Technische Universität Berlin
Electrical Engineering and Computer Science
Institute of Software Engineering
and Theoretical Computer Science
Algorithmics and Computational Complexity (AKT)

Parameterized Algorithmics of 2-Club Cluster Graph Modification

Bachelor thesis

von Aleksander Figiel

zur Erlangung des Grades „Bachelor of Science“ (B. Sc.)
im Studiengang Informatik

Erstgutachter:	Prof. Dr. Rolf Niedermeier
Zweitgutachter:	Prof. Dr. Stephan Kreutzer
Betreuer:	Anne-Sophie Himmel,
	Dr. André Nichterlein,
	Prof. Dr. Rolf Niedermeier

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und eigenhändig sowie ohne unerlaubte fremde Hilfe und ausschließlich unter Verwendung der aufgeführten Quellen und Hilfsmittel angefertigt habe.

Die selbstständige und eigenständige Anfertigung versichert an Eides statt:

Berlin, den	____________________	____________________________
	Datum	Unterschrift

Zusammenfassung

Wir befassen uns mit drei eng verwandten Graphenmodifikationsproblemen aus der Perspektive der parametrisierten Komplexität. Diese Probleme sind 2-Club Cluster Editing, 2-Club Cluster Vertex Deletion und 2-Club Cluster Edge Deletion, worin es das Ziel ist einen gegebenen Graphen so zu transformieren, dass jede Komponente einen Durchmesser von zwei hat mit der kleinstmöglichen Anzahl eingefügter und gelöschter Kanten, gelöschter Knoten beziehungsweise gelöschter Kanten. Bemerkenswerte Anwendungsgebiete schließen graphbasiertes Clustering ein—die Aufteilung von Knoten in Cluster, so dass die Knoten in einem Cluster stark miteinander verbunden sind. Eine vorherige Arbeit resultierte in Algorithmen mit Laufzeit $\mathcal{O}^*(3.31^k)$ beziehungsweise $\mathcal{O}^*(2.74^k)$ für die letzten zwei Probleme. Wir zeigen ein W[2]-Härte-Ergebniss für 2-Club Cluster Editing, was bedeutet, dass die Existenz eines FPT Algorithmuses sehr unwahrscheinlich ist. Auf der positiven Seite entwickeln wir mehrere Datenreduktionsregeln und untere Schranken für 2-Club Cluster Vertex Deletion und 2-Club Cluster Edge Deletion und implementieren einen Solver für das erste von den Beiden. Wir haben erfolgreich Probleminstanzen aus einem biologischen Datensatz mit dichten Graphen bis zu 250 Knoten gelöst.

Abstract

We study three closely related graph modification problems from a parametrized complexity perspective. These problems are 2-Club Cluster Editing, 2-Club Cluster Vertex Deletion and 2-Club Cluster Edge Deletion, where the goal is to transform a graph into a 2-club cluster graph, that is a graph in which component has diameter at most two, with as few edge insertion and deletions, vertex deletions, and edge deletions, respectively. Notable areas of application include graph-based data clustering—the partitioning of vertices into highly interrelated clusters. Previous work resulted in $\mathcal{O}^*(3.31^k)$ and $\mathcal{O}^*(2.74^k)$ algorithms for the last two of the problems. We present a W[2]-hardness result for 2-Club Cluster Editing, which means it is likely not fixed-parameter tractable (FPT) with respect to the number of allowed edge modifications. On the positive side, we develop multiple data reduction rules and lower bounds for 2-Club Cluster Vertex Deletion and 2-Club Cluster Edge Deletion, and implement a solver for the former. We have successfully solved problem instances from a biological dataset with dense graphs up to 250 vertices.

Contents

1 Introduction

The clustering or partitioning of data is an important step used for data analysis. One of the most popular clustering algorithms is k-means clustering, where datapoints represented as vectors from $\mathbb{R}^n$ are to be split into k clusters, such that the points in each cluster are close to its center and distanced from other clusters as well as possible.

An emerging domain is *graph-based data clustering*, where the data is no longer represented as vectors in $\mathbb{R}^n$, but as vertices of a graph $G = (V, E)$. The edges of the graph can carry diverse semantic meaning such as similarity or interaction of datapoints and can optionally have weights to represent the strength of interaction or similarity. The goal of graph-based data clustering is the separation or clustering of vertices into disjoint sets in which vertices are highly interrelated—which means that they induce dense subgraphs such as cliques. However, it is unlikely that the vertices of a graph can be partitioned nicely into many large cliques, because in them no edge can be missing.

For this reason the graph-based clustering problem is often a graph modification problem. The graph is to be transformed with as few modifications as possible such that the resulting graph has a desired structure, for example that each component forms a clique. The resulting transformed graph defines the clustering of vertices. Vertices in the same component of the transformed graph are part of the same cluster. Standard graph modification operations are: inserting edges, deleting edges, and deleting vertices and typically one only allows one or two modification types. These modification problems are often NP-hard and have been studied from a computational complexity perspective, particularly fixed-parameter tractability (FPT) with respect to the number of allowed modifications [Bor+16; Böc12; Guo+10].

One of the most popular graph modification problems is CLUSTER EDITING where one wants to find the number of edge insertions and deletions that is necessary to transform a graph into a *cluster graph*, that is a graph in which each component is a clique. In particular, clustering of graphs representing protein sequence similarities has had much appeal [Wit+10].

The condition of the components forming cliques is, however, very restrictive. The choice of a clustering model is very use-case dependent and there is no model that is well-suited for all applications. Alternative models that relax the conditions for density, minimum vertex degree or diameter have been proposed and include: μ-cliques, k-cores, s-plexes and s-clubs, among others [Lee+10]. Distance-based relaxation models have been mostly unexplored, particularly in the area of graph modification problems. An s-club is a graph which has diameter at most s, which means any two vertices have a distance of at most s. A 1-club is equivalent to a clique, whereas 0-clubs are isolated vertices, which means only s-clubs with $s \geq 2$ are interesting from the viewpoint of distance based relaxations. In our work we will be focusing on the first of them, that is 2-clubs. The 2-club relaxation model has been proposed for the analysis of biological

[BBT05] and social networks [PYB12]. These graphs have the property that any two vertices are adjacent or have a common neighbor, which can be thought of as "two hop transitivity", and already is a quite relaxed model.

