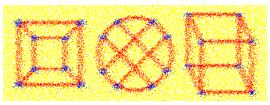

Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 25, no. 1, pp. 521–547 (2021)
DOI: 10.7155/jgaa.00570

# On 2-Clubs in Graph-Based Data Clustering: Theory and Algorithm Engineering

*Aleksander Figiel* *Anne-Sophie Himmel* *André Nichterlein* *Rolf Niedermeier*

TU Berlin, Faculty IV, Algorithmics and Computational Complexity

| | | |
|---|---|---|
| Submitted: February 2021 | Reviewed: August 2021 | Revised: August 2021 |
| Accepted: October 2021 | Final: October 2021 | Published: October 2021 |
| Article type: Regular Paper | Communicated by: M Nöllenburg | |

**Abstract.** Editing a graph into a disjoint union of clusters is a standard optimization task in graph-based data clustering. Here, complementing classical work where the clusters shall be cliques, we focus on clusters that shall be 2-clubs, that is, subgraphs of diameter at most two. This naturally leads to the two NP-hard problems 2-Club Cluster Editing (the editing operations are edge insertions and edge deletions) and 2-Club Cluster Vertex Deletion (the editing operations are vertex deletions). Answering an open question from the literature, we show that 2-Club Cluster Editing is W[2]-hard with respect to the number of edge modifications, thus contrasting the fixed-parameter tractability result for the classical Cluster Editing problem (considering cliques instead of 2-clubs). Then, focusing on 2-Club Cluster Vertex Deletion, which is easily seen to be fixed-parameter tractable, we show that under standard complexity-theoretic assumptions it does not have a polynomial-size problem kernel when parameterized by the number of vertex deletions. Nevertheless, we develop several effective data reduction and pruning rules, resulting in a competitive solver, outperforming a standard ILP formulation in most instances of an established biological test data set.

AF was partially supported by DFG Project FPTinP NI 369/18. ASH was supported by DFG Project FPTinP NI 369/16. An extended abstract of this work appears in the proceedings of the 12th International Conference on Algorithms and Complexity (CIAC '21) [17].

*E-mail addresses:* a.figiel@tu-berlin.de (Aleksander Figiel) andre.nichterlein@tu-berlin.de (André Nichterlein) rolf.niedermeier@tu-berlin.de (Rolf Niedermeier)

# 1 Introduction

Graph-based data clustering is among the most important application domains for graph modification problems [36]. Roughly speaking, the goal herein is to transform a given graph into (usually disjoint) clusters, thereby performing as few modification operations (edge deletions, edge insertions, vertex deletions) as possible. Problems of this type are typically NP-hard. The perhaps most prominent problem herein is Cluster Editing (also known as Correlation Clustering), where the clusters are requested to be cliques and one is allowed to perform both edge insertions and edge deletions. Cluster Editing and its variants have applications in, among others, bioinformatics, information retrieval, and psychometrics [1, 2, 38].

There has been a lot of algorithmic work on Cluster Editing, e.g., see the surveys by Böcker and Baumbach [4] and by Crespelle et al. [9]. In addition, also the variant where one modifies the input graph by vertex deletions received significant interest [8, 14, 24, 37].

Arguably, for many data science applications the request that the clusters have to be cliques is too rigid. Hence, the consideration of clique relaxations for defining clusters gained attention in graph-based data clustering [3, 20, 30, 32]. In this work, we focus on so-called *2-clubs* as clusters [30, 32]: these are diameter-at-most-two graphs (thus, cliques are 1-clubs). As opposed to finding cliques, finding 2-clubs of size at least $k$ is fixed-parameter tractable with respect to $k$ [22, 23, 35]. Note that 2-clubs have been used in the context of biological data analysis [26, 33]. Moreover, 2-clubs have been studied in the context of covering vertices in a graph [11, 12, 13].

Closest to our work is the one by Liu et al. [30]: they studied the edge editing variant (referred to as 2-Club Cluster Editing) and the vertex/edge deletion variants (referred to as 2-Club Cluster Vertex/Edge Deletion). In particular, Liu et al. [30] proved the NP-hardness of all three variants and provided search-tree algorithms based on extensive case distinctions that run in $3.31^k \cdot n^{O(1)}$ time for 2-Club Cluster Vertex Deletion and in $2.74^k \cdot n^{O(1)}$ time for 2-Club Cluster Edge Deletion (thus showing that both variants are fixed-parameter tractable); here $k$ denotes the required number of edge/vertex deletions and $n$ denotes the number of vertices in the input. Continuing and complementing this work, we contribute the following three main results:

1. Answering an open question of Liu et al. [30], in Section 2 we show that 2-Club Cluster Editing is W[2]-hard with respect to the number of modified edges (deletions and insertions), hence most likely not fixed-parameter tractable. This stands in sharp contrast to the problems Cluster Editing [19] and the more general $s$-Plex Cluster Editing [20][1], both known to be fixed-parameter tractable for the parameter number of edge modifications. The W[2]-hardness seems surprising when considering the fact that while Cluster Editing is fixed-parameter tractable [4] and 2-Club Cluster Editing is presumably not, by way of contrast finding cliques is presumably not fixed-parameter tractable (since W[1]-hard [15]) while finding 2-clubs is.

2. Complementing fixed-parameter tractability and kernelization results for Cluster Vertex Deletion [8, 24, 37] and $s$-Plex Cluster Vertex Deletion [3], in Section 2 we show that, other than these related problems and despite being easily seen to be fixed-parameter tractable for the parameter solution size, 2-Club Cluster Vertex Deletion is unlikely to have a polynomial-size problem kernel.[2]

[1] This is the generalization of Cluster Editing where the clusters are requested to be $s$-plexes (and not cliques); an $s$-plex is a subgraph where each vertex is connected to all other vertices of the $s$-plex except for at most $s-1$ vertices. Notably, a clique is a 1-plex.

[2] It has been featured as an open problem whether the edge deletion variant $s$-Club Cluster Edge Deletion has a polynomial-size problem kernel [9, 30].

3. In Sections 3 to 5, we explore the fixed-parameter tractability of 2-Club Vertex Deletion from a more practical angle and develop several efficient data reduction rules together with effective search-tree pruning rules. Performing an empirical evaluation with standard biological data, we show that our tuned algorithmic approach (based on branching and data reduction) in most relevant cases clearly outperforms a standard ILP formulation solved with CPLEX, thus providing a state-of the art software tool for the vertex deletion variant of graph-based data clustering with 2-clubs.

**Preliminaries.** All graphs considered in our work are undirected and simple. For a graph $G = (V, E)$ we set $n := |V|$ and $m := |E|$. We denote with $\binom{V}{2}$ the set of all two-element subsets of $V$. For a vertex $v \in V$, we denote by $N_G(v) := \{w \in V \mid \{v, w\} \in E\}$ the *open neighborhood* of $v$ and by $N_G[v] := N_G(v) \cup \{v\}$ the *closed neighborhood* of $v$. The *degree* of $v$ is $\deg_G(v) := |N_G(v)|$. For a vertex subset $V' \subseteq V$, let $N_G[V'] := \bigcup_{v \in V'} N_G[v]$. If it is clear from the context, then we omit $G$ from the subscripts. We denote by $G[V']$ the subgraph of $G$ induced by the vertex set $V' \subseteq V$ and by $G[E']$ the subgraph of $G$ with edge set $E' \subseteq E$, that is, $G[E'] := (V, E')$. The graph $G - v$ is obtained by deleting $v \in V$ from $G$, that is $G - v := G[V \setminus \{v\}]$.

A path $P$ in $G$ is an ordered sequence of pairwise distinct vertices $v_1, v_2, \ldots, v_{k+1} \in V$ such that $\{v_i, v_{i+1}\} \in E$ for all $i \in \{1, \ldots, k\}$. It is also an *induced path* if these are the only edges between its vertices. The length of $P$ is $k$. We denote by $P_n$ a path on $n$ vertices. The *distance* of two vertices $s, t \in V$, denoted by $\mathrm{dist}_G(s, t)$, is the length of a shortest path connecting $s$ and $t$ if one exists, and $\infty$ otherwise. The *diameter* of a graph is the maximum distance of any two vertices, formally $\max_{s,t \in V} \mathrm{dist}_G(s, t)$. A graph is said to be *connected* if there exists a path between all pairs of its vertices. A (connected) *component* of a graph $G$ is a maximal vertex set $S \subseteq V$ such that $G[S]$ is connected.

**$s$-Clubs.** An *$s$-club* is a graph of diameter at most $s$. A *clique* is a 1-club. Furthermore, an *$s$-club cluster graph* is a graph in which each component is an $s$-club. In this paper, we consider the following two problems, where $E \triangle F := (E \setminus F) \cup (F \setminus E)$ denotes the *symmetric difference* of two sets $E$ and $F$.

$s$-Club Cluster Editing
**Input:** An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
**Question:** Is there an edge set $F \subseteq \binom{V}{2}$ with $|F| \leq k$ such that $G[E \triangle F]$ is an $s$-club cluster graph?

$s$-Club Cluster Vertex Deletion
**Input:** An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
**Question:** Is there a vertex subset $S \subseteq V$ with $|S| \leq k$ such that $G[V \setminus S]$ is an $s$-club cluster graph?

An edge set $F \subseteq \binom{V}{2}$ such that $G[E \triangle F]$ is an $s$-club cluster graph is called an *$s$-club editing set* and a vertex set $S \subseteq V$ such that $G[V \setminus S]$ is an $s$-club cluster graph is called an *$s$-club vertex deletion set*.

**2-Clubs.** A 2-club is a graph with diameter at most two. This means that for all pairs of vertices $u, v \in V$ it holds that $u$ and $v$ are adjacent or have at least one common neighbor. Note that 2-clubs are *non-hereditary*, that is, if $G$ is a 2-club, then deleting vertices from $G$ may destroy this property. This is a significant difference in comparison with cliques.

Using terminology of Liu et al. [30], we call a path $stuv$ in $G$ a *restricted* $P_4$ if $\text{dist}_G(s,v) = 3$. That is, a restricted $P_4$ is a shortest path connecting $s$ and $v$ and is thus also an induced $P_4$. The following characterization is easy to verify:

**Observation 1 ([30, Lemma 3])** *A graph $G$ is a 2-club cluster graph if and only if it contains no restricted $P_4$.*

**Parameterized Algorithmics.** We use standard notation and terminology from parameterized complexity. A parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ is a set of pairs $(I,k)$, where $I$ denotes the problem instance and $k$ is the parameter. Problem $\Pi$ is *fixed-parameter tractable* (FPT) if there exists an algorithm solving any instance of $\Pi$ in $f(k)\cdot|I|^c$ time, where $f$ is some computable function and $c$ is some constant. A *parameterized reduction* from a parameterized problem $\Pi \subseteq \Sigma^* \times \mathbb{N}$ to a parameterized problem $\Pi' \subseteq \Sigma^* \times \mathbb{N}$ is a function which maps any instance $(I,k) \in \Sigma^* \times \mathbb{N}$ to another instance $(I',k') \in \Sigma^* \times \mathbb{N}$ such that (1) $(I',k')$ can be computed from $(I,k)$ in FPT time, (2) $k' \leq g(k)$ for some computable function $g$, and (3) $(I,k) \in \Pi \iff (I',k') \in \Pi'$. If $\Pi$ is W[$t$]-hard, $t \geq 1$, then such a parameterized reduction shows that also $\Pi'$ is W[$t$]-hard, that is, presumably not fixed-parameter tractable. A *reduction to a problem kernel* is a parameterized self-reduction (from $\Pi$ to $\Pi$) such that $(I',k')$ can be computed in polynomial time and $|I'| \leq g(k)$. If $g$ is a polynomial, then $(I',k')$ is called a *polynomial kernel*. Problem kernels are usually achieved by applying *data reduction rules*. Given an instance $(I,k)$, a data reduction rule computes in polynomial time a new instance $(I',k')$. We call a data reduction rule *safe* if $(I,k) \in \Pi \iff (I',k') \in \Pi$.

