A Fast Branching Algorithm for Cluster Vertex Deletion*

Anudhyan Boral[1], Marek Cygan[2], Tomasz Kociumaka[2], and Marcin Pilipczuk[3]

[1] Chennai Mathematical Institute, Chennai, India,
anudhyan@cmi.ac.in
[2] Institute of Informatics, University of Warsaw, Poland,
[cygan|kociumaka]@mimuw.edu.pl
[3] Department of Informatics, University of Bergen, Norway
Marcin.Pilipczuk@ii.uib.no

Abstract. In the family of clustering problems we are given a set of objects (vertices of the graph), together with some observed pairwise similarities (edges). The goal is to identify clusters of similar objects by slightly modifying the graph to obtain a cluster graph (disjoint union of cliques).
Hüffner et al. [LATIN 2008, Theory Comput. Syst. 2010] initiated the parameterized study of Cluster Vertex Deletion, where the allowed modification is vertex deletion, and presented an elegant $\mathcal{O}(2^k k^9 + nm)$-time fixed-parameter algorithm, parameterized by the solution size. In the last 5 years, this algorithm remained the fastest known algorithm for Cluster Vertex Deletion and, thanks to its simplicity, became one of the textbook examples of an application of the iterative compression principle. In our work we break the 2^k-barrier for Cluster Vertex Deletion and present an $\mathcal{O}(1.9102^k(n+m))$-time branching algorithm.

1 Introduction

The problem to cluster objects based on their pairwise similarities has arisen from applications both in computational biology [6] and machine learning [5]. In the language of graph theory, as an input we are given a graph where vertices correspond to objects, and two objects are connected by an edge if they are observed to be similar. The goal is to transform the graph into a cluster graph (a disjoint union of cliques) using a minimum number of modifications.

The set of allowed modifications depends on the particular problem variant and an application considered. Probably the most studied variant is the Cluster Editing problem, known also as Correlation Clustering, where we seek for a minimal number of edge edits to obtain a cluster graph. The study of Cluster Editing includes [3, 4, 14, 20, 31] and, from the parameterized perspective, [7–11, 15, 16, 19, 22–24, 27–29].

The main principle of parameterized complexity is that we seek algorithms that are efficient if the considered parameter is small. However, the distance

* Partially supported by NCN grant N206567140 and Foundation for Polish Science.

measure in Cluster Editing, the number of edge edits, may be quite large in practical instances, and, in the light of recent lower bounds refuting the existence of subexponential FPT algorithms for Cluster Editing [19, 27], it seems reasonable to look for other distance measures (see e.g. Komusiewicz's PhD thesis [27]) and/or different problem formulations.

In 2008, Hüffner et al. [25, 26] initiated the parameterized study of the Cluster Vertex Deletion problem (ClusterVD for short). Here, the allowed modifications are vertex deletions.

Cluster Vertex Deletion (ClusterVD) **Parameter:** k
Input: An undirected graph G and an integer k.
Question: Does there exist a set S of at most k vertices of G such that $G \setminus S$ is a cluster graph, i.e., a disjoint union of cliques?

In terms of motivation, we want to refute as few objects as possible to make the set of observations completely consistent. Since a vertex deletion removes as well all its incident edges, we may expect that this new editing measure may be significantly smaller in practical applications than the edge-editing distance.

As ClusterVD can be equivalently stated as the problem of hitting, with minimum number of vertices, all induced P_3s (paths on 3 vertices) in the input graph, ClusterVD can be solved in $\mathcal{O}(3^k(n+m))$ time by a straightforward branching algorithm [13], where n and m denote the number of vertices and edges of G, respectively. The dependency on k can be improved by considering more elaborate case distinction in the branching algorithm, either directly [21], or via a general algorithm for 3-Hitting Set [17]. Hüffner et al. [26] provided an elegant $\mathcal{O}(2^k k^9 + nm)$-time algorithm, using the iterative compression principle [30] and a reduction to the weighted maximum matching problem. This algorithm, presented at LATIN 2008 [25], quickly became one of the textbook examples of an application of the iterative compression technique.

In our work we pick up this line of research and obtain the fastest algorithm for (unweighted) ClusterVD.

Theorem 1. Cluster Vertex Deletion *can be solved in* $\mathcal{O}(1.9102^k(n+m))$ *time and polynomial space on an input* (G, k) *with* $|V(G)| = n$ *and* $|E(G)| = m$.

The source of the exponential 2^k factor in the time complexity of the algorithm of [26] comes from enumeration of all possible intersections of the solution we are looking for with the previous solution of size $(k+1)$. As the next step in each subcase is a reduction to the weighted maximum matching problem (with a definitely nontrivial polynomial-time algorithm), it seems hard to break the 2^k-barrier using the approach of [26]. Hence, in the proof of Theorem 1 we go back to the bounded search tree approach. However, to achieve the promised time bound, and at the same time avoiding very extensive case analysis, we do not follow the general 3-Hitting Set approach. Instead, our methodology is to carefully investigate the structure of the graph and an optimum solution around a vertex already guessed to be not included in the solution. We note that a somehow similar approach has been used in [26] to cope with a variant of ClusterVD where we restrict the number of clusters in the resulting graph.

