Cluster Editing with Overlapping Communities

Emmanuel Arrighi
University of Trier, Germany

Matthias Bentert
University of Bergen, Norway

Pål Grønås Drange[1]
University of Bergen, Norway

Blair D. Sullivan
University of Utah, Salt Lake City, UT, USA

Petra Wolf
University of Bergen, Norway

Abstract

Cluster Editing, also known as correlation clustering, is a well-studied graph modification problem. In this problem, one is given a graph and allowed to perform up to k edge additions and deletions to transform it into a cluster graph, i.e., a graph consisting of a disjoint union of cliques. However, in real-world networks, clusters are often overlapping. For example, in social networks, a person might belong to several communities – e.g. those corresponding to work, school, or neighborhood. Another strong motivation comes from language networks where trying to cluster words with similar usage can be confounded by homonyms, that is, words with multiple meanings like "bat". The recently introduced operation of *vertex splitting* is one natural approach to incorporating such overlap into Cluster Editing. First used in the context of graph drawing, this operation allows a vertex v to be replaced by two vertices whose combined neighborhood is the neighborhood of v (and thus v can belong to more than one cluster). The problem of transforming a graph into a cluster graph using at most k edge additions, edge deletions, or vertex splits is called Cluster Editing with Vertex Splitting and is known to admit a polynomial kernel with respect to k and an $O(9^{k^2} + n + m)$-time (parameterized) algorithm. However, it was not known whether the problem is NP-hard, a question which was originally asked by Abu-Khzam et al. [Combinatorial Optimization, 2018]. We answer this in the affirmative. We further give an improved algorithm running in $O(2^{7k \log k} + n + m)$ time.

2012 ACM Subject Classification Mathematics of computing → Graph algorithms

Keywords and phrases graph modification, correlation clustering, vertex splitting, NP-hardness, parameterized algorithm

Digital Object Identifier 10.4230/LIPIcs.IPEC.2023.2

Funding *Matthias Bentert*: European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 819416).
Pål Grønås Drange: Research Council of Norway under grant Parameterized Complexity for Practical Computing (NFR, no. 274526).
Blair D. Sullivan: Gordon & Betty Moore Foundation under grant GBMF4560.

1 Introduction

Correlation clustering is a fundamental problem in data mining and machine learning that aims to identify groups of similar objects based on their pairwise similarity or dissimilarity. This problem arises in various domains, including social network analysis, image segmentation,

[1] corresponding author

document classification, gene expression analysis, and many more. Broadly speaking, the goal of correlation clustering is to partition the objects into clusters such that objects within the same cluster are more similar to each other than to objects in different clusters.

Correlation clustering has also been studied in algorithmic graph theory [34]. In this setting, the input is a graph and two vertices share an edge if they are similar. The goal is then to add or remove a minimum number of edges such that the resulting graph is a *cluster graph*, that is, a disjoint unions of cliques. This problem is typically known as Cluster Editing and formally defined as follows.

Cluster Editing

Input:	A graph $G = (V, E)$ and an integer k.
Question:	Does there exist a set $F \subseteq \binom{V}{2}$ of size at most k such that $G = (V, E \triangle F)$ is a cluster graph?

Here, $\triangle$ is the symmetric difference between two sets. Equivalently, one can ask whether there is a sequence $(\sigma_1, \sigma_2, \ldots, \sigma_\ell)$ of length $\ell \leq k$ where each σ_i describes either an edge addition or an edge removal between two vertices such that performing all operations in the sequence yields a cluster graph.

Cluster Editing has been extensively studied in terms of both classic and parameterized complexity. Cai [9] showed a general result implying an $O(3^k \cdot n^3)$-time algorithm for Cluster Editing. This running time has since been improved several times [23, 25, 29]. Moreover, it is known that Cluster Editing admits a polynomial kernel with respect to k [21].

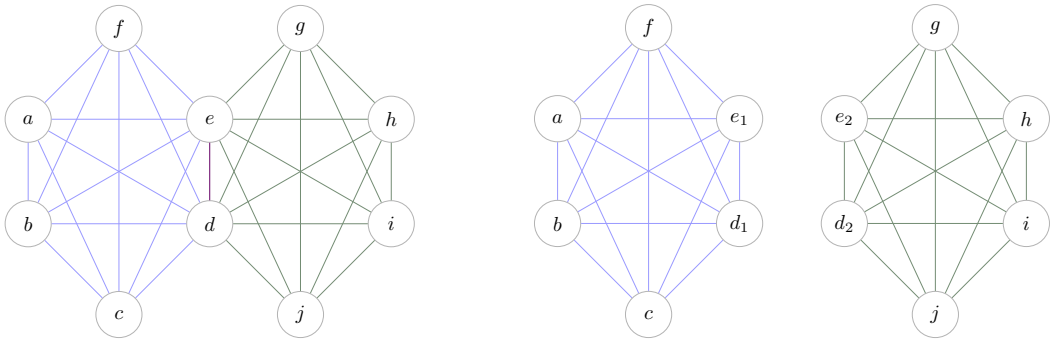
However, it has been observed that real-world data usually does not neatly conform to such an equivalence relation [27, 36, 40]. This lead to the study of detecting so-called *overlapping communities* [5, 31], with applications in categorizing videos [35], classifying emotions in music [41] and sentiments in online posts [33], document classification [18, 32], and protein clustering based on amino-acid sequences [8].