**Organization of the Paper.** We prove in Section 2 the W[2]-hardness of 2-Club Cluster Editing and that 2-Club Cluster Vertex Deletion does not admit a polynomial kernel with respect to $k$ unless NP $\subseteq$ coNP / poly. We then present in Section 3 an ILP-formulation and a branch&bound algorithm for 2-Club Cluster Vertex Deletion. In Section 4, we provide implementation details for the solver which we experimentally evaluate in Section 5. We conclude in Section 6.

## 2 Hardness Results

It is easy to see that 2-Club Cluster Vertex Deletion is fixed-parameter tractable with respect to solution size $k$ [30]: By Observation 1, it is enough to recursively search for a restricted $P_4$ $stuv$ and delete a vertex to separate $s$ and $v$. In contrast, we subsequently show that 2-Club Cluster Editing is W[2]-hard with respect to solution size $k$, answering an open question of Liu et al. [30]. Intuitively, the hardness is due to the fact that there is a "non-local" way of destroying a restricted $P_4$ with edge insertions, see Fig. 1 for an illustration.

The basic idea of our parameterized reduction from Dominating Set[3] is inspired by a parameterized reduction by Gao et al. [18, Theorem 1] who showed hardness for the problem of reducing the diameter of a given graph to two by inserting at most $k$ edges. In our reduction we need to take care of the possibility to delete edges, which changes several details of the construction. Dominating Set remains W[2]-hard with respect to $k$ for graphs with diameter two [31]. Thus we can assume that the Dominating Set instance has diameter two.

[3] Given an undirected graph $G = (V,E)$ and an integer $k$, the question is whether there is a dominating set $V' \subseteq V$ (that is, $N[V'] = V$) of size at most $k$.

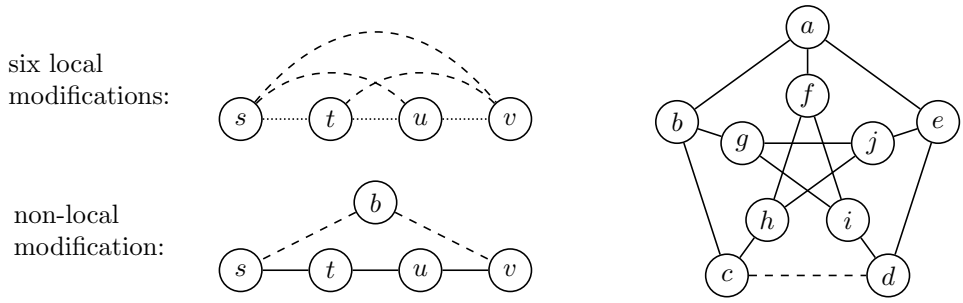



Figure 1: *Top left:* The six "local" modifications to destroy a restricted $P_4$. Any edge that is inserted (dashed edges) or deleted (dotted edges) has both its ends in $\{s, t, u, v\}$.
*Bottom left:* A non-local modification (the two inserted edges are dashed), where $b$ can be any vertex other than $s, t, u$, and $v$.
*Right:* The dashed edge $\{c, d\}$ indicates the single optimal solution (inserting the edge, the resulting Petersen graph has diameter two) which is a non-local modification. Since the distance between $c$ and $d$ was four, the insertion of edge $\{c, d\}$ is not part of any local modification.

**Theorem 1** 2-Club Cluster Editing *is W[2]-hard with respect to $k$, even if the input graph has diameter three, a minimum cut larger than $k$, and deleting one vertex results in a 2-club.*

**Proof:** Let $G = (V, E)$ be a graph with diameter two. We construct a graph $G'$ in such a way that $G$ has a dominating set of size at most $k$ if and only if $G'$ has a 2-club editing set of size at most $k$. The graph $G' = (V', E')$ consists of the following parts: the original graph $G$, a clique $C \subseteq V'$ of cardinality $(n+1)^2$, and a single vertex $x$. We assign two indices for the vertices $c_{i,j} \in C$ such that $i, j \in \{0, \ldots, n\}$. The vertices in $V$ only have one index: $v_i \in V$, $i \in \{1, \ldots, n\}$. In addition to the existing edges in $G$ and $C$, add the following edges:

- for each $j \in \{0, \ldots, n\}$, add $\{x, c_{0,j}\}$, and
- for each $c_{i,j} \in C, i \neq 0$, add $\{v_i, c_{i,j}\}$.

The resulting graph $G'$ has $\mathcal{O}(n^4)$ edges and $\mathcal{O}(n^2)$ vertices and can be constructed in polynomial time; see Fig. 2 for an example. Note that $G'$ has diameter three, a minimum cut of $n + 1 > k$, and deleting $x$ results in a 2-club (recall that $G$ has diameter two).

We claim that there exists a dominating set of size at most $k$ for $G$ if and only if there exists a 2-club editing set of size at most $k$ for $G'$ (which only inserts edges).

"$\Rightarrow$": Let $D$ be a dominating set for $G$ with $|D| \leq k$, and $F := \{\{x, v\} \mid v \in D\}$. Let $H := G'[E' \triangle F]$. For every $v_i \in V$, either $v_i \in D$ and then $\text{dist}_H(x, v_i) = 1$, or $v_i \notin D$ and then $v_i$ has a neighbor in $D$ and thus $\text{dist}_H(x, v_i) = 2$. This means that $H$ is a 2-club cluster graph and $F$ is a 2-club editing set for $G'$ with $|F| \leq k$.

"$\Leftarrow$": Let $F$ be a 2-club editing set for $G'$ with $|F| \leq k$ and $H = G'[E' \triangle F]$ be the resulting 2-club cluster graph. Assume without loss of generality that $F$ is minimal. Removing any edge would only be optimal if $H$ contained more than one 2-club cluster. Note that the size of a minimum cut of $G'$ is $n + 1$ and that $k < n$. Hence, there is only one 2-club cluster in a solution and no edge is removed.

For any inserted edge $\{a, b\} \in F$ exactly one of the following cases applies, since the distance between $x$ and some $v_i \in V$ has to be reduced by means of inserting $\{a, b\}$.

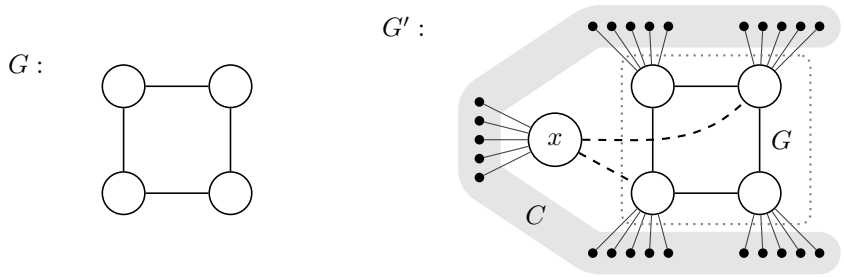



Figure 2: *Left:* A $C_4$ as an example for a Dominating Set instance $G$ with $k = 2$.
*Right:* The constructed graph $G'$ consisting of the $C_4$ (vertices within the dotted box), vertex $x$, and clique $C$ (small black vertices within gray background); for clarity, edges within the clique are not drawn. The two dashed edges indicate a possible solution for $G'$, that is, after inserting the two dashed edges $G'$ will be a 2-club and the corresponding endpoints (excluding $x$) form a dominating set in $G$.

- $\{a, b\} = \{v_i, x\}$: Then $\text{dist}_H(x, v_i) = 1$ and for $a \in N_G(v_i)$ we have $\text{dist}_H(x, a) \leq 2$. We interpret this as $v_i$ being a dominating vertex in $G$.

- $\{a, b\} = \{v_i, c_{0,j}\}$: This edge enables a path of length two from $v_i$ to $x$ via $c_{0,j}$. This means that this edge is only of benefit to $v_i$. Then $F' = (F \setminus \{v_i, c_{0,j}\}) \cup \{x, v_i\}$ is also a 2-club editing set with $|F| = |F'|$.

- $\{a, b\} = \{v_i, v_j\}$: This means that one of the vertices has an edge to $x$. Without loss of generality assume that $\{x, v_i\} \in F$. Note that $F$ is only minimal if $\{x, v_j\} \notin F$, as the edge $\{v_i, v_j\}$ is only of benefit to $v_j$ and no other vertices since it enables a path of length two from $v_j$ to $x$ via $v_i$. Then $F' = (F \setminus \{v_i, v_j\}) \cup \{x, v_j\}$ is also a 2-club editing set with $|F| = |F'|$.

- $\{a, b\} = \{v_i, c_{j,k}\}, j \neq i, j \neq 0$: This means that there is an edge $\{x, c_{j,k}\} \in F$, otherwise $F$ would not be minimal. The edge $\{v_i, c_{j,k}\}$ enables a path of length two from $v_i$ to $x$ via $c_{j,k}$. This means that the edge is of no benefit to any other vertices. Then $F' = (F \setminus \{v_i, c_{j,k}\}) \cup \{x, v_i\}$ is also a 2-club editing set with $|F| = |F'|$.

- $\{a, b\} = \{x, c_{i,j}\}, i \neq 0$: This edge enables a path of length two from $v_i$ to $x$ via $c_{i,j}$. In the previous case, we have seen that there exists an $F'$ with $\{x, c_{i,j}\} \in F'$ such that there exists no edge $\{v_\ell, c_{i,j}\} \in F'$ with $\ell \neq i$. This means that the edge $\{x, c_{i,j}\}$ is of no benefit to any other vertices. Then $F'' = (F' \setminus \{x, c_{i,j}\}) \cup \{x, v_i\}$ is also a 2-club editing set with $|F| = |F''|$.

Altogether, we know that there exists an $F'$ with $|F'| = |F|$ such that $F'$ is a 2-club editing set of the form $\{\{x, v\} \mid v \in D\}$ for some $D \subseteq V$. This means that $D$ is a dominating set for $G$ with $|D| \leq k$.

Summarizing, the reduction from $(G, k)$ to $(G', k)$ is a parameterized reduction from Dominating Set for graphs with diameter two to 2-Club Cluster Editing. Since Dominating Set is W[2]-hard for graphs of diameter two [31], this yields that 2-Club Cluster Editing is also W[2]-hard. □

**No Polynomial Kernel for 2-Club Cluster Vertex Deletion.** Next, we use the OR-cross-composition framework of Bodlaender et al. [7] to prove that 2-Club Cluster Vertex Deletion does not admit a polynomial kernel with respect to the solution size $k$. To this end, we introduce some notions used in the proof. Given an NP-hard problem $L$, an equivalence relation $\mathcal{R}$ on the instances of $L$ is a *polynomial equivalence relation* if

(i) one can decide for any two instances in time polynomial in their sizes whether they belong to the same equivalence class, and

(ii) for any finite set $S$ of instances, $\mathcal{R}$ partitions $S$ into at most $(\max_{x\in S}|x|)^{O(1)}$ equivalence classes.

**Definition 1** *Given an* NP*-hard problem $L$, a parameterized problem $P$, and a polynomial equivalence relation $\mathcal{R}$ on the instances of $L$, an* OR-cross-composition *of $L$ into $P$ (with respect to $\mathcal{R}$) is an algorithm that takes $\ell$ $\mathcal{R}$-equivalent instances $\mathcal{I}_1,\ldots,\mathcal{I}_\ell$ of $L$ and constructs in time polynomial in $\sum_{i=1}^{\ell}|\mathcal{I}_\ell|$ an instance $(\mathcal{I},k)$ such that*

1. *$k$ is polynomially upper-bounded in $\max_{1\le i\le\ell}|\mathcal{I}_i|+\log(\ell)$ and*
2. *$(\mathcal{I},k)$ is a yes-instance for $P$ if and only if there is at least one $\ell'\in[\ell]$ such that $\mathcal{I}_{\ell'}$ is yes-instance for $L$.*

If a parameterized problem $P$ admits an OR-cross-composition for some NP-hard problem $L$, then $P$ does not admit a polynomial kernel with respect to its parameterization, unless NP $\subseteq$ coNP / poly [7].