In our work we are interested in modifying graphs into *2-club cluster graphs*, that is graphs in which each component is a 2-club. Liu, Zhang, and Zhu [LZZ12] introduced three NP-hard graph modification problems: 2-Club Cluster Editing, 2-Club Cluster Edge Deletion and 2-Club Cluster Vertex Deletion, the last two of which are fixed-parameter tractable with respect to the number of modifications, but tractability of the first is an open problem. In 2-Club Cluster Editing the goal is to transform the graph into a 2-club cluster graph with as few edge insertions and deletions as possible, whereas for 2-Club Vertex and Edge Deletion with as few vertex and edge deletions, respectively. The prominence of Cluster Editing means that 2-Club Cluster Editing is of particular interest to us.

Related work Many graph modification problems have been analyzed in literature. Cao and Chen [CC12], Böcker [Böc12] and many others studied kernels and fast branching algorithms for (Weighted) Cluster Editing. Boral et al. [Bor+16] and Hüffner et al. [Hüf+10] developed $\mathcal{O}^*(1.92^k)$ and $\mathcal{O}^*(2^k)$ algorithms for Cluster Vertex Deletion. For clique relaxations Guo et al. [Guo+10] developed a polynomial kernel and an efficient search tree for s-Plex Cluster Editing. In the field of 2-clubs we have the work of Liu, Zhang, and Zhu [LZZ12] with $\mathcal{O}^*(3.31^k)$ and $\mathcal{O}^*(2.74^k)$ algorithms for 2-Club Cluster Vertex Deletion and 2-Club Cluster Edge Deletion, respectively. Hartung, Komusiewicz, and Nichterlein [HKN15] studied fast ways of finding the largest 2-club in a graph, whereas Castelli et al. [Cas+19] use a genetic algorithm for finding the k largest 2-clubs in a graph. Komusiewicz et al. [Kom+19] explore different variants of well-connected 2-clubs and efficient ways to find them.

Our contribution In Chapter 3 we present a W[2]-hardness result for 2-Club Cluster Editing, meaning that 2-Club Cluster Editing is likely not FPT with respect to the number of allowed modifications. On the positive side, we extend the work of Liu, Zhang, and Zhu [LZZ12] on the fixed-parameter tractability of 2-Club Cluster Vertex Deletion and 2-Club Cluster Edge Deletion and develop multiple data reduction rules and lower bounds for the two problems in Chapter 4 and Chapter 5, respectively. For this we use a standard depth-bounded search tree approach. We also extensively use constraints by identifying vertices or edges that must not be deleted and use this information to improve our reduction rules and lower bounds. The deletion of a vertex being much more powerful than the deletion of an edge, means that likely fewer vertex than edge deletions are necessary to transform the graph into a 2-club cluster graph. We therefore hope to find better results for 2-Club Cluster Vertex Deletion than for 2-Club Cluster Edge Deletion. For this reason we focus our efforts mainly on 2-Club Cluster Vertex Deletion, but also describe similarities and differences in 2-Club Cluster Edge Deletion. We explore a setting in which vertices can have positive integer weights and its surprisingly simple equivalence to the unweighted variant. We make use of vertex weights to explicitly model our constraints, by assigning infinite weight to vertices, but also allow the merging of vertices. Finally, we develop a solver for 2-Club Cluster Vertex Deletion using our findings and conduct some experiments on a biological dataset with dense graphs and describe our results in Chapter 6. We have successfully applied our solver on this dataset to solve instances with up to 250 vertices and 15,000 edges, where up to 60 vertices had to be deleted to transform the graphs into 2-club cluster graphs. The maximum running time of our solver was 4 minutes, but took only up to 10 seconds in most cases.

2 Preliminaries

We will first introduce basic knowledge about graph theory and parameterized complexity that we will need for the rest of this thesis. In Section 2.4 we will describe important properties of 2-clubs that need to be considered.

2.1 Graph theory

A graph $G = (V, E)$ is a pair of a set V of vertices and a set $E \subseteq \{\{v, w\} \mid v, w \in V, v \neq w\}$ of edges. This means there are no self-loops or multi-edges. We will often denote by n and m the number of vertices and edges, respectively.

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 will call a path on n vertices a P_n.

A graph is said to be *connected* if there exists a path between all pairs of vertices from $V(G)$ in G. A (connected) *component* of a graph is a maximal vertex set $S \subseteq V$ such that $G[S]$ is connected. A *cut vertex* is a vertex whose deletion increases the number of components.

The following list summarizes most of the notation that we will use:

$\binom{V}{2}$	the set of all possible pairs, formally $\{\{a, b\} \mid a, b \in V, a \neq b\}$
$E \triangle F$	the *symmetric difference* of two sets, formally $(E \setminus F) \cup (F \setminus E)$
$V(G)$	the vertex set of G
$E(G)$	the edge set of G
$N_G(v)$	the *open neighborhood* of v, formally $\{\{v, a\} \in E \mid a \in V\}$
$N_G[v]$	the *closed neighborhood* of v, formally $N_G(v) \cup \{v\}$
$\deg_G(v)$	the *degree* of $v \in V$, formally $\lvert N_G(v)\rvert$
$\Delta(G)$	the *maximum degree* of G, formally $\max_{v \in V} \deg_G(v)$
$\delta(G)$	the *minimum degree* of G, formally $\min_{v \in V} \deg_G(v)$
$G[V']$	the subgraph of G induced by $V' \subseteq V$, formally $(V', E \cap \binom{V'}{2})$
$G[E']$	the subgraph of G induced by $E' \subseteq E$, formally (V, E')
$G - v$	the graph obtained after deleting the vertex $v \in V$, formally $G[V \setminus \{v\}]$