More precisely, the main observation in the proof of Theorem 1 is that, if for some vertex v we know that there exists a minimum solution S not containing v, in the neighbourhood of v the CLUSTERVD problem reduces to VERTEX COVER. Let us define N_1 and N_2 to be the vertices at distance 1 and 2 from v, respectively, and define the auxiliary graph H_v to be a graph on $N_1 \cup N_2$ having an edge for each edge of G between N_1 and N_2 and for each non-edge in $G[N_1]$. In other words, two vertices are connected by an edge in H_v if, together with v, they form a P_3 in G. We observe that a minimum solution S not containing v needs to contain a vertex cover of H_v. Moreover, one can show that we may greedily choose a vertex cover with inclusion-wise maximal intersection with N_2, as deleting vertices from N_2 helps us resolve the remaining part of the graph.

Branching to find the 'correct' vertex cover of H_v is very efficient, with worst-case $(1, 2)$ (i.e., golden-ratio) branching vector. However, we do not have the vertex v beforehand, and branching to obtain such a vertex is costly. Our approach is to get as much gain as possible from the vertex cover-style branching on the graph H_v, to be able to balance the loss from some inefficient branches used to obtain the vertex v to start with. Consequently, we employ involved analysis of properties and branching algorithms for the auxiliary graph H_v.

Note that the algorithm of Theorem 1 can be pipelined with the kernelization algorithm of 3-HITTING SET [1], yielding the following corollary.

Corollary 2. CLUSTER VERTEX DELETION *can be solved in* $\mathcal{O}(1.9102^k k^4 + nm)$ *time and polynomial space on an input* (G, k) *with* $|V(G)| = n$ *and* $|E(G)| = m$.

However, due to the $\mathcal{O}(nm)$ summand in the complexity of Corollary 2, for a wide range of input instances the running time bound of Theorem 1 is better than the one of Corollary 2. In fact, the advantage of our branching approach is that the obtained dependency on the graph size in the running time is linear, whereas with the approach of [26], one needs to spend at least quadratic time either on computing weighted maximum matching or on kernelizing the instance.

In the full version [12] we also analyse the co-cluster setting, where one aims at obtaining a co-cluster graph instead of a cluster one, and show that the linear dependency on the size of the input can be maintained also in this case.

The paper is organised as follows. We give some preliminary definitions and notation in Section 2. In Section 3 we analyse the structural properties of the auxiliary graph H_v. Then, in Section 4 we prove Theorem 1, with the main tool being a subroutine branching algorithm finding all relevant vertex covers of H_v.

2 Preliminaries

We use standard graph notation. All our graphs are undirected and simple. For a graph G, by $V(G)$ and $E(G)$ we denote its vertex- and edge-set, respectively. For $v \in V(G)$, the set $N_G(v) = \{u \mid uv \in E(G)\}$ is the neighbourhood of v in G and $N_G[v] = N_G(v) \cup \{v\}$ is the closed neighbourhood. We extend these notions to sets of vertices $X \subseteq V(G)$ by $N_G[X] = \bigcup_{v \in X} N_G[v]$ and $N_G(X) = N_G[X] \setminus X$. We omit the subscript if it is clear from the context. For a set $X \subseteq V(G)$ we also define $G[X]$ to be the subgraph induced by X and $G \setminus X$ is a shorthand for $G[V(G) \setminus X]$. An even cycle is a cycle with an even number of edges, and an even path is a path with an even number of edges. A set $X \subseteq V(G)$ is called a *vertex cover* of G if $G \setminus X$ is edgeless. By $\texttt{MinV}(G)$ we denote the size of the minimum vertex cover of G.

In all further sections, we assume we are given an instance (G, k) of Cluster Vertex Deletion, where $G = (V, E)$. That is, we use V and E to denote the vertex- and edge-set of the input instance G.

A P_3 is an ordered set of 3 vertices (u, v, w) such that $uv, vw \in E$ and $uw \notin E$. A graph is a cluster graph iff it does not contain any P_3; hence, in ClusterVD we seek for a set of at most k vertices that hits all P_3s. We note also the following.

Lemma 3. *Let G be a connected graph which is not a clique. Then, for every $v \in V(G)$, there is a P_3 containing v.*

Proof. Consider $N(v)$. If there exist vertices $u, w \in N(v)$ such that $uw \notin E(G)$ then we have a P_3 (u, v, w). Otherwise, since $N[v]$ induces a clique, we must have $w \in N(N[v])$ such that $uw \in E(G)$ for some $u \in N(v)$. Thus we have a P_3, (v, u, w) involving v. □

If at some point a vertex v is fixed in the graph G, we define sets $N_1 = N_1(v)$ and $N_2 = N_2(v)$ as follows: $N_1 = N_G(v)$ and $N_2 = N_G(N_G[v])$. That is, N_1 and N_2 are sets of vertices at distance 1 and 2 from v, respectively. For a fixed $v \in V$, we define an auxiliary graph H_v with $V(H_v) = N_1 \cup N_2$ and

$$E(H_v) = \{uw \mid u, w \in N_1, uw \notin E\} \cup \{uw \mid u \in N_1, w \in N_2, uw \in E\}.$$

Thus, H_v consists of the vertices in N_1 and N_2 along with non-edges among vertices of N_1 and edges between N_1 and N_2. Note that N_2 is an independent set in H_v. Observe the following.