**Theorem 2** 2-Club Cluster Vertex Deletion *does not admit a polynomial kernel with respect to $k$ unless* NP $\subseteq$ coNP / poly*.*

**Proof:** To show the result, we provide an OR-cross-composition from 2-Club Cluster Vertex Deletion to itself. To this end, we define $\mathcal{R}$ as follows: two instances $(G_1,k_1)$, $(G_2,k_2)$ of 2-Club Cluster Vertex Deletion are equivalent with respect to $\mathcal{R}$ iff $k_1=k_2$. Since the solution size is at most $n$, this gives a polynomial equivalence relation.

Given $\ell$ $\mathcal{R}$-equivalent instances $(G_1=(V_1,E_1),k),\ldots,(G_\ell=(V_\ell,E_\ell),k)$, we construct a new instance $(G'=(V',E'),k')$ as follows. Without loss of generality, assume that $\ell$ is a power of two (otherwise copy instances until $\ell$ is a power of two). We set $k':=k+\log\ell$. To describe $G'$, we need a simple *selection-gadget* consisting of two stars with $k'+1$ leaves each where the two center vertices are adjacent. Observe that in the selection-gadget the leaves of one star are at distance three to the leaves of the other star. Moreover, since each star has more than $k'$ leaves, the only possibility to transform a selection-gadget into a 2-club cluster graph is to delete one of the two center vertices.

We can now define $G'$: To this end, we recursively create an "instance-selector" that forces the selection of exactly one instance $G_i$ as shown in Fig. 3. First, add a selection-gadget with the two center vertices $c_L$ and $c_R$ (left and right). Second, recursively build the two graphs $G_L,G_R$ composing $G_1,\ldots,G_{\ell/2}$ and $G_{\ell/2+1},\ldots,G_\ell$ respectively until $G_L,G_R$ consist of only one input instance. Make every vertex in $G_L$ (in $G_R$) adjacent to $c_L$ (to $c_R$). Note that this recursive procedure has recursion depth $\log\ell$.

The construction of $(G',k')$ can clearly be done in polynomial time. It remains to show the correctness, that is, $(G',k')$ is a yes-instance if and only if there is a yes-instance $(G_i,k)$, $i\in[\ell]$.

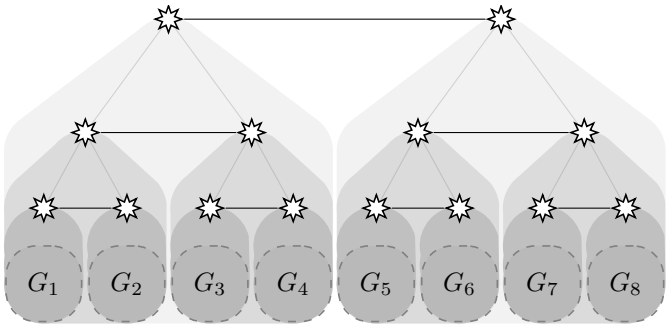



Figure 3: Illustration of the construction for Theorem 2 exemplified for $\ell = 8$. Star-shaped vertices have $k' + 1$ additional leaves and are connected to all vertices in the gray-shaded area below them.

"$\Rightarrow$:" Let $S' \subseteq V'$ be a minimal solution of size at most $k + \log \ell$ for $G'$. Observe that, by construction of $G'$, at least one of the two center vertices $c_L$ and $c_R$ of the "topmost" selection-gadget has to be in $S'$. Assume without loss of generality that $c_L \in S$ (the other case is completely analogous). Observe that the connected component $C_R$ in $G' - c_L$ containing $c_R$ is a 2-club since $c_R$ is a universal vertex in $C_R$. Since $S'$ is minimal, it follows that $S'$ contains no vertex in $C_R$. By construction, the connected component containing the graphs $G_1, \ldots, G_{\ell/2}$ contains a selection-gadget where again one of the two center vertices has to be in $S'$. By induction on the recursion depth one can show that $S'$ contains exactly $\log \ell$ center vertices of the selection-gadgets. Moreover, there is exactly one graph $G_i$ such that all $\log \ell$ center vertices adjacent to the vertices in $G_i$ are in $S'$. Since $S'$ is a solution for $G'$ it follows that $S' \cap V_i$ is a solution of size at most $k$ for $G_i$.

"$\Leftarrow$:" Let $i \in [\ell]$ be such that $(G_i, k)$ is a yes-instance and let $S \subseteq V_i$ be the solution for $G_i$, $|S| \leq k$. The solution $S' \subseteq V'$ for $G'$ consists of $S$ and every adjacent center vertex in a selection-gadget. Observe that $|S'| \leq k + \log \ell$ since, by construction, there are $\log \ell$ selection-gadgets that contain a vertex adjacent to $G_i$. Thus, it remains to show that $G' - S'$ is a 2-club cluster graph. To this end, observe that, by assumption, $G_i - S$ is a 2-club cluster graph. Note that each graph $G_j$, $i \neq j$, is in $G' - S'$ in a connected component with a center vertex $c$ of a selection-gadget such that the other center vertex of this gadget is in $S'$. Note that each such center vertex $c$ is a universal vertex in its connected component in $G' - S'$. Thus a connected component containing $c$ forms a 2-club. Since each connected component of $G' - S'$ contains vertices of at least one graph $G_j$, it follows that $G' - S'$ is a 2-club cluster graph. □

# 3 Algorithms for 2-Club Cluster Vertex Deletion

In this section, we first formulate 2-Club Cluster Vertex Deletion as an Integer Linear Program (ILP) and then introduce a branch&bound algorithm solving a generalization of 2-Club Cluster Vertex Deletion. The basis of our branch&bound algorithm is a simple search tree (see Section 3.1), which is refined by data reduction rules (see Section 3.2) and lower bounds (see Section 3.3). We use the ILP-formulation in our experiments to evaluate our branch&bound algorithm.

**ILP Formulation.** By Observation 1, a graph is a 2-club cluster graph if and only if it contains no restricted $P_4$. Recall that a restricted $P_4$ is an induced $P_4$ *stuv* that is also a shortest path between $s$ and $t$. Thus, there exists no vertex $w \in N(s) \cap N(v)$ in the common neighborhood of $s$ and $v$. The deletion of a vertex cannot create any new induced path but it might "promote" an induced $P_4$ to a restricted $P_4$. Hence, if $N(s) \cap N(v) = \emptyset$ for any induced $P_4$ *stuv* in $G$, then at least one vertex from *stuv* must be deleted.

We introduce a variable $x_v$ for each vertex $v \in V$. This variable has a value of 1 if and only if $v$ is in the 2-club vertex deletion set. This leads to the following 0/1-ILP formulation:

$$\begin{aligned} \text{min:} \quad & \sum_{v \in V} x_v \\ \text{such that} \quad & x_s + x_t + x_u + x_v + \sum_{b \in N(s) \cap N(v)} (1 - x_b) \geq 1 && \text{for all induced } P_4\text{'s } \textit{stuv} \text{ in } G, \\ & x_v \in \{0, 1\} && \text{for all } v \in V. \end{aligned}$$

## 3.1 Search Tree

A graph is a 2-club cluster graph if and only if there exists no restricted $P_4$ (Observation 1). This observation yields a straight-forward search tree of size $O(4^k)$ for 2-Club Cluster Vertex Deletion. To shrink the search tree, we employ the concept of permanent vertices. A vertex is *permanent* if it is not allowed to be removed from the graph. Additionally, we introduce weights on the vertices. These two concepts will allow us to apply a wider range of data reduction rules and lower bounds in the search tree algorithm. The resulting problem is defined as follows:

Generalized 2-Club Cluster Vertex Deletion (Gen2CVD)

**Input:** An undirected graph $G = (V, E)$, an integer $k \in \mathbb{N}$, a set $F \subseteq V$ of permanent vertices, and a weight function $w \colon V \to \mathbb{N}^+$.

**Question:** Is there an $S \subseteq V$ with $w(S) \leq k$ and $S \cap F = \emptyset$ such that $G[V \setminus S]$ is a 2-club cluster graph?

Note that an instance $(G, k)$ of 2-Club Cluster Vertex Deletion is clearly equivalent to the instance $(G, k, \emptyset, w \equiv 1)$ of Gen2CVD.

Our algorithm uses a simple branching rule that takes a restricted $P_4$ and branches into all four cases of deleting one vertex which implies updates of the set $F$ of permanent vertices in each branch. If some vertex of the restricted $P_4$ *stuv* is already in $F$, then we skip the corresponding case in the branching. Thus, the branching itself "grows" the set $F$ of permanent vertices that will reduce the cases to be considered later in the branching. Moreover, if more than one restricted $P_4$ exists, then the algorithm chooses one with most vertices in $F$ and uses the weights of the vertices as tiebreaker.

**Branching Rule 1** *Let $\mathcal{I} = (G, k, F, w)$ be an instance of* Gen2CVD. *If $G$ is not a 2-club cluster graph, then find a restricted $P_4$ stuv and split $\mathcal{I}$ into four smaller instances $\mathcal{I}_s, \mathcal{I}_t, \mathcal{I}_u, \mathcal{I}_v$ as follows:*

- $\mathcal{I}_s = (G - s, k - w(s), F, w)$,
- $\mathcal{I}_t = (G - t, k - w(t), F \cup \{s\}, w)$,
- $\mathcal{I}_u = (G - u, k - w(u), F \cup \{s, t\}, w)$,

- $\mathcal{I}_v = (G - v, k - w(v), F \cup \{s, t, u\}, w)$.

*If an instance $\mathcal{I}_x$ was derived by removing a permanent vertex $x \in \{s, t, u, v\} \cap F$, then do not branch on $\mathcal{I}_x$.*

**Lemma 1** *Branching Rule 1 is correct.*

**Proof:** If $G$ is already a 2-club cluster graph, then there is nothing to do. Otherwise, assume that $G$ contains $stuv$, a restricted $P_4$. We have to show that $\mathcal{I}$ is a yes-instance if and only if at least one of the instances $\mathcal{I}_s, \mathcal{I}_t, \mathcal{I}_u, \mathcal{I}_v$ is a yes-instance.

"$\Leftarrow$": Let $S'$ be a 2-club vertex deletion set for one of the four instances where the vertex $x \in \{s, t, u, v\}$ was deleted. Because all four instances have a set of permanent vertices that is a superset of $F$, the set $S := S' \cup \{x\}$ is a 2-club vertex deletion set for $\mathcal{I}$ unless $x$ is permanent, in which case the instance $\mathcal{I}_x$ would have been skipped.

"$\Rightarrow$": If $\mathcal{I}$ is a yes-instance, then at least one of the vertices $s, t, u$, or $v$ has to be deleted. Let $S$ be a 2-club vertex deletion set for $G$ with $w(S) \leq k$ and $S \cap F = \emptyset$. If $s \in S$, then $\mathcal{I}_s$ is clearly a yes-instance; otherwise if $t \in S$, then $\mathcal{I}_t$ is a yes-instance; otherwise if $u \in S$, then $\mathcal{I}_u$ is a yes-instance; otherwise $v \in S$ and $\mathcal{I}_v$ is a yes-instance. □

An additional feature of Branching Rule 1 is that we can skip branches where a permanent vertex would have been deleted. Combining Branching Rule 1 with the fact that one can find a restricted $P_4$ by running a breadth-first-search starting from each vertex, we arrive at the following.

