instance of Vertex Cover where the maximum degree in G is 3. We construct the graph G' as follows:

- construct a copy of G,
- for each vertex $v \in V(G)$, construct a path P_v on $3d$ vertices and identify one end-vertex of P_v with v.

Note that the maximum degree in G' is four. We claim that G has a vertex cover of size at most k if and only if (G',k) is a yes-instance of d-Dominated Cluster Deletion.

If G has a vertex cover S of size at most k, then each connected component of $G - S$ is either P_v or $P_v - v$ for $v \in V(G)$. Observe that $\gamma(P_v) = \gamma(P_v - v) = d$. Thus, S is a solution to the instance (G', k) of d-Dominated Cluster Deletion, that is, (G', k) is a yes-instance.

For the opposite direction, assume that (G', k) is a yes-instance of Dominated Cluster Deletion, that is, there is a set $S' \subseteq V(G')$ of at most k vertices such that for each connected component C of $G' - S'$, $\gamma(C) \leq d$. We construct the set $S \subseteq V(G)$ from S' by replacing each vertex $u \in S' \setminus V(G)$ by the nearest vertex in G. This means that if u is a vertex of $P_v - v$ for some $v \in V(G)$ then u is replaced by v. Clearly, $|S| \leq |S'| \leq k$. We claim that S is a vertex cover of G. The proof is by contradiction. Assume that there is an edge $\{u, v\}$ of G such that $u \notin S$ and $v \notin S$. By the construction of S, we have that $V(P_u) \cap S' = \emptyset$ and $V(P_v) \cap S = \emptyset$. This means that the vertices of P_u and P_v are in the same component C of $G' - S'$. However, any set D_u dominating the vertices of $P_u - u$ has size at least d. Symmetrically, we need at least d vertices to dominate the vertices of $P_v - v$. This implies that $\gamma(C) \geq 2d > d$; a contradiction. We obtain that S is a vertex cover of G of size at most k. This concludes the proof. ◀

We next observe that Dominated Cluster Deletion with $k = 0$ is equivalent to Dominating Set on connected input graphs. In Dominating Set, we ask whether a graph G has a dominating set of size at most d. It is well-known that Dominating Set is W[2]-complete when parameterized by the solution size and hence, an algorithm with running time $f(d) \cdot n^{\mathcal{O}(1)}$ would contradict the basic assumptions of parameterized complexity that FPT $\neq$ W[2] [11]. Furthermore, the existence of an algorithm running in $f(k) \cdot n^{o(d)}$ time would contradict the Exponential Time Hypothesis (ETH) of Impagliazzo, Paturi, and Zane [17] (we refer to the textbook by Cygan et al. [7] for an introduction to computational lower bounds based on the ETH). Finally, Dominating Set is NP-hard on graphs of maximum degree 3 [5].

▶ **Observation 2.** *Dominated Cluster Deletion with $k = 0$ is NP-hard on subcubic graphs,* W[2]*-hard on general graphs, and any $f(k) \cdot n^{o(d)}$-time algorithm for it would contradict the ETH.*

Finally, we show that Dominated Cluster Deletion is fixed-parameter tractable when parameterized by $k + d + \Delta$.

▶ **Proposition 3.** *Dominated Cluster Deletion can be solved in $\mathcal{O}((d(\Delta + 1) + 1)^k \cdot 3^{d(\Delta+1)} \cdot (n + m))$ time.*

Proof. We construct a simple branching algorithm as follows. First, we find any connected subgraph with $d(\Delta + 1) + 1$ vertices. Note that any connected component in the graph obtained after removing a solution S can contain at most $d(\Delta + 1)$ vertices as it has a dominating set of size d and the maximum degree is Δ. Hence, each of the d vertices in the dominating set can dominate at most $(\Delta + 1)$ vertices each. Thus, at least one vertex of the chosen connected subgraph is contained in the solution S and we branch on that vertex and reduce k by one. After reaching depth k in the search tree, we discard the current branch if we can still find a connected subgraph with $d(\Delta + 1) + 1$ vertices. Each remaining branch represents a graph in which each connected component contains at most $d(\Delta + 1)$ vertices. For each such component, we can compute an optimal solution in $\mathcal{O}(3^{d(\Delta+1)}(n + m))$ time by trying all possible choices of assigning vertices in the subgraph to the solution S, the dominating set, or the rest of the graph. We can then check in linear time whether the current assignment is a valid solution and remember the solution that assigns a minimum number

of vertices to S. Finally, we sum up all the optimal values for k in each of the connected components to check whether the current branch in the search tree leads to a solution. If any branch leads to a solution, then we return yes and otherwise we return no. The correctness follows from the fact that we exhaustively search for all possible solutions and the running time is in $\mathcal{O}((d(\Delta+1)+1)^k \cdot 3^{d(\Delta+1)} \cdot (n+m))$. This concludes the proof. ◀

3 An FPT algorithm for d-Dominated Cluster Deletion

In this section, we show d-Dominated Cluster Deletion parameterized by k is fixed-parameter tractable. This also shows that Dominated Cluster Deletion is contained in XP.

