Fixed-Parameter Algorithms for Cluster Vertex Deletion*

Falk Hüffner† Christian Komusiewicz‡ Hannes Moser§
Rolf Niedermeier

Institut für Informatik, Friedrich-Schiller-Universität Jena,
Ernst-Abbe-Platz 2, D-07743 Jena, Germany
{hueffner,ckomus,moser,niedermr}@minet.uni-jena.de

Abstract

We initiate the first systematic study of the NP-hard Cluster Vertex Deletion (CVD) problem (unweighted and weighted) in terms of fixed-parameter algorithmics. In the unweighted case, one searches for a minimum number of vertex deletions to transform a graph into a collection of disjoint cliques. The parameter is the number of vertex deletions. We present efficient fixed-parameter algorithms for CVD applying the fairly new iterative compression technique. Moreover, we study the variant of CVD where the maximum number of cliques to be generated is prespecified. Here, we exploit connections to fixed-parameter algorithms for (weighted) Vertex Cover.

1 Introduction

Graph modification problems form a core topic in algorithmic graph theory with many applications. In particular, cluster graph modification problems [36] have recently received considerable interest. Here, the basic problem is, given an undirected graph G, to find a minimum number of editing operations that transform G into a collection of disjoint complete subgraphs, a *cluster graph*. Each of these disjoint complete subgraphs is called a *cluster*. Herein, the three

*An extended abstract of this paper appeared in Proceedings of the 8th Latin American Theoretical Informatics Symposium (LATIN 2008), April 7–11, 2008, Armação dos Búzios, Brazil, volume 4957 in Lecture Notes in Computer Science, pages 711–722, Springer, 2008.

†Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4, and the Edmond J. Safra foundation.

‡Supported by a PhD fellowship of the Carl-Zeiss-Stiftung.

§Supported by the Deutsche Forschungsgemeinschaft, project ITKO (iterative compression for solving hard network problems), NI 369/5, and project AREG (algorithms for generating quasi-regular structures in graphs), NI 369/9.

standard editing operations are adding edges, deleting edges, and deleting vertices. For instance, Cluster Editing asks whether a graph can be transformed into a cluster graph by altogether at most k edge additions and edge deletions. Cluster Editing is NP-complete; it recently has shown particularly useful for clustering biological data [10, 33]. Whereas also a factor-2.5 polynomial-time approximation for Cluster Editing is known [3, 4, 38], in practical applications fixed-parameter algorithms (combined with some heuristics) providing optimal solutions seem to dominate [5, 6, 10, 33]. For a background on fixed-parameter algorithmics we refer to [12, 15, 29]. Parameterized complexity studies for Cluster Editing were initiated by Gramm et al. [21] and have been further pursued in a series of papers [5, 6, 10, 13, 20, 22, 32, 33]. A previously shown bound of $O(1.92^k + n^3)$ for an n-vertex graph [20] can be improved by combining a linear-time problem kernelization algorithm [13] that yields an instance with $O(k^2)$ vertices with the currently best claimed running time of $O(1.82^k+n^3)$ [6] to get an algorithm with running time $O(1.82^k+n+m)$, where m is the number of edges in the graph. Moreover, problem kernels, based on efficient data reduction rules, with only $O(k)$ vertices are known [13, 22], the best upper bound currently being $4k$ [22].

Whereas Cluster Editing has been subject to intensive research, its "sister problem" Cluster Vertex Deletion so far has been widely neglected. Here, we aim at finding a vertex set of minimum weight such that its deletion transforms a given graph into a cluster graph.[1]

> Weighted Cluster Vertex Deletion
> **Instance:** An undirected graph $G = (V, E)$, a vertex weight function $\omega : V \to [1, \infty)$, and a nonnegative number k.
> **Question:** Is there a vertex set $X \subseteq V$ with $\sum_{v \in X} \omega(v) \leq k$ such that deleting all vertices in X from G results in a cluster graph (i. e., a graph where every connected component forms a complete graph)?

The unweighted version asks whether there exists a subset $X \subseteq V$ such that $|X| \leq k$ (in other words, all vertices have weight exactly one). We call a set of vertices whose deletion produces a cluster graph a *CVD set*.

Motivation. Like Cluster Editing, Cluster Vertex Deletion may find applications in graph-modeled data clustering: Assume that we have a number of samples, some of which are equivalent (e. g., DNA samples, some of which are from the same species) and a method to test two samples for equivalence. A graph is formed where each vertex corresponds to a sample and an edge between two vertices is added when their samples are tested as equivalent. In the absence of errors, the resulting graph is a cluster graph, where each connected component corresponds to an equivalence class (e. g., a species). However, an unknown subset of samples may be contaminated and can produce unpredictable comparisons to other samples. An optimal solution for unweighted Cluster Vertex

[1] Parameterized problems (as follows) usually are formulated as decision problems—all our algorithms will also solve the corresponding optimization problem within the same time bounds.

Deletion, that is, a minimum-cardinality set of vertices whose deletion produces a cluster graph, then provides the most parsimonious explanation for the data under this model. This clearly extends to the weighted case.

Cluster Vertex Deletion can also be seen as the problem of making a symmetric relation transitive by omitting a minimum number of elements. The related problem of making an antisymmetric relation transitive by omitting a minimum number of elements is also known as Feedback Vertex Set in tournaments (see, e. g., [11, 34] for fixed-parameter tractability results).

A further application for Cluster Vertex Deletion is the computation of a maximum clique of a graph with the help of a CVD set. Suppose a CVD set X has been computed. Clearly, a maximum clique is at least as large as the largest cluster of the remaining cluster graph $G[V \setminus X]$. If there is a clique that is even larger, then it must contain some vertices of X. The vertex set $S \subseteq X$ that is contained in the maximum clique must induce a complete graph. For each such vertex set S, we can find the maximum clique that contains S and vertices of G by finding the cluster in G that contains the most vertices that are neighbors of S. Therefore, we can find a maximum clique of a graph by enumerating all subsets S of X, checking whether the respective subset induces a complete graph, and then finding the cluster in $G[V \setminus X]$ that contains the most vertices that are neighbors of all vertices in S. A clique that has maximum size of all cliques thus found is a maximum clique of G. A similar approach can be used to find the maximum independent set of G. Therefore, the problem of finding a maximum clique or a maximum independent set of a graph G can be parameterized by the size of the CVD set of G, or by the size of the CVD set of the complement graph $\bar{G}$, respectively.

Finally, in comparison to Cluster Editing, a small parameter value k (that is, the number of editing operations) appears even more likely for Cluster Vertex Deletion, since the vertex deletion operation is more powerful, making a parameterized approach particularly meaningful here.

Known results. By general results for vertex deletion problems for hereditary graph properties [27], it follows that already unweighted Cluster Vertex Deletion is NP-complete. The optimization version is MaxSNP-hard [28]. Only few specific results for (unweighted) Cluster Vertex Deletion are known.[2] These are based on the simple observation that a graph is a cluster graph if and only if it does not contain an induced P_3, a path on three vertices.[3] Using this characterization, one directly obtains fixed-parameter tractability [7] as well as a factor-3 polynomial-time approximation algorithm. Gramm et al. [20] used an elaborate case distinction found with computer help to derive a

[2] Jansen et al. [26] studied the closely related problem of finding d pairwise disjoint cliques with maximum overall number of vertices, motivated by applications in scheduling. Note that, other than in Cluster Vertex Deletion, they allowed to have edges between cliques. Jansen et al. gave polynomial-time algorithms for special graph classes, contrasting the NP-complete general case.