**Proposition 3** Generalized 2-Club Cluster Vertex Deletion *can be solved in $O(4^k \cdot nm)$ time.*

## 3.2 Data Reduction Rules

Data reduction rules may be considered the most valuable contribution of parameterized algorithmics to algorithm engineering [27]. In this section, we will introduce polynomial-time data reduction rules that can be applied in each step of our search tree algorithm. We categorize these rules into two types: The first type removes vertices and shrinks the graph. The second type increases the set of permanent vertices, which in turn can trigger data reduction rules of the first type, and decreases the number of branches

**Shrinking the Graph.** We start with describing rules of the first type. First, note that we can always safely remove a connected component of the graph that is already a 2-club.

**Reduction Rule 1** *If a graph contains a connected component $C$ that is a 2-club, then delete all vertices in $C$.*

Now, consider a graph with $k + 1$ restricted $P_4$'s that only overlap in one vertex $v$ as shown in Fig. 4. We clearly have to delete $v$. This basic observation can be generalized to the case with vertex weights and some overlap in the the restricted $P_4$'s: Let $\mathcal{P}$ be a set of restricted $P_4$'s overlapping in $v$ (and possibly other vertices) with a lower bound of at least $k + 1$ on the cost of resolving the $P_4$'s in $\mathcal{P}$ without removing $v$. Then, we can delete $v$. In here, we need to specify the mentioned lower bound. To this end, our idea is that if each vertex $u$ is contained in at most $w(u)$ many restricted $P_4$'s in $\mathcal{P}$, then each restricted $P_4$ requires a cost of at least one to remove, thus resolving $\mathcal{P}$ costs at least $|\mathcal{P}|$. If each vertex $u$ in a restricted $P_4$ is contained in less than $w(u)$

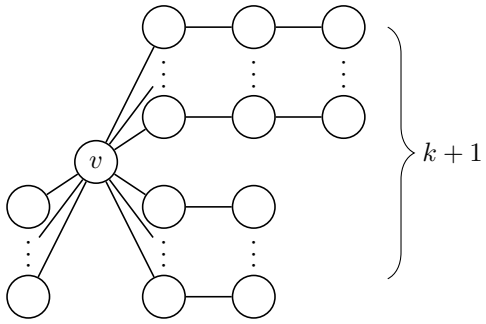



Figure 4: A graph with $k+1$ restricted $P_4$'s that overlap only in vertex $v$.

many restricted $P_4$'s in $\mathcal{P}$, then this lower bound is not tight. We improve the lower bound by allowing $\mathcal{P}$ to be a multiset, that is, we count some restricted $P_4$'s in $\mathcal{P}$ more than once towards the lower bound on the cost. Formally, we arrive at the following data reduction rule.

**Reduction Rule 2** *Let $\mathcal{P}$ be a multiset of restricted $P_4$'s that each contain the vertex $v$ and such that each vertex $u \in V$ other than $v$ is contained in at most $w(u)$ many restricted $P_4$'s in $\mathcal{P}$. If $|\mathcal{P}| \geq k+1$, then delete $v$ and decrease $k$ by $w(v)$.*

**Lemma 2** *Reduction Rule 2 is safe.*

**Proof:** Let $v$ and $\mathcal{P}$ be as above. Clearly, deleting $v$ would eliminate all restricted $P_4$'s in $\mathcal{P}$. Let $u \in V$ be some other vertex. Denote by $\ell$ the number of $P_4$'s in $\mathcal{P}$ that contain $u$, which means that deleting $u$ eliminates $\ell$ $P_4$'s in $\mathcal{P}$. The cost of deleting $u$ is $w(u) \geq \ell$. This means that eliminating all $P_4$'s in $\mathcal{P}$ without deleting $v$ has a cost of at least $|\mathcal{P}| \geq k+1$, which exceeds the budget of $k$. □

For the next data reduction rule, we consider *twins*, that is, two vertices with either the same closed neighborhood or the same open neighborhood.

**Observation 2** *If a graph is a 2-club and contains two twins $u, v$, then after deleting one of them, the graph remains a 2-club. Likewise, a 2-club to which a twin is added remains a 2-club.*

Due to Observation 2, we can delete one of the twins because in an optimal solution they are always in the same 2-club.

**Reduction Rule 3** *Given two vertices $u, v \in V$ such that either $N[u] = N[v]$ or $N(u) = N(v)$, delete $v$ and set $w(u)$ to $w(u) + w(v)$.*

**Lemma 3** *Reduction Rule 3 is safe.*

**Proof:** We have to show that $(G, w, k)$ is a yes-instance if and only if $(G', w', k)$ is a yes-instance.

"⇐": Let $S'$ be an optimal 2-club vertex deletion set with $w'(S') \leq k$. If $S'$ removes $u$, then $S = S' \cup \{v\}$ is a 2-club vertex deletion set for $G$ with $w'(S') = w(S)$. Otherwise, by Observation 2, $S'$ is also a 2-club vertex deletion set for $G$.

"⇒": Let $S$ be an optimal 2-club vertex deletion set with $w(S) \leq k$ and $H = G[V \setminus S]$. We claim that $S$ either removes both $u$ and $v$ or neither of them. Assume without loss of generality that $S$ only removes $u$. Then, by Observation 2, the set $S' = S \setminus \{u\}$ would be a 2-club vertex

deletion set as $u$ and $v$ would be twins in $H' = G[V \setminus S']$. Since $S'$ is smaller than $S$, this is a contradiction to the optimality of $S$. □

If a restricted $P_4$ *stuv* has to be eliminated, then one has to potentially branch into four cases. If some of the vertices of *stuv* are permanent, then this decreases this number branches. If three vertices of *stuv* are permanent, then there is only one possible branch.

**Reduction Rule 4** *If a graph contains a restricted $P_4$ stuv such that only one vertex $x \in \{s, t, u, v\}$ is not permanent, then delete $x$ and decrease $k$ by $w(x)$.*

The next rule can be used to shrink some subgraphs in which all vertices are permanent.

**Reduction Rule 5** *Let $v$ be a vertex in $G$ that is not permanent. Let $C$ be a component of $G - v$ that is a 2-club with its vertices being marked permanent. Let $d$ be the maximum distance of a vertex in $C$ to $v$. Replace $C$ with $d$ new permanent vertices that together with $v$ induce a path of length $d$.*

**Lemma 4** *Reduction Rule 5 is safe.*

**Proof:** Let $C$ and $v$ be as in Reduction Rule 5. Note that the vertices in $C$ are all marked permanent. If any vertex $u \in C$ is part of a restricted $P_4$, then $v$ must also be part of the restricted $P_4$. This means that only the distance between $u$ and $v$ is important and the path with at most $d + 1$ vertices that starts in $v$ is sufficient to represent all vertices that have some distance (which is at most three) to $v$. □

**Increasing the set of Permanent Vertices.** Note that Reduction Rules 4 and 5 both require permanent vertices in order to be triggered. So far, the only way to add new permanent vertices is with Branching Rule 1. Subsequently, we discuss a faster way of producing permanent vertices via data reduction rules.

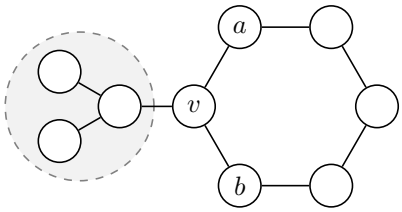
For the following data reduction rule, we need the concept of a *bridge vertex*. We call a vertex $b$ a bridge vertex if for some $s, v \in N(b)$ there exists an induced $P_4$ *stuv* for some $t, u \in V$. We say that $b$ bridges *stuv*.

A vertex $v \in V$ is a *cut vertex* if the deletion of $V$ increases the number of connected components. If a 2-club $C$ becomes isolated by removing a cut vertex $v$, then the vertices of $C$ can be safely excluded from the solution (by marking them as permanent) under the following premise: If any vertex from that 2-club $C$ was included in the solution, then it can be replaced by $v$ to yield another solution of the same size. This premise holds if $v$ is not a bridge vertex and if $w(v) \leq \min_{u \in C} w(u)$ see Fig. 5a. For an example of the application of the corresponding Reduction Rule 6.

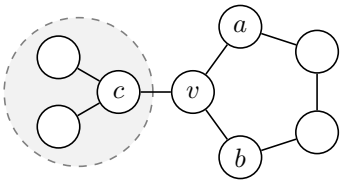
**Reduction Rule 6** *Let $v$ be a cut vertex in $G$ that is not a bridge vertex and not permanent and let $C$ be a component in $G - v$. If $C$ is a 2-club and $w(v) \leq \min_{u \in C} w(u)$, then mark the vertices in $C$ as permanent.*

**Lemma 5** *Reduction Rule 6 is safe.*

**Proof:** If $C$ is a 2-club, then it contains no restricted $P_4$. If any vertex in $C$ is part of a restricted $P_4$, then $v$ must be on this path. Suppose that an optimal solution $S$ removes some vertices $F \subseteq C$, $F \neq \emptyset$. Removing $F$ is only needed to eliminate $P_4$'s that start in $C$, which also necessarily contain $v$. If $v \in S$, then $S$ is not optimal. Otherwise, we claim that $S' = (S \setminus F) \cup \{v\}$ is another optimal solution. This holds true because $v$ eliminates the same $P_4$'s as $F$ does, has




(a) Reduction Rule 6 can be applied on vertex $v$. The gray area contains vertices that can be marked as permanent.




(b) Reduction Rule 6 cannot be applied on vertex $v$ because $v$ is a bridge for $a$ and $b$.

Figure 5: Examples of graphs with a cut vertex $v$ (all vertex weights are one). The gray area is a 2-club that is isolated from the rest of the graph after deleting $v$. Note that in the left graph the removal of $v$ increases the distance of $a$ and $b$ to five. Thus no induced $P_4$ exists between $a$ and $b$.

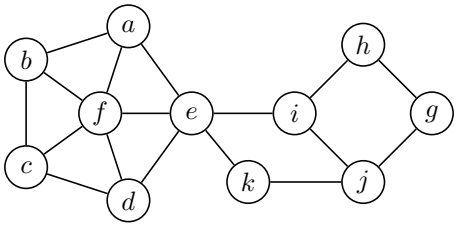



Figure 6: An example to demonstrate robustness (all vertex weights are one). For instance, we have $\text{robust}(a,d) = 2$ because vertices $e$ and $f$ have to be deleted to "promote" the induced $P_4$ *abcd* to a restricted one. Other robustness values are: $\text{robust}(f,j) = \text{robust}(h,k) = 0$ and $\text{robust}(h,j) = \infty$. Two optimal 2-club cluster vertex deletion sets are $\{e,j\}$ and $\{i,k\}$.

weight $w(v) \leq w(F)$, and because $v$ is not a bridge vertex. Hence, it follows that removing $v$ cannot contribute to the creation of a restricted $P_4$. □

See Fig. 5b for an example why we require $v$ to be a non-bridge vertex in Reduction Rule 6.

Reduction Rule 6 can be slightly generalized. To this end, we define the *robustness* of two vertices $s$ and $v$ as the number of vertices that need to be deleted before a restricted $P_4$ can be created between $s$ and $v$:

$$\text{robust}(s,v) := \begin{cases} \infty & \text{if there is no induced } P_4 \textit{ stuv} \text{ for any } t,u \in V, \\ w(N(s) \cap N(v)) & \text{otherwise.} \end{cases}$$

For an example see Fig. 6. An induced $P_4$ *stuv* is a restricted $P_4$ if and only if $\text{robust}(s,v) = 0$. For the induced $P_4$ *stuv* the set $U := N(s) \cap N(v)$ of vertices needs to be deleted before *stuv* is "promoted" to a restricted $P_4$. We will say that the deletion of the vertices in $U$ *contributes to the creation* of the restricted $P_4$ *stuv*.

In Reduction Rule 6, the deletion of $v$ cannot decrease the robustness of any two vertices because $v$ is not a bridge vertex. However, we do not want to restrict consideration to just vertices that cannot decrease robustness. We adapt this rule to also allow $v$ to be a bridge vertex, but we still need to guarantee that the deletion of $v$ cannot contribute to the creation of a restricted $P_4$. For this we consider our remaining budget $k$ and conclude that if the robustness in the neighborhood is

sufficiently high, then we can still mark the 2-clubs as permanent. Additionally, we do not need to consider how removing $v$ affects the robustness between vertices in 2-clubs that would be isolated, because we already know that they are 2-clubs and do not have restricted $P_4$'s.