▶ **Theorem 4.** *For every positive integer d, d-Dominated Cluster Deletion can be solved in $f(k,d) \cdot n^{\mathcal{O}(d)}$ time.*

We remark that the exponential dependency on d of the polynomial part of the running time seems unavoidable by Observation 2. We prove Theorem 4 by applying the *recursive understanding* technique introduced by Chitnis et al. [6] and using the meta theorem of Lokshtanov et al. [20]. The meta theorem considerably simplifies the arguments. However, the proof becomes nonconstructive and we can only claim a nonuniform FPT algorithm. Still, a uniform FPT algorithm may be given using the techniques of Chitnis et al. [6] or the dynamic programming approach of Cygan et al. [8].

The meta theorem of Lokshtanov et al. [20] allows to reduce solving a graph problem on general graphs to highly connected graphs. Let p, q be positive integers. A graph G is said to be (p,q)-*unbreakable* if for every separation (A,B) of G of order at most q, either $|A \setminus B| \leq p$ or $|B \setminus A| \leq p$, that is, G has no separator of size at most q that partitions the graph into two parts of size at least $p+1$ each.

We remind that in *monadic second-order logic* (MSO) on graphs, we have logical connectives $\vee$, $\wedge$, $\neg$, and variables for vertices, edges, and sets of vertices and edges. We can apply the quantifiers $\forall$ and $\exists$ to these variables and use the predicates (i) $x \in X$ where x is a vertex (respectively, edge) variable and X is a vertex (respectively, edge) set variable, (ii) $inc(u,e)$ where u is a vertex variable and e is an edge variable that denotes that u is incident to e, (iii) $adj(u,v)$ where u and v are vertex variables denoting the adjacency of u and v, and (iv) the equality predicate for variables of the same type. The *counting monadic second-order logic* (CMSO) extends MSO by allowing testing whether the cardinality of a set is equal to p modulo q for integers $0 \leq p < q$ with $q \geq 2$. Given a CMSO sentence φ, the task of the model-checking problem for φ (denoted by CMSO[φ]) is to decide whether $G \models \varphi$ for a graph G. Lokshtanov et al. [20] proved the following result.

▶ **Proposition 5** ([20])**.** *Let φ be a CMSO sentence. For all positive integers q, there exists a positive integer p such that if there exists an algorithm that solves CMSO[φ] on (p,q)-unbreakable graphs in $\mathcal{O}(n^c)$ time for some $c > 4$ then there exists an algorithm that solves CMSO[φ] on general graphs in $\mathcal{O}(n^c)$ time.*

To apply Proposition 5, we observe that Dominated Cluster Deletion can be expressed in MSO.

▶ **Observation 6.** *For given $d \geq 1$ and $k \geq 0$, there is a MSO sentence φ (depending on d and k) such that $G \models \varphi$ if and only if (G,k) is a yes-instance of Dominated Cluster Deletion.*

Proof. For $k = 0$, (G, k) is a yes-instance of Dominated Cluster Deletion if and only if every connected component C of G has a dominating set of size at most d, that is, it holds that $\exists v_1 \in V(C) \colon \ldots \exists v_d \in V(C) \colon \forall w \in V(C) \colon (w = v_1) \vee \cdots \vee (w = v_d) \vee adj(w, v_1) \vee \cdots \vee adj(w, v_d)$. Further, we have that $C = G[X]$ for $X \subseteq V(G)$ is a connected component of G if and only if $G[X]$ is connected and for any proper superset Y of X, $G[Y]$ is not connected. It is standard to express the connectivity of $G[X]$ as follows: $\forall Z \subset X \colon \exists u \in Z \colon \exists v \in X \setminus Z \colon adj(u, v)$. This proves that the property that $(G, 0)$ is a yes-instance of Dominated Cluster Deletion can be expressed in MSO.

For $k \geq 1$, (G, k) is a yes-instance of Dominated Cluster Deletion if and only if $(G, 0)$ is a yes-instance or $\exists v_1 \in V(G) \ldots \exists v_k \in V(G)$ such that every connected component C of $G - \{v_1, \ldots, v_k\}$ has a dominating set of size at most d. It is straightforward to modify the formula for $k = 0$ to get an MSO sentence for the case $k \geq 1$. ◀

We next show that d-Dominated Cluster Deletion is fixed-parameter tractable for unbreakable graphs. More precisely, we prove the following lemma.

