Fixed-Parameter Algorithms for Cluster Vertex Deletion

Falk Hüffner*, Christian Komusiewicz**, Hannes Moser***, and 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. Our iterative compression algorithm for CVD seems to be the first nontrivial application of this fairly new technique to a problem that is not a feedback set problem. Moreover, we study the variant of CVD where the number of cliques to be generated is specified. Here, we detect 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 [21] 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*. Herein, the three 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 [6, 19]. Whereas also a factor-2.5 polynomial-time approximation for Cluster Editing is known [3, 23], in practical applications fixed-parameter algorithms (combined with some heuristics) providing optimal solutions seem to dominate [4, 6, 19]. Parameterized complexity studies for Cluster Editing were initiated by Gramm et al. [11]

* Supported by the Deutsche Forschungsgemeinschaft, Emmy Noether research group PIAF (fixed-parameter algorithms), NI 369/4.
** 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 have been further pursued in a series of papers [4, 6, 7, 10, 12, 18, 19]. A previously shown bound of $O(1.92^k + n^3)$ for an n-vertex graph [10] can be improved by combining a linear-time problem kernel [7] with the currently best claimed running time of $O(1.83^k + n^3)$ [4] to get an algorithm with running time $O(1.83^k + n + m)$, 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 [7, 12], the best upper bound currently being $4k$ [12].

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 minimum number of vertices such that their 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 subset $X \subseteq V$ 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).

Motivation. As 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 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. 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, making a parameterized approach particularly meaningful here.

Known results. By general results for vertex deletion problems for hereditary graph properties, it follows that already unweighted Cluster Vertex Deletion is NP-complete [15]. Only few specific results for (unweighted) Cluster

[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.

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 of three vertices.[3] Gramm et al. [10] used an elaborate case distinction found with computer help to derive a 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 [22]. 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 [8], implying an $O(2.25^k + n^3)$ running time for weighted Cluster Vertex Deletion.

New results. One of our main results is an elegant iterative compression algorithm for weighted Cluster Vertex Deletion using matching techniques, running in $O(2^k k^9 + n^3)$ time. Notably, this seems to be the first nontrivial application of the technique of iterative compression (described by Reed et al. [20]; see also [16, Chapter 11]) to a non-feedback set problem. We extend our studies to the (also NP-hard) case where the number of clusters to be generated is given by a second parameter d. Such studies have also been undertaken for Cluster Editing [9, 12, 21], 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, leading to the NP-hardness of so-called d-Cluster Vertex Deletion also for $d > 1$. Since d-Cluster Vertex Deletion is already NP-hard for $d = 1$, 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: $O(3^k + n^3)$, $O(1.40^k k^{3d} + n^3)$, and $O(1.84^{k+d} + n^3)$. Depending on the value of d, each of these algorithms may be preferable in certain constellations.

[2] Jansen et al. [14] 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 simply writing of containment of a P_3 in a graph we actually always refer to an induced P_3.

```
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' ←ReduceRule1(G')
 6          G' ←ReduceRule2(G')
 7          G' ←ReduceRule3(G')
 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'
```

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

In the latter two algorithms, fixed-parameter algorithms for weighted Vertex Cover play a decisive role.

Due to the lack of space, most details are deferred to the full version of this paper.

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 [13] and [16, Chapter 11]. We first describe how to employ a compression routine, and then the compression routine itself. We call a set of vertices whose deletion produces a cluster graph a *CVD set*.

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 Fig. 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 new vertices from $V \setminus X$. The idea is to try by brute force all $2^{|X|} - 1$ 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 subset $S \subseteq V$ such that $G[S]$ and $G \setminus S$ are cluster graphs.
Task: Find a set $X' \subseteq V \setminus S$ such that $G \setminus X'$ is a cluster graph and $\sum_{v \in X'} \omega(x)$ is minimum.

The instance can now be simplified by a series of data reduction rules. We call a connected component in a cluster graph a *cluster*.

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

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

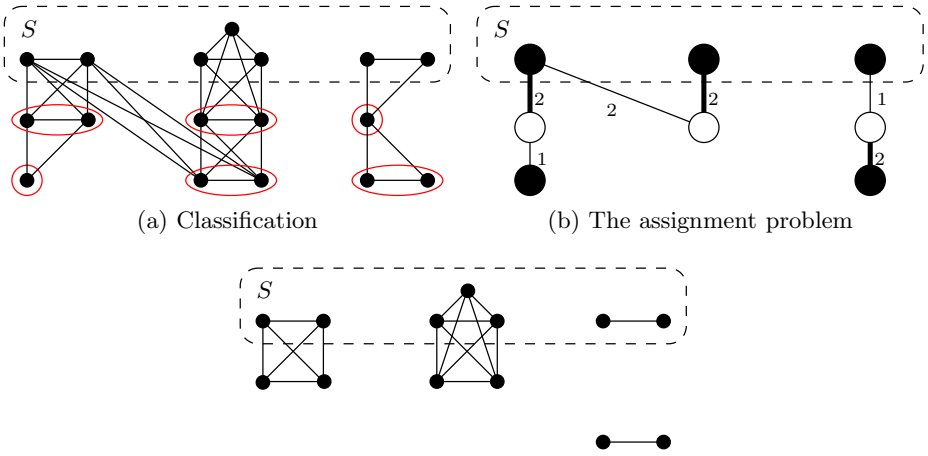
Reduction Rule 3. *Remove connected components that are complete graphs.*