**Reduction Rule 7** *Given a vertex $v$ in $G$ that is not permanent. Let $C_1, \ldots, C_\ell$ be the components of $G - v$ that are 2-clubs, and $H = G - v - C_1 - \cdots - C_\ell$. If for all pairs of vertices $a, b \in N_G(v) \cap V(H)$ $\text{robust}_G(a, b) > k$ and for all 2-club components $C_i$, $i \in \{1, \ldots, \ell\}$, we have $w(v) \leq \min_{u \in C_i} w(u)$, then mark all vertices in $C_1, \ldots, C_\ell$ as permanent.*

**Lemma 6** *Reduction Rule 7 is safe.*

**Proof:** If $C_i$ is a 2-club, then it contains no restricted $P_4$. If any vertex in $C_i$ is part of a restricted $P_4$, then $v$ must be on this path. Suppose that an optimal solution $S$ removes some vertices $F \subseteq C_i$ with $F \neq \emptyset$. Removing $F$ is only needed to eliminate $P_4$'s that start in $C_i$, which also necessarily contain $v$. If $v \in S$, then $S$ is not optimal. Otherwise we claim that $S' = (S \backslash F) \cup \{v\}$ is another optimal solution. Deleting $v$ reduces the robustness between its neighbors. However, deleting $v$ cannot create a restricted $P_4$. If deleting $v$ created a restricted $P_4$, then this $P_4$ would need to start and end in two neighbors of $v$. Deleting $v$ cuts off the 2-clubs $C_1, \ldots, C_\ell$ which means no restricted $P_4$'s were created in them, which means the neighbors of $v$ in these 2-clubs need not be considered further. The only other vertices that could be affected are those in $U = N_G(v) \cap V(H)$. Because the pairwise robustness of vertices in $U$ is at least $k + 1$, this means that vertices with a total budget greater than $k$ need to be removed before there can be a restricted $P_4$ that starts and ends in $U$. Because the budget $k$ does not allow this to happen, replacing $F$ by the single vertex $v$ with $w(v) \leq \sum_{u \in F} w(u)$ to obtain $S'$ results in another optimal solution. □

A 2-club vertex deletion set $S$ is clearly not optimal if a vertex $v$ can be removed from it and $S' = S \setminus \{v\}$ remains a 2-club vertex deletion set.

**Observation 3** *Let $S$ be a 2-club vertex deletion set of $G$. If $N[v] \subseteq S$ for some $v \in V$, then $S \backslash \{v\}$ is also a 2-club vertex deletion set.*

One could use Observation 3 as a simple test of whether a 2-club vertex deletion set $S$ is minimal. Clearly we do not have to wait until we have found a 2-club vertex deletion set $S$ to apply this test. A *partial 2-club vertex deletion set* $S'$ is a set of vertices which were removed from $G$ along the way from the root to a branching node of the search tree. The test can be applied to $S'$ in the same way as if it would be applied to $S$. Additionally, if the removal of any vertex in $G$ would cause this test to fail, then this vertex must not be removed.

**Reduction Rule 8** *Let $S'$ be a partial 2-club vertex deletion set of $G$ constructed at some stage of the branching. If $|N(v) \setminus S'| = 1$ for any $v \in S'$, then mark the unique vertex $x \in N(v) \setminus S'$ as permanent.*

**Lemma 7** *Reduction Rule 8 is safe.*

**Proof:** Let $S'$ and $x$ be as above. Any 2-club vertex deletion set $S$ with $(S' \cup \{x\}) \subseteq S$ is not a minimal deletion set by Observation 3. This implies that $x$ cannot be in any minimal solution containing $S'$ and hence we can mark $x$ as permanent. □

### 3.3 Lower Bounds

Another way to shrink the size of a search tree are *lower bounds*. A lower bound can be thought of as a function $\ell(G)$ of the graph $G$ such that $\ell(G) \leq |S|$, where $S$ is an optimal 2-club vertex deletion set for $G$. Lower bounds are a very practical way to shrink the size of the search tree, because if once for the solution size parameter $k$ it holds that $k < \ell(G)$, then we know that there is no solution. The basic idea behind our first lower bound was already used in Reduction Rule 2 (see also discussion before Reduction Rule 2).

**Lower Bound 1** *Let $\mathcal{P} = \{p_1, \ldots, p_\ell\}$ be a multiset of restricted $P_4$'s in $G$ such that each vertex $v \in V$ is contained in at most $w(v)$ many restricted $P_4$'s in $\mathcal{P}$. Then a minimum vertex deletion set for $G$ has size at least $\ell$.*

**Proof:** Let $\mathcal{P}$ be as above and $v \in V$ be any vertex. Denote by $r$ the number of restricted $P_4$'s in $\mathcal{P}$ that contain $v$, which means deleting $v$ eliminates $r$ restricted $P_4$'s in $\mathcal{P}$. The cost of deleting $v$ is $w(v) \geq r$. This means eliminating all restricted $P_4$'s in $\mathcal{P}$ has a cost of at least $|\mathcal{P}|$. Clearly, if a restricted $P_4$ in $\mathcal{P}$ is not eliminated, then the graph is not a 2-club cluster graph. □

Next, we exploit the size of a minimum vertex cut set. A *cut set* $D$ of a graph $G$ is a set of vertices such that $G - D$ is disconnected. We know that an optimal 2-club vertex deletion set $S$ for $G$ splits it into one or more 2-clubs. This means that $S$ must be a vertex cut set or $G - S$ is a single 2-club. The following lower bound incorporates these options:

**Lower Bound 2** *Let $G$ be a connected graph with vertex weights, let $C$ be the maximum-weight 2-club in $G$, and let $D$ be the minimum-weight vertex cut set of $G$. Then a minimum-weight 2-club vertex deletion set for $G$ has size at least* $\min(w(V \setminus C), w(D))$.

**Proof:** Let $S$ be an optimal 2-club vertex deletion set and let $G' := G[V \setminus S]$ be the resulting 2-club cluster graph. If $G'$ contains at least two components, then $S$ is a vertex cut set for $G$ and $w(S) \geq w(D)$. If $G'$ has only one component, then $V \setminus S = V(G')$ is the maximum-weight 2-club in $G$ and $w(S) = w(V \setminus C)$. □

## 4 Implementation of the Branch&Bound Algorithm

In this section, we discuss some implementation details of our algorithm for Gen2CVD that have been left open in Section 3. For example we did not say how to compute a set of restricted $P_4$'s used in Lower Bound 1. It is clear that ideally we would like to find such a set with maximum size. This is likely an NP-hard problem (refer to Itai et al. [25]), which is why in a practical implementation we would rather have a fast heuristic that offers good results in most cases. We will now discuss heuristics used in our solver (see Section 5 for corresponding experimental results) and other implementation challenges of interest.

**Determining the Minimum 2-Club Vertex Deletion Set Size.** In order to find the minimum 2-club vertex deletion set size we simply try increasing values for the budget $k$, as can be seen in Algorithm 1. Naturally, our solver also outputs the 2-club vertex deletion set that was found.

**Algorithm 1** Finding a minimum 2-club vertex deletion set.

**Input:** An undirected graph $G = (V, E)$
**Output:** A minimum 2-club vertex deletion set of $G$
1: initialize the weight function $w$ with $w(v) = 1$ for each $v \in V$
2: $(G', \infty, F', w') \leftarrow$ apply data reduction rules to $(G, k = \infty, F = \emptyset, w)$
3: $k \leftarrow$ LOWER BOUND$(G', w', F')$
4: **while** $(G', k, F', w')$ is a no-instance of Gen2CVD **do** // Algorithm described in Section 3
5: $\quad k \leftarrow k + 1$
6: **return** found solution of size $k$

**Branching Rule 1.** This is the branching rule that allows us to mark some vertices as permanent and to skip branches in which a permanent vertex would have been deleted (see Section 3.1). It also allows us to freely choose any restricted $P_4$ to branch on. It is highly advantageous to choose a $P_4$ that contains permanent vertices because for each permanent vertex we are allowed to skip one out of a total of four branches. For this reason we select a restricted $P_4$ that contains the most permanent vertices, and if there is more than one, then we select the one in which the average weight of non-permanent vertices is the highest (ties broken arbitrarily) because then on average $k$ is decreased by a larger value in the branches and thus the search tree is smaller.

**Handling Multiple Connected Components.** Each connected component can be solved separately, potentially reducing the search space drastically. However, we do not know how to distribute the budget $k$ among these components. As in Algorithm 1, we try to solve each component with as little budget as possible, first trying small values for $k$ and then increasing it by one each time. An improvement is to sort all components by size and when solving the last component to give it all remaining budget, which prevents us from trying many different $k$-values for the last component. In a single iteration, this is a only a minor improvement. However, the effect is much more noticeable when the graph repeatedly decomposes into one large component and a few smaller ones.

**Disjoint Restricted $P_4$'s.** Reduction Rule 2 allows us to delete a vertex $v$ if there is a multiset $\mathcal{P}$ of $k + 1$ restricted $P_4$'s that each contain $v$, but otherwise each vertex $u$ can only be present in at most $w(u)$ many restricted $P_4$'s. We would like to find a maximum set $\mathcal{P}$ and then test if its size is at least $k + 1$. However, this proved to be quite challenging. For our implementation we use a heuristic that does not guarantee finding a maximum set.

We focus on finding a maximum set that only contains those restricted $P_4$'s that start in vertex $v$. This means we do not try to find $P_4$'s where $v$ might be in the "middle" (see Fig. 4). This can then be modeled as a simple maximum flow problem, see Algorithm 2 for pseudo code.

Because for a restricted $P_4$ $stuv$ the distance from $v$ to $u$, $t$, and $s$ is one, two and three, respectively, we partition the vertices in the graph into three sets $D_1, D_2, D_3$. No restricted $P_4$ can contain two vertices from the same $D_i$, which is why our flow graph only contains edges from $D_i$ to $D_{i+1}$. We make sure that a vertex $u$ is part of at most $w(u)$ many $P_4$'s by splitting it into two vertices connected by an edge with $w(u)$ capacity. As a result there can be only flow along paths of type $su_1^{\text{in}}u_1^{\text{out}}u_2^{\text{in}}u_2^{\text{out}}u_3^{\text{in}}u_3^{\text{out}}t$, and sending a flow of value 1 along such a path means adding the restricted $P_4$ $vu_1u_2u_3$ to $\mathcal{P}$. The final maximum flow in $G_{\text{flow}}$ does not uniquely identify a set $\mathcal{P}$; however, the value of the maximum flow tells us the size of all maximum size $\mathcal{P}$'s. Because we are only interested in the size of the set $\mathcal{P}$, this is all we need.