▶ **Lemma 7.** *Let p and k be positive integers. Then, d-Dominated Cluster Deletion can be solved in $(p + k)^{\mathcal{O}(k+d)} \cdot n^{\mathcal{O}(d)}$ time on (p, k)-unbreakable graphs.*

Proof. Let (G, k) be an instance of d-Dominated Cluster Deletion where G is (p, k)-unbreakable.

Suppose that (G, k) is a yes-instance, that is, there is a set $S \subseteq V(G)$ of at most k vertices such that for each connected component C of $G - S$, $\gamma(C) \leq d$. Because G is a (p, k)-unbreakable graph, $G - S$ has a connected component C with at least $n - |S| - p \geq n - k - p$ vertices and the total number of vertices in the other connected components of $G - S$ is at most p. We call the component C *big*. Denote by $C_1, \ldots, C_\ell$ the connected components of $G - S$ that are distinct from C and let D be a dominating set of C of size at most d. Observe that $V(C) \subseteq N_G[D] \subseteq V(C) \cup S$ and $V(C_i) \subseteq V(G) \setminus N_G[D]$ for all $i \in \{1, \ldots, \ell\}$. Furthermore, if $S' = S \setminus N_G[D]$ then $S = S' \cup N_G(\bigcup_{i=1}^{\ell} V(G_i))$. These observations lead to the following algorithm.

We guess a dominating set D of the (potential) big connected component by trying all possible choices of at most d vertices of G. Let $R = V(G) \setminus N_G[D]$. We immediately discard the choice of D if $|R| > p + k$ because such sets D cannot be dominating sets of the big component. From now on, we assume that D is fixed and $|R| \leq p + k$.

Next, we guess the set of at most k vertices $S' = R \cap S$ for a potential solution S to (G, k) by trying all possible choices of such a set. For each choice of S', we consider $H = G[R \setminus S']$ and find the connected components $C_1, \ldots, C_\ell$ of H. For each $i \in \{1, \ldots, \ell\}$, we verify whether C_i has a dominating set of size at most d and immediately discard the current choice of S' if this does not hold. Further, we construct $S = S' \cup N_G(\bigcup_{i=1}^{\ell} V(C_i))$ and check whether $|S| \leq k$. If this holds then we conclude that S is a solution, return it, and stop. Otherwise, we discard the current choice of S'.

If we do not find a solution for all choices of S' for the given D, we discard the choice of D. If all sets D are discarded, then we conclude that (G, k) is a no-instance.

Our structural observations about yes-instances immediately imply the correctness of the algorithm. To evaluate the running time, observe that we have at most n^d choices of D that can be generated in $n^{\mathcal{O}(d)}$ time. For each choice of D with $|R| \leq p + k$, we have at most $(p + k)^k$ possibilities to choose S'. We can find connected components of H in $\mathcal{O}(p + k)$ time, and we can check in time $(p + k)^{\mathcal{O}(d)}$ whether each connected component can be dominated by at most d vertices. Thus, the total running time is $(p + k)^{\mathcal{O}(k+d)} \cdot n^{\mathcal{O}(d)}$. This concludes the proof. ◀

Combining Observation 6 and Lemma 7, we obtain Theorem 4 as a direct corollary of Proposition 5.

4 An FPT algorithm for 1-Dominated Cluster Deletion

In this section, we demonstrate a uniform algorithm for Dominated Cluster Deletion for the special case $d = 1$, that is, for 1-Dominated Cluster Deletion. The crucial property that makes the problem significantly easier for $d = 1$ is that each cluster is of diameter at most two. We use a simple branching algorithm to reduce the input graph to a connected graph of diameter at most two. Then we can exploit the following structural property of a solution.

▶ **Lemma 8.** *Let G be a graph of diameter at most two and let (A, B) be a separation of G with $X = A \cap B$. Suppose that for $S \subseteq V(G)$, each connected component of $G - S$ is dominated by a single vertex and it holds that (i) $(A \setminus B) \setminus S \neq \emptyset$, (ii) $(B \setminus A) \setminus S \neq \emptyset$, and (iii) $X \cap S = \emptyset$. Then for every connected component C of $G - S$, any vertex $v \in V(C)$ dominating C is in X. Furthermore, the vertices of each connected component of $G[X]$ are in the same connected component of $G - S$ and the vertices of distinct connected component of $G[X]$ are in separate components of $G - S$.*

Proof. Let C be a connected component of $G - S$. Assume for the sake of a contradiction that there is a vertex $v \in V(C)$ that dominates C such that $v \in A \setminus B$. By (ii), there is a vertex $u \in (B \setminus A) \setminus S$. Since the diameter of G is at most two, there is a (u, v)-path P of length at most two. Since u and v are separated by X, $P = uxv$ for some $x \in X$. Note that (iii) implies that $x \notin S$. Then, $u \in V(C)$. However, u is not dominated by v; a contradiction. This proves the first claim.