[3] In the remainder of this work, when writing of containment of a P_3 in a graph we refer to an induced P_3.

search tree algorithm running in $O(2.26^k m)$ time for an m-edge graph. This can be improved to $O(2.08^k + n^3)$, n denoting the number of vertices, by using a straightforward reduction of unweighted Cluster Vertex Deletion to the 3-Hitting Set problem (transforming each induced P_3 into a three-element set) and employing a sophisticated algorithm for 3-Hitting Set [37]. Moreover, kernelization results for 3-Hitting Set [1] also imply an $O(k^2)$-vertex problem kernel for unweighted Cluster Vertex Deletion, which can be found in $O(n^3)$ time. A weighted Cluster Vertex Deletion instance can be easily transformed into a weighted 3-Hitting Set instance. With this transformation, an $O(k^3)$-vertex problem kernel result for weighted 3-Hitting Set [2] can be adapted to weighted Cluster Vertex Deletion. Moreover, weighted 3-Hitting Set possesses an elaborate search tree algorithm based on case distinction [14], implying an $O(2.25^k + n^3)$ running time for weighted Cluster Vertex Deletion.

Iterative compression, as first described by Reed et al. [35], is a technique for the development of fixed-parameter algorithms. For an introduction and a survey on state of the art results that employ iterative compression we refer to Guo et al. [23]. Hüffner [24] gives experimental evaluations of some iterative compression algorithms, demonstrating their usefulness on real-world and synthetic data.

New results. One of our main results is an elegant iterative compression algorithm for weighted Cluster Vertex Deletion using matching techniques, running in $O(2^k k^9 + nm)$ time.[4] We extend our studies to the (also NP-hard) case where the number of clusters to be generated is limited by a second parameter d. Such studies have also been undertaken for Cluster Editing [19, 22, 36], but note that for Cluster Editing clearly $d \leq 2k$. By way of contrast, since vertex deletion is a stronger operation than edge deletion, in the case of Cluster Vertex Deletion also $d > 2k$ is possible. Observe that $d = 1$ yields the Clique problem; therefore, a parameterization only with respect to the parameter d is meaningless. Considering the combined parameter (d, k), however, we can provide further fixed-parameter tractability results. First, we nontrivially extend the kernelization result for weighted Cluster Vertex Deletion to a problem kernel for weighted d-Cluster Vertex Deletion, again achieving an $O(k^3)$-vertex problem kernel. Based on this, we develop three fixed-parameter algorithms for weighted d-Cluster Vertex Deletion with the following running times: $O(2^k \cdot k^9 + nm)$, $O(1.40^k \cdot k^{3d} + nm)$, and $O(1.84^{k+d} + nm)$. Depending on the value of d, each of these algorithms may be preferable in certain constellations. In the latter two algorithms, fixed-parameter algorithms for weighted Vertex Cover play a decisive role.

Notation. In this paper, for a graph $G = (V, E)$ and a vertex set $S \subseteq V$, let $G[S]$ be the subgraph of G induced by S and $G \setminus S := G[V \setminus S]$, and

[4] A similar algorithm for Cluster Vertex Deletion later has also been described by Fomin et al. [16].

```
CompressCVD(G, X)
 1  X' ← X
 2  for each ∅ ≠ S ⊆ X:
 3      if G[S] is a cluster graph:
 4          G' ← G \ (X \ S);  R ← V(G' \ S)
 5          G' ←ReductionRule1(G', S)
 6          G' ←ReductionRule2(G', S)
 7          G' ←ReductionRule3(G', S)
 8          Classify each vertex u in R according to N(u) ∩ S
 9          H ← auxiliary graph
10          M ← maximum weight matching in H
11          Delete all vertices not in a class corresponding to an edge in M
12          D ← vertices deleted in lines 4–7 and 11
13          if ω(D) < ω(X'):
14              X' ← D
15  return X'
```

Figure 1: Pseudo-code for CompressCVD, where $\omega(A) := \sum_{v \in A} \omega(v)$ for $A \subseteq V$.

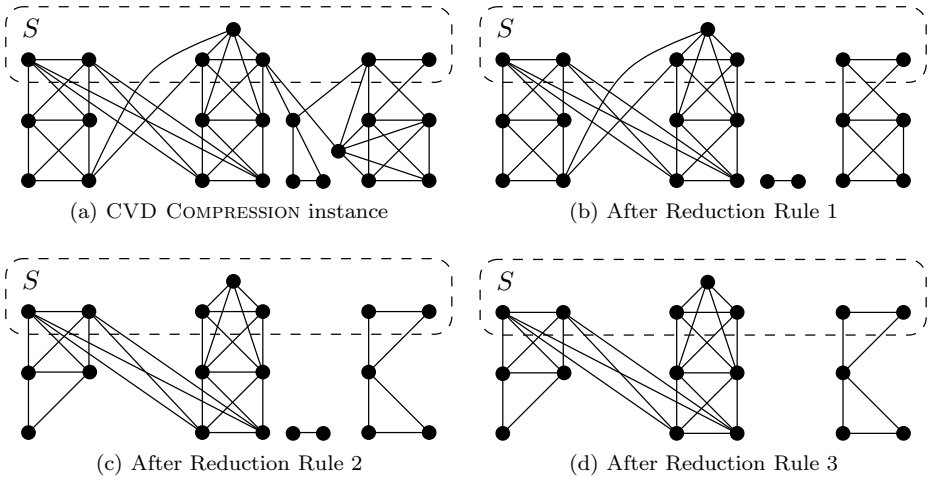
let $N(v) := \{u \in V \mid \{u, v\} \in E\}$. We write $V(G)$ to denote the vertex set of G.

2 Iterative compression for Cluster Vertex Deletion

We now describe a novel iterative compression algorithm for weighted Cluster Vertex Deletion. General considerations about iterative compression algorithms can be found in [23, 25] and [29, Chapter 11]. We first describe how to employ a compression routine, and then the compression routine itself.

The general idea behind our iterative compression is as follows. We start with $V' = \emptyset$ and $X = \emptyset$; clearly, X is a CVD set for $G[V']$. Iterating over all graph vertices, step by step we add one vertex $v \notin V'$ from V to both V' and X. Then X is still a CVD set for $G[V']$, although possibly not a minimum one. We can, however, obtain a minimum one by applying the compression routine CompressCVD. It takes a graph G and a CVD set X for G, and returns a minimum CVD set for G. Therefore, it is a loop invariant that X is a minimum-size CVD set for $G[V']$. Since eventually $V' = V$, we obtain an optimal solution for G once the algorithm returns X.

In the rest of this section, we describe the compression routine CompressCVD following the pseudo-code in Figure 1. For this, consider a smaller CVD set X' as a modification of the larger CVD set X. This modification retains some vertices $Y \subsetneq X$, while the other vertices $S := X \setminus Y$ are replaced by at most $|S|-1$ new vertices from $V \setminus X$. The idea is to try by brute force all $2^{|X|}-1$



(a) CVD Compression instance (b) After Reduction Rule 1

(c) After Reduction Rule 2 (d) After Reduction Rule 3

Figure 2: Data reduction in the compression routine.

partitions of X into such sets Y and S (line 2). For each such partition, the vertices from Y are immediately deleted, since we already decided to take them into the CVD set. In the resulting instance $G' = (V', E') := G \setminus Y$, it remains to find an optimal CVD set that is disjoint from S. This is a much easier task than finding a CVD set in general; in fact, it can be done in polynomial time using data reduction and maximum matching.