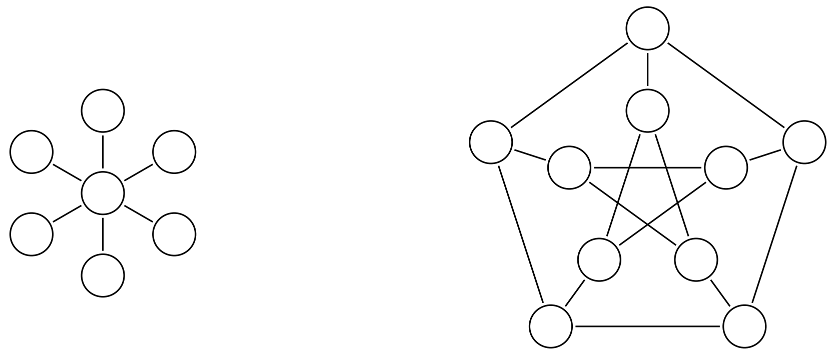

(a) A star with six leaves (b) The Petersen graph

Figure 2.1: Examples of 2-clubs. Notice that after removing any vertex from the Petersen graph it is no longer a 2-club. The Petersen graph with its complex yet highly symmetrical structure is infamous for being a small counterexample to many optimistic assumptions.

$G - e$	the graph obtained after deleting the edge $e = \{a, b\} \in E$, formally $G[E \setminus \{e\}]$
ab	a shorthand for the edge $\{a, b\} \in E$
$\text{dist}_G(s, t)$	the *distance* of $s, t \in V$, i.e., the length of the shortest path connecting s and t or ∞ if none exists

When in context it is clear which graph we are working with then we will not use G in the subscripts and only write $\deg(v)$ instead of $\deg_G(v)$, for example. We will call two vertices u, v *twins* if either $N_G(u) = N_G(v)$ or $N_G[u] = N_G[v]$, the former sometimes being called *false twins* and the latter *true twins*, however, we will not make use of this distinction. The *diameter* of a graph is the maximum distance of any two vertices, formally $\max_{s,t \in V} \text{dist}_G(s, t)$.

A *clique* or *complete graph* is a graph G of the form $(V, \binom{V}{2})$. A *tree* is a connected graph with $n-1$ edges. An s-club is a graph of diameter at most s. *Stars* are particularly simple 2-clubs, which are trees that have at most one vertex with degree greater than one. For examples of 2-clubs see Figure 2.1. A *cluster graph* is a graph in which each component is a clique, likewise an *s-club cluster graph* is a graph in which each component is an s-club.

2.2 Parameterized complexity

A problem of size n and with a parameter k is said to be fixed-parameter tractable if it can be solved in $f(k) \cdot n^{\mathcal{O}(1)}$ time, where f is a computable function. We sometimes use the notation $\mathcal{O}^*(f(k))$ for algorithms for FPT problems, in which omit the polynomial term dependent on the size of the input problem. Fixed-parameter tractable problems form the class *FPT*. FPT algorithms often make use of *data reduction rules*. A data reduction rule is a polynomial-time algorithm, which *reduces* an instance $(\mathcal{I}, k)$ to an instance $(\mathcal{I}', k')$. Typically the reduced instance is smaller, that is, $|\mathcal{I}'| \leq |\mathcal{I}|$ and $k' \leq k$. The data reduction rule is said to *safe* if $(\mathcal{I}, k)$ and $(\mathcal{I}', k')$ are equivalent, i.e., that $(\mathcal{I}, k)$

is a yes-instance if, and only if, $(\mathcal{I}', k')$ is a yes-instance. We are only interested in safe data reduction rules. A *kernel* is an alogrithm that reduces an instance $(\mathcal{I}, k)$ to an equivalent instance $(\mathcal{I}', k')$ with $|\mathcal{I}'| + k' \leq f(k')$ for some computable function f. It is also called a polynomial kernel if f is a polynomial.

FPT algorithms are often based on *depth-bounded search trees* [Nie06]. They recursively explore the space of possible solutions by identifying a "small subset of the input instance" such that at least one element from this subset is part of an optimal solution. This small subset is often guaranteed to have a fixed size. The algorithm then *branches* into appropriately many cases to check which of these elements can be found in an optimal solution. This is done in an recursive manner and a parameter k, often representing the desired solution size, is decreased by d_i in the i-th branch. The search tree does not branch further if k were to become negative, thus bounding its depth. The resulting recursive call graph is a tree. For the running time analysis of such FPT algorithms it is important to analyze the number of recursive calls or the *size* of the search tree. This can be done by solving the following recurrence relation:

$$\begin{aligned} T(k) &= 1 && \text{for } k \leq 0 \\ T(k) &= T(k - d_1) + \cdots + T(k - d_\ell) && \text{otherwise} \end{aligned}$$

The vector $(d_1, \ldots, d_\ell)$ is called a *branching vector*. The size of the search tree can be computed by solving $z^k = z^{k-d_1} + \cdots + z^{k-d_\ell}$ for z. The root α with the largest absolute value then gives $T(k) = k^{\mathcal{O}(1)} \cdot \alpha^k$ and if α is a single root, then $T(k) = \mathcal{O}(\alpha^k)$. The *branching number* corresponding to the branching vector $(d_1, \ldots, d_\ell)$ is $|\alpha|$.