The overlapping clusters model we study was introduced by Abu-Khzam et al. [2], who augmented Cluster Editing with a *vertex-splitting* operation which enables vertices to belong to multiple clusters. Specifically, in addition to allowing edge modification, a vertex v can be *split* into two vertices v_1, v_2 such that $N(v_1) \cup N(v_2) = N(v)$, where N is the (open) neighborhood of a vertex (see Figure 1 for an example). In the same article [2], they show that this generalized problem admits a quadratic vertex kernel and an FPT algorithm. A slightly tighter analysis than the paper provides reveals that the algorithm runs in $O(9^{k^2} + n + m)$ time. Abu-Khzam et al. [1] later provided a greedy algorithm, but it was still not known whether the problem is actually NP-hard, a question posed by the former set of authors. We settle this question by showing that the following problem is indeed NP-complete.

Cluster Editing with Vertex Splitting (CEVS)

Input:	A graph $G = (V, E)$ and an integer k.
Question:	Does there exist a sequence $\sigma = (\sigma_1, \sigma_2, \ldots, \sigma_\ell)$ of length $\ell \leq k$, where each σ_i is an edge addition, an edge removal, or a vertex splitting, such that performing the operations in σ turns G into a cluster graph?

Related work. In addition to the aforementioned work [1, 2], overlapping clustering has been studied from the graph-editing point of view before, but under a less natural restriction. Fellows et al. [16] consider the problem where each vertex (and edge) is allowed to participate in a *fixed number* of clusters. In real-world applications, however, there is no fixed number s such that *every* vertex belongs to at most s cliques. Indeed, some vertices naturally belong to many clusters and some to few.



(a) A graph with two overlapping clusters that can be separated by two vertex splits.

(b) The graph after splitting d into d_1 and d_2 and e into e_1 and e_2.

Figure 1 An example of an instance where the budget needed for Cluster Editing is 8, and where the resulting graph is quite different from the input graph (d has to be disconnected completely from (e.g.) g, h, i, and j). In CEVS, we only need a budget of 2, and the resulting graph is very similar to the input graph.

To the best of our knowledge, the first use of vertex splitting in graph modification is the Planar Vertex Splitting used in graph drawing [15], a problem shown to be FPT quite recently by Nöllenburg et al. [30]. Even more recently, an FPT algorithm was given for obtaining a 2-layer planar drawing of a graph after at most k vertex splits [3]. In addition, vertex splitting has also been studied in the context of reducing a graph's pathwidth by Baumann, Pfretzschner, and Rutter [6], who also show that the problem Π-Vertex Splitting is definable in MSO_2 (provided that Π is), making the problem FPT for graphs of bounded treewidth.

Finally, we would also like to highlight that finding ever faster algorithms for overlapping clustering and community detection is a major area of study in complex networks [4, 7, 11, 19, 20, 39, 43].

2 Preliminaries

For a positive integer n, we use $[n] = \{1, 2, \ldots, n\}$ to denote the set of all positive integers up to n. All logarithms in this paper use 2 as their base.

We use standard graph-theoretic notation and refer the reader to the textbook by Diestel [12] for commonly used definitions. For an introduction to parameterized complexity, fixed-parameter tractability, and kernelization, we refer the reader to the textbooks by Flum and Grohe [17] and Cygan et al. [10]. The only non-standard graph concept used in this work are critical cliques as introduced by Lin et al. [28].

▶ **Definition 1.** *A* critical clique *is a subset of vertices C that is maximal with the properties that*

1. *C is a clique*
2. *there exists $U \subseteq V(G)$ s.t. $N[v] = U$ for all $v \in C$.*

▶ **Lemma 2** ([2, 28])**.** *Every vertex appears in exactly one critical clique.*

We denote the critical clique in which a vertex v appears by $CC(v)$. The *critical clique quotient graph* $\mathcal{C}$ of G contains a vertex for each critical clique in G and two vertices are adjacent if and only if the two respective critical cliques C_1 and C_2 are adjacent, that is,

there is an edge between each vertex in C_1 and each vertex in C_2. Note that by the definition of critical cliques this is equivalent to the condition that at least one edge $\{u, v\}$ with $u \in C_1$ and $v \in C_2$ exists. The main reason that critical cliques turn out to be useful when studying CEVS is captured by the following lemma.

▶ **Lemma 3** ([2]). *Let σ be an optimal solution to CEVS. Then for each critical clique C_i and each clique S_j in the cluster graph reached after performing the operations in σ, either $C_i \subseteq S_j$ or $C_i \cap S_j = \emptyset$.*

To show NP-hardness, we reduce from the well-known NP-complete problem 3-SAT.