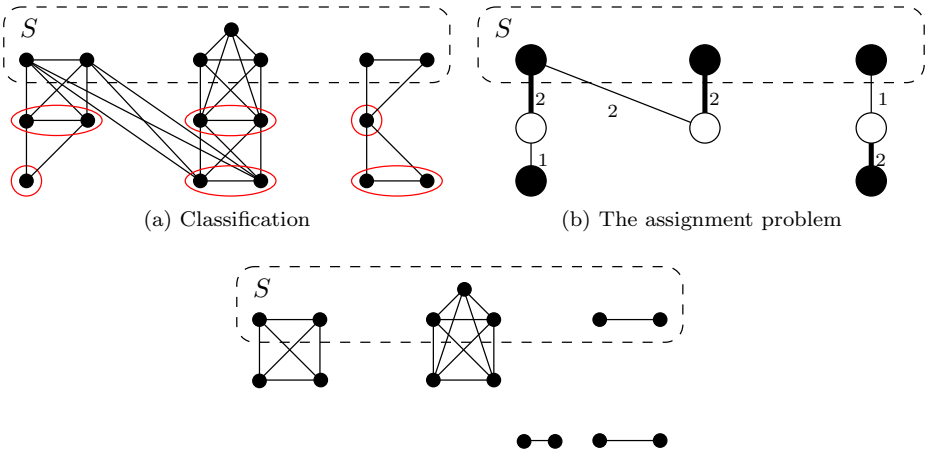
First, we discard partitions where S does not induce a cluster graph (line 3); these cannot lead to a solution, since we determined that none of the vertices in S would be deleted. Further, $R := V' \setminus S$ also induces a cluster graph, since $R = V \setminus X$ and X is a CVD set. Therefore, the following problem remains:

CVD Compression
Instance: An undirected graph $G = (V, E)$, a vertex weight function $\omega : V \to [1, \infty)$, and a vertex set $S \subseteq V$ such that $G[S]$ and $G \setminus S$ are cluster graphs.
Task: Find a vertex set $X' \subseteq V \setminus S$ such that $G \setminus X'$ is a cluster graph and $\sum_{v \in X'} \omega(x)$ is minimum.

An example instance is shown in Figure 2a. The instance can now be simplified by a series of data reduction rules. The results are shown in Figures 2b–d.

Reduction Rule 1. *Delete all vertices in $R := V \setminus S$ that are adjacent to more than one cluster in $G[S]$.*

Proof of correctness. If a vertex $v \in R$ is adjacent to vertices u and w in different clusters in S, then uvw induces a P_3 that can only be removed by deleting v (because the vertices in S cannot be deleted). □



(a) Classification (b) The assignment problem

(c) Final resulting cluster graph

Figure 3: Assignment problem in the iterative compression, unweighted case.

Reduction Rule 2. *Delete all vertices in R that are adjacent to some, but not all vertices of a cluster in $G[S]$.*

Proof of correctness. If a vertex $v \in R$ is adjacent to a vertex u, but not to a vertex w in a cluster in S, then uwv induces a P_3, which can only be removed by deleting v. □

Reduction Rule 3. *Remove connected components that are complete graphs.*

Proof of correctness. No optimal solution deletes vertices in such components. □

After Reduction Rules 1–3 have been applied, the instance is much simplified (Figure 2d). In each cluster in $G[R]$, we can divide the vertices into equivalence classes according to their neighborhood in S; each class then contains either vertices adjacent to all vertices of a particular cluster in $G[S]$, or the vertices adjacent to no vertex in S (see Figure 3a). This classification is useful because of the following lemma.

Lemma 1. *In an optimal* CVD COMPRESSION *solution, for each cluster in $G[R]$, the vertices of at most one class are present.*

Proof. Clearly, it is never useful to delete only some, but not all vertices of a class, since if that led to an optimal solution, we could always re-add the deleted vertices without introducing new P_3's. Further, if $v \in R$ is adjacent to some $w \in S$, and u is a vertex from the same cluster as v, but from a different class, then uvw is a P_3; therefore, we cannot keep vertices from two different classes within a cluster. □

Because of Lemma 1, the remaining task is an assignment of each cluster in $G[R]$ to one of its classes (corresponding to the preservation of this class, and the deletion of all other classes within the cluster) or to nothing (corresponding to the complete deletion of the cluster). However, we cannot do this independently for each cluster; we must not choose two classes from different clusters in $G[R]$ which are adjacent to the same cluster in $G[S]$, since that would create a P_3. This can be modelled as a weighted bipartite matching problem in an auxiliary graph H, where each edge corresponds to a possible choice. The graph H is constructed as follows (see Figure 3b):

- Add a vertex for every cluster in $G[R]$ (white vertices).
- Add a vertex for every cluster in $G[S]$ (black vertices in S).
- For a cluster C_S in $G[S]$ and a cluster C_R in $G[R]$, add an edge between the vertex for C_S and the vertex for C_R if there is a class in C_R adjacent to C_S. This edge corresponds to choosing this class for C_R and is weighted with the total weight of the vertices in this class.
- Add a vertex for each class in a cluster C_R that is not adjacent to a cluster in $G[S]$ (black vertices outside S), and connect it to the vertex representing C_R. Again, this edge corresponds to choosing this class for C_R and is weighted with the total weight of the vertices in this class.

Since we only added edges between a black and a white vertex, H is bipartite. The task is now to find a *maximum-weight bipartite matching*, that is, a set of edges of maximum weight where no two edges have an endpoint in common. This allows any choice for a cluster, as long as no two clusters share edges to the same cluster in $G[S]$. The following lemma shows that this is a valid approach:

Lemma 2. *A maximum-weight bipartite matching in H provides an optimal* CVD COMPRESSION *solution.*

Proof. Each edge in a matching corresponds to a class in a cluster of $G[R]$. The CVD COMPRESSION solution is to delete all vertices in R but those of the selected classes. The matching cannot select two classes within the same cluster, since the corresponding edges have an endpoint in common; similarly, it cannot select two classes that share a connection to the same cluster in $G[S]$. Therefore, a matching yields a feasible solution. By Lemma 1, an optimal CVD COMPRESSION solution corresponds to an assignment of each cluster to one of its classes or to nothing, and therefore, it corresponds to a matching. Finally, the weight of a matching corresponds to the weight of the vertices not deleted from R, and therefore a maximum-weight matching corresponds to an optimal CVD COMPRESSION solution. □

Figure 3c shows the resulting cluster graph for our example after deleting the vertex sets corresponding to edges that are not selected by the maximum-weight matching shown in Figure 3b by bold edges. Note that the size of the solution can be upper-bounded by $k + 1$, since $\forall v \in V : \omega(v) \geq 1$. Altogether, we obtain

Proposition 1. *Weighted* Cluster Vertex Deletion *can be solved in* $O(2^k \cdot n^2(m + n \log n))$ *time.*

Proof. The correctness of the algorithm has already been shown. It remains to bound the running time. We can find clusters in S and R in $O(m)$ time by depth-first search within $G[S]$ and $G[R]$. Reduction Rule 1 can then be executed in $O(m)$ time. If Reduction Rule 1 has been applied, Reduction Rule 2 can be executed in $O(m)$ time by examining the degree of each vertex in R. Reduction Rule 3 can be executed in $O(m)$ time. Finally, we need to find a maximum weight matching in a bipartite graph with at most n vertices and at most m edges, which can be done in $O(n(m + n \log n))$ time [17]. Therefore, we can solve CVD Compression in the same time. The number of vertices in an intermediary solution X to be compressed is bounded by $k + 1$, because any such X consists of an optimal solution for a subgraph of G plus a single vertex. In one compression step, CVD Compression is thus solved $O(2^k)$ times, and there are n compression steps, yielding a total of $O(2^k \cdot n^2(m+n \log n))$ time. □

For the unweighted case, we can get better running times, since integer weighted matchings can be found faster than general weighted ones.