The W-hierarchy: $FPT = W[0] \subseteq W[1] \subseteq \cdots \subseteq W[P]$ is an important hierarchy of NP-hard problems. All of these inclusion are believed to be strict [Cyg+15], meaning that the "deeper" a problem is in the W-hierarchy the less likely is its fixed-parameter tractability. In order to prove for example W[2]-hardness of a problem B one needs to find a *parameterized reduction* from a W[2]-hard problem A to B. A parameterized reduction is an algorithm that given an instance $(\mathcal{I}, k)$ of A outputs an equivalent instance $(\mathcal{I}', k')$ of B such that $k' \leq g(k)$ for some computable function g and the running time of the algorithm is $f(k) \cdot |\mathcal{I}|^{\mathcal{O}(1)}$.

For more details about the topic of parameterized complexity we refer to the books by Cygan et al. [Cyg+15], Downey and Fellows [DF13], and Niedermeier [Nie06].

2.3 Problem definitions

What follows is a list of problems relevant for our thesis.

Vertex Cover
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there a $U \subseteq V$ with $|U| \leq k$ such that each edge has at least one end in U?

Dominating Set
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there a $U \subseteq V$ with $|U| \leq k$ such that each vertex in $V \setminus U$ has at least one neighbor in U?

d-Hitting Set
Input: A collection $\mathcal{C}$ of subsets of size d of a finite set S and an integer $k \in \mathbb{N}$
Question: Is there a subset $S' \subseteq S$ with $|S'| \leq k$ such that S' contains at least one element from each of the sets in $\mathcal{C}$?

Cluster Editing
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there an $F \subseteq \binom{V}{2}$ with $|F| \leq k$ such that $G[E \triangle F]$ is a cluster graph?

Weighted Cluster Editing
Input: A graph G, a weight function $w : E \to \mathbb{N}$ and an integer $k \in \mathbb{N}^+$.
Question: Is there an $F \subseteq \binom{V}{2}$ with $w(F) \coloneqq \sum_{e \in F} w(e) \leq k$ such that $G[E \triangle F]$ is a cluster graph?

Cluster Vertex Deletion
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there an $S \subseteq V$ with $|S| \leq k$ such that $G[V \setminus S]$ is a cluster graph?

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

s-Club Cluster Vertex Deletion
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there an $S \subseteq V$ with $|S| \leq k$ such that $G[V \setminus S]$ is an s-club cluster graph?

s-Club Cluster Edge Deletion
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there an $F \subseteq E$ with $|F| \leq k$ such that $G[E \setminus F]$ is an s-club cluster graph?

s-Club
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there an $S \subseteq V$ with $|S| \geq k$ such that $G[S]$ is a s-club?

While these are all decision problems, one is usually also interested in finding the solution for a yes-instance. We will be using the terms minimum solution and optimal solution interchangeably. The domination number $\gamma(G)$ is the size of a minimum dominating set of G. Similarly, $\tau(G)$ is the size of a minimum vertex cover of G. Vertex Cover is famously fixed-parameter tractable with respect to k, whereas Dominating Set is W[2]-complete [DF13; Nie06].

Cluster Editing is equivalent to 1-Club Editing and Cluster Vertex Deletion is equivalent to 1-Club Vertex Deletion. Notice also how Vertex Cover is equivalent to 0-Club Vertex Deletion.

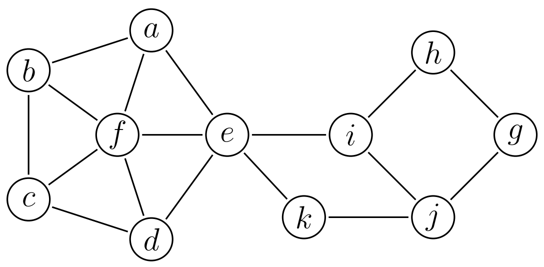


Figure 2.2: An example to demonstrate robustness. For example we have the following robustness: $\text{robust}(a, d) = 2, \text{robust}(f, j) = \text{robust}(h, k) = 0$ and $\text{robust}(h, j) = \infty$. Two optimal 2-club cluster vertex deletion sets are $\{e, j\}$ and $\{i, k\}$.

2.4 2-Club specifics

A 2-club is a graph with diamater at most two, which 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. We will use terminology from Liu, Zhang, and Zhu [LZZ12] and call a path $stuv$ in G a *restricted* P_4 if $\text{dist}_G(s, v) = 3$. In this case, the restricted P_4 is a shortest path connecting s and v and it is also an induced P_4.

Observation 2.1. *G is a 2-club cluster graph if and only if it contains no restricted P_4's.*

This alternate 2-club characterization is important for fixed parameter tractability of 2-Club Cluster Vertex Deletion and 2-Club Cluster Edge Deletion. While a restricted P_4 exists the graph cannot be a 2-club cluster graph. Therefore, one of the three edges or four vertices needs to be deleted. If edge insertions are allowed this is no longer so simple, as we will see in Chapter 3.

2-Clubs are *non-hereditary*, that is, if G is a 2-club, then for a $V' \subseteq V$ the graph $G[V']$ is not necessarily a 2-club (for example see the Petersen graph in Figure 2.1). For 2-Club Cluster Vertex Deletion we introduce the following terminology. We will 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 $stuv$. Because restricted P_4's are induced paths and the deletion of a vertex cannot create more induced paths, this means that only the deletion of bridge vertices can create additional restricted P_4's. However, deleting a single vertex may not be enough to create a restricted P_4. We therefore introduce the following *robustness* function to find the number of vertices that need to be deleted before between some vertices s and v a restricted P_4 can be created.

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

For an example see Figure 2.2

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 $stuv$. We will call a set S such that $G[V \setminus S]$ a 2-club cluster graph a 2-club vertex deletion set.

3 2-Club Cluster Editing

Some work has been done analyzing problems closely related to 2-Club Cluster Editing such as 2-Club Cluster Edge Deletion and 2-Club Cluster Vertex Deletion. The last two are fixed-parameter tractable with respect to the solution set size k [LZZ12]. The core idea behind the algorithms was to search for a restricted P_4 *stuv* and delete a vertex or an edge to separate s and v. This is correct, because while a restricted P_4 connecting s and v exists, the graph is not a 2-club cluster graph. These observations were essential to show the fixed-parameter tractability of the two problems.