Lemma 4. *For $u, w \in N_1 \cup N_2$, we have $uw \in E(H_v)$ iff u, w and v form a P_3 in G.*

Proof. For every $uw \in E(H_v)$ with $u, w \in N_1$, (u, v, w) is a P_3 in G. For $uw \in E(H_v)$ with $u \in N_1$ and $w \in N_2$, (v, u, w) forms a P_3 in G. In the other direction, for any P_3 in G of the form (u, v, w) we have $u, w \in N_1$ and $uw \notin E$, thus $uw \in E(H_v)$. Finally, for any P_3 in G of the form (v, u, w) we have $u \in N_1$, $w \in N_2$ and $uw \in E$, hence $uw \in E(H_v)$. □

We call a subset $S \subseteq V$ a *solution* when $G \setminus S$ is a cluster graph, that is, a collection of disjoint cliques. A solution with minimal cardinality is called a *minimum solution*.

Our algorithm is a typical branching algorithm, that is, it consists of a number of *branching steps*. In a step $(A_1, A_2, \ldots, A_r)$, $A_1, A_2, \ldots, A_r \subseteq V$, we independently consider r subcases. In the i-th subcase we look for a minimum solution S containing A_i: we delete A_i from the graph and decrease the parameter k

by $|A_i|$. If k becomes negative, we terminate the current branch and return a negative answer from the current subcase.

The *branching vector* for a step $(A_1, A_2, \ldots, A_r)$ is $(|A_1|, |A_2|, \ldots, |A_r|)$. It is well-known (see e.g. [18]) that the number of final subcases of a branching algorithm is bounded by $\mathcal{O}(c^k)$, where c is the largest positive root of the equation $1 = \sum_{i=1}^{r} x^{-|A_i|}$ among all branching steps $(A_1, A_2, \ldots, A_r)$ in the algorithm.

At some places, the algorithm makes a greedy (but optimal) choice of including a set $A \subseteq V$ into the constructed solution. We formally treat it as length-one branching step (A) with branching vector $(|A|)$.

3 The auxiliary graph H_v

In this section we investigate properties of the auxiliary graph H_v. Hence, we assume that a ClusterVD input (G, k) is given with $G = (V, E)$, and a vertex $v \in V$ is fixed.

3.1 Basic properties

First, note that an immediate consequence of Lemma 4 is the following.

Corollary 5. *Let S be a solution such that $v \notin S$. Then S contains a vertex cover of H_v.*

In the other direction, the following holds.

Lemma 6. *Let X be a vertex cover of H_v. Then, in $G \setminus X$, the connected component of v is a clique.*

Proof. Suppose the connected component of v in $G \setminus X$ is not a clique. Then by Lemma 3, there is a P_3 involving v. Such a P_3 is also present in G. However, by Lemma 4, as X is a vertex cover of H_v, X intersects such a P_3, a contradiction. □

Lemma 7. *Let S be a solution such that $v \notin S$. Denote by X the set $S \cap V(H_v)$. Let Y be a vertex cover of H_v. Suppose that $X \cap N_2 \subseteq Y \cap N_2$. Then $T := (S \setminus X) \cup Y$ is also a solution.*

Proof. Since Y (and hence, $T \cap V(H_v)$) is a vertex cover of H_v and $v \notin T$, we know by Lemma 6 that the connected component of v in $G \setminus T$ is a clique. If T is not a solution, then there must be a P_3 contained in $Z \setminus T$, where $Z = V \setminus (\{v\} \cup N_1)$. But since $S \cap Z \subseteq T \cap Z$, $G \setminus S$ would also contain such a P_3. □

Lemma 7 motivates the following definition. For vertex covers of H_v, X and Y, we say that Y *dominates* X if $|Y| \leq |X|$, $Y \cap N_2 \supseteq X \cap N_2$ and at least one of these inequalities is sharp. Two vertex covers X and Y are said to be *equivalent* if $X \cap N_2 = Y \cap N_2$ and $|X \cap N_1| = |Y \cap N_1|$. We note that the first aforementioned relation is transitive and strongly anti-symmetric, whereas the second is an equivalence relation.

As a corollary of Lemma 7, we have:

Corollary 8. *Let S be a solution such that $v \notin S$. Suppose Y is a vertex cover of H_v which either dominates or is equivalent to the vertex cover $X = S \cap V(H_v)$. Then $T := (S \setminus X) \cup Y$ is also a solution with $|T| \leq |S|$.*

3.2 Special cases of H_v

We now carefully study the cases where H_v has small vertex cover or has a special structure, and discover some possible greedy decisions that can be made.