Theorem 1. *Unweighted* Cluster Vertex Deletion *can be solved in* $O(2^k \cdot km\sqrt{n} \log n)$ *time.*

Proof. For each matching instance, we can use an algorithm for integer weighted matching with a maximum weight of $C = n$ [18], yielding a running time of $O(m\sqrt{n} \log(nC)) = O(m\sqrt{n} \log n)$. Further, in iterative compression, we can save some iteration rounds by starting with a large subgraph for which we can still find in polynomial time a solution of size at most k. For Cluster Vertex Deletion, we can find a solution of size at most $3k$ by simply repeatedly finding a P_3 and then taking all three of its vertices. We can then start the iteration with G lacking these at most $3k$ vertices and an empty CVD set, after which we will need only $3k$ iteration rounds. □

Using problem kernelization, we can reach an additive running time of the form $O(f(k) + \text{poly}(n))$. First, we improve the running time bound on the enumeration of P_3's, compared to the trivial running time bound of $O(n^3)$.

Proposition 2. *All* P_3*'s of a graph can be enumerated in* $O(nm)$ *time.*

Proof. We can enumerate the P_3's by enumerating for each vertex $v \in V$ the P_3's in which v has degree two. This is done by scanning through v's adjacency list. For each vertex u in the adjacency list, we check for all vertices w that appear after u in the adjacency list of v whether u and w are adjacent. If this is not the case, then we have found a P_3 uvw and output it. The running time of this approach can be bounded as follows: for each vertex v, we spend $O(\deg(v) \cdot \deg(v))$ time, since we scan through its adjacency list, and for each position in the adjacency list, we scan once again through the part of the list after this position. Testing adjacency can be performed in constant time via an adjacency

matrix. Therefore, the total running time can be bounded as $O(\sum_{v \in V} \deg(v) \cdot \deg(v)) = O(\sum_{v \in V} \deg(v) \cdot n) = O(nm)$. □

Applying this algorithm for enumeration of P_3's together with a kernelization for 3-Hitting-Set [1] leads to the following running time.

Theorem 2. *Unweighted* Cluster Vertex Deletion *can be solved in* $O(2^k \cdot k^6 \log k + nm)$ *time.*

Proof. In $O(nm)$ time, we enumerate the P_3's of G. We then apply the linear-time kernelization by Abu-Khzam and Fernau [2], which gives us an instance with $O(k^3)$ vertices. To this instance, we apply the kernelization algorithm that yields a kernel with $O(k^2)$ vertices (running in $O(k^9)$ time) [1]. Finally, we apply Theorem 1. □

Curiously, we can use this unweighted algorithm as a subroutine to speed up the weighted case: if we have a solution for an unweighted instance, we can get an optimal weighted solution by executing the compression routine once. This works because the compression only requires that the set X to compress is a CVD set, and does not make any assumptions about its weight.

Theorem 3. *Weighted* Cluster Vertex Deletion *can be solved in* $O(2^k \cdot k^9 + nm)$ *time.*

Proof. Using the kernelization by Abu-Khzam and Fernau [2], we can in $O(nm)$ time first shrink the instance to $O(k^3)$ vertices. Then we can use the algorithm for the unweighted problem for all compression steps except the last one. For a kernel of size k^3, this takes $O(2^k \cdot k^9 + nm)$ time using Theorem 1. A single compression for the weighted problem takes $O(2^k \cdot n(m + n \log n))$ time, giving a time of $O(2^k(k^9 + k^6 \log k)) = O(2^k k^9)$ for the kernelized instance. In total, we arrive at the claimed bound. □

In fact, we even have a stronger parameterization in Theorem 3 when compared to Proposition 1: as parameter k, we can use the number of vertices in an optimal unweighted solution, which is less than or equal to the number of vertices in an optimal weighted solution, which in turn is less than or equal to the minimum weight of a weighted solution.

Since the matching subproblem is the bottleneck of the algorithm, it would be nice to replace it with something simpler. However, it is straightforward to show that the assignment problem in the last step of the compression routine is as hard as the task of finding a maximum weight matching in a bipartite graph, even after applying Reduction Rules 1–3. This indicates that the bottleneck of computing the maximum weight matching might actually be very difficult to overcome with our approach.

3 Cluster Vertex Deletion with a limited number of clusters

In clustering applications, it is often desirable to limit the number of output clusters, for example to simplify manual verification of the solution. The deletion of vertices should then produce a *d-cluster graph*, that is, a graph comprising *at most* d clusters.

Weighted d-Cluster Vertex Deletion
Instance: An undirected graph $G = (V, E)$, a vertex weight function $\omega : V \to [1, \infty)$, and a nonnegative number k.
Question: Is there a vertex set $X \subseteq V$ with $\sum_{v \in X} \omega(v) \leq k$ such that deleting all vertices in X from G results in a d-cluster graph?

Since the property of being a d-cluster graph is hereditary, it follows that d-Cluster Vertex Deletion is NP-complete [27].

We can show that slightly modifying the kernelization approach from Section 1 that makes use of a result for 3-Hitting Set [2] leads to a problem kernel.

Theorem 4. *Weighted d-Cluster Vertex Deletion admits a problem kernel containing $O(k^3)$ vertices, and it can be found in $O(nm)$ time.*

Proof. Since every vertex weight in a weighted d-Cluster Vertex Deletion instance is at least 1, any solution set has size at most k. Therefore, we can use the kernelization algorithm for 3-Hitting Set by Abu-Khzam and Fernau [2] that produces a kernel that contains $O(k^3)$ elements. This kernelization algorithm removes elements for two reasons. First, it removes all elements from the weighted 3-Hitting Set instance that appear in too many subsets, because they must belong to any 3-hitting set. Hence, these elements are added to the solution. Second, it removes elements that do not appear in any subset, because they must not belong to any minimal 3-hitting set. For details, we refer to [2].

Next, we describe our kernelization algorithm and bound the kernel size and the running time of the procedure.

Let G be the graph of the weighted d-Cluster Vertex Deletion instance. First, we enumerate the P_3's of G in $O(nm)$ time (see Proposition 2) and create a 3-Hitting-Set instance in which every P_3 corresponds to a subset of size 3. Since the minimum vertex weight is 1, the solution has size at most k.

Then, we perform all reduction rules of Abu-Khzam and Fernau [2] except the deletion of elements that do not appear in any subset. Let S be the set of elements after we have performed these reduction rules. Clearly, S contains $O(k^3)$ elements that appear in subsets and an unbounded number of elements that do not appear in any subset. We then transform the weighted 3-Hitting Set instance back into a weighted d-Cluster Vertex Deletion instance by creating the graph $G[S]$. This graph has $O(k^3)$ vertices that appear in induced P_3's and an unbounded number of vertices that do not appear in P_3's. The vertices that do not appear in any induced P_3 are part of isolated clusters. In Cluster

Vertex Deletion, we can remove all of these vertices from the graph since adding these vertices to a CVD set is never optimal. However, in weighted d-Cluster Vertex Deletion, we might have to add some of these vertices to the d-CVD set in order to delete surplus clusters. We now describe how we can remove some of the isolated clusters from the graph without removing any clusters that we might have to add to the d-CVD set.