3.1 W[2]-hardness

In the context of restricted P_4's for 2-Club Cluster Editing we consider ways of eliminating a restricted P_4 *stuv* with only a single edge deletion or insertion:

- inserting $\{s,v\}, \{t,v\}$ or $\{s,u\}$,
- deleting $\{s,t\}, \{t,u\}$ or $\{u,v\}$.

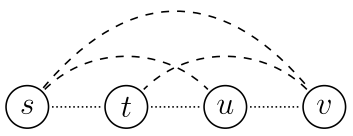
We call these six operations *local resolutions* (see Figure 3.1a). It is not immediately obvious whether these six local resolutions are sufficient to construct a minimum 2-club editing set F. After all, other ways of eliminating restricted P_4's exist—meaning those that add edges $\{a,b\}$ such that $a \in \{s,t,u,v\}$ and $b \in V \setminus \{s,t,u,v\}$, which we will call *non-local resolutions* (see Figure 3.1b). What is more, we also notice that by inserting new edges into the graph, new restricted P_4's could be created. An unlucky choice of a restricted P_4 to eliminate might lead to the editing cost of the graph in all six branches to be at least as high as it was before, meaning that that specific restricted P_4 should not have been branched on.

Indeed, we find cases where it is not sufficient to branch on an arbitrary P_4 (see Figure 3.2a). In particular, cases exist where branching on any restricted P_4 does not yield an optimal solution (see Figure 3.2b). This means that considering paths of length three and their local resolutions is not always sufficient to find an optimal solution.

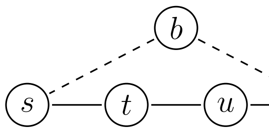
This begs the question if 2-Club Cluster Editing is at all fixed-parameter tractable with respect to k. We show that this is very unlikely. To this end, we show that 2-Club Cluster Editing is W[2]-hard with respect to k.

A problem closely related to 2-Club Cluster Editing is the following.

Diameter-2 Augmentation
Input: A graph G and an integer $k \in \mathbb{N}$.
Question: Is there an $F \subseteq \binom{V}{2} \setminus E$ with $|F| \leq k$ such that $G[E \cup F]$ is a graph with diameter two?

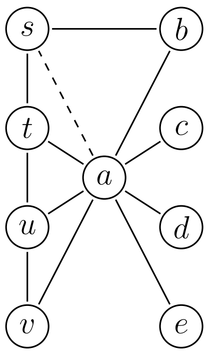


(a) The six local resolutions. Any edge that is inserted or deleted has both its ends in $\{s, t, u, v\}$.

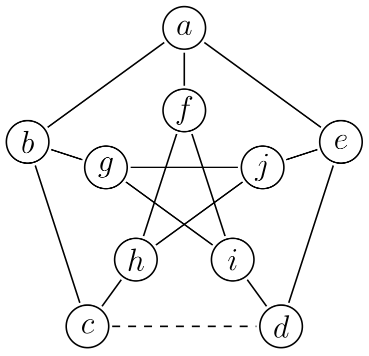


(b) A non-local resolution, where b can be any vertex other than s, t, u and v. It is possible that at most one of the edges $\{s, b\}$ or $\{v, b\}$ is already present.

Figure 3.1: Examples of ways in which a restricted P_4 *stuv* can be resolved. Dashed edges are edges to be inserted, whereas dotted ones are to be deleted.



(a) No local resolution of *stuv* yields an optimal solution



(b) No local resolution of any restricted P_4 yields an optimal solution

Figure 3.2: Cases where the optimistic assumption of branching on any restricted P_4 fails. The dashed edges indicate the single optimal solution. Notice that in Figure 3.2b the distance of c and d is four. Therefore we cannot insert the edge $\{c, d\}$ as part of a local resolution without first decreasing their distance.

The only difference between 2-Club Cluster Editing and Diameter-2 Augmentation is that the latter does not allow edges to be deleted. Gao, Hare, and Nastos [GHN13] already claim that Diameter-2 Augmentation is W[2]-hard with respect to k, by constructing a parameterized reduction from Dominating Set; however, we have not been able to verify their results. We use an idea similar to the one by Gao, Hare, and Nastos [GHN13], which involves a parameterized reduction from Dominating Set, to prove W[2]-hardness for 2-Club Cluster Editing. Dominating Set remains W[2]-hard for graphs with diameter two [Lok+13], which allows us to assume that the Dominating Set instance has diameter two.

Theorem 3.1. 2-Club Cluster Editing *is W[2]-hard with respect to k.*

Proof. Let $G = (V, E)$ be graph with diameter two. We construct a graph G' in such a way that G has a minimum dominating set of size k if and only if G' has a minimum 2-club editing size of k. The graph $G' = (V', E')$ can be broken down into following parts: the original graph G, a clique C of size $(n + 1)^2$ and a single vertex x. We assign two indices for the vertices $c_{i,j} \in C$ such that $i, j \in \{0, \dots, n\}$. The vertices in V only

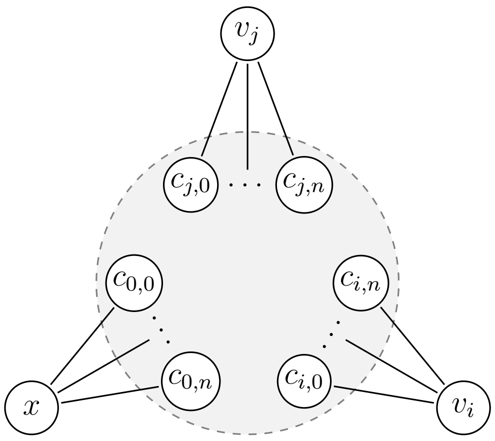


Figure 3.3: A schematic picture of the construction of G' from the proof of Theorem 3.1. The vertices in the gray circle form a clique, but only the vertices connected to v_i, v_j or x are shown. The edge between v_i and v_j may be present, depending on G.

have one index: $v_i \in V$, $i \in \{1, \ldots, n\}$. In addition to the existing edges in G and C we add the following edges:

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

The graph G' has $\mathcal{O}(n^4)$ edges and $\mathcal{O}(n^2)$ vertices. For a schematic picture of G' see Figure 3.3. Notice that the only pairs of vertices with distance three are x and $v_i \in V$, all others have distance at most two.