Lemma 9. *Suppose X is a vertex cover of H_v. Then there is a minimum solution S such that either $v \notin S$ or $|X \setminus S| \geq 2$.*

Proof. Suppose S is a minimum solution such that $v \in S$ and $|X \setminus S| \leq 1$. We are going to convert S to another minimum solution T that does not contain v.

Consider $T := (S \setminus \{v\}) \cup X$. Clearly, $|T| \leq |S|$. Since T contains X, a vertex cover, by Lemma 6, the connected component of v in $G \setminus T$ is a clique. Thus, there is no P_3 containing v. Since any P_3 in $G \setminus T$ which does not include v must also be contained in $G \setminus S$, contradicting the fact that S is a solution, we obtain that T is also a solution. Hence, T is a minimum solution. □

Corollary 10. *If $\texttt{MinV}(H_v) = 1$ then there is a minimum solution S not containing v.*

Lemma 11. *Let C be the connected component of G containing v, and assume that neither C nor $C \setminus \{v\}$ is a cluster graph. If $X = \{w_1, w_2\}$ is a minimum vertex cover of H_v, then there exists a connected component $\widehat{C}$ of $G \setminus \{v\}$ that is not a clique and $\widehat{C} \cap \{w_1, w_2\} \neq \emptyset$.*

Proof. Assume the contrary. Consider a component $\widehat{C}$ of $C \setminus \{v\}$ which is not a clique. Since v must be adjacent to each connected component of $C \setminus \{v\}$, $\widehat{C} \cap N_1$ must be non-empty. For any $w \in \widehat{C} \cap N_1$, we have that $w_1, w_2 \neq w$ and $ww_1, ww_2 \notin E(G)$, since otherwise the result follows. If $uw \in E(G)$ with $u \in N_2$, then, as $\{w_1, w_2\}$ is a vertex cover of H_v we must have $u = w_1$ or $u = w_2$. We would then have w_1 or w_2 contained in a non-clique $\widehat{C}$, contradicting our assumption. Hence $uw \in E(G) \Rightarrow u \in N_1$. Thus $\widehat{C} \subseteq N_1$. As w_1 and w_2 are not contained in $\widehat{C}$ and they cover all edges in H_v, $\widehat{C}$ must be an independent set in H_v. In $G \setminus \{v\}$, therefore, $\widehat{C}$ must be a clique, a contradiction. □

We now investigate the case when H_v has a very specific structure. The motivation for this analysis will become clear in Section 4.3.

A *seagull* is a connected component of H_v that is isomorphic to a P_3 with middle vertex in N_1 and endpoints in N_2. The graph H_v is called an *s-skein* if it is a disjoint union of s seagulls and some isolated vertices.

Lemma 12. *Let $v \in V$. Suppose that H_v is an s-skein. Then there is a minimum solution S such that $v \notin S$.*

Proof. Let H_v consist of seagulls $(x_1, y_1, z_1), (x_2, y_2, z_2), \ldots, (x_s, y_s, z_s)$. That is, the middle vertices y_i are in N_1, while the endpoints x_i and z_i are in N_2. If $s = 1$, $\{y_1\}$ is a vertex cover of H_v and Corollary 10 yields the result. Henceforth, we assume $s \geq 2$.

Let X be the set N_1 with all the vertices isolated in H_v removed. Clearly, X is a vertex cover of H_v. Thus, we may use X as in Lemma 9 and obtain a minimum solution S. If $v \notin S$ we are done, so let us assume $|X \setminus S| \geq 2$. Take arbitrary i such that $y_i \in X \setminus S$. As $|X \setminus S| \geq 2$, we may pick another $j \neq i$, $y_j \in X \setminus S$. The crucial observation from the definition of H_v is that (y_j, y_i, x_i) and (y_j, y_i, z_i) are P_3s in G. As $y_i, y_j \notin S$, we have $x_i, z_i \in S$. Hence, since the choice of i was arbitrary, we infer that for each $1 \leq i \leq s$ either $y_i \in S$ or $x_i, z_i \in S$, and, consequently, S contains a vertex cover of H_v. By Lemma 6, $S \setminus \{v\}$ is also a solution in G, a contradiction. □

4 Algorithm

In this section we show our algorithm for ClusterVD, proving Theorem 1. The algorithm is a typical branching algorithm, where at each step we choose one branching rule and apply it. In each subcase, a number of vertices is deleted, and the parameter k drops by this number. If k becomes negative, the current subcase is terminated with a negative answer. On the other hand, if k is non-negative and G is a cluster graph, the vertices deleted in this subcase form a solution of size at most k.

4.1 Preprocessing

At each step, we first preprocess simple connected components of G.

Lemma 13. *For each connected component C of G, in linear time, we can:*

1. *conclude that C is a clique; or*
2. *conclude that C is not a clique, but identify a vertex w such that $C \setminus \{w\}$ is a cluster graph; or*
3. *conclude that none of the above holds.*

Proof. On each connected component C, we perform a depth-first search. At every stage, we ensure that the set of already marked vertices induces a clique.