For the second claim, it is straightforward to see that the vertices of each connected component of $G[X]$ are in the same connected component of $G - S$ because of (iii). Let C_1 and C_2 be distinct connected components of $G[X]$. Notice that any vertex of G adjacent to some vertices in both connected components is either in $A \setminus B$ or $B \setminus A$. According to the first claim of the lemma, such vertices cannot be dominating vertices for the cluster. This proves that the vertices of distinct connected components of $G[X]$ are in separate components of $G - S$ and this concludes the proof. ◀

We are now in a position to prove the main theorem of this section.

▶ **Theorem 9.** *1-Dominated Cluster Deletion can be solved in $\mathcal{O}((k+3)^{k+5/2} \cdot nm)$ time.*

Proof. We use the method of bounded search trees to solve the problem. Our recursive branching algorithm, called Cluster (Algorithm 1), takes as input a graph G and an integer k, and returns either Yes or No depending on whether (G, k) is a yes- or a no-instance. For simplicity, we solve the decision problem, but the algorithm can easily be modified to output a solution if it exists.

We analyze the algorithm to show its correctness. Trivially, if $k < 0$ then (G, k) is a no-instance. Hence, we correctly return No and quit in Line 2. Also, if the set of vertices becomes empty then we should return Yes as is done in Line 3.

If G has a connected component C that can be dominated by one vertex, then this component is already a required cluster and we can exclude it from the consideration. This is done in Lines 4–5.

In Lines 7–14, we deal with disconnected graphs. In this case, we solve the problem for each connected component separately, and for each component C_i, we find the minimum k_i such that (C_i, k_i) is a yes-instance. Notice that because of the previous step, each connected component is not dominated by a single vertex. Thus, we consider only k_i satisfying $1 \leq k_i \leq k - 1$. Then, we verify whether the solutions for components can be combined. From now on, we assume that G is connected.

Algorithm 1 CLUSTER(G, k).

Input: A graph G and an integer k.
Result: Yes or No.

begin
  if $k < 0$ **then** return No and **quit**;
  if $V(G) = \emptyset$ **then** return No and **quit**;
  if *G has a connected component C with $\gamma(G) = 1$* **then**
    call CLUSTER$(G - V(C), k)$, return the result, and **quit**
  end
  if *G is disconnected* **then**
    find connected components $C_1, \ldots, C_\ell$ of G;
    foreach $i \in \{1, \ldots, \ell\}$ **do**
      find minimum $1 \leq k_i \leq k - 1$ s.t. CLUSTER(C_i, k_i) returns Yes and
      set $k_i := k$ if CLUSTER$(C_i, k - 1)$ returns No
    end
    return Yes if $k_1 + \cdots + k_\ell \leq k$ and return No, otherwise, and **quit**
  end
  if *there is a shortest path $v_1v_2v_3v_4$* **then**
    foreach $i \in \{1, 2, 3, 4\}$ **do**
      call CLUSTER$(G - v_i, k - 1)$;
      return Yes and **quit** if the algorithm returns Yes
    end
    return No and **quit**
  end
  if *there is $v \in V(G)$ s.t. $|V(G) \setminus N_G[v]| \leq k$* **then**
    return Yes and **quit**
  end
  find a separation (A, B) of G with the separator $X = A \cap B$ of size at most k
  and return No and **quit** if such a separation does not exist;
  foreach $v \in X$ **do**
    call CLUSTER$(G - v, k - 1)$;
    return Yes and **quit** if the algorithm returns Yes
  end
  call CLUSTER$(G - (A \setminus B)|, k - |A \setminus B|)$ and
  return Yes and **quit** if the algorithm returns Yes;
  call CLUSTER$(G - (B \setminus A)|, k - |B \setminus A|)$ and
  return Yes and **quit** if the algorithm returns Yes;
  if *there is $v \in V(G) \setminus X$ s.t. v has neighbors in distinct components of $G[X]$* **then**
    call CLUSTER$(G - v, k - 1)$;
    return Yes and **quit** if the algorithm returns Yes
  end
  return No
end

In Lines 15–21, we analyze the case when G has a shortest path on four vertices. If $v_1v_2v_3v_4$ is a shortest (v_1, v_4)-path then the vertices v_1, v_2, v_3, v_4 cannot belong to the same cluster because they cannot be dominated by the same vertex. This implies that at least one vertex of such a path should be included in any solution. We branch on four possible options and return Yes if we have a yes-instance for one of the branches. Otherwise, we return No and stop. From this moment, we assume that G has no shortest paths of length three, that is, the diameter of G is at most two.