3-SAT

Input:	A CNF formula ϕ where each clause contains exactly three distinct literals.
Question:	Is ϕ satisfiable?

Even stronger hardness results than NP-hardness can be achieved if one assumes the *exponential time hypothesis* (ETH) to be true. The ETH, formulated by Impagliazzo, Paturi, and Zane [24], states that there exists some positive real number s such that 3-SAT on N variables and M clauses cannot be solved in $2^{s(N+M)}$ time.

3 NP-hardness

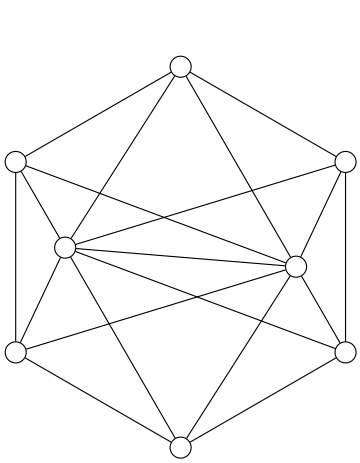
In this section, we show that Cluster Editing with Vertex Splitting is NP-hard, thereby resolving an open problem posed by Abu-Khzam et al. [2].

▶ **Theorem 4.** *CEVS is NP-complete. Moreover, assuming ETH, there is no $2^{o(n+m)}$-time or $2^{o(k)} \cdot \mathrm{poly}(n)$-time algorithm for it.*

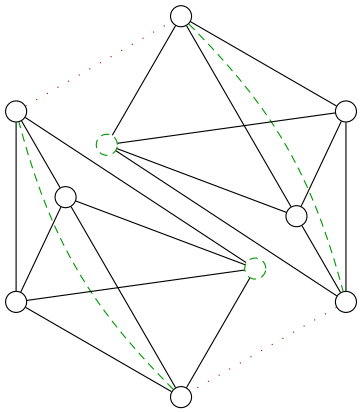
Proof. Since containment in NPis obvious (non-deterministically guess the sequence of operations and check that the resulting graph is indeed a cluster graph), we focus on the NP-hardness and present a reduction from 3-SAT. Therein, we will use two gadgets, a *variable gadget* and a *clause gadget*. The variable gadget is a wheel graph with two (connected) center vertices. An example of this graph is depicted on the left side of Figure 2. We call this graph with t vertices on the outside W_t and we will only consider instances with $t \bmod 6 = 0$, that is, $t = 6a$ for some positive integer a. The clause gadget is a "crown graph" as depicted in Figure 3(a).

More precisely, for each variable x_i, we construct a variable gadget G_i which is a W_{6a} where a is the number of clauses that contain either x_i or $\neg x_i$. For each clause C_j, we construct a clause gadget H_j, that is, a K_5 with the edges of a triangle removed. We arbitrarily assign each of the three vertices of degree two in H_j to one literal in C_j. Finally, we connect the variable and clause gadgets as follows. If a variable x_i appears in a clause C_j, then let u be the vertex in H_j assigned to x_i (or $\neg x_i$). Moreover, let b be the number such that C_j is the b^{th} clause containing either x_i or $\neg x_i$ and let $c = 6(b-1)$. Let the vertices on the outer cycle of G_i be $v_1, v_2, \ldots, v_{6a}$. If C_j contains the literal x_i, then we add the three edges $\{u, v_{c+1}\}, \{u, v_{c+2}\}, \{u, v_{c+3}\}$. If C_j contains the literal $\neg x_i$, then we add the three edges $\{u, v_{c+2}\}, \{u, v_{c+3}\}, \{u, v_{c+4}\}$. To complete the reduction, we set $k = 35M - 2N$, where M is the number of clauses and N is the number of variables.

We next show that the reduction is correct, that is, the constructed instance of CEVS is a yes-instance if and only if the original formula ϕ of 3-SAT is satisfiable. To this end, first assume that ϕ is satisfiable and let β be a satisfying assignment. For each variable x_i we will



(a) The graph W_6 (variable gadget).



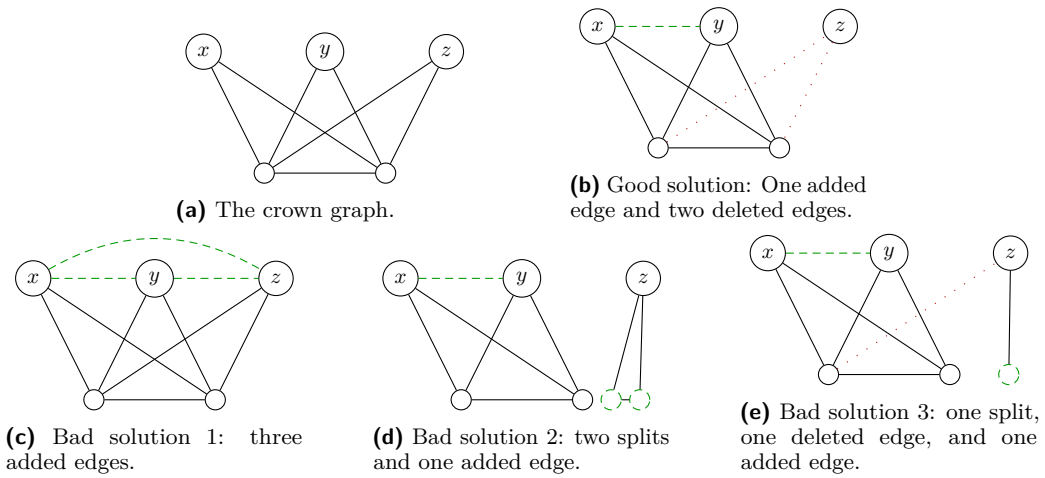
(b) One of the three ways of transforming W_6 into two K_5's using six operations.