Clearly, we either have to delete all vertices of a cluster or none: a d-CVD set that deletes some but not all vertices of a cluster is non-optimal since we could reinsert these deleted vertices into the graph without increasing the number of clusters and obtain a better solution. Furthermore, we cannot delete any clusters in which the sum of the vertex weights is more than k. Hence, we can remove each of these clusters from the graph, decreasing d by 1, because any such cluster is already fixed as one cluster of the final d-cluster graph. All remaining clusters contain at most k vertices. Clearly, we can delete at most k clusters. Therefore, we only have to keep the k clusters that have the lowest weight. All others can be removed without decreasing k, decreasing d by 1 for each removed isolated cluster. The remaining graph then contains at most k isolated clusters with at most k vertices each, resulting in $O(k^2)$ vertices that are part of isolated clusters. Together with the $O(k^3)$ vertices that are not part of isolated clusters, the graph then has $O(k^3)$ vertices overall. Clearly, all steps can be performed in $O(nm)$ time. □

The kernelization result implies that d-Cluster Vertex Deletion is fixed-parameter tractable with respect to the parameter k. A straightforward approach to solve d-Cluster Vertex Deletion is to apply a search tree algorithm that branches on the different cases to destroy a P_3 by vertex deletion, deleting a different vertex in each branch. Since the minimum vertex weight is 1, the parameter is reduced by at least 1 in each search tree branch. Let k' be the sum of the weights of the vertices that may still be deleted at a given search tree node. Branching is performed as long as the graph contains a P_3 and $k' \geq 1$. If $k' < 1$, and the graph still contains a P_3, then we have not found a d-CVD set of weight at most k and we cannot delete further vertices. If otherwise the graph is P_3-free, then it is a cluster graph albeit one that might contain more than d clusters. Therefore, we delete the cluster of minimum weight until either G is a d-cluster graph, which means that we have found a solution, or $k' < 1$, which means that no solution was found in this search tree branch.

Clearly, this search tree procedure finds a d-CVD set of minimum weight, since it explores all possibilities to destroy the P_3's of the graph and afterwards optimally removes surplus clusters. A straightforward combination of search tree, kernelization and the interleaving technique [30] leads to the following running time.

Proposition 3. *Weighted d-Cluster Vertex Deletion can be solved in running time $O(3^k + nm)$.*

In the remainder of this section, we describe three different algorithms that improve on this trivial search tree algorithm. The first algorithm is a modifi-

cation of the iterative compression algorithm from Section 2 and also uses the parameter k. The second and third algorithm use the combined parameter (d, k) but are preferable if d is small compared to k.

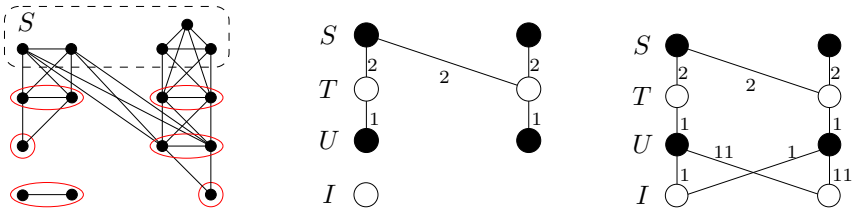
3.1 An iterative compression algorithm for d-Cluster Vertex Deletion

In the following, we describe how to modify the algorithm from Section 2 in order to ensure that the graph remaining after the removal of the CVD set comprises at most d clusters. The only part of the algorithm that has to be modified is the compression routine. The problem that has to be solved by this compression routine can be formulated as follows:

> d-CVD Compression
> **Instance:** An undirected graph $G = (V, E)$, a vertex weight function $\omega : V \to [1, \infty)$, and a vertex set $S \subseteq V$ such that $G[S]$ and $G \setminus S$ are d-cluster graphs.
> **Task:** Find a vertex set $X' \subseteq V \setminus S$ such that $G \setminus X'$ is a d-cluster graph and $\sum_{v \in X'} \omega(x)$ is minimum.

The main outline of the procedure remains the same, that is, we first perform data reduction, and subsequently solve the remaining problem with a matching algorithm. Recall that $R := V \setminus S$ denotes the set of vertices that may still be deleted in order to obtain a d-cluster graph. We can apply Reduction Rules 1 and 2 of the original algorithm, since in these reduction rules we only delete vertices that appear in a P_3 together with two vertices of S, and hence have to be deleted in order to destroy this P_3. In contrast, we cannot apply Reduction Rule 3, since this reduction rule removes isolated clusters from the graph. However, we might have to delete some of these clusters in order to obtain a graph comprising at most d clusters.

After performing Reduction Rules 1 and 2, we divide the vertices of each cluster in $G[R]$ into equivalence classes according to their neighborhood in S in the same way as it was done in the algorithm from Section 2. This situation is depicted in Figure 4a. From these equivalence classes and the clusters in $G[S]$, we create an auxiliary graph H, in which each vertex represents an equivalence class, the difference being that now this graph contains isolated vertices, representing the isolated clusters that were not removed. An example of this graph is shown in Figure 4b. We partition the vertices of H as follows: the set S in H contains the vertices that correspond to clusters in $G[S]$, the vertices in T correspond to sets of vertices in G that are adjacent to vertices in S, the vertices in U correspond to sets of vertices in G that are not adjacent to vertices in S, and the vertices in I correspond to isolated clusters in G. Not deleting a set of vertices that corresponds to a vertex in U means that we create an additional cluster. Clearly, if $|S| + |U| + |I| \leq d$, then we always obtain a d-cluster graph, and hence we can remove the isolated vertices from this auxiliary graph H and solve the matching problem as in Section 2. Otherwise, we transform H in or-



(a) A graph after application of Reduction Rules 1 and 2.

(b) The initial auxiliary graph H.

(c) The modified auxiliary graph H.

Figure 4: Example of the compression procedure for 3-Cluster Vertex Deletion.

der to obtain a graph whose maximum weight matching provides a minimum weight d-CVD set.

Since we cannot delete any vertices from S, we can create at most $d' := d - |S|$ additional clusters. First, we apply a data reduction rule:

Reduction Rule 4. *If $d' > |U|$, then remove the vertex corresponding to the isolated cluster of maximum weight from I and set $d' := d' - 1$.*

Proof of correctness. Even if all clusters corresponding to vertices from U are not deleted, there is at least one isolated cluster that we do not need to delete, and it is optimal to not delete the isolated cluster of maximum weight. □

After this reduction rule has been exhaustively applied, we can assume that $d' \leq |U|$. We now modify the auxiliary graph H as follows:

- Connect each vertex in I with each vertex in U. To each new edge, assign a weight that equals the weight of the cluster that corresponds to the vertex in I.
- Add $|U| - d'$ additional vertices to H. Let I' be the set of these additional vertices. Connect each vertex in I' to all vertices in U, and assign a weight that is larger than the sum of all other weights in H to every new edge.

The goal of these modifications is to ensure that the cluster graph corresponding to the maximum weight matching has at most d clusters. A matching edge between a vertex from U to a vertex from I models that the cluster corresponding to the vertex of U must be deleted, since we decided to keep the cluster corresponding to the vertex of I. With the additional vertices I', we model that at least $|U| - d'$ clusters corresponding to vertices in U must be deleted. In the following lemma, we prove the correctness of this approach.

Lemma 3. *Let H be an auxiliary graph constructed as described. A maximum weight matching of H provides an optimal d-CVD Compression solution.*

Proof. We show that each maximum matching provides a d-CVD solution. The optimality of the solution can be demonstrated in complete analogy to Lemma 2.

In order to ensure that the resulting graph is a d-cluster graph, the number of clusters that do not contain vertices in S can be at most $d' = d - |S|$. We have to create an additional cluster whenever a vertex of U is matched via an edge between T and U, or via an edge between a vertex in I and U. An edge between a vertex in U and a vertex in I' does not create an additional cluster.