When we enter a new vertex, w, adjacent to a marked vertex v, we attempt to maintain the above invariant. We check if the number of marked vertices is equal to the number neighbours of w which are marked; if so then the new vertex w is marked. Since w is adjacent to every marked vertex, the set of marked vertices remains a clique. Otherwise, there is a marked vertex u such that $uw \notin E(G)$, and we may discover it by iterating once again over edges incident to w. In this case, we have discovered a P_3 (u, v, w) and C is not a clique. At least one of u, v, w must be deleted to make C into a cluster graph. We delete each one of them, and repeat the algorithm (without further recursion) to check if the remaining graph is a cluster graph. If one of the three possibilities returns a cluster graph, then (2) holds. Otherwise, (3) holds.

If we have marked all vertices in a component C while maintaining the invariant that marked vertices form a clique, then the component C is a clique. $\square$

For each connected component C that is a clique, we disregard C. For each connected component C that is not a clique, but $C \setminus \{w\}$ is a cluster graph for some w, we may greedily delete w from G: we need to delete at least one vertex from C, and w hits all P_3s in C. Thus, henceforth we assume that for each connected component C of G and for each $v \in V(C)$, $C \setminus \{v\}$ is not a cluster graph. In other words, we assume that we need to delete at least two vertices to solve each connected component of G.

4.2 Accessing H_v in linear time

Let us now fix a vertex $v \in V$ and let C be its connected component in G. Note that, as H_v contains parts of the complement of G, it may have size superlinear in the size of G. Therefore we now develop a simple oracle access to H_v that allows us to claim linear dependency on the graph size in the time bound.

Lemma 14. *Given a designated vertex $v \in V$, one can in linear time either compute a vertex w of degree at least 3 in H_v, together with its neighbourhood in H_v, or explicitly construct the graph H_v.*

Proof. First, mark vertices of N_1 and N_2. Second, for each vertex of G compute its number of neighbours in N_1 and N_2. This information, together with $|N_1|$, suffices to compute degrees of vertices in H_v. Hence, we may identify a vertex of degree at least 3 in H_v, if it exists. For such a vertex w, computing $N_{H_v}(w)$ takes time linear in the size of G. If no such vertex w exists, the complement of $G[N_1]$ has size linear in $|N_1|$ and we may construct H_v in linear time in a straightforward manner. $\square$

In the algorithm of Theorem 1, we would like to make a decision depending on the size of the minimum vertex cover of H_v. By the preprocessing step, C is not a clique, and by Lemma 3, H_v contains at least one edge, thus $\texttt{MinV}(G) \geq 1$. We now note that we can find a small vertex cover of G in linear time.

Lemma 15. *In linear time, we can determine whether H_v has a minimum vertex cover of size 1, of size 2, or of size at least 3. Moreover, in the first two cases we can find the vertex cover in the same time bound.*

Proof. We use Lemma 14 to find, in linear time, a vertex w with degree at least 3, or generate H_v explicitly. In the latter case, H_v has vertices of degree at most 2, and it is straightforward to compute its minimum vertex cover in linear time.

If we find a vertex w of degree at least 3 in H_v, then w must be in any vertex cover of size at most 2. We proceed to delete w and restart the algorithm of Lemma 14 on the remaining graph to check if H_v in $G \setminus w$ has a vertex cover of size 0 or 1. We perform at most 2 such restarts. Finally, if we do not find a vertex cover of size at most 2, it must be the case that the minimum vertex cover contains at least 3 vertices. $\square$

4.3 Subroutine: branching on H_v

We are now ready to present a branching algorithm that guesses the 'correct' vertex cover of H_v, for a fixed vertex v. That is, we are now working in the setting where we look for a minimum solution to ClusterVD on (G, k) not containing v, thus, by Corollary 5, containing a vertex cover of H_v. Our goal is to branch into a number of subcases, in each subcase picking a vertex cover of H_v. By Corollary 8, our branching algorithm, to be correct, needs only to generate at least one element from each equivalence class of the 'equivalent' relation, among maximal elements in the 'dominate' relation.

The algorithm consists of a number of branching steps; in each subcase of each step we take a number of vertices into the constructed vertex cover of H_v and, consequently, into the constructed minimum solution to ClusterVD on G. At any point, the first applicable rule is applied.

First, we disregard isolated vertices in H_v. Second, we take care of large-degree vertices.

Rule 1 *If there is a vertex $u \in V(H_v)$ with degree at least 3 in H_v, include either u or $N_{H_v}(u)$ into the vertex cover. That is, use the branching step $(u, N_{H_v}(u))$.*

Note that Rule 1 yields a branching vector $(1, d)$, where $d \geq 3$ is the degree of u in H_v. Henceforth, we can assume that vertices have degree 1 or 2 in H_v. Assume there exists $u \in N_1$ of degree 1, with $uw \in E(H_v)$. Moreover, assume there exists a minimum solution S containing u. If $w \in S$, then, by Lemma 7, $S \setminus \{u\}$ is also a solution, a contradiction. If $w \in N_2 \setminus S$, then $(S \setminus \{u\}) \cup \{w\}$ dominates S. Finally, if $w \in N_1 \setminus S$, then $(S \setminus \{u\}) \cup \{w\}$ is equivalent to S. Hence, we infer the following greedy rule.