Figure 2 The graph W_{6t} requires $8t-2$ edits and any solution with exactly $8t-2$ edits results in a disjoint union of K_5s.

partition G_i into K_5's as follows. Let a be the value such that G_i is isomorphic to W_{6a}. If β sets x_i to true, then we remove the edge $\{v_{3j}, v_{3j+1}\}$ and add the edge $\{v_{3j+1}, v_{3j+3}\}$ for each integer $1 \leq j \leq 2a$ (where values larger than $6a$ are taken modulo $6a$). If β sets x_i to false, then we remove the edge $\{v_{3j+1}, v_{3j+2}\}$ and add the edge $\{v_{3j+2}, v_{3j+4}\}$ for each $1 \leq j \leq 2a$. Moreover, we split the two center vertices $2a-1$ times. In total, we use $8a-2$ modifications to transform G_i into a collection of K_5's. Since each clause contains exactly three literals and we add six vertices for each variable appearance, the sum of lengths of cycles in all variable gadgets combined is $18M$. Hence, in all variable gadgets combined, we perform $24M-2N$ modifications.

Next, we modify the crown graphs. To this end, let C_j be a clause and let H_j be the constructed clause gadget. Since β is a satisfying assignment, at least one variable appearing in C_j satisfies it. If multiple such variables exist, then we pick any one. Let x_i be the selected variable and let u be the vertex in H_j assigned to x_i. We first turn H_j into a K_4 and an isolated vertex by removing the two edges incident to u in H_j and add the missing edge between the two vertices assigned to different variables. Finally, we look at the edges between variable gadgets and clause gadgets. For the vertex u, note that by construction the three vertices that u is adjacent to in G_i already belong to a K_5 and hence we can add two edges to the two (split) centers of the variable gadget to get to a K_6. For the two other vertices in H_j that have edges to vertices in variable gadgets, we remove all three such edges, that is, six edges per clause. Hence, we use $3+2+6=11$ modifications for each clause. Since the total number of modifications is $35M-2N$ and the resulting graph is a collection of K_4's, K_5's, and K_6's, the constructed instance of CEVS is a yes-instance.

For the other direction, suppose the constructed instance of CEVS is a yes-instance. We first show that $24M-2N$ modifications are necessary to transform all variable gadgets into cluster graphs and that this bound can only be achieved if each time exactly three consecutive vertices on the cycles are contained in the same K_5. To this end, consider any variable gadget G_i. By construction, G_i is isomorphic to W_{6a} for some integer a. By the counting argument from above, we show that at least $8a-2$ modifications are necessary. Note first that some edge in the cycle has to be removed or some vertex on the cycle has to be split as otherwise any solution would contain a clique with all vertices in the cycle and



(a) The crown graph.

(b) Good solution: One added edge and two deleted edges.

(c) Bad solution 1: three added edges.

(d) Bad solution 2: two splits and one added edge.

(e) Bad solution 3: one split, one deleted edge, and one added edge.

Figure 3 The crown graph with its four solutions of size 3. The *good solution* is the only solution with three operations that creates at least one isolated vertex.