After Rules 1–3 have been applied, the instance is much simplified. 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 Fig. 2a). This classification is useful because of the following lemma.

Lemma 1. *In an optimal* CVD compression *solution, for each cluster in $G[R]$, either the vertices of exactly one class are present, or the whole cluster is deleted.*

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 connected 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 Fig. 2b):

– Add a vertex for every cluster in $G[R]$ (white vertices).



(a) Classification (b) The assignment problem

(c) Final resulting cluster graph

Fig. 2: Assignment problem in the iterative compression, unweighted case.

- 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 connected 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 connected 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.*

Fig. 2c 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 Fig. 2b 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.*

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

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

Problem kernelization leads to the following.

Theorem 2. *Unweighted* Cluster Vertex Deletion *can be solved in* $O(2^k \cdot k^6 \log k + n^3)$ *time.*

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 does only require 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* $O(2^k \cdot k^9 + n^3)$ *time.*

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, we can 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 fixed number of clusters

In clustering applications, the number of desired clusters is often known. The deletion of vertices should then produce a *d-cluster graph*, that is, a graph comprising *exactly* 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 subset $X \subseteq V$ with $\sum_{v \in X} \omega(v) \leq k$ such that deleting all vertices in X from G results in a d-cluster graph?

1-Cluster Vertex Deletion is equivalent to Clique, since a 1-cluster graph is a complete graph. Hence, 1-Cluster Vertex Deletion is NP-complete. More generally, we have the following.

Proposition 2. d-Cluster Vertex Deletion *is NP-complete for any constant integer* d.

3.1 An $O(3^k + n^3)$ time algorithm

We start with describing a simple search tree algorithm for weighted d-Cluster Vertex Deletion parameterized by the weight k of a solution set. In the search tree, we branch on induced P_3's until the graph is a cluster graph, and then remove surplus clusters in case the graph contains more than d clusters. Before starting the search tree procedure, we perform data reduction. The subsequent problem kernel result makes use of the corresponding result for 3-Hitting Set [2].

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

After kernelization, we perform a search tree procedure. We branch into three 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 removed 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 remove further vertices. If otherwise the graph contains no P_3, then it is a cluster graph. Let S be the set of vertices that were removed so far. We distinguish four cases.
1. $k' < 0$. The weight of S exceeds k. Therefore, no solution was found.
2. $G \setminus S$ comprises less than d clusters. We can discard S, since S is not a d-CVD set and no superset of S is a d-CVD set.
3. $G \setminus S$ comprises more than d clusters. We compute the sum of the vertex weights for all remaining clusters, and remove a cluster with minimum weight until either $G \setminus S$ is a d-cluster graph (then Case 4 applies) or $k' < 1$ (no solution set was found in this search tree branch).
4. $G \setminus S$ is a d-cluster graph. In this case, S is a d-CVD set of weight at most k. 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 (in Case 3). Below, we bound the running time of the described algorithm.

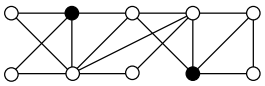
Theorem 5. *Weighted d-Cluster Vertex Deletion can be solved in running time $O(3^k + n^3)$.*

3.2 An $O(1.40^k \cdot k^{3d} + n^3)$ time algorithm

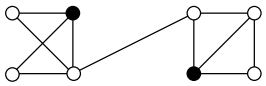
Now we present an algorithm that solves weighted d-Cluster Vertex Deletion via the computation of minimum weight vertex covers.[4]

The idea is to try all independent sets of size d and to solve weighted d-Cluster Vertex Deletion for the case that these vertices are not removed

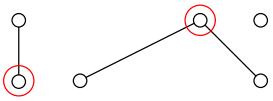
[4] A *vertex cover* of a graph is a set C of graph vertices such that every graph edge has at least one endpoint in C.



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



(b) After Rule 4.