Rule 2 *If there is a vertex $u \in N_1$ of degree 1 in H_v, include $N_{H_v}(u)$ into the vertex cover without branching. (Formally, use the branching step $(N_{H_v}(u))$.)*

Now we assume vertices in N_1 are of degree exactly 2 in H_v. Suppose we have vertices $u, w \in N_1$ with $uw \in E(H_v)$. We would like to branch on u as in Rule 1, including either u or $N_{H_v}(u)$ into the vertex cover. However, note that in the case where u is deleted, Rule 2 is triggered on w and consequently the other neighbour of w is deleted. Hence, we infer the following rule.

Rule 3 *If there are vertices $u, w \in N_1$, $uw \in E(H_v)$ then include either $N_{H_v}(w)$ or $N_{H_v}(u)$ into the vertex cover, that is, use the branching step $(N_{H_v}(w), N_{H_v}(u))$.*

Note that Rule 3 yields the branching vector $(2, 2)$.

We are left with the case where the maximum degree of H_v is 2, there are no edges with both endpoints in N_1, and no vertices of degree one in N_1. Hence H_v must be a collection of even cycles and even paths (recall that N_2 is an independent set in H_v). On each such cycle C, of $2l$ vertices, the vertices of N_1 and N_2 alternate. Note that we must use at least l vertices for the vertex cover of C. By Lemma 7 it is optimal to greedily select the l vertices in $C \cap N_2$.

Rule 4 *If there is an even cycle C in H_v with every second vertex in N_2, include $C \cap N_2$ into the vertex cover without branching. (Formally, use the branching step $(C \cap N_2)$.)*

For an even path P of length $2l$, we have two choices. If we are allowed to use $l+1$ vertices in the vertex cover of P, then, by Lemma 7, we may greedily take $P \cap N_2$. If we may use only l vertices, the minimum possible number, we need to choose $P \cap N_1$, as it is the unique vertex cover of size l of such path. Hence, we have an $(l, l+1)$ branch with our last rule.

Rule 5 *Take the longest possible even path P in H_v and either include $P \cap N_1$ or $P \cap N_2$ into the vertex cover. That is, use the branching step $(P \cap N_1, P \cap N_2)$.*

In Rule 5, we pick the longest possible path to avoid the branching vector $(1, 2)$ as long as possible; this is the worst branching vector in the algorithm of this section. Moreover, note that if we are forced to use the $(1, 2)$ branch, the graph H_v has a very specific structure.

Lemma 16. *If the algorithm of Section 4.3 may only use a branch with the branching vector $(1, 2)$, then H_v is an s-skein for some $s \geq 1$.*

We note that the statement of Lemma 16 is our sole motivation for introducing the notion of skeins and proving their properties in Lemma 12.

We conclude this section with an observation that the oracle access to H_v given by Lemma 14 allows us to execute a single branching step in linear time.

4.4 Main algorithm

We are now ready to present our algorithm for Theorem 1. We assume the preprocessing (Lemma 13) is done. Pick an arbitrary vertex v. We first run the algorithm of Lemma 15 to determine if H_v has a small minimum vertex cover. Then we run the algorithm of Lemma 14 to check if H_v is an s-skein for some s.

We consider the following cases.

1. $\texttt{MinV}(H_v) = 1$ **or** H_v **is an** s**-skein for some** s**.** Then, by Corollary 10 and Lemma 12, we know there exists a minimum solution not containing v. Hence, we run the algorithm of Section 4.3 on H_v.
2. $\texttt{MinV}(H_v) = 2$ **and** H_v **is not a 2-skein.**[4] Assume the application of Lemma 15 returned a vertex cover $X = \{w_1, w_2\}$ of H_v. By Lemma 9, we may branch into the following two subcases: in the first we look for minimum solutions containing v and disjoint with X, and in the second, for minimum solutions not containing v.

   In the first case, we first delete v from the graph and decrease k by one. Then we check whether the connected component containing w_1 or w_2 is not a clique. By Lemma 11, for some $w \in \{w_1, w_2\}$, the connected component

[4] Note that the size of a minimum vertex cover of an s-skein is exactly s, so this case is equivalent to '$\texttt{MinV}(H_v) = 2$ and H_v is not an s-skein for any s'.

of $G \setminus \{v\}$ containing w is not a clique; finding such w clearly takes linear time. We invoke the algorithm of Section 4.3 on H_w.
In the second case, we invoke the algorithm of Section 4.3 on H_v.
3. $\mathtt{MinV}(H_v) \geq 3$ **and** H_v **is not an** s**-skein for any** $s \geq 3$**.** We branch into two cases: we look for a minimum solution containing v or not containing v. In the first branch, we simply delete v and decrease k by one. In the second branch, we invoke the algorithm of Section 4.3 on H_v.