Further, in Lines 22–24, we check whether the instance admits a trivial solution with one cluster. This holds if there is $v \in V(G)$ such that $|V(G) \setminus N_G[v]| \leq k$ – we simply delete all the vertices that are not dominated by v. Now, we can assume that for each solution S, $G - S$ has at least two components.

In Lines 25–26, we find a separation (A, B) of G with the separator $X = A \cap B$ of size at most k (we remind that we only consider separations (A, B) where both $A \setminus B$ and $B \setminus A$ are nonempty). If such a separation does not exist, then for each $S \subseteq V(G)$ of size at most k, $G - X$ is connected. Hence, (G, k) may have only a solution with one cluster. However, this case was already excluded. Thus, we correctly return No and stop in Line 26. From now on, we assume that a separation (A, B) of order at most k exists.

In Lines 27–30, we branch on the possibility that X contains a vertex of a solution. We return Yes and stop if we find such a solution. Next, in Lines 31–34, we check whether there is a solution that includes either every vertex of $A \setminus B$ or every vertex of $B \setminus A$.

Finally, we conclude that any solution to (G, k) excludes the vertices of X and, furthermore, each solution contains at least one vertex of $A \setminus B$ and at least one vertex of $B \setminus A$. Then by Lemma 8, for any solution S to (G, S), it holds that for each connected component C of $G - S$, any vertex $v \in V(C)$ dominating C is in X and, furthermore, the vertices of each connected component of $G[X]$ are in the same connected component of $G - S$ and the vertices of distinct connected component of $G[X]$ are in separate components of $G - S$. Recall that we excluded the case when (G, k) has a solution with one cluster. Hence, if (G, k) is a yes-instance then $G[X]$ should have at least two connected components. Let C_1 and C_2 be two distinct connected components of $G[X]$. Because the diameter of G is at most two, we have that for every $u \in V(C_1)$ and every $v \in V(C_2)$, there is a vertex $x \in V(G) \setminus X$ such that uxv is a path in G. Because u and v are in distinct connected components of $G - S$, it holds that $x \in S$. We conclude that if (G, k) is a yes-instance then there is at least one vertex $v \in V(G) \setminus X$ which has neighbors in two distinct connected components of $G[X]$ and all such vertices are included in any solution S. We use this in Lines 35-39. We either find v and obtain that that $(G - v, k - 1)$ is a yes-instance or conclude that (G, k) is a no-instance. This concludes the correctness proof.

To evaluate the running time, notice that in Lines 15–21, we have four recursive calls. If we go to Lines 27-38, then we have at most $|X| + 3 \leq k + 3$ branches. For each branch, we reduce the parameter by at least one. Thus, the total number of leaves of the search tree is at most $(k+3)^k$. We can find connected components in $\mathcal{O}(n+m)$ time using standard techniques and we can check in $\mathcal{O}(n)$ time whether a connected graph has a dominating vertex. A shortest path on four vertices can be found in $\mathcal{O}(nm)$ time by performing a breadth-first search from each vertex. The existence of a vertex v such that $|V(G) \setminus N_G[v]| \leq k$ can be checked in $\mathcal{O}(m)$ time. To verify whether G has a separation of order k, we can use the algorithm of Gabow [13] with running time $\mathcal{O}((n + \min(k^{5/2}, kn^{3/4}))m)$. In fact, for undirected graphs, m can be replaced by kn, but we use the bound in the weaker form $\mathcal{O}(k^{5/2}nm)$. Finally, note that in Lines 9–11, for each $i \in \{1, \ldots, \ell\}$, we compute the minimum k_i such that (C_i, k_i) is a yes-instance. For this, we can use binary search, which adds an additional $\log k$-factor to

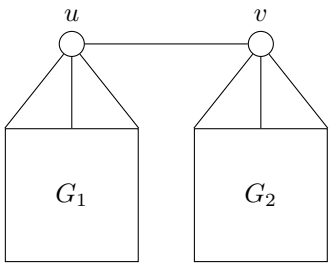

Figure 2 An illustration of the construction in the proof of Theorem 10.

the running time. However, $k_i \leq k-1$ and this means that the steps in Lines 9–11 can be executed in time $\mathcal{O}((k+3)^{k+5/2} \cdot nm)$. Summarizing, we obtain that the total running time is in $\mathcal{O}((k+3)^{k+5/2} \cdot nm)$. This concludes the proof. ◀

5 (No) Polynomial Kernels

In this section, we show some results related to the existence of polynomial kernels. Our main result here is that Dominated Cluster Deletion does not admit a polynomial kernel with respect to $k+c$ even if $d=1$. We mention that the construction is identical to a reduction by Figiel et al. [12] that shows that 2-Club Cluster Vertex Deletion does not have a polynomial kernel with respect to the solution size k. The two problems are loosely related as a graph with a universal vertex (a dominating set of size one) is a 2-Club. However, the other direction is not true, that is, not every 2-club has a dominating set of size one. The two results therefore do not directly follow from one another.