**Algorithm 2** Heuristic for Reduction Rule 2

**Input:** A graph $G = (V, E)$, with a weight function $w$, a vertex $v$ and $k \in \mathbb{N}$
**Output:** `true` if Reduction Rule 2 can be applied to remove $v$ from $G$

Create a directed graph $G_{\text{flow}}$ containing only the vertices $s$ and $t$
**for each** $i \in \{1, 2, 3\}$ // Split vertices into layers based on distance
    $D_i \leftarrow$ all vertices with distance $i$ to $v$ in $G$
**for each** $u \in D_1 \cup D_2 \cup D_3$ // Limit flow through a vertex
    add the vertices $u_{\text{in}}$ and $u_{\text{out}}$ to $G_{\text{flow}}$
    add the edge $(u_{\text{in}}, u_{\text{out}})$ with a capacity of $w(u)$ to $G_{\text{flow}}$
**for each** $u \in D_1$ // Connect layer 1 to source
    add the edge $(s, u_{\text{in}})$ with infinite capacity to $G_{\text{flow}}$
**for each** $i \in \{1, 2\}$ // Connect the layers
    **for each** $u \in D_i$
        **for each** $x \in D_{i+1}$ with $\{u, x\} \in E$
            add the edge $(u_{\text{out}}, x_{\text{in}})$ with infinite capacity to $G_{\text{flow}}$
**for each** $u \in D_3$ // Connect layer 3 to sink
    add the edge $(u_{\text{out}}, t)$ with infinite capacity to $G_{\text{flow}}$
$f \leftarrow$ maximum $s$-$t$ flow in $G_{\text{flow}}$
**if** $f > k$ **then return** `true`
**return** `false`

**Disjoint Restricted $P_4$'s Lower Bound.** Here we use a multiset $\mathcal{P}$ of restricted $P_4$'s such that each vertex $v$ is present in at most $w(v)$ many restricted $P_4$'s. The size of this set is then the lower bound. We compute the set $\mathcal{P}$ using a greedy heuristic. Each vertex has a counter initialized with the value of its weight. This counter keeps track of how many times this vertex can be used in a restricted $P_4$. We iterate over all vertices in $V$ by increasing degree and for each vertex $s$ we look for a restricted $P_4$ $stuv$ such that for all four vertices of this $P_4$ their counter is positive. The restricted $P_4$ is not chosen randomly, but rather we select a $P_4$ that minimizes the sum of degrees of its vertices. Finding such a $P_4$ takes $\mathcal{O}(n+m)$ time. The $P_4$ $stuv$ is then implicitly added to $\mathcal{P}$ by decrementing the counter for $s, t, u$ and $v$ by one. If the counter for $s$ remains positive, then we repeat this step and search for another $P_4$.

The reason for minimizing the sum of the degrees is that selecting a $P_4$ which contains many high-degree vertices might overlap with—and therefore likely exclude—many other restricted $P_4$'s (see Fig. 7).

# 5 Experimental Evaluation

In this section, we present experimental results for our Gen2CVD solver. In Section 5.1, we describe the used solvers and their configurations. In Section 5.2, we discuss our biological dataset. Then in Section 5.3, we describe our experiments.

## 5.1 Setup

We implemented our branch&bound algorithm (see Sections 3 and 4) for Generalized 2-Club Cluster Vertex Deletion in C++ (we use the algorithm to solve 2-Club Cluster Vertex

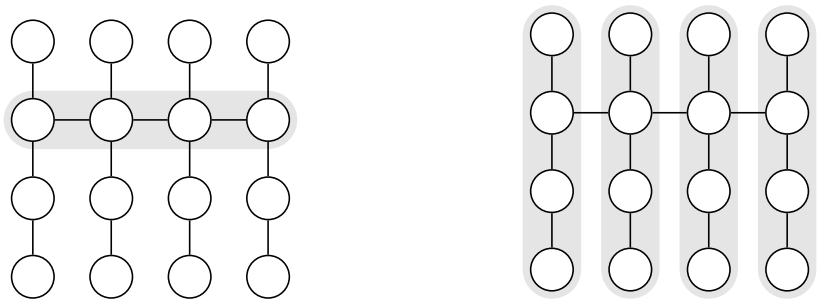


Figure 7: Two maximal sets of disjoint restricted $P_4$'s in the same graph, the corresponding restricted $P_4$'s are represented by gray paths. The restricted $P_4$ on the left contains the highest degree vertices, each of which could have been in its own disjoint restricted $P_4$ like on the right. Our heuristic for Lower Bound 1 (see Section 3.3) prevents such a bad case.

Deletion).[4] This solver (called `solverALL` in the following) computes a 2-club vertex deletion set size of minimum cost and outputs the solution set. It uses all data reduction rules described in Section 3.2 and Lower Bound 1.[5] Note that the implementations of some data reduction rules and lower bounds are based on heuristics; see Section 4 for details.

We will compare the performance of our solver against the ILP formulation from Section 3 solved using CPLEX (we will refer to this solver as `CPLEX`). All experiments were run on a machine with an Intel Xeon W-2125 8-core, 4.0 GHz CPU and 256GB of RAM running Ubuntu 18.04. We used a recent version of CPLEX, 12.8, for our experiments. Analogously to previous work for the related Cluster Editing [6], we focus on implementations computing optimal solutions. Thus, we also require CPLEX to compute optimal solutions.

We use mostly default parameters and only set `mip tolerances mipgap` and `absmipgap` to zero and enabled `emphasis numerical`. CPLEX can use up to 32 threads by default. Even though we had 8 cores available, in our experiments CPLEX usually only used four. This is an advantage of CPLEX, because our solver was only written to use a single thread. Our solver only needs up to 20MB of RAM, but we have seen CPLEX to use even 30GB and more of RAM on the instances described in Section 5.2. For the running time measurements of our solver and CPLEX we used wallclock time. For running time measurements of CPLEX we excluded the time it takes to build the ILP model, which can have $\Theta(n^4)$ constraints. For instances with 250 vertices this process can take 20 seconds, and sometimes even 60. However, in the vast majority of cases, the build time was at most 30% of the total running time.

## 5.2 Dataset

For our analysis we used a real-world biological dataset[6] that has been used for the evaluation of Weighted Cluster Editing solvers [5, 34]. The vertices in the graphs represent protein sequences and between each pair of vertices there is an edge whose weight represents some sort of similarity of the proteins. The edge weights can be positive or negative.

[4] The source code is available at `https://fpt.akt.tu-berlin.de/software/two-club-editing/two-club-vertex-deletion.zip` and includes the source code for the ILP formulation using CPLEX.

[5] Our implementation of Lower Bound 2 in Section 3.3 was far too slow in order to be of use.

[6] The dataset is available at `https://bio.informatik.uni-jena.de/data/#cluster_editing_data`

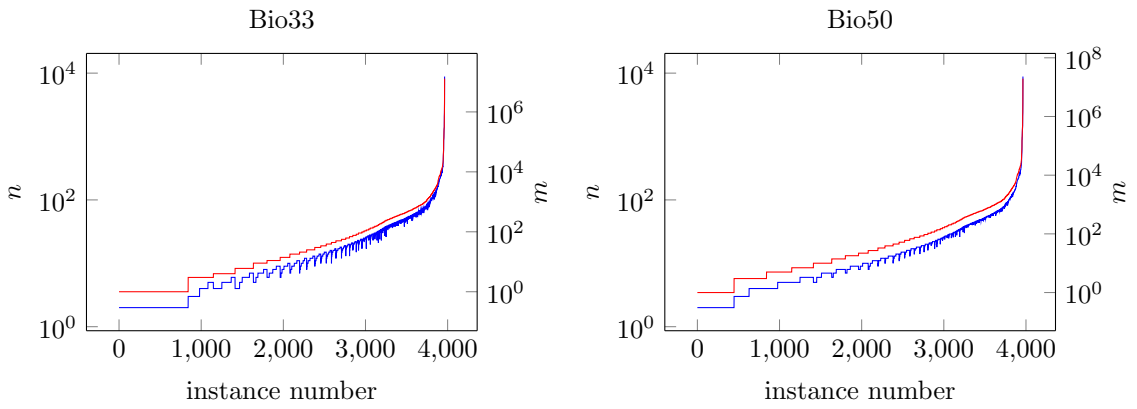



Figure 8: Two graphs showing the number $n$ of vertices in blue (lower line) and the number $m$ of edges in red (upper line) in the Bio33 (left) and Bio50 (right) instances (on a log-scale). The instance numbers for the Bio33 and Bio55 instances were selected such that the number of edges increases with the instance number. There are about 15 instances with more than 500 vertices, the largest of which has nearly 9000 vertices.

A graph with weighted edges does not match the input of 2-Club Cluster Vertex Deletion. Hartung and Hoos [21] used the following conversion for their (unweighted) Cluster Editing solver: first sort the edges by descending weight, keep the first $c\%$ of edges for some $c \in [0, 100]$ and discard their weight. We additionally delete degree zero vertices from the graphs. Hartung and Hoos [21] used the values $c = 33$, $c = 50$, and $c = 66$, which is also what we did, and they obtained three datasets, which we will refer to as Bio33, Bio50 and Bio66, respectively. Our experiment results for Bio66 are fairly similar to Bio50, which is why we will only discuss results for Bio33 and Bio50.

The Bio33 and Bio50 datasets each contain 3964 instances. See Fig. 8 for the number of vertices and edges in the instances. The "noise" in the number of vertices is a result of deleting degree-zero vertices from the graphs. In Fig. 8 we can see that these datasets contain many instances with less than 50 vertices and a few with around 8000 vertices. Our results show that the vast majority of instances with less than 100 vertices can be solved within less than a second. For this reason we focused on the harder instances. From each dataset we only kept instances with 50–250 vertices. (On the larger instances the main memory of 256GB in our machine was insufficient for solving the ILP.) After this filtering, Bio33 contains 430 instances, whereas Bio50 contains 446 instances.

## 5.3 Results

We next analyze the performance of our solver in detail. To this end, we start with comparing the theoretical bounds with the results of our experiments. The number of branches in our search tree is far below the theoretical worst-case bound of $4^k$ (even far below the $3.31^k$ bound of the search tree of Liu et al. [30]) given in Proposition 3 (see Fig. 9). This is a clear indication that the data reduction rules and lower bounds have a strong impact on our solver. Another observation derived from Fig. 9 is that the impact of the number of input graph vertices on the running time is quite significant. The reason for this is the high polynomial running time for executing the data reduction rules and computing lower bounds: Our best upper bound on the running time (in

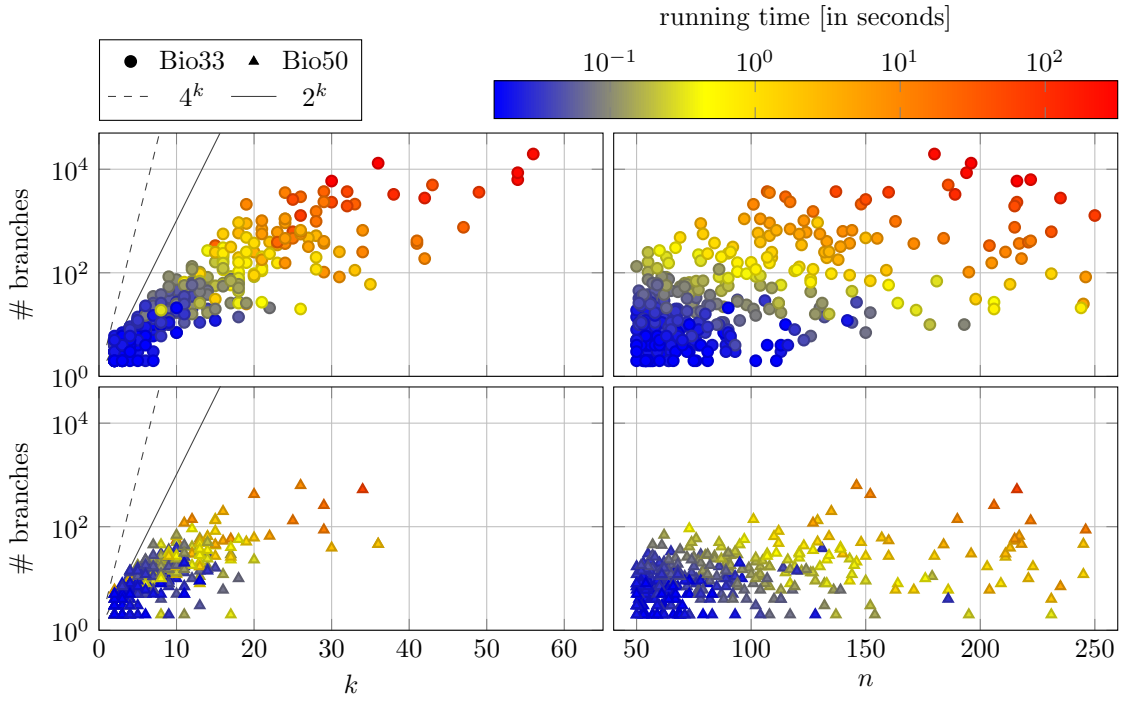



Figure 9: Diagrams illustrating the impact of $k$ and $n$ on the running time and number of branches (that is, number of times the branching function is called). All four diagrams use the same scale on the $y$-axis. The diagrams on top of each other also use the same $x$-axis. The values on the $x$-axis are the optimal solution size (left two diagrams) and number of input graph vertices (right two diagrams). The top two diagrams show the results for the Bio33-instances; the bottom two for the Bio50 instances.

terms of $n$) of one recursive step (including data reduction and lower bounds) is $O(n^4)$. One of the bottlenecks in the running time is Reduction Rule 2, where we solve up to $n$ maximum flow instances. We show subsequently that the high running-time cost for the data reduction rules is justified.