We claim there exists a minimum 2-club editing set for G' (which only inserts edges) of the form $\{\{x, v\} \mid v \in D\}$, where D is a minimum dominating set for G. We will denote by $\eta(G')$ the size of a minimum 2-club editing set for G'. Recall that $\gamma(G)$ is the size of a minimum dominating set of G. We will now show that $\gamma(G) = \eta(G')$.

$\gamma(G) \geq \eta(G')$: Let D be a minimum dominating set for G, and $F = \{\{x, v\} \mid v \in D\}$. Let $H = G'[E' \triangle F]$ be the graph transformed by F. Every v_i is either in D, and then $\mathrm{dist}_H(x, v_i) = 1$, or $v_i \notin D$, then v_i has a neighbor in D and thus $\mathrm{dist}_H(x, v_i) = 2$. This means H is a 2-club cluster graph and F a 2-club editing set for G'.

$\eta(G') \geq \gamma(G)$: Let F be any minimum 2-club editing set for G' and $H = G'[E' \triangle F]$ be the resulting 2-club cluster graph. Notice that the minimum cut of G' is $n+1$. Removing any edge would only be optimal if H contained more than one 2-club cluster. However, since $\eta(G') \leq \gamma(G) \leq n$, a 2-club editing set of size $\gamma(G) \leq n$ can be constructed from a minimum dominating set for G. As such, no minimum 2-club editing set F deletes any edges from G'.

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

- $\{a, b\} = \{v_i, x\}$: Then $\mathrm{dist}_H(x, v_i) = 1$ and for $a \in N_G(v_i)$ $\mathrm{dist}_H(x, a) \leq 2$. 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 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 minimum 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 $\{x, v_i\} \in F$. Notice that F is only optimal 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 the edge 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 minimum 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 optimal. The edge $\{v_i, c_{j,k}\}$ enables a path of length two from v_i to x via $c_{j,k}$. This means 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 minimum 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 $c_{i,j}$. By the case above we know there exists an F' with $\{x, c_{i,j}\} \in F'$ such that there exists no edge $\{v_k, c_{i,j}\} \in F'$ with $k \neq i$. This means 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 minimum 2-club editing set with $|F| = |F''|$.

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

As such $\gamma(G) = \eta(G')$ and the reduction from (G, k) to (G', k) is a valid parameterized reduction from Dominating Set for graphs with diameter two to 2-Club Cluster Editing. Since Dominating Set remains W[2]-hard for graphs of diameter two [Lok+13] this means 2-Club Cluster Editing is also W[2]-hard. □

Our parameterized reduction from Dominating Set to 2-Club Cluster Editing can also be used to show W[2]-hardness for Diameter-2 Augmentation, because by our construction an optimal 2-club editing set does not delete any edges, which means that it is also an optimal diameter two augmentation set.

3.2 Useful algorithmic properties

Despite being W[2]-hard we would like to describe some properties of this problem that could be useful for a solver. A trivial way to approach the problem would be to try all $F \subseteq \binom{V}{2}$ of size at most k. The running time would then be $\mathcal{O}^*(\binom{n^2}{k}) = \mathcal{O}^*(n^{2k})$.

We noticed that local resolutions of a restricted P_4 are not enough to construct an optimal 2-club editing set. The only other way to resolve $stuv$ is by "constructing a bridge", i.e., finding a vertex $b \in V \setminus \{s, t, u, v\}$ and inserting the edges $\{s, b\}$ and $\{v, b\}$ if they are not in E already. Since $\text{dist}_G(s, v) = 3$, at least one of the edges is not present. If we try all six local resolutions and all $n - 4$ non-local resolutions of $stuv$ this gives us a search tree algorithm with running time $\mathcal{O}^*((n + 2)^k)$.

We notice that the number of edges that can be inserted or removed from a graph can be in the order of n^2. However, there is a simple type of 2-club that has only $n-1$ edges—a star with $n-1$ leaves. This means that we can make any graph into a 2-club cluster graph with at most $n-1$ edge modifications by selecting any vertex v and inserting edges to any vertex that is not a neighbor of v. This leads to the following upper bound for the minimum 2-club editing cost of a graph.

Observation 3.2. *The minimum 2-club editing set size of a graph $G = (V, E)$ is at most $|V| - \Delta(G) - 1$.*

Due to the number of edges that need to be potentially deleted to split a graph into multiple 2-clubs being in the order of n^2 it is likely that the previously noted upper bound is also a minimum 2-club editing set size for many dense graphs. In the context of graph-based clustering such a solution would likely not be very satisfactory. It might be more meaningful to find a constrained solution that does not modify too many edges adjacent to a vertex (refer to Komusiewicz and Uhlmann [KU12]).

Next we want to study the effect of inserting or deleting an edge. We consider an instance (G, k) of 2-Club Cluster Editing and some simple search tree algorithm that solves this problem recursively by branching. Let F' be a partial 2-club editing set that was constructed at some stage of the search tree algorithm. We are left with the graph $G' = G[E \triangle F']$. Solving this problem recursively involves now searching for a 2-club editing set F'' with $F'' \leq k - |F'|$ for G', because then the set $F = F'' \triangle F'$ would be a 2-club editing set for G. However, it would be strange if $F'' \cap F' \neq \emptyset$, because that would mean that F'' either deleted edges inserted by F' or inserted edges deleted by F'. The search tree would have then "paid" for one edge modification from its budget k only to pay once more to later undo it. Undoing edge modifications in the search tree means losing budget for two edge modifications. Constraining the solver to find an F'' with $F'' \cap F' = \emptyset$ is then much more desirable.