▶ **Theorem 10.** *1-Dominated Cluster Deletion does not admit a polynomial kernel with respect to $k+c$ unless* NP $\subseteq$ coNP / poly *even if the input graph is connected.*

Proof. We show this result by providing a OR-cross-composition (see [4]) for 1-Dominated Cluster Deletion from itself. Recall that 1-Dominated Cluster Deletion is NP-complete by Proposition 1. Without loss of generality, we start with $T = 2^t$ instances of 1-Dominated Cluster Deletion, where each instance has the same number n of vertices, the same value of $k < n-1$, and where each of the input graphs is connected.

The main ingredient for our proof is a construction to merge two instances $I_1 = (G_1, k)$ and $I_2 = (G_2, k)$ into one instance $I^* = (G^*, k^*)$. We first prove that I^* is a yes-instance if and only if at least one of I_1 and I_2 is a yes-instance. Afterwards, we will show how to use this construction to get a cross-composition for all t instances.

The construction is depicted in Figure 2 and works as follows. We take the disjoint union of the two graphs G_1 and G_2 and add two vertices u and v. We make u adjacent to all vertices in G_1 and v adjacent to all vertices in G_2. Moreover, we add the edge $\{u, v\}$. Finally, we set the parameter $k^* = k+1$.

It remains to show that I^* is a yes-instance if and only if at least one of I_1 and I_2 is a yes-instance. First assume, that one of I_1 and I_2 is a yes-instance. Since both cases are completely analogous, we assume without loss of generality that I_1 is a yes instance. A solution for I^* consists of an optimal solution for I_1 and the vertex u. Note that adding u to the solution disconnects the graphs G_1 and G_2. Moreover, the connected component including G_2 also contains v and therefore has a domination number of one.

For the other direction, assume that the constructed instance contains a solution S of size at most $k+1$. Note that since we assume $k < n-1$, we need to include either u or v in S as otherwise we would need to include all $n > k^*$ vertices in either G_1 or G_2 into S as one

vertex of each instance together with u and v induces a P_4 and no vertex is adjacent to both a vertex in G_1 and a vertex in G_2. We assume without loss of generality that u is included in S. Note that this leaves the graph G_1 as one connected component and since we already know that $u \in S$, it holds that S contains at most k vertices from G_1 which results in each connected component of G_1 being dominated by a single vertex. Thus I_1 is a yes-instance.

To complete the reduction, we iteratively half the number of instances by partitioning all instances into arbitrary pairs and merge the two instances in a pair into one instance. Note that in each iteration, all of our assumption still hold as all instances have $d = 1$, the number of vertices in I^* is $2n + 2 > k^* + 1$, and G^* is connected. After $\log(T)$ iterations, we are left with a single instance which is a yes-instance if and only if at least one of the T original instances is a yes-instance. The time required to compute the cross composition is in $O(\sum_{i\in[T]} |G_i| \cdot \log(T))$ which is polynomial in $\sum_{i\in[T]} |G_i|$ as each graph G_i contains at least one vertex. Moreover, the parameter k in the constructed instance is $k + \log(T)$ and the degeneracy of the instance is at most $n + \log(T)$ as we can put all the vertices in the T original instances first into the degeneracy ordering and afterwards add the newly added vertices in increasing order of their degree. Note that each vertex in an original instance has at most $n + \log(T)$ neighbors and each of the newly added vertices has at most $\log(T)$ neighbors "above themself". Thus, all requirements of a cross-composition are met and the proof is completed. ◀

We mention in passing that it is simple to generalize Theorem 10 to any constant $d > 0$. In each step, we simply add the following graph to each of the two newly added vertices u and v. We show the construction only for v. We start with a path $(w_1, w_2, \ldots, w_{3d-2})$ with $3d - 2$ vertices and make w_1 adjacent to v. Next, we iteratively replace each vertex w_i by a clique of size $k + \log(T) + 1$ vertices, where each vertex has the same neighborhood as w_i had. Note that we cannot fully remove any of the cliques and hence whenever we remove neither u nor v, then we are left with a connected graph which requires at least $2d > d$ vertices to dominate. Whenever we remove one of the two vertices, say u, then we can dominate the connected component of v by selecting v and one copy of each vertex in the clique corresponding to the vertices $w_3, w_6, \ldots, w_{3(d-1)}$. The rest of the proof is completely analogous to the proof of Theorem 10.