4.5 Complexity analysis

In the previous discussion we have argued that invoking each branching step takes linear time. As in each branch we decrease the parameter k by at least one, the depth of the recursion is at most k. In this section we analyse branching vectors occurring in our algorithm. To finish the proof of Theorem 1 we need to show that the largest positive root of the equation $1 = \sum_{i=1}^{r} x^{-a_i}$ among all possible branching vectors $(a_1, a_2, \ldots, a_r)$ is strictly less than 1.9102.

As the number of resulting branching vectors in the analysis is rather large, we use a Python script for automated analysis[5]. The main reason for a large number of branching vectors is that we need to analyse branchings on the graph H_v in case when we consider v not to be included in the solution. Let us now proceed with formal arguments.

Analysis of the algorithm of Section 4.3. In a few places, the algorithm of Section 4.3 is invoked on the graph H_v and we know that $\mathtt{MinV}(H_v) \geq h$ for some $h \in \{1, 2, 3\}$. Consider the branching tree $\mathbb{T}$ of this algorithm. For a node $x \in V(\mathbb{T})$, the *depth* of x is the number of vertices of H_v deleted on the path from x to the root. We mark some nodes of $\mathbb{T}$. Each node of depth less than h is marked. Moreover, if a node x is of depth $d < h$ and the branching step at node x has branching vector $(1, 2)$, we infer that graph H_v at this node is an s-skein for some $s \geq h - d$, all descendants of x in $V(\mathbb{T})$ are also nodes with branching steps with vectors $(1, 2)$. In this case, we mark all descendants of x that are within distance (in $\mathbb{T}$) less than $h - d$. Note that in this way we may mark some descendants of x of depth equal or larger than h.

We split the analysis of an application of the algorithm of Section 4.3 into two phases: the first one contains all branching steps performed on marked nodes, and the second on the remaining nodes. In the second phase, we simply observe that each branching step has branching vector not worse than $(1, 2)$. In the first phase, we aim to write a single branching vector summarizing the phase, so that with its help we can balance the loss from other branches when v is deleted.

We remark that, although in the analysis we aggregate some branching steps to prove better time bound, we always aggregate only a constant number of branches (that is, we analyse the branching on marked vertices only for constant

[5] Available at `http://www.mimuw.edu.pl/~malcin/research/cvd` and in the full version [12].

h). Consequently, we maintain a linear dependency on the size of the graph in the running time bound.

The main property of the marked nodes in $\mathbb{T}$ is that their existence is granted by the assumption $\mathtt{MinV}(H_v) \geq h$. That is, each leaf of $\mathbb{T}$ has depth at least h, and, if at some node x of depth $d < h$ the graph H_v is an s-skein, we infer that $s \geq h - d$ (as the size of minimum vertex cover of an s-skein is s) and the algorithm performs s independent branching steps with branching vectors $(1, 2)$ in this case. Overall, no leaf of $\mathbb{T}$ is marked.

To analyse such branchings for $h = 2$ and $h = 3$ we employ the Python script. The procedure `branch_Hv` generates all possible branching vectors for the first phase, assuming the algorithm of Section 4.3 is allowed to pick branching vectors (1), $(1, 3)$, $(2, 2)$ or $(1, 2)$ (option `allow_skein` enables/disables the use of the $(1, 2)$ vector in the first branch). Note that all other vectors described in Section 4.3 may be simulated by applying a number of vectors (1) after one of the aforementioned branching vectors.

Analysis of the algorithm of Section 4.4.

Case 1. Here the algorithm of Section 4.3 performs branchings with vectors not worse than $(1, 2)$.

Case 2. If v is deleted, we apply the algorithm of Section 4.3 to H_w, yielding at least one branching step (as the connected component with w is not a clique). Hence, in this case the outcoming branching vector is any vector that came out of the algorithm of Section 4.3, with all entries increased by one (for the deletion of v). Recall that in the algorithm of Section 4.3, the worst branching vector is $(1, 2)$, corresponding to the case of H_w being a skein. Consequently, the words branching vector if v is deleted is $(2, 3)$.

If v is not deleted, the algorithm of Section 4.3 is applied to H_v. The script invokes the procedure `branch_Hv` on $h = 2$ and `allow_skein=False` to obtain a list of possible branching vectors. For each such vector, we append entries $(2, 3)$ from the subcase when v is deleted.

Case 3. The situation is analogous to the previous case. The script invokes the procedure `branch_Hv` on $h = 3$ and `allow_skein=False` to obtain a list of possible branching vectors. For each such vector, we append the entry (1) from the subcase when v is deleted.

Summary. We infer that the largest root of the equation $1 = \sum_{i=1}^{r} x^{-a_i}$ occurs for branching vector $(1, 3, 3, 4, 4, 5)$ and is less than 1.9102. This branching vector corresponds to Case 3 and the algorithm of Section 4.3, invoked on H_v, first performs a branching step with the vector $(1, 3)$ and in the branch with 1 deleted vertex, finds H_v to be a 2-skein and performs two independent branching steps with vectors $(1, 2)$.