We now consider such a constrained search tree, and a partial 2-club editing set F' constructed at some stage of the branching. We will call an edge $\{a, b\}$ *permanent* if it was inserted, i.e., $\{a, b\} \in F' \setminus E$. Likewise, we will call an edge $\{a, b\}$ *forbidden* if was deleted, i.e., $\{a, b\} \in E \cap F'$. Permanent and forbidden edges have been used previously by Böcker et al. [Böc+09] for their Weighted 2-Club Editing solver.

A fairly obvious observation is the following:

Observation 3.3. *The existence of a permanent edge $\{a, b\}$ enforces that a and b must be in the same 2-club.*

Furthermore, if a permanent edge $\{b, c\}$ exists, then a, b and c must be in the same 2-club. We notice that if two vertices a and c must be in the same 2-club, then it is not optimal to delete the edge $\{a, c\}$, if it exists, which means that this edge can also be marked as permanent. Permanent edges have a transitivity property.

We can further constrain the search tree to only find 2-club editing sets F, such that in in each 2-club in $G[E \triangle F]$ no edge was deleted by F, because such solutions would clearly not be optimal, because the edge can be simply added back. With such a constraint in place this leads us to the following observation:

Observation 3.4. *The existence of a forbidden edge $\{a, b\}$ enforces that a and b must be in different 2-clubs.*

This opposite function of forbidden and permanent edges has a fruitful consequence.

Observation 3.5. *Let F' be a partial solution constructed at some stage of the branching, and $A, B \subseteq V$ be two disjoint sets of vertices that must be in the same 2-club, as enforced by F'. If for some $a \in A$ and $b \in B$ the edge $\{a, b\}$ is forbidden, then all edges between A and B need to be deleted.*

We will later capture some of these ideas more formally for 2-Club Cluster Edge Deletion in Chapter 5.

4 2-Club Cluster Vertex Deletion

To solve 2-Club Cluster Vertex Deletion we use an approach typical for FPT algorithms involving a search tree, which we describe in Section 4.1. We introduce a set of permanent vertices, which is initially empty, and we constrain ourselves to finding a 2-club vertex deletion set that does not include a permanent vertex. These permanent vertices will act as additional information about the graph, which we obtain during the execution of our solver, and can be used in many different areas of our algorithm in each of which it yields small improvements. Permanent vertices serve a similar purpose as permanent and forbidden edges in 2-Club Cluster Editing, which were also used by Böcker et al. [Böc+09] for Cluster Editing. We describe multiple data reduction rules, for deleting vertices (with and without the help of permanent vertices), but also for marking vertices as permanent (Section 4.2). We then develop lower bounds (Section 4.3) to further reduce the size of the search tree. In Section 4.4 we switch to a setting that allows vertices to have weights. We do this to create data reduction rules which would not work without vertex weights, particularly "merging" of twin vertices. We explain how our rules for the unweighted variant still apply to the weighted one, in many cases without restrictions due to weights.

An important idea that we use repeatedly is to find many vertex-disjoint restricted P_4's. Finding a maximum set of vertex-disjoint P_4's is NP-hard [IPS82], thus finding such a set of restricted P_4's is likely also NP-hard. A practical implementation would then employ heuristics.

In Chapter 6 we experimentally evaluate an implementation of the algorithm described in this chapter. There we explore the impact of the techniques used involving permanent vertices and twin merging. We also compare our solver against an ILP formulation, which we describe next, that is solved using CPLEX.

ILP Formulation We start by devising an integer linear programming formulation for 2-Club Cluster Vertex Deletion. Although compact, it manages to capture some important properties of the problem.

The only requirement that our graph has to fulfill in the end is that it is a 2-club cluster graph, i.e., that it contains no restricted P_4. Recall that a restricted P_4 is an induced P_4 for which there is no vertex bridging it, i.e., a vertex whose neighborhood includes the two ends of the P_4. The deletion of any vertex cannot create any new induced path. However, the deletion of a bridge vertex might "promote" an induced P_4 to a restricted P_4. This means that for any induced P_4 $stuv$ in G, if $N(s) \cap N(v) = \emptyset$, 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 ILP formulation:

$$\begin{array}{lll} \text{minimize:} & \sum_{v \in V} x_v & \\ \text{subject to:} & x_s + x_t + x_u + x_v + \sum_{b \in N(s) \cap N(v)} (1 - x_b) \geq 1 & \text{for all induced } P_4\text{'s } stuv \text{ in } G \\ & x_v \in \{0, 1\} & \text{for all } v \in V \end{array}$$

From this ILP formulation it is easy to see that deleting a vertex can cause some constraints to no longer hold, i.e., create restricted P_4's, and therefore force us to delete additional vertices. However, deleting just one vertex may not be enough to create a restricted P_4. This effect is due to the non-hereditary property of 2-clubs.

If we could omit the sum over the common neighbors of s and v and thus only consider restricted P_4's rather than induced P_4's, then 2-Club Cluster Vertex Deletion could be trivially reduced to 4-Hitting Set and we could immediately benefit from having a kernel with at most $\mathcal{O}(k^3)$ vertices and a 4-approximation algorithm [Abu10]. However, it remains open whether 2-Club Cluster Vertex Deletion has a polynomial kernel.

4.1 Search tree

The characterization of 2-club clusters graphs as graphs free of restricted P_4's yields a straight-forward search tree algorithm, described by Branching Rule 4.1. We use this as a basis of our solver.

Branching Rule 4.1. *Let $\mathcal{I} = (G, k)$ be an instance of* 2-Club Cluster Vertex Deletion. *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:*

- $\mathcal{I}_s = (G - s, k - 1)$,
- $\mathcal{I}_t = (G - t, k - 1)$,
- $\mathcal{I}_u = (G - u, k - 1)$,
- $\mathcal{I}_v = (G - v, k - 1)$.