this would require at least $18a^2 - 9a > 8a - 2$ edge additions (since the degree of each of the $6a$ vertices in the cycle would need to increase from 2 to $6a - 1$). We next analyze how many modifications are necessary to separate b vertices from the outer cycle into a clique. We require at least two modifications for the center vertices (either splitting them or removing the edges between them and the first vertex that we want to separate) and one operation to separate the cycle on the other end (either splitting a vertex or removing an edge of the cycle). For $b \in \{1, 2\}$ these operations are enough. For $b \geq 3$, we need to add $\binom{b}{2} - (b - 1)$ edges (all edges in a clique of size b minus the already existing edges of a path on b vertices). Note that the "average cost" per separated vertex (number of operations divided by b) is minimized (only) with $b = 3$ with a cost of 4 for three vertices. Hence, to separate all but c vertices from the cycle, we require at least $4(6a - c)/3$ operations. The cost for making the remaining c vertices into a clique requires again $\binom{c}{2} - (c - 1)$ edge additions. Analogously, the optimal solution is to have $c = 3$ with just a single edge addition. Thus, the minimum number of required operations is at least $4(6a - 3)/3 + 2 = 8a - 2$ (where the $+2$ comes from the initial edge removal and the final edge addition between the last $c = 3$ vertices) and this value can only be reached by partitioning the cycle into triples which each form a K_5 with the two center vertices. Note that it is always preferable to delete an edge on the outer cycle and not split one of the two incident edges as splitting a vertex increases the number of vertices on the cycle and thus invokes a higher average cost. Next, we analyze the clause gadget and the edges between the different gadgets. We start with the latter. Let u be a vertex in a clause gadget H_j with (three) incident edges to some variable gadget. The only way to not use at least three operations to deal with the three edges is if u is an isolated vertex or if the three neighbors do not have two more neighbors in the current solution. In the former case, we can (possibly) add the two edges between u and the two centers of the respective variable gadgets to build a K_6. In the latter case, we have used at least three operations more in the variable gadget than intended (either by removing edges between neighbors of u and the center vertices or by splitting all neighbors of u). Since each vertex in a variable gadget is only adjacent to at most one vertex in a clause gadget, this cannot lead to an overall reduction in the number of operations and we can therefore ignore this latter case.

We are now in a position to argue that at least eleven modifications are necessary for each clause gadget. First, note that at least three operations are required to transform a crown into a cluster graph. Possible ways of achieving this are depicted in Figure 3. In each of these possibilities at most one vertex becomes an isolated vertex. To make two vertices independent, at least four operations are required and for three isolated vertices at least five operations are required. As shown above, at least two operations are required for each isolated vertex with edges to variable gadgets and at least three operations are required for non-isolated vertices with edges to variable gadgets. Thus, at least eleven operations are required for each clause gadget and eleven operations are sufficient if and only if the three vertices incident to one of the vertices in H_j belong to the same K_5 in the variable gadget.

By the argument above, at least $24M - 2N + 11M = k$ operations are necessary and since the constructed instance is a yes-instance, there is a way to cover all variable gadgets with K_5's such that for each clause there is at least one vertex whose three neighbors in a variable gadget belong to the same K_5. Let C_j be a clause, let u be a vertex with all three neighbors in the same K_5, and let x_i be the variable corresponding to this variable gadget. If x_i appears positively in C_j, then v_{3i+1}, v_{3i+2}, and v_{3i+3} belong to the same K_5 for each i and we set x_i to true. If x_i appears negatively in C_j, then v_{3i+2}, v_{3i+3}, and v_{3i+4} belong to the same K_5 for each i and we set x_i to false. Note that we never set a variable to both true and false in this way. We set all remaining variables arbitrarily to true or false. By construction, the variable x_i satisfies C_j and since we do the same for all clauses, all clauses are satisfied, that is, the original formula ϕ is satisfiable. Thus, the constructed instance is equivalent to the original 3-SAT instance.

Since the reduction can clearly be computed in polynomial time, this concludes the proof for the NP-hardness. For the ETH-hardness, observe that $k, n, m \in O(N + M)$. This implies that there are no $2^{o(n+m)}$-time or $2^{o(k)} \cdot \operatorname{poly} n$-time algorithm for CEVS unless the ETH fails [24]. ◀

In contrast to the reduction for Cluster Editing [26], our reduction does not produce instances with constant maximum degree. We instead observe that in our reduction, the maximum degree of the produced instances depends only on the maximum number of times a variable appears in a clause. Combining this with the fact that 3-SAT remains NP-hard when restricted to instances where each variable appears in at most four clauses [37], we obtain the following corollary.

▶ **Corollary 5.** *CEVS remains NP-hard on bounded-degree graphs.*

4 A faster algorithm

In this section, we improve upon the known $O(9^{k^2} + n + m)$-time algorithm and present an algorithm running in $O(2^{7k \log k} + n + m)$ time. The general outline of the two algorithms is fairly similar. We first compute a kernel with $O(k^2)$ vertices as well as all critical cliques in linear time [2]. We then guess[2] which critical cliques belong to the same clique (cluster) in the solution. The main difference between the two algorithms is how these guesses are made.

▶ **Theorem 6.** *CEVS can be solved in $O(2^{7k \log k} + n + m)$ time.*

[2] Whenever we pretend to guess something, we iterate over all possibilities and assume in the presentation that we are currently in an iteration that yields a solution.

Proof. First, we compute the critical clique of each vertex and the critical clique quotient graph $\mathcal{C}$ of G in linear time [28]. As shown by Abu-Khzam et al. [2], we may assume that $\mathcal{C}$ contains at most $4k$ vertices. By Lemma 3, we can also assume that all vertices in a critical clique belong to the same clique in the graph G' reached after performing an optimal solution σ. Let $\mathcal{X} = (S_1, S_2, \ldots, S_\ell)$ be the set of cliques in G'. Note that $\mathcal{X}$ contains $\ell \leq 2k$ cliques as each operation can complete at most two cliques of the solution (removing an edge between two cliques or splitting a vertex contained in both cliques). Hence, if there are more than $2k$ cliques in the solution, then we cannot reach the solution with k operations. To streamline the following argumentation, we will cover the vertices in $\mathcal{C}$ by cliques $S_1, S_2, \ldots, S_\ell$ and assume that an optimal solution contains exactly $2k$ cliques by allowing some of the cliques to be empty. Next, we iterate over all possible colorings of the vertices in $\mathcal{C}$ using $\ell+1$ colors $0, 1, 2, \ldots, \ell$. Note that there are at most $(\ell+1)^{4k} \in O((2k+1)^{4k})$ such colorings.