This analysis concludes the proof of Theorem 1. We remark that the worst branching vector in Case 2 is $(2, 2, 3, 3, 3)$ (with solution $x < 1.8933$), corresponding to the case with single $(1, 2)$-branch when v is deleted and a 2-skein in the case when v is kept. Obviously, the worst case in Case 1 is the golden-ratio branch $(1, 2)$ with solution $x < 1.6181$.

5 Conclusions and open problems

We have presented a new branching algorithm for Cluster Vertex Deletion. We hope our work will trigger a race for faster FPT algorithms for ClusterVD, as it was in the case of the famous Vertex Cover problem.

Repeating after Hüffner et al. [26], we would like to re-pose here the question for a linear vertex-kernel for ClusterVD. As ClusterVD is a special case of the 3-Hitting Set problem, it admits an $\mathcal{O}(k^2)$-vertex kernel in the unweighted case and an $\mathcal{O}(k^3)$-vertex kernel in the weighted one [1, 2]. However, Cluster Editing is known to admit a much smaller $2k$-vertex kernel, so there is a hope for a similar result for ClusterVD.

References

1. F. N. Abu-Khzam. A kernelization algorithm for d-hitting set. *Journal of Computer and System Sciences*, 76(7):524–531, 2010.
2. F. N. Abu-Khzam and H. Fernau. Kernels: Annotated, proper and induced. In *Proceedings of IWPEC 2006*, volume 4169 of *Lecture Notes in Computer Science*, pages 264–275. Springer, 2006.
3. N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: Ranking and clustering. *Journal of the ACM*, 55(5):23:1–23:27, 2008.
4. N. Alon, K. Makarychev, Y. Makarychev, and A. Naor. Quadratic forms on graphs. In *Proceedings of STOC 2005*, pages 486–493. ACM, 2005.
5. N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56:89–113, 2004.
6. A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–297, 1999.
7. S. Böcker. A golden ratio parameterized algorithm for cluster editing. *Journal of Discrete Algorithms*, 16:79–89, 2012.
8. S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009.
9. S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.
10. S. Böcker and P. Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011.
11. H. L. Bodlaender, M. R. Fellows, P. Heggernes, F. Mancini, C. Papadopoulos, and F. A. Rosamond. Clustering with partial information. *Theoretical Computer Science*, 411(7-9):1202–1211, 2010.
12. A. Boral, M. Cygan, T. Kociumaka, and M. Pilipczuk. Fast branching algorithm for cluster vertex deletion. *CoRR*, abs/1306.3877, 2013.

13. L. Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996.
14. M. Charikar and A. Wirth. Maximizing quadratic programs: Extending Grothendieck's inequality. In *Proceedings of FOCS 2004*, pages 54–60. IEEE Computer Society, 2004.
15. P. Damaschke. Fixed-parameter enumerability of cluster editing and related problems. *Theory of Computing Systems*, 46(2):261–283, 2010.
16. M. R. Fellows, J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011.
17. H. Fernau. A top-down approach to search-trees: Improved algorithmics for 3-hitting set. *Algorithmica*, 57(1):97–118, 2010.
18. F. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in theoretical computer science. Springer Berlin Heidelberg, 2010.
19. F. V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and Y. Villanger. Tight bounds for parameterized complexity of cluster editing. In *Proceedings of STACS 2013*, volume 20 of *LIPIcs*, pages 32–43. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
20. I. Giotis and V. Guruswami. Correlation clustering with a fixed number of clusters. *Theory of Computing*, 2(1):249–266, 2006.
21. 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.
22. 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.
23. J. Guo, I. A. Kanj, C. Komusiewicz, and J. Uhlmann. Editing graphs into disjoint unions of dense clusters. *Algorithmica*, 61(4):949–970, 2011.
24. J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. A more relaxed model for graph-based data clustering: s-plex cluster editing. *SIAM Journal of Discrete Mathematics*, 24(4):1662–1683, 2010.
25. F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. In *Proceedings of LATIN 2008*, volume 4957 of *Lecture Notes in Computer Science*, pages 711–722. Springer, 2008.
26. F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010.
27. C. Komusiewicz. *Parameterized Algorithmics for Network Analysis: Clustering & Querying*. PhD thesis, Technische Universität Berlin, 2011. Available at `http://fpt.akt.tu-berlin.de/publications/diss-komusiewicz.pdf`.
28. C. Komusiewicz and J. Uhlmann. Alternative parameterizations for cluster editing. In *Proceedings of SOFSEM 2011*, volume 6543 of *Lecture Notes in Computer Science*, pages 344–355. Springer, 2011.
29. F. Protti, M. D. da Silva, and J. L. Szwarcfiter. Applying modular decomposition to parameterized cluster editing problems. *Theory of Computing Systems*, 44(1):91–104, 2009.
30. B. A. Reed, K. Smith, and A. Vetta. Finding odd cycle transversals. *Operations Research Letters*, 32(4):299–301, 2004.
31. R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.