Fig. 9 also displays that the Bio33 instances are in general harder for our solver than the Bio50 instances. The reason for this is that the Bio50 instances are more dense and allow to cluster into less 2-clubs of larger size with fewer vertex removals.

**Comparisons.** We next compare our solver `solverALL` to several variants of itself where we deactivate key features, and to `CPLEX`. The comparisons are illustrated in Fig. 10. The first row of plots in Fig. 10 shows that if we deactivate the data reduction rules, then the performance becomes much worse, especially on the harder instances that require more than 10 seconds to solve. On average, `solverALL` (with all data reduction rules) is 6.7 times faster on the Bio33 instances and 3 times faster on the Bio50 instances than our solver without data reduction rules. This is in stark contrast to the kernelization lower bound given in Theorem 2 and gives hope to find small parameters based on which one may perform a mathematical analysis yielding polynomial kernel

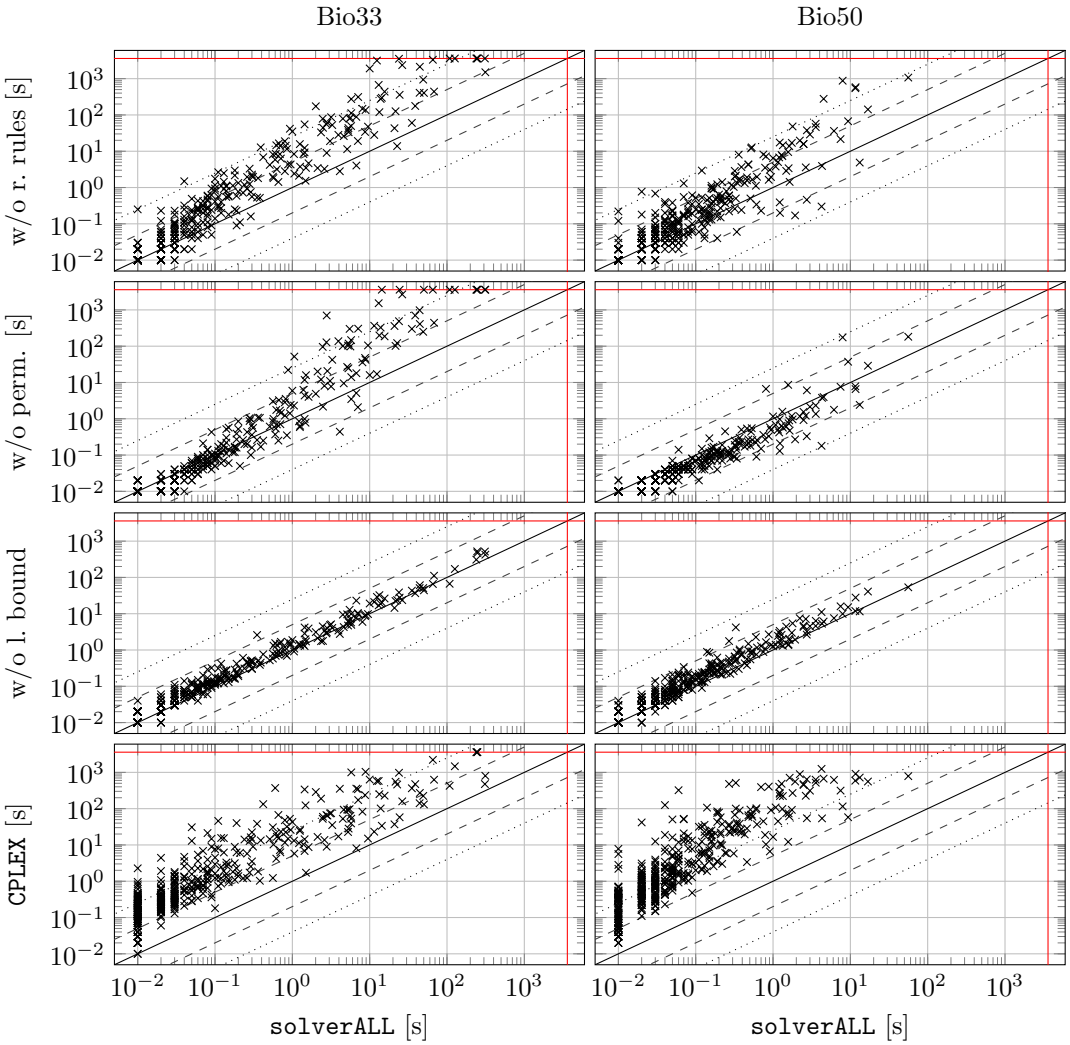



Figure 10: Running time comparison (in seconds) of different configurations of our solver and CPLEX on two datasets (left: Bio33, right: Bio55). Each cross represents one instance with the $x$- and $y$-coordinates indicating the running time of the respective solver (in seconds). Thus, a cross above (below) the solid diagonal indicates that the solver on the $x$-axis ($y$-axis) is faster on the corresponding instance. The diagonal lines mark running time factors of 1 (solid), 5 (dashed) and 25 (dotted). Crosses on the solid horizontal red line (at 3600 seconds) indicate timeouts. In each plot the running time of our `solverALL` is displayed at the $x$-axis. The $y$-axis shows in each row of the plots a different solver; these are from top to bottom: Three configurations of our solver where certain features are disabled (first all data reduction rules, then permanent vertices with the corresponding data reduction rules that require permanent vertices (Reduction Rules 5 to 8), finally without Lower Bound 1). The last row shows the comparison against CPLEX.

sizes.

The plots in the second row of Fig. 10 show the effect of turning off permanent vertices and the corresponding data reduction rules (Reduction Rules 5 to 8). Note that in the Bio50 dataset the variant without permanent vertices is faster on most instances, very likely due to Reduction Rule 7 being an expensive data reduction rule. However, the results for the Bio33 dataset show a different picture. In fact, one can see in both data sets that turning off the feature of permanent vertices solves the easier instances even faster and slows down the solver on the harder instances. The lack of "hard" instances in the Bio50 dataset (see also Fig. 9) is the reason for the variant without permanent vertices being faster there. On average, `solverALL` (with permanent vertices) is 5 times faster on the Bio33 instances but 1.6 times slower on the Bio50 instances.

The plots in the third row of Fig. 10 show that the impact of Lower Bound 1 is much smaller than the impact of the data reduction rules and the permanent vertices. While the `solverALL` is faster on most instances, the gap does not improve for the harder instances as in the previous two comparisons (see row one and two in Fig. 10). On average, `solverALL` (with lowerbounds) is 1.4 times faster on the Bio33 instances and 1.5 times faster on the Bio50 instances. This indicates that the data reduction rules and permanent vertices are more impactful than Lower Bound 1.

The plots in the last row of Fig. 10 show that our solver is almost always faster than CPLEX by a factor of 5–25 for Bio33 and a factor of 25–100 for Bio50. On average, `solverALL` is 29.3 times faster on the Bio33 instances and 103.6 times faster on the Bio50 instances. For Bio33 it appears that for harder instances CPLEX is not much slower than our solver. On Bio50, CPLEX does very poorly compared to our solver. This is likely due to the minimum 2-club vertex deletion set size (the parameter in our FPT-algorithm) on these graphs being smaller than for Bio33. Moreover, the process for building the ILP model for CPLEX is usually fairly fast, but for larger instances it can take up to 60 seconds. For example, in Bio50 there is a graph with 205 vertices and 10455 edges which is already a 2-club cluster graph. It takes about 50 seconds to create the ILP model, and when exported to a file it takes up to 1.6GB (uncompressed) and includes 5.8 million constraints, whereas the original graph only takes up 72kB stored in an edge list format.

Summarizing, our solver outperforms a standard ILP-formulation solved with CPLEX. To this end, good data reduction rules are crucial to the practical performance of our solver. Testing whether the performance of the ILP can be improved by the use of data reduction and row generation approaches remains a task for future work.

**2-Club Cluster Vertex Deletion Solutions.** In Fig. 11 (top row) we compare the sizes of a minimum cluster vertex deletion set (CVD) and a minimum 2-club vertex deletion set (2CVD) on our datasets. For Bio33 there is a much stronger correlation between these two values than for Bio50. For Bio33 the CVD solution size is around 2–4 times larger than the 2CVD solution size, but on many Bio50 instances the CVD solution size can be very large while the 2CVD solution size is below five.

We next compare the number of clusters created by both problems, as shown in the bottom row of Fig. 11. As expected, the 2CVD solution creates for most instances much less clusters (while deleting fewer vertices). Note that all solvers we employ compute minimum-size solutions where the number of clusters is not optimized. Thus, if there are multiple optimal solutions that create a different number of clusters, then we have no control which optimal solution is picked. We believe that this issue causes two instances (one Bio33 and one Bio50 instance) having a smaller number of clusters when using the CVD solution (see the two points above the solid line in Fig. 11).

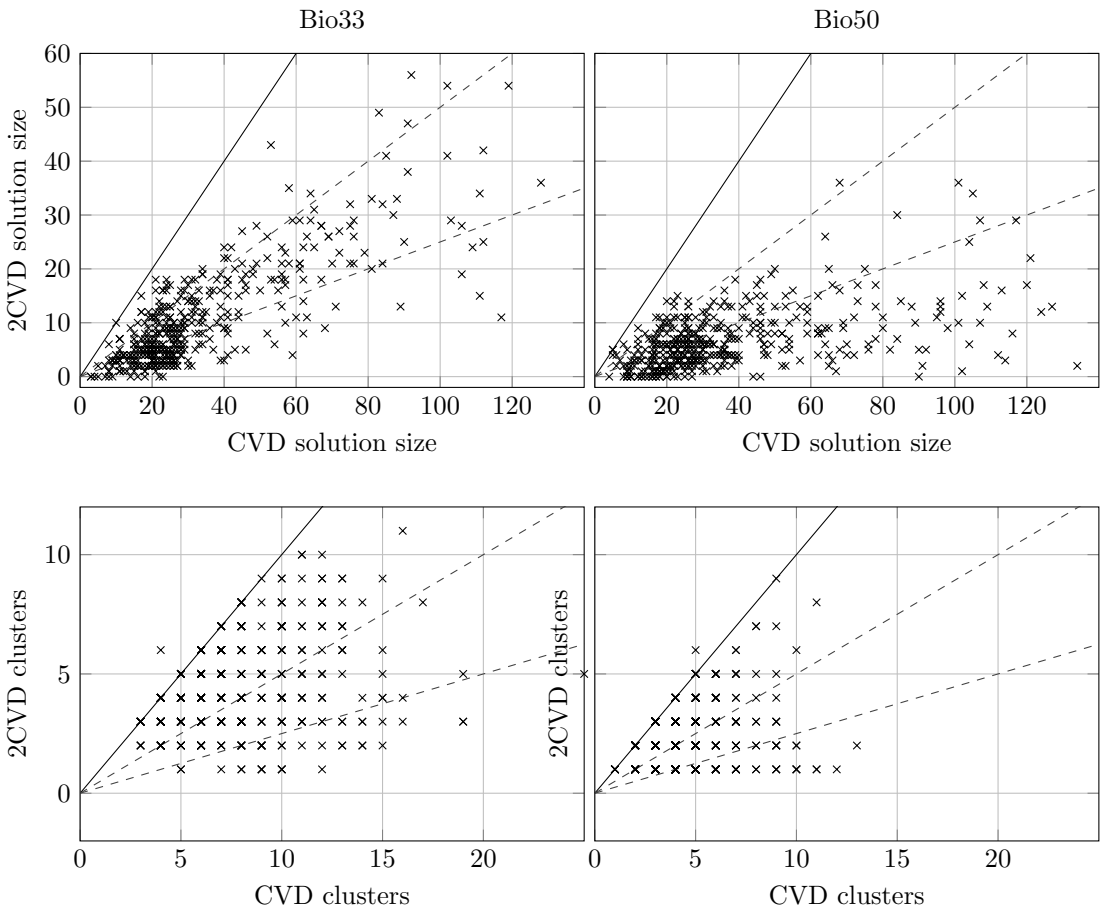



Figure 11: Comparison of the solution size (top row) and cluster size for Cluster Vertex Deletion (≙1-Club Cluster Vertex Deletion) and 2-Club Cluster Vertex Deletion for two datasets (left: Bio33, right: Bio55). The solid line is $y = x$, and the dashed ones are $y = 0.5x$ and $y = 0.25x$