The idea behind the coloring is the following. All colors $1, 2, \ldots, \ell$ will correspond to the cliques $S_1, S_2, \ldots, S_\ell$, that is, we try to find a solution where all (critical cliques corresponding to) vertices of the same color (except for color 0) belong to the same clique in the solution. The color 0 indicates that the vertex will belong to multiple cliques in the solution, that is, that all vertices in the respective critical clique will be split. Since each such split operation reduces k by one, we can reject any coloring in which the number of vertices in critical cliques corresponding to vertices with color 0 is more than k. In particular, we can reject any coloring in which more than k vertices have color 0.

Next, we guess two indices $i \in [k], j \in [\ell]$ and assume that the i^{th} vertex of color 0 belongs to S_j or we guess that all vertices of color 0 have been assigned to all cliques they belong to. Note that in each iteration there are $k\ell + 1$ possibilities and since each guess (except for the last one) reduces k by at least one, we can make at most k guesses (after k guesses we know that the next guess has to be that all vertices have been fully assigned). Hence, there are at most $(k\ell+1)^k = (2k^2+1)^k \in O((2k+1)^{2k})$ such guesses.

It remains to compute the best solution corresponding to each possible sequence of guesses. To this end, we first iterate over each pair of vertices and remove an existing edge between them if we guessed that the two vertices do not appear in a common clique. Moreover, we add an edge between them if such an edge does not already exist and we guessed that there is a clique S_i which contains both vertices. Finally, we perform all split operations. Therein, we iteratively split one vertex v into two vertices u_1 and u_2 where u_1 will be the vertex in some clique S_i and u_2 might be split further in the future. The vertex u_1 is adjacent to all vertices that are guessed to belong to S_i. The vertex u_2 is adjacent to all vertices that u was adjacent to, except for vertices that are adjacent to u_1 and not guessed to also belong to some other clique S_j which (some part of) u_2 belongs to.

Since our algorithm basically performs an exhaustive search, it will find an optimal solution. It only remains to analyze the running time. We first compute the kernel in $O(n+m)$ time. We then try $O((2k+1)^{4k})$ possible colorings of $\mathcal{C}$ and for each coloring $O((2k+1)^{2k})$ sequences of guesses. Afterwards, we compute the solution in $O(k^3)$ time. Thus, the overall running time is in $O((2k+1)^{6k} \cdot k^3 + n + m) \subseteq O(2^{7k \log k} + n + m)$. ◀

We mention in passing that while the constants in the running time of our algorithm can probably be improved, a completely new approach is required if one wants a single-exponential-time algorithm. This is due to the fact that the number of possible partitions of $O(k)$ critical cliques into clusters grows super-exponentially (roughly as fast as $k!$) even if no vertex-splitting operations are allowed.

5 Conclusion

On the one hand, we resolve an open question from the literature by showing that CEVS is NP-complete. We also show that, assuming the ETH, there are no $2^{o(n+m)}$-time or $2^{o(k)} \cdot \text{poly}(n)$-time algorithms for CEVS. On the other hand, we give an $O(2^{7k \log k} + n + m)$-time algorithm, beating the previously best $O(9^{k^2} + n + m)$ algorithm. This leaves the following gap.

▶ **Open problem.** *Does there exist a* $2^{O(k)} \cdot \text{poly}(n)$*-time algorithm for CEVS?*

However, even resolving this question should only be seen as a starting point for a much larger undertaking. While we do understand the parameterized complexity of CEVS with respect to k reasonably well, there are still a lot of open questions regarding structural parameters of the input graph. Moreover, one might also study the approximability of CEVS as the trivial constant-factor approximation of Cluster Editing does not carry over if we allow vertex splitting. In case CEVS turns out to be hard to approximate, one might then continue with studying FPT-approximation (schemes) and approximation kernels (also known as lossy kernels).

Regarding real-world applications, it has been observed that requiring communities to be cliques is too restrictive in some settings. To circumvent this issue, many relaxations of cliques such as s-cliques, s-clubs, s-plexes, k-cores, and γ-quasi-cliques have been proposed. It would also be interesting to study vertex-splitting operations in the respective graph editing problems for these relaxations.