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

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

Fig. 3: Example of the algorithm for Weighted 2-Cluster Vertex Deletion when a size-2 independent set of vertices that cannot be deleted is given. Black vertices are permanent.

from the graph. Since in a d-cluster graph any set of d vertices from d different clusters forms an independent set, at least one of the independent sets of size d must be a set of vertices that remain in the graph.

Suppose that such an independent set D of size 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 Fig. 3. First, we perform the following reduction rule.

Reduction Rule 4. *Delete all vertices from the graph that are not adjacent to any vertex in D and all vertices that are adjacent to more than one vertex in D.*

The correctness of Rule 4 is obvious; an example of its application is given in Fig. 3b. 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 of Rule 4. 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: if u and v are adjacent, then remove the edge $\{u, v\}$; otherwise, insert the edge $\{u, v\}$. Furthermore, remove all permanent vertices. After this, we have obtained G' (for an example of this construction see Fig. 3c).

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 3c and 3d.

Lemma 3. *Let G be a graph with a size-d independent set of permanent vertices that is reduced with respect to Rule 4 and G' a graph constructed as described above. Then, a vertex set X is a vertex cover of G' iff X is a d-*CVD *set of G.*

We now bound the running time of computing a d-CVD set of a graph, once a size-d independent set that may not be deleted is given. It fundamentally relies on a fixed-parameter algorithm for weighted Vertex Cover [17].

Lemma 4. *Let $G = (V, E)$ be a graph and $D \subseteq V$ an independent set of size 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.*

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 $O(1.40^k \cdot k^{3d} + n^3)$.*

For the unweighted case, we can apply the current fastest algorithm for unweighted Vertex Cover by Chen et al. [5], yielding an improved running time.

Theorem 7. *Unweighted d-*Cluster Vertex Deletion *can be solved in running time $O(1.28^k \cdot k^{3d} + n^3)$.*

3.3 An $O(1.84^{k+d} + n^3)$ time algorithm

First, we apply the kernelization algorithm from Theorem 4 that produces a problem kernel consisting of $O(k^3)$ vertices. Next, we perform a search tree algorithm that branches on forbidden subgraphs. For a vertex in a forbidden subgraph, we have two choices: either we have to delete this vertex, or this vertex is one of the remaining vertices in the d-cluster graph. Whenever a vertex v is deleted, the combined parameter $k + d$ decreases by $\omega(v) \geq 1$. Furthermore, explicitly not deleting a vertex means that we assign a cluster to this vertex. Again, we call such a vertex *permanent*. If the permanent vertex does not have any neighbors that are marked as permanent, 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 removed 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 5. *If G contains a P_3 with two permanent vertices u, v and one non-permanent vertex w, then remove w from G and set $k' := k' - \omega(w)$.*

Clearly, if $k' < 1$, then we cannot remove 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 have assigned all clusters. This means that there is an independent set of d permanent vertices. By Lemma 4, 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 sketch the branching rules in case $k' \geq 1$ and $d' > 0$. After application of Reduction Rule 5, 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. Next, we branch on isolated clusters that do not contain permanent vertices. 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.

Theorem 8. *Weighted d-Cluster Vertex Deletion can be solved in running time $O(1.84^{k+d} + n^3)$.*

4 Outlook

Are there any nontrivial polynomial-time approximation algorithms for Cluster Vertex Deletion (weighted and unweighted)? 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 [7, 12], 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. Finally, all our results are worst-case estimates. Practical tests based on algorithm engineering seem promising.

Acknowledgments. We thank the anonymous *LATIN* referees for pointing out some inconsistencies in the submitted manuscript and for other comments that have improved the presentation of this paper.

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. Proofs of conjectures in "Aggregating inconsistent information: Ranking and clustering". Technical Report TR-719-05, Department of Computer Science, Princeton University, 2005.

[4] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. PEACE: Parameterized and exact algorithms for cluster editing. Manuscript, Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Sept. 2007.

[5] 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.

[6] 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.

[7] 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.

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

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

[10] 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.

[11] 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.

[12] J. Guo. A more effective linear kernelization for cluster editing. In *Proc. ESCAPE 2007*, volume 4614 of *LNCS*, pages 36–47. Springer, 2007.

[13] F. Hüffner, R. Niedermeier, and S. Wernicke. Techniques for practical fixed-parameter algorithms. *The Computer Journal*, 2007. To appear.

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

[15] 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.

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

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

[18] F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized bicluster editing. In *Proc. 2nd IWPEC*, volume 4169 of *LNCS*, pages 1–12. Springer, 2006. To appear under the title "Applying modular decomposition to parameterized cluster editing problems" in *Theory of Computing Systems.*

[19] 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.

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

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

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

[23] A. van Zuylen and D. P. Williamson. Deterministic algorithms for rank aggregation and other ranking and clustering problems. In *Proc. 5th WAOA*, LNCS. Springer, 2007. To appear.