Each of the $|U| - d'$ vertices in I' is matched in a maximum weight matching, because of our choice of weights. Therefore, at most d' of the vertices in U are matched via edges that are not incident to vertices in I'. The matching thus contains at most d' edges that create new clusters. Since there were $d - d'$ clusters in $G[S]$, the graph corresponding to the matching is a d-cluster graph. □

The modifications that have to be applied to the auxiliary graph can be performed in $O(k^6)$ time, since this graph contains $O(k^3)$ vertices. We can thus bound the running time of the algorithm as in Section 2.

Theorem 5. *Weighted d-Cluster Vertex Deletion can be solved in $O(2^k \cdot k^9 + nm)$ time, and unweighted d-Cluster Vertex Deletion can be solved in $O(2^k \cdot k^6 \log k + nm)$ time.*

3.2 Solving d-Cluster Vertex Deletion via reduction to Vertex Cover

Here, we present an algorithm that solves weighted d-Cluster Vertex Deletion via the computation of minimum weight vertex covers.[5] Compared to the fixed-parameter tractability result from Section 3.1, which is only based on the parameter k, we now employ the combined parameter (d, k).

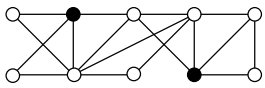
The idea is to try all independent sets of size at most d and to solve weighted d-Cluster Vertex Deletion for the case that these vertices are not deleted from the graph. Since in a d-cluster graph any set of vertices from different clusters forms an independent set, at least one of the independent sets of size at most d must be a set of vertices that remain in the graph.

Suppose that such an independent set D of size at most d is given. We call the vertices in D *permanent*. In the following, we describe how to compute the minimum weight d-CVD set of such a graph; an example is shown in Figure 5. First, we perform the following reduction rule.

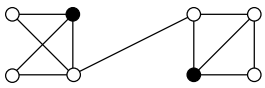
Reduction Rule 5. *Delete all vertices from the graph that are not adjacent to any vertex in the independent set D and all vertices that are adjacent to more than one vertex in D.*

The correctness of Reduction Rule 5 is obvious; an example of its application is given in Figure 5b. For each deleted vertex v, we decrease k by $\omega(v)$. Let G be a graph with a size-d independent set of permanent vertices after application

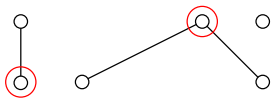
[5] A *vertex cover* of a graph is a set C of graph vertices such that every graph edge has at least one endpoint in C.



(a) The original graph G with two non-adjacent permanent vertices.



(b) After Reduction Rule 5.



(c) The graph G' with a vertex cover X (marked with circles).

(d) The 2-cluster graph after the removal of X.

Figure 5: Example of the algorithm for 2-Cluster Vertex Deletion when a size-2 independent set of vertices that cannot be deleted is given. Black vertices are permanent.

of Reduction Rule 5. All non-permanent vertices of G are adjacent to exactly one permanent vertex. To produce a cluster graph, we also have to ensure that all neighbors of a permanent vertex are adjacent, and neighbors of different permanent vertices are non-adjacent. These two attributes can be encoded into a graph G' such that a vertex cover of G' is a vertex set whose removal establishes the attributes in G. We construct the graph G' from G as follows: For any pair u, v of non-permanent vertices that is adjacent to the same permanent vertex, we do the following:

- remove the edge $\{u, v\}$, if u and v are adjacent;
- insert the edge $\{u, v\}$, otherwise.

Furthermore, we remove all permanent vertices. After this, we have obtained G' (for an example of this construction see Figure 5c).

In the following lemma, we show that a vertex cover of G' is a d-CVD set of G; an example of this equivalence is shown in Figures 5c and 5d.

Lemma 4. *Let G be a graph with a size-d independent set of permanent vertices that is reduced with respect to Reduction Rule 5 and G' a graph constructed as described above. Then, a vertex set X is a vertex cover of G' if and only if X is a d-*CVD *set of G.*

Proof. Let X be a vertex cover of G', and let u and v be two non-permanent vertices in $V \setminus X$. We show that u and v are adjacent in G if and only if they are neighbors of the same permanent vertex. Since G is reduced with respect to Reduction Rule 5, this implies that $G \setminus X$ is a d-cluster graph.

Clearly, u and v are nonadjacent in G'. If u and v are both adjacent to the same permanent vertex p in G, then they have to be adjacent in G, because

of the way we constructed G'. Hence, the neighborhood of any permanent vertex $p \in D$ in $G \setminus X$ is a clique. Otherwise, if u and v are adjacent to different permanent vertices in G, then they cannot be adjacent in G, since in the construction of G' from G we did not modify any edges between vertices that are neighbors of different permanent vertices. Hence, there are no edges between neighbors of different permanent vertices in $G \setminus X$. Therefore, $G \setminus X$ comprises exactly d clusters, which means that X is a d-CVD set of G.

The same reasoning can be applied to show that a d-CVD set of G that does not delete any permanent vertices is a vertex cover of G'. □

We now bound the running time of computing a d-CVD set of a graph, once an independent set of size at most d that may not be deleted is given. It fundamentally relies on a fixed-parameter algorithm for weighted Vertex Cover [31].

Lemma 5. *Let $G = (V, E)$ be a graph and $D \subseteq V$ an independent set of size at most d. A minimum weight d-CVD set of G of weight at most k that does not delete any vertex $v \in D$ can be computed in $O(1.40^k + n^2)$ time.*

Proof. We bound the running time of the algorithm that was described; the correctness follows from its description.

Applying Reduction Rule 5 can be performed in $O(n^2)$ time. Constructing G' also runs in $O(n^2)$ time. The computation of minimum weight vertex covers of weight at most k can be done in $O(1.40^k + kn)$ time [31]. Overall, this amounts to the claimed running time. □

Combining this approach with the kernelization algorithm from Theorem 4, we achieve the following running time.

Theorem 6. *Weighted d-Cluster Vertex Deletion can be solved in running time $O(1.40^k \cdot k^{3d} + nm)$.*

Proof. The kernelization algorithm runs in $O(nm)$ time. After kernelization the graph comprises $O(k^3)$ vertices. Hence, there are $\binom{k^3}{d} = O(k^{3d})$ possible choices for independent sets of size d. By Lemma 5, we can find a minimum weight d-CVD set of a graph of size $O(k^3)$ in which an independent set D of size at most d cannot be deleted in $O(1.40^k + k^6) = O(1.40^k)$ time. This is done for all $O(k^{3d})$ sets, and the best solution is stored. Clearly, the overall running time is $O(1.40^k \cdot k^{3d} + nm)$. □

For the unweighted case, we can apply the currently fastest algorithm for unweighted Vertex Cover by Chen et al. [9] in combination with the size $O(k^2)$ kernel for unweighted 3-Hitting Set, yielding an improved running time.

Theorem 7. *Unweighted d-Cluster Vertex Deletion can be solved in running time $O(1.28^k \cdot k^{2d} + nm)$.*

3.3 Combining branching and reduction to Vertex Cover

Clearly, the algorithm from Theorem 6 is only feasible for small values of d. In this section, we describe an algorithm that is useful if neither d nor k is very small, and combines an improved branching strategy with the algorithm from Theorem 6.

First, we apply the kernelization algorithm from Theorem 4. Next, we perform a search tree algorithm that branches on forbidden subgraphs. For a vertex v in a forbidden subgraph, we have two choices: either we have to delete v, or v is one of the remaining vertices in the d-cluster graph. If v is deleted, the combined parameter $k + d$ decreases by $\omega(v) \geq 1$. Explicitly not deleting v means that we assign a cluster to v. In this case, we call v *permanent*. If v does not have any permanent neighbors, then we have assigned a new cluster. Hence, $k + d$ also decreases by 1.