Finally, a related area of study is clustering of bipartite data, which is modelled by Bicluster Editing and which has received significant attention recently [13, 22, 38, 42]. Overlapping structures are also relevant in the bipartite case [14]. To the best of our knowledge, nothing is known about Bicluster Editing with Vertex Splitting. We mention that there are two natural versions in the bipartite case and both of them seem worth studying. The two versions differ in whether or not one requires that all copies of a split vertex lie on the same side of a bipartition in a solution. On the one hand, the additional requirement makes sense if the data is inherently bipartite. This happens for example if each vertex represents either a researcher or a paper. On the other hand, if edges reflect something like a seller-buyer relationship, then it is plausible that a person both sells and buys.

References

1 Faisal N. Abu-Khzam, Joseph R. Barr, Amin Fakhereldine, and Peter Shaw. A greedy heuristic for cluster editing with vertex splitting. In *Proceedings of the 4th International Conference on Artificial Intelligence for Industries (AI4I '21)*, pages 38–41. IEEE, 2021.

2 Faisal N. Abu-Khzam, Judith Egan, Serge Gaspers, Alexis Shaw, and Peter Shaw. Cluster editing with vertex splitting. In *Combinatorial optimization*, pages 1–13. Springer, 2018.

3 Reyan Ahmed, Stephen Kobourov, and Myroslav Kryven. An FPT algorithm for bipartite vertex splitting. In *Proceedings of the 30th International Symposium on Graph Drawing and Network Visualization (GD '22)*, pages 261–268. Springer International Publishing, 2022.

4 Sanjeev Arora, Rong Ge, Sushant Sachdeva, and Grant Schoenebeck. Finding overlapping communities in social networks: toward a rigorous approach. In *Proceedings of the 13th ACM Conference on Electronic Commerce (EC '12)*, pages 37–54. Association for Computing Machinery, 2012.

5 Sanghamitra Bandyopadhyay, Garisha Chowdhary, and Debarka Sengupta. FOCS: Fast overlapped community search. *IEEE Transactions on Knowledge and Data Engineering*, 27(11):2974–2985, 2015.

6 Jakob Baumann, Matthias Pfretzschner, and Ignaz Rutter. Parameterized complexity of vertex splitting to pathwidth at most 1. *CoRR*, abs/2302.14725, 2023.

7 Jeffrey Baumes, Mark Goldberg, and Malik Magdon-Ismail. Efficient identification of overlapping communities. In *Proceedings of the 2005 IEEE International Conference on Intelligednce and Security Informatics (ISI '05)*, pages 27–36. Springer, 2005.

8 Francesco Bonchi, Aristides Gionis, and Antti Ukkonen. Overlapping correlation clustering. *Knowledge and Information Systems*, 35(1):1–32, 2013.

9 Leizhen Cai. Fixed-parameter tractability of graph modification problems for hereditary properties. *Information Processing Letters*, 58(4):171–176, 1996. doi:10.1016/0020-0190(96)00050-6.

10 Marek Cygan, Fedor V. Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized algorithms*. Springer, 2015.

11 George B. Davis and Kathleen M. Carley. Clearing the FOG: Fuzzy, overlapping groups for social networks. *Social Networks*, 30(3):201–212, 2008. doi:10.1016/j.socnet.2008.03.001.

12 Reinhard Diestel. *Graph Theory*. Springer, 2005.

13 Pål Grønås Drange, Felix Reidl, Fernando S. Villaamil, and Somnath Sikdar. Fast biclustering by dual parameterization. In *Proceedings of the 10th International Symposium on Parameterized and Exact Computation (IPEC '15)*, pages 402–413. Schloss Dagstuhl — Leibniz-Zentrum für Informatik, 2015.

14 Nan Du, Bai Wang, Bin Wu, and Yi Wang. Overlapping community detection in bipartite networks. In *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT '08)*, pages 176–179, 2008.

15 Peter Eades and Candido Ferreira Xavier de Mendonça Neto. Vertex splitting and tension-free layout. In *Proceedings of the 3rd International Symposium on Graph Drawing and Network Visualization (GD '95)*, pages 202–211. Springer, 1995.

16 Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Graph-based data clustering with overlaps. In *Proceedings of the 15th Annual International Conference on Computing and Combinatorics (COCOON '09)*, pages 516–526. Springer, 2009.

17 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Springer, 2006.

18 Reynaldo Gil-García and Aurora Pons-Porrata. Dynamic hierarchical algorithms for document clustering. *Pattern Recognition Letters*, 31(6):469–477, 2010. doi:10.1016/j.patrec.2009.11.011.

19 Mark Goldberg, Stephen Kelley, Malik Magdon-Ismail, Konstantin Mertsalov, and Al Wallace. Finding overlapping communities in social networks. In *Proceedings of the 2nd IEEE International Conference on Social Computing (SC '10)*, pages 104–113, 2010. doi:10.1109/SocialCom.2010.24.

20 Steve Gregory. An algorithm to find overlapping community structure in networks. In *Proceedings of 2007 Knowledge Discovery in Databases (PKDD '07)*, pages 91–102. Springer, 2007.

21 Jiong Guo. A more effective linear kernelization for cluster editing. *Theoretical Computer Science*, 410(8-10):718–726, 2009. doi:10.1016/j.tcs.2008.10.021.