# 6 Conclusion

We investigated the NP-hard task of modifying graphs into 2-club cluster graphs. While we proved W[2]-hardness for 2-Club Cluster Editing parameterized by the solution size, for 2-Club Cluster Vertex Deletion we faced a somewhat more positive complexity behavior, including both fixed-parameter tractability and a no-polynomial-size problem kernel result. Indeed, the results for 2-Club Cluster Vertex Deletion were promising enough to further study the potential of algorithms to be used in applications. Indeed, we developed and engineered a competitive branch&bound algorithm algorithm for the fixed-parameter tractable 2-Club Cluster Vertex Deletion problem.

On the theoretical side, we left open whether our "no-poly-kernel" result for 2-Club Cluster Vertex Deletion parameterized by solution size transfers to the closely related 2-Club Clus-

ter Edge Deletion problem, a further open problem from the literature [9, 30]. Moreover, it would be interesting to see whether our results also generalize to using $s$-clubs with $s \geq 3$. As to other 2-club-related graph modification problems deserving to be studied, one could consider overlapping clusters [16] or use stricter 2-club models such as well-connected 2-clubs [28, 39]. Limiting the number of local manipulations [29] is another restriction worthwhile investigations. On the empirical and algorithm engineering side, note that while our solver showed strong performance when working with biological data, preliminary experiments showed that this is less so when attacking social network data: Our solver successfully dealt with only five of the 58 test instances taken from the 10th DIMACS challenge [10] whereas the ILP solved 10 of these instances. In each of the five instances the practically measured search tree size of our solver was lower than $2^k$ (as in the biological data set). While these bounds are better than the best known theoretical bounds for 2-Club Cluster Vertex Deletion from Liu et al. [30], these instances are considerably larger than the biological data and thus the polynomial overhead is too large (see also Fig. 9). A further extended algorithm engineering effort with a focus on large sparse graphs seems necessary to tackle these instances.

**Acknowledgment.** We thank the anonymous reviewers for their valuable feedback.

## References

[1] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1–3):89–113, 2004. doi:10.1023/B:MACH.0000033116.57574.95.

[2] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3/4):281–292, 1999. doi:10.1089/106652799318274.

[3] R. van Bevern, H. Moser, and R. Niedermeier. Approximation and tidying—A problem kernel for $s$-plex cluster vertex deletion. *Algorithmica*, 62(3-4):930–950, 2012. doi:10.1007/s00453-011-9492-7.

[4] S. Böcker and J. Baumbach. Cluster editing. In *Proceedings of the 9th International Conference on Computability in Europe (CiE '13)*, volume 7921 of *LNCS*, pages 33–44. Springer, 2013. doi:10.1007/978-3-642-39053-1_5.

[5] 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. doi:10.1016/j.tcs.2009.05.006.

[6] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011. doi:10.1007/s00453-009-9339-7.

[7] H. L. Bodlaender, B. M. P. Jansen, and S. Kratsch. Kernelization lower bounds by cross-composition. *SIAM Journal on Discrete Mathematics*, 28(1):277–305, 2014. doi:10.1137/120880240.

[8] A. Boral, M. Cygan, T. Kociumaka, and M. Pilipczuk. A fast branching algorithm for cluster vertex deletion. *Theory of Computing Systems*, 58(2):357–376, 2016. doi:10.1007/s00224-015-9631-7.

[9] C. Crespelle, P. G. Drange, F. V. Fomin, and P. A. Golovach. A survey of parameterized algorithms and the complexity of edge modification. *CoRR*, abs/2001.06867, 2020. URL: https://arxiv.org/abs/2001.06867.

[10] DIMACS. Graph partitioning and graph clustering. 10th DIMACS challenge, 2012.

[11] R. Dondi and M. Lafond. On the tractability of covering a graph with 2-clubs. In *Proceedings of the 22nd International Symposium on Fundamentals of Computation Theory (FCT '19)*, volume 11651 of *LNCS*, pages 243–257. Springer, 2019. doi:10.1007/978-3-030-25027-0_17.

[12] R. Dondi, G. Mauri, F. Sikora, and I. Zoppis. Covering a graph with clubs. *Journal of Graph Algorithms and Applications*, 23(2):271–292, 2019. doi:10.7155/jgaa.00491.

[13] R. Dondi, G. Mauri, and I. Zoppis. On the tractability of finding disjoint clubs in a network. *Theoretical Computer Science*, 777:243–251, 2019. doi:10.1016/j.tcs.2019.03.045.

[14] M. Doucha and J. Kratochvíl. Cluster vertex deletion: A parameterization between vertex cover and clique-width. In *Proceedings of the 37th International Symposium on Mathematical Foundations of Computer Science (MFCS '12)*, volume 7464 of *LNCS*, pages 348–359. Springer, 2012. doi:10.1007/978-3-642-32589-2_32.

[15] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Springer, 2013.

[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. doi:10.1016/j.disopt.2010.09.006.

[17] A. Figiel, A.-S. Himmel, A. Nichterlein, and R. Niedermeier. On 2-clubs in graph-based data clustering: Theory and algorithm engineering. In *Proceedings of the 12th International Conference on Algorithms and Complexity (CIAC '21)*, volume 12701 of *LNCS*, pages 216–230. Springer, 2021. doi:10.1007/978-3-030-75242-2_15.

[18] Y. Gao, D. R. Hare, and J. Nastos. The parametric complexity of graph diameter augmentation. *Discrete Applied Mathematics*, 161(10-11):1626–1631, 2013. doi:10.1016/j.dam.2013.01.016.

[19] 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. doi:10.1007/s00224-004-1178-y.

[20] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. A more relaxed model for graph-based data clustering: $s$-plex cluster editing. *SIAM Journal on Discrete Mathematics*, 24(4):1662–1683, 2010. doi:10.1137/090767285.

[21] S. Hartung and H. H. Hoos. Programming by optimisation meets parameterised algorithmics: A case study for cluster editing. In *Proceedings of the 9th Learning and Intelligent OptimizatioN Conference (LION '15)*, volume 8994 of *LNCS*, pages 43–58. Springer, 2015. doi:10.1007/978-3-319-19084-6_5.

[22] S. Hartung, C. Komusiewicz, and A. Nichterlein. Parameterized algorithmics and computational experiments for finding 2-clubs. *Journal of Graph Algorithms and Applications*, 19(1):155–190, 2015. doi:10.7155/jgaa.00352.

[23] S. Hartung, C. Komusiewicz, A. Nichterlein, and O. Suchý. On structural parameterizations for the 2-club problem. *Discrete Applied Mathematics*, 185:79–92, 2015. doi:10.1016/j.dam.2014.11.026.

[24] 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. doi:10.1007/s00224-008-9150-x.

[25] A. Itai, Y. Perl, and Y. Shiloach. The complexity of finding maximum disjoint paths with length constraints. *Networks*, 12(3):277–286, 1982. doi:10.1002/net.3230120306.

[26] S. Jia, L. Gao, Y. Gao, J. Nastos, X. Wen, X. Huang, and H. Wang. Viewing the meso-scale structures in protein-protein interaction networks using 2-clubs. *IEEE Access*, 6:36780–36797, 2018. doi:10.1109/ACCESS.2018.2852275.

[27] C. Komusiewicz, A. Nichterlein, and R. Niedermeier. Parameterized algorithmics for graph modification problems: On interactions with heuristics. In *Proceedings of the 41st International Workshop on Graph-Theoretic Concepts in Computer Science (WG '15)*, volume 9224 of *LNCS*, pages 3–15. Springer, 2015. doi:10.1007/978-3-662-53174-7_1.

[28] C. Komusiewicz, A. Nichterlein, R. Niedermeier, and M. Picker. Exact algorithms for finding well-connected 2-clubs in sparse real-world graphs: Theory and experiments. *European Journal of Operational Research*, 275(3):846–864, 2019. doi:10.1016/j.ejor.2018.12.006.

[29] C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. doi:10.1016/j.dam.2012.05.019.

[30] H. Liu, P. Zhang, and D. Zhu. On editing graphs into 2-club clusters. In *Proceedings of the Joint International Conference Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM '12)*, volume 7285 of *LNCS*, pages 235–246. Springer, 2012. doi:10.1007/978-3-642-29700-7_22.

[31] D. Lokshtanov, N. Misra, G. Philip, M. S. Ramanujan, and S. Saurabh. Hardness of $r$-dominating set on graphs of diameter $(r+1)$. In *Proceedings of the 8th International Symposium on Parameterized and Exact Computation (IPEC '13)*, volume 8246 of *LNCS*, pages 255–267. Springer, 2013. doi:10.1007/978-3-319-03898-8_22.

[32] N. Misra, F. Panolan, and S. Saurabh. Subexponential algorithm for $d$-cluster edge deletion: Exception or rule? *Journal of Computer and System Sciences*, 113:150–162, 2020. doi:10.1016/j.jcss.2020.05.008.

[33] S. Pasupuleti. Detection of protein complexes in protein interaction networks using $n$-clubs. In *Proceedings of the 6th European Conference on Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics (EvoBIO '08)*, volume 4973 of *LNCS*, pages 153–164. Springer, 2008. doi:10.1007/978-3-540-78757-0_14.

[34] S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truss, and S. Böcker. Exact and heuristic algorithms for weighted cluster editing. In *Proceedings of the 6th Computational Systems Bioinformatics Conference (CSB '07)*, pages 391–401. World Scientific, 2007. URL: https://doi.org/10.1142/9781860948732_0040.

[35] A. Schäfer, C. Komusiewicz, H. Moser, and R. Niedermeier. Parameterized computational complexity of finding small-diameter subgraphs. *Optimization Letters*, 6(5):883–891, 2012. doi:10.1007/s11590-011-0311-5.

[36] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1–2):173–182, 2004. doi:10.1016/j.dam.2004.01.007.

[37] D. Tsur. Faster parameterized algorithm for cluster vertex deletion. *Theory of Computing Systems*, 65(2):323–343, 2021. doi:10.1007/s00224-020-10005-w.

[38] E. Ulitzsch, Q. He, V. Ulitzsch, A. Nichterlein, H. Molter, R. Niedermeier, and S. Pohl. Combining clickstream analyses and graph-modeled data clustering for identifying common response processes. *Psychometrika*, 86:190–214, 2021. doi:10.1007/s11336-020-09743-0.

[39] O. Yezerska, F. M. Pajouh, and S. Butenko. On biconnected and fragile subgraphs of low diameter. *European Journal of Operational Research*, 263(2):390–400, 2017. doi:10.1016/j.ejor.2017.05.020.