Let k' be the sum of the weights of the vertices that may still be deleted at a given search tree node and d' the number of clusters that may still be assigned. Before branching, we perform the following data reduction rule.

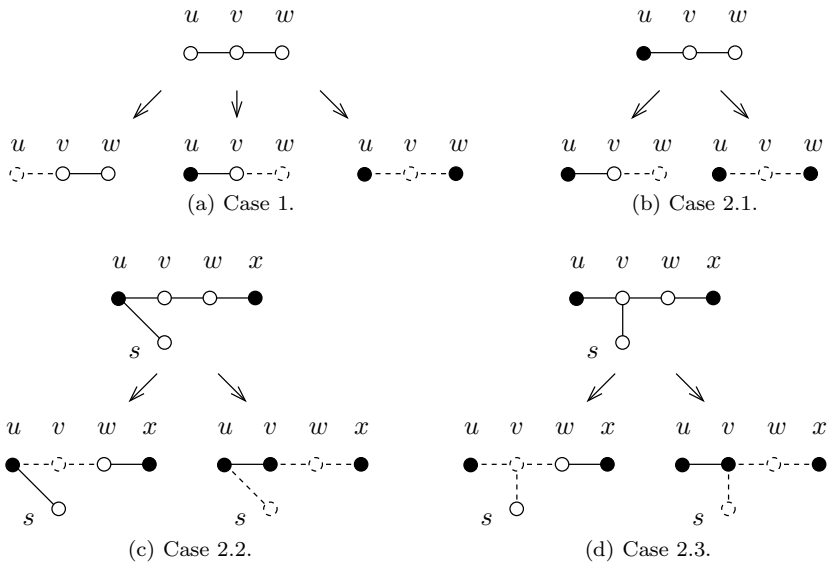
Reduction Rule 6. *If G contains a P_3 with two permanent vertices u, v and one non-permanent vertex w, then delete w from G and set $k' := k' - \omega(w)$.*

Proof of correctness. Since we may not delete any of the two permanent vertices u and v, we have to delete vertex w in order to eliminate the P_3. □

Clearly, if $k' < 1$, then we cannot delete any vertices and either the graph is already a d-cluster graph or this particular branch of the search tree is a dead end. Furthermore, if $d' = 0$, then we cannot assign further clusters. This means that there is an independent set of d permanent vertices. By Lemma 5, we can find a d-CVD set of such a graph in $O(1.40^k + k^6) = O(1.40^k)$ time. In the following, we describe the branching rules for the case $k' \geq 1$ and $d' > 0$. After application of Reduction Rule 6, every P_3 contains at most one permanent vertex.

First, we branch on P_3's that consist of vertices that are not adjacent to permanent vertices. If such a P_3 does not exist, then we branch on P_3's that contain a permanent vertex u that is not the middle vertex of the P_3. Finally, we show that if none of the other cases applies, then we can find a minimum weight d-CVD set of the graph by computing a minimum weight vertex cover. In the following, we describe the branching rules in detail; each of them is also shown in Figure 6.

Case 1: There is a P_3 uvw such that u, v, and w are not adjacent to permanent vertices. We branch into three cases. In the first case, we delete u; the parameter $k' + d'$ decreases by at least 1. In the second case, we mark u as permanent and delete w; the parameter $k' + d'$ decreases by at least 2 (one new assigned cluster and one deleted vertex). In the third case we mark u and w as permanent and delete v; the parameter $k' + d'$ decreases by at least 3 (two new assigned clusters and one deleted vertex).



(a) Case 1. (b) Case 2.1. (c) Case 2.2. (d) Case 2.3.

Figure 6: Branching rules on P_3's that are not adjacent to permanent vertices (Case 1) and P_3's that contain one permanent non-middle vertex (Cases 2.1–2.3). Dashed vertices and edges are deleted, black vertices are permanent.

Case 2: There is a P_3 uvw such that u is permanent and v and w are non-permanent. We consider several subcases of this case.

Case 2.1: Vertex w has no permanent neighbors. We branch into two cases. In the first branch, we delete w; the parameter decreases by at least 1. In the second branch, we mark w as permanent and thus delete v; the parameter decreases by at least 2 (one new assigned cluster and one deleted vertex).

Case 2.2: Vertex w has a permanent neighbor x and u (or x) has a non-permanent neighbor s that is not adjacent to v (or w). Without loss of generality assume that u has a neighbor s that is not adjacent to v. We branch into two cases. In the first branch, we delete v; the parameter decreases by at least 1. In the second branch, we mark v as permanent and may thus delete s and w; the parameter decreases by at least 2.

Case 2.3: Vertex w has a permanent neighbor x and v (or w) has a non-permanent neighbor s that is not adjacent to u (or x). Without loss of generality assume that v has a neighbor s that is not adjacent to u. We branch into two cases. In the first branch, we delete v; the parameter decreases

by at least 1. In the second branch, we mark v as permanent and may thus delete s and w; the parameter decreases by at least 2.

Case 2.4: Otherwise. If none of the other subcases of Case 2 applies, then the following must hold: Clearly, vertex w has a permanent neighbor x. Otherwise, Case 2.1 would apply. Furthermore, v is adjacent to all non-permanent neighbors of u, since Case 2.2 does not apply. Also, u is adjacent to all non-permanent neighbors of v other than w, since Case 2.3 does not apply. Finally, there is no permanent neighbor of u that is not adjacent to v and vice versa, since we performed Reduction Rule 6. Hence, $N[v] = N[u] \cup \{w\}$. Using the same arguments we can show that $N[w] = N[x] \cup \{v\}$. We delete that vertex of v and w which has lower weight; no branching takes place. For the correctness, consider the following: Clearly, we have to delete at least one of v and w. Let v be the vertex of lower weight of v and w. Suppose that there is a d-CVD set S that contains w, where $\omega(w) \geq \omega(v)$. In case $v \in S$, we can reinsert w into the graph without violating the d-cluster property (because w is only adjacent to neighbors of x and to v). Obviously, the weight of the solution decreases. In case $v \notin S$, we can delete v from the graph and reinsert w into the graph, again without violating the d-cluster property. Therefore, deleting the vertex that has lower weight is optimal.

Case 3: Otherwise. The following must hold: Since Case 1 does not apply, every P_3 uvw contains at least one vertex that is adjacent to a permanent vertex. Furthermore, if a P_3 uvw contains a permanent vertex, then this permanent vertex is v. Otherwise, Case 2 would apply. Therefore, every connected component either contains a permanent vertex or it is an isolated cluster. Also, if a non-permanent vertex v is adjacent to a permanent vertex u, then all of v's neighbors must be adjacent to u. Otherwise, there would be a P_3 uvw in which the permanent vertex is not v.

If the graph contains more than d connected components, then we must remove isolated clusters to decrease the number of clusters (all other connected components contain a permanent vertex and thus cannot be completely deleted). We remove isolated clusters of minimum weight until either $k' < 1$ or the graph contains exactly d connected components. In the first case, we cannot obtain a d-cluster graph, in the second case, we can mark the remaining isolated clusters as permanent. Then, there is an independent set of permanent vertices of size at most d and we can thus compute a d-CVD set with weight at most k' in $O(1.40^{k'})$ time (Lemma 5).

Theorem 8. *Weighted d-Cluster Vertex Deletion can be solved in running time $O(1.84^{k+d} + nm)$.*

Proof. We prove the theorem by bounding the running time of the described algorithm. The correctness of the algorithm follows from the given description.