We next show that d-Dominated Cluster Deletion parameterized by $k + \Delta$ admits a polynomial kernel.

▶ **Proposition 11.** *d-Dominated Cluster Deletion parameterized by $k + \Delta$ admits a polynomial kernel of size $\mathcal{O}(k^2\Delta^3)$.*

Proof. First, we check for each connected component of the input graph whether it contains a dominating set of size d in $\mathcal{O}(n^{d+2})$ time. Note that since we assume d to be a constant, this is a polynomial running time. Moreover, each such connected component can be removed from the input graph to create a smaller equivalent instance. If afterwards, the number of connected components in the graph is larger than k, then we return a trivial no-instance as we need to include at least one vertex from each connected component into the solution S, meaning that any solution has size at least $k+1$. Finally, we bound the size of each connected component as follows. If the number of vertices in one connected component is larger than $k + (k(\Delta - 1) + 1)d(\Delta + 1) \in \mathcal{O}(k\Delta^2)$, then we again return a trivial no-instance as removing any solution S of size at most k leaves at most $k(\Delta - 1) + 1$ connected components and hence by the pigeonhole principle, at least one connected component has size larger than $d(\Delta + 1)$. Note that this connected component cannot contain a dominating set of size at most d and hence the entire instance is a no-instance.

The number remaining vertices is now at most $k \cdot (k + (k(\Delta - 1) + 1)d(\Delta + 1)) \in \mathcal{O}(k^2\Delta^2)$ and since the maximum degree is Δ, the number of vertices and edges is in $\mathcal{O}(k^2\Delta^3)$. This concludes the proof. ◀

Next, we show that the previous result cannot be generalized to Dominated Cluster Deletion even if we include the parameter d to the kernel and restrict k to be 0.

▶ **Observation 12.** *Dominated Cluster Deletion does not admit a polynomial kernel with respect to $d + \Delta$ even if $k = 0$ unless* NP $\subseteq$ coNP / poly.

Proof. We show a simple AND-composition from Dominating Set on connected input graphs. Given t instances of Dominating Set where all instances are connected and all instances ask for a dominating set of the same size d, we simply take the disjoint union of all of these graphs and set $k = 0$. If all t instances are yes-instances, then each connected component of the constructed graph has a dominating set of size at most d, that is, the constructed instance is a yes-instance. If the constructed instance is a yes-instance, then also each connected component of the constructed graph has a dominating set of size at most d as $k = 0$. Thus, all t original instances are yes-instances. Note that $k = 0$, the value of d is the same as in each input instance, and the maximum degree is at most the number of vertices in one of the t input instances. Thus, we have provided an AND-cross-composition and this concludes the proof. ◀

Note that the previous reduction produces a disconnected graph. We show that this is necessary as Dominated Cluster Deletion admits a polynomial kernel when parameterized by $k + d + \Delta$ if the input graph is connected.

▶ **Observation 13.** *Dominated Cluster Deletion admits a kernel of size $\mathcal{O}(kd\Delta^3)$ when parameterized by $k + d + \Delta$ if the input graph is connected.*

Proof. If the number of vertices in the input graph is at most $kd\Delta^2 + k + d(\Delta + 1) - kd$, then we return the input graph as a trivial kernel of size $\mathcal{O}(kd\Delta^3)$. Otherwise we return a trivial no-instance as a kernel. We next show that this is justified. Assume towards a contradiction that the input instance has more vertices but is a yes-instance. Then, it contains a set S of at most k vertices such that removing S from the graph results in connected components which can each be dominated by at most d vertices. Note that removing a set of k vertices of degree at most Δ from a connected graph can result in a graph with at most $k(\Delta - 1) + 1$ connected components. Moreover, the dominating set of size d in each of these can dominate at most $d(\Delta) + 1$ vertices. Thus, the entire graph contains at most $k + (k(\Delta - 1) + 1)d(\Delta + 1) = kd\Delta^2 + k + d(\Delta + 1) - kd$ vertices, a contradiction. ◀

6 Conclusion

In this work, we initiated the study of Dominated Cluster Deletion and showed an almost complete tetrachotomy regarding its parameterized complexity when parameterized by combinations of k, d, c, and Δ. We remark that we leave two questions unanswered. First, is Dominated Cluster Deletion fixed-parameter tractable or W[1]-hard when parameterized by $k + d + c$? We conjecture that it is fixed-parameter tractable and the algorithm may be obtained by combining the recursive understanding technique [6, 20] and the results of Telle and Villanger [21]. Second, does d-Dominated Cluster Deletion parameterized by k admit a polynomial kernel when d and c are constants?