22 Jiong Guo, Falk Hüffner, Christian Komusiewicz, and Yong Zhang. Improved algorithms for bicluster editing. In *Proceedings of the 5th International Conference on Theory and Applications of Models of Computation (TAMC '08)*, pages 445–456. Springer, 2008.

23 Falk Hüffner, Christian Komusiewicz, Hannes Moser, and Rolf Niedermeier. Fixed-parameter algorithms for cluster vertex deletion. *Theory of Computing Systems*, 47(1):196–217, 2010. doi:10.1007/s00224-008-9150-x.

24 Russell Impagliazzo, Ramamohan Paturi, and Francis Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4):512–530, 2001. doi:10.1006/jcss.2001.1774.

25 Christian Komusiewicz and Johannes Uhlmann. Alternative parameterizations for cluster editing. In *Proceedings of the 2011 Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*, pages 344–355. Springer, 2011.

26 Christian Komusiewicz and Johannes Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. doi:10.1016/j.dam.2012.05.019.

27 Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Statistical properties of community structure in large social and information networks. In *Proceedings of the 17th International Conference on World Wide Web (WWW '08)*, pages 695–704. Association for Computing Machinery, 2008.

28 Guo-Hui Lin, Tao Jiang, and Paul E. Kearney. Phylogenetic k-root and Steiner k-root. In *Proceedings of the 11th International Symposium on Algorithms and Computation (ISAAC '00)*, pages 539–551. Springer, 2000.

29 Neeldhara Misra, Fahad Panolan, and Saket Saurabh. Subexponential algorithm for d-cluster edge deletion: Exception or rule? *Journal of Computer and System Sciences*, 113:150–162, 2020.

30 Martin Nöllenburg, Manuel Sorge, Soeren Terziadis, Anaïs Villedieu, Hsiang-Yun Wu, and Jules Wulms. Planarizing graphs and their drawings by vertex splitting. In *Proceedings of the 30th International Symposium on Graph Drawing and Network Visualization (GD '22)*, pages 232–246. Springer, 2022.

31 Lorenzo Orecchia, Konstantinos Ameranis, Charalampos Tsourakakis, and Kunal Talwar. Practical almost-linear-time approximation algorithms for hybrid and overlapping graph clustering. In *Proceedings of the 39th International Conference on Machine Learning (ICML '22)*, pages 17071–17093. PMLR, 2022.

32 Airel Pérez-Suárez, José Fco. Martínez-Trinidad, Jesús A. Carrasco-Ochoa, and José E. Medina-Pagola. An algorithm based on density and compactness for dynamic overlapping clustering. *Pattern Recognition*, 46(11):3040–3055, 2013. doi:10.1016/j.patcog.2013.03.022.

33 Hafiz Hassaan Saeed, Khurram Shahzad, and Faisal Kamiran. Overlapping toxic sentiment classification using deep neural architectures. In *Proceedings of the 2018 IEEE International Conference on Data Mining Workshops (ICDMW '18)*, pages 1361–1366. IEEE Computer Society, 2018. doi:10.1109/ICDMW.2018.00193.

34 Satu Elisa Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007. doi:10.1016/j.cosrev.2007.05.001.

35 Cees G. M. Snoek, Marcel Worring, Jan C. van Gemert, Jan-Mark Geusebroek, and Arnold W. M. Smeulders. The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th ACM International Conference on Multimedia (MM '06)*, pages 421–430. Association for Computing Machinery, 2006.

36 Lei Tang and Huan Liu. Scalable learning of collective behavior based on sparse social dimensions. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management (CIKM '09)*, pages 1107–1116. Association for Computing Machinery, 2009.

37 Craig A. Tovey. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics*, 8(1):85–89, 1984.

38 Dekel Tsur. Faster parameterized algorithm for bicluster editing. *Information Processing Letters*, 168, 2021. doi:10.1016/j.ipl.2021.106095.

39 Qinna Wang and Eric Fleury. Uncovering overlapping community structure. In *Proceedings of the 2nd International Workshop on Complex Networks (COMPLEX '10)*, pages 176–186. Springer, 2010.

40 Xufei Wang, Lei Tang, Huiji Gao, and Huan Liu. Discovering overlapping groups in social media. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM '10)*, pages 569–578, 2010.

41 Alicja Wieczorkowska, Piotr Synak, and Zbigniew W. Raś. Multi-label classification of emotions in music. In *Proceedings of the 2006 Intelligent Information Processing and Web Mining (IIPWM '04)*, pages 307–315. Springer, 2006.

42 Mingyu Xiao and Shaowei Kou. A simple and improved parameterized algorithm for bicluster editing. *Information Processing Letters*, 2022.

43 Shihua Zhang, Rui-Sheng Wang, and Xiang-Sun Zhang. Identification of overlapping community structure in complex networks using fuzzy c-means clustering. *Physica A: Statistical Mechanics and its Applications*, 374(1):483–490, 2007.