As shown in Theorem 4, kernelization can be done in $O(nm)$ time. We bound the size of the search tree by analyzing the branching vectors and their

branching number; for details on this type of analysis, we refer to [29, Chapter 8]. In the search tree, the branching vector with the largest branching number is $(1, 2, 3)$ from Case 1. The branching number of this vector is 1.84. Hence, the search tree has size $O(1.84^{k+d})$. The computation of minimum weight vertex covers is also performed by a search tree procedure, whose largest branching number is 1.40. Hence, applying the fixed-parameter algorithm for weighted Vertex Cover in case $d' = 0$ does not increase the worst-case search tree size.

Every step at a search tree node can be computed in polynomial time. By interleaving kernelization and branching, the running time for the search tree can be improved from $O(1.84^{k+d} \cdot \text{poly}(k))$ to $O(1.84^{k+d})$ [30]. Together with the time needed for the kernelization ($O(nm)$), we arrive at the claimed overall running time bound. □

4 Outlook

It is open to improve the trivial factor-3 approximation for Cluster Vertex Deletion. Cai et al. [8] improved the approximation factor for (unweighted) Feedback Vertex Set in tournaments, which is also characterized by a 3-vertex forbidden subgraph, from 3 to 2.5; perhaps similar techniques are applicable here. Moreover, the exponential upper bounds for our search-tree based algorithms should be improvable. More importantly, for the unweighted case of Cluster Editing, $O(k)$-vertex problem kernels are known [13, 22], whereas correspondingly for Cluster Vertex Deletion only an $O(k^2)$-vertex kernel is known. Also, improving the $O(k^3)$-vertex problem kernel for the weighted case would be desirable. Further problem variants, for example a variation of d-Cluster Vertex Deletion that specifies the exact number of desired clusters could be practically relevant. While the algorithms from Subsections 3.2 and 3.3 can be applied to this problem, the method of iterative compression is not applicable, since the property of comprising a fixed number of clusters is not hereditary. Finally, all our results are worst-case estimates. Practical tests based on algorithm engineering seem promising.

Acknowledgments. We are grateful to anonymous referees for pointing out some inconsistencies in the manuscript of the submitted conference version and for other comments that have improved the presentation.

References

[1] F. N. Abu-Khzam. Kernelization algorithms for d-hitting set problems. In *Proc. 10th WADS*, volume 4619 of *LNCS*, pages 434–445. Springer, 2007.

[2] F. N. Abu-Khzam and H. Fernau. Kernels: Annotated, proper and induced. In *Proc. 2nd IWPEC*, volume 4169 of *LNCS*, pages 264–275. Springer, 2006.

[3] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. In *Proc. 37th STOC*, pages 684–693. ACM, 2005.

[4] N. Ailon, M. Charikar, and A. Newman. Proofs of conjectures in "Aggregating inconsistent information: Ranking and clustering". Technical Report TR-719-05, Department of Computer Science, Princeton University, 2005.

[5] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. A fixed-parameter approach for weighted cluster editing. In *Proc. 6th APBC*, volume 5 of *Series on Advances in Bioinformatics and Computational Biology*, pages 211–220. Imperial College Press, 2008.

[6] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. Going weighted: Parameterized algorithms for cluster editing. In *Proc. 2nd COCOA*, volume 5165 of *LNCS*, pages 1–12. Springer, 2008.

[7] L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.

[8] M.-C. Cai, X. Deng, and W. Zang. An approximation algorithm for feedback vertex sets in tournaments. *SIAM Journal on Computing*, 30(6):1993–2007, 2001.

[9] J. Chen, I. A. Kanj, and G. Xia. Improved parameterized upper bounds for vertex cover. In *Proc. 31st MFCS*, volume 4162 of *LNCS*, pages 238–249. Springer, 2006.

[10] F. Dehne, M. A. Langston, X. Luo, S. Pitre, P. Shaw, and Y. Zhang. The cluster editing problem: Implementations and experiments. In *Proc. 2nd IWPEC*, volume 4169 of *LNCS*, pages 13–24. Springer, 2006.

[11] M. Dom, J. Guo, F. Hüffner, R. Niedermeier, and A. Truß. Fixed-parameter tractability results for feedback set problems in tournaments. In *Proc. 6th CIAC*, volume 3998 of *LNCS*, pages 320–331. Springer, 2006.

[12] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[13] M. R. Fellows, M. A. Langston, F. A. Rosamond, and P. Shaw. Efficient parameterized preprocessing for cluster editing. In *Proc. 16th FCT*, volume 4639 of *LNCS*, pages 312–321. Springer, 2007.

[14] H. Fernau. Parameterized algorithms for hitting set: The weighted case. In *Proc. 6th CIAC*, volume 3998 of *LNCS*, pages 332–343. Springer, 2006.

[15] J. Flum and M. Grohe. *Parameterized Complexity Theory.* Springer, 2006.

[16] F. V. Fomin, S. Gaspers, D. Kratsch, M. Liedloff, and S. Saurabh. Iterative compression and exact algorithms. In *Proc. 33rd MFCS*, volume 5162 of *LNCS*, pages 335–346. Springer, 2008.

[17] M. L. Fredman and R. E. Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM*, 34(3):596–615, 1987.

[18] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM Journal on Computing*, 18(5):1013–1036, 1989.

[19] I. Giotis and V. Guruswami. Correlation clustering with a fixed number of clusters. *Theory of Computing*, 2:249–266, 2006.

[20] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39 (4):321–347, 2004.

[21] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38 (4):373–392, 2005.

[22] J. Guo. A more effective linear kernelization for cluster editing. In *Proc. 1st ESCAPE*, volume 4614 of *LNCS*, pages 36–47. Springer, 2007. Long version to appear in *Theoretical Computer Science*.

[23] J. Guo, H. Moser, and R. Niedermeier. Iterative compression for exactly solving NP-hard minimization problems. In *Algorithmics of Large and Complex Networks*, LNCS. Springer, 2008. To appear.

[24] F. Hüffner. *Algorithms and Experiments for Parameterized Approaches to Hard Graph Problems*. PhD thesis, Institut für Informatik, Friedrich-Schiller-Universität Jena, 2007.

[25] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 51(1):7–25, 2008.

[26] K. Jansen, P. Scheffler, and G. Woeginger. The disjoint cliques problem. *RAIRO Recherche Opérationnelle*, 31(1):45–66, 1997.

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

[28] C. Lund and M. Yannakakis. The approximation of maximum subgraph problems. In *Proc. 20th ICALP*, volume 700 of *LNCS*, pages 40–51. Springer, 1993.

[29] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Number 31 in Oxford Lecture Series in Mathematics and Its Applications. Oxford University Press, 2006.

[30] R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Information Processing Letters*, 73:125–129, 2006.

[31] R. Niedermeier and P. Rossmanith. On efficient fixed-parameter algorithms for weighted vertex cover. *Journal of Algorithms*, 47(2):320–331, 2003.

[32] F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 2008. To appear.

[33] S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truß, and S. Böcker. Exact and heuristic algorithms for weighted cluster editing. In *Proc. 6th CSB*, volume 6 of *Computational Systems Bioinformatics*, pages 391–401. Imperial College Press, 2007.

[34] V. Raman and S. Saurabh. Parameterized algorithms for feedback set problems and their duals in tournaments. *Theoretical Computer Science*, 351(3):446–458, 2006.

[35] B. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.

[36] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1–2):173–182, 2004.

[37] M. Wahlström. *Algorithms, Measures and Upper Bounds for Satisfiability and Related Problems*. PhD thesis, Department of Computer and Information Science, Linköpings universitet, 2007.

[38] A. van Zuylen and D. P. Williamson. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In *Proc. 5th WAOA*, volume 4927 of *LNCS*, pages 260–273. Springer, 2008.