References

1 Faisal N. Abu-Khzam, Rebecca L. Collins, Michael R. Fellows, Michael A. Langston, W. Henry Suters, and Christopher T. Symons. Kernelization algorithms for the vertex cover problem: Theory and experiments. In *Proceedings of the 6th Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 62–69. SIAM, 2004.

2 Deepak Ajwani, Rob H. Bisseling, Katrin Casel, Ümit V. Çatalyürek, Cédric Chevalier, Florian Chudigiewitsch, Marcelo Fonseca Faraj, Michael R. Fellows, Lars Gottesbüren, Tobias Heuer, George Karypis, Kamer Kaya, Jakub Lacki, Johannes Langguth, Xiaoye Sherry Li, Ruben Mayer, Johannes Meintrup, Yosuke Mizutani, François Pellegrini, Fabrizio Petrini, Frances A. Rosamond, Ilya Safro, Sebastian Schlag, Christian Schulz, Roohani Sharma, Darren Strash, Blair D. Sullivan, Bora Uçar, and Albert-Jan Yzelman. Open problems in (hyper)graph decomposition. *CoRR*, abs/2310.11812, 2023. `arXiv:2310.11812`.

3 Stéphane Bessy, Marin Bougeret, Dimitrios M. Thilikos, and Sebastian Wiederrecht. Kernelization for graph packing problems via rainbow matching. In *Proceedings of the 34th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 3654–3663. SIAM, 2023.

4 Hans L. Bodlaender, Bart M. P. Jansen, and Stefan Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014.

5 Valentin Bouquet, François Delbot, Christophe Picouleau, and Stéphane Rovedakis. On minimum dominating sets in cubic and (claw, H)-free graphs. *CoRR*, abs/2002.12232, 2020. `arXiv:2002.12232`.

6 Rajesh Chitnis, Marek Cygan, MohammadTaghi Hajiaghayi, Marcin Pilipczuk, and Michal Pilipczuk. Designing FPT algorithms for cut problems using randomized contractions. *SIAM Journal on Computing*, 45(4):1171–1229, 2016.

7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

8 Marek Cygan, Daniel Lokshtanov, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. Minimum bisection is fixed-parameter tractable. *SIAM Journal on Computing*, 48(2):417–450, 2019.

9 Reinhard Diestel. *Graph Theory*. Springer, 2012.

10 Rodney G. Downey and Michael R. Fellows. Fixed-parameter tractability and completeness I: Basic results. *SIAM Journal on Computing*, 24(4):873–921, 1995.

11 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

12 Aleksander Figiel, Anne-Sophie Himmel, André Nichterlein, and Rolf Niedermeier. On 2-clubs in graph-based data clustering: Theory and algorithm engineering. *Journal of Graph Algorithms and Applications*, 25(1):521–547, 2021.

13 Harold N. Gabow. Using expander graphs to find vertex connectivity. *Journal of the ACM*, 53(5):800–844, 2006.

14 M. R. Garey, David S. Johnson, and Larry J. Stockmeyer. Some simplified NP-complete graph problems. *Theoretical Computer Science*, 1(3):237–267, 1976.

15 Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010.

16 David 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)*, pages 40:1–40:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024.

17 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001.

18 John M. Lewis and Mihalis Yannakakis. The node-deletion problem for hereditary properties is NP-complete. *Journal of Computer and System Sciences*, 20(2):219–230, 1980.

19 Hong Liu, Peng Zhang, and Daming Zhu. On editing graphs into 2-club clusters. In *Proceedings of the 2nd Joint International Conference on Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM)*, pages 235–246. Springer, 2012.

20 Daniel Lokshtanov, M. S. Ramanujan, Saket Saurabh, and Meirav Zehavi. Reducing CMSO model checking to highly connected graphs. In *Proceedings of the 45th International Colloquium on Automata, Languages, and Programming (ICALP)*, pages 135:1–135:14. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018.

21 Jan Arne Telle and Yngve Villanger. FPT algorithms for domination in sparse graphs and beyond. *Theoretical Computer Science*, 770:62–68, 2019.

22 Dekel Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory of Computing Systems*, 65(2):323–343, 2021.

23 René van Bevern, Hannes Moser, and Rolf Niedermeier. Approximation and tidying – A problem kernel for s-plex cluster vertex deletion. *Algorithmica*, 62(3-4):930–950, 2012.