Lemma 4.1. *Branching Rule 4.2 is correct.*

Proof. If G is already a 2-club cluster graph, then there is nothing to do. Otherwise assume that G contains a restricted P_4 $stuv$. 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 with $|S| \leq k - 1$ be a 2-club vertex deletion set for one of the four instances $\mathcal{I}_x$ with $x \in \{s, t, u, v\}$. Then the set $S' = S \cup \{x\}$ with $|S'| \leq k$ is a 2-club vertex deletion set for the instance $\mathcal{I}$.

"$\Rightarrow$": 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 $|S| \leq k$. For any $x \in \{s, t, u, v\}$ if $x \in S$, then the instance $\mathcal{I}_x$ must also be a *yes*-instance with $S' = S \setminus \{x\}$ being its 2-club vertex deletion set and $|S'| \leq k - 1$. □

Here the branching number is four, because we split into four branches in which k is reduced by one. A more extensive case analysis can be made that yields a branching rule whose branching number is 3.31 [LZZ12]. However, we concentrate on data reduction rules and other strategies to significantly reduce the number of total recursive calls.

In the remainder of this section we focus on improving our branching rule through constraints. For this we introduce *permanent vertices* and constrain our solver to never delete a permanent vertex. This also means that a permanent vertex is never part of a 2-club vertex deletion set that we find. We will use permanent vertices to encode additional information about our graph. Knowing that a vertex must not be deleted might not be enough information to benefit from immediately, but we can "save" this information for later by marking the vertex as permanent. As we will see in Section 4.2 and Section 4.4 we can benefit from having many permanent vertices to apply more data reduction rules and improve our lower bounds. We formally capture permanent vertices in the following problem definition.

CONSTRAINED 2-CLUB CLUSTER VERTEX DELETION (CSTR2CVD)
Input: A graph $G = (V, E)$, an integer $k \in \mathbb{N}$ and a set $F \subseteq V$ of permanent vertices.
Question: Is there an $S \subseteq V$ with $|S| \leq k$ and $S \cap F = \emptyset$ such that $G[V \setminus S]$ is a 2-club cluster graph?

An instance (G, k) of 2-CLUB CLUSTER VERTEX DELETION is clearly equivalent to the instance $(G, k, \emptyset)$ of CSTR2CVD.

Suppose there is some restricted P_4 *stuv* and an optimal solution of size k removes s and v. Then we know that, this solution will be found in (at least) two branches of Branching Rule 4.1. To reduce this overlap we extend Branching Rule 4.1 and introduce the following branching rule.

Branching Rule 4.2. *Let $\mathcal{I} = (G, k, F)$ be an instance of* CSTR2CVD. *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:*

- $\mathcal{I}_s = (G - s, k - 1, F)$,
- $\mathcal{I}_t = (G - t, k - 1, F \cup \{s\})$,
- $\mathcal{I}_u = (G - u, k - 1, F \cup \{s, t\})$,
- $\mathcal{I}_v = (G - v, k - 1, F \cup \{s, t, u\})$.

If an instance $\mathcal{I}_x$ was derived by removing a permanent vertex $x \in \{s, t, u, v\} \cap F$, then skip $\mathcal{I}_x$.

Lemma 4.2. *Branching Rule 4.2 is correct.*

Proof. If G is already a 2-club cluster graph, then there is nothing to do. Otherwise assume that G contains a restricted P_4 *stuv*. 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 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 $|S| \leq k$ 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. □

With this we have developed a simple search tree for Cstr2CVD, but also integrated the generation of permanent vertices directly into the branching. An added benefit of Branching Rule 4.2 is that we can skip branches where a permanent vertex would have been deleted.

4.2 Data reduction rules

We now present some data reduction rules. Finding powerful data reduction rules for 2-Club Cluster Vertex Deletion is difficult, due to 2-clubs being non-hereditary. Recall that robust(s, v) tells us the number of vertices that need to be deleted before a restricted P_4 *stuv* is created. The effect of deleting "wrong" vertices might lead to a large increase of restricted P_4's. These effects must be considered when designing safe data reduction rules.

For this reason we were only able to find rather simple data reduction rules. In Section 4.4 we generalize a few rules by incorporating information about permanent vertices by considering a scenario where vertices have weights.

The following reduction rule is obviously safe, because no vertex from a component that already is a 2-club needs to be deleted.

Reduction Rule 4.1. *If G contains a component C that is a 2-club, then delete all vertices in C.*

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

Reduction Rule 4.2. *If the graph G 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 1.*

Lemma 4.3. *Reduction Rule 4.2 is safe.*

Proof. Let *stuv* and x be as above. At least one vertex from *stuv* has to be deleted, but three vertices are permanent, which leaves no choice but to delete x. □

The next rule uses that sometimes with the budget k we cannot afford not to delete a specific vertex. For an example see the vertex v in Figure 4.1.

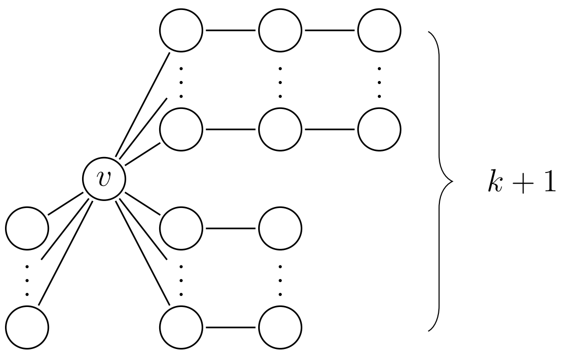


Figure 4.1: An example of a graph that has $k+1$ P_4's that overlap only in v. Here we also illustrate the two types of restricted P_4's that can contain v: either v is at the beginning or it is the second vertex in the restricted P_4. The vertex v could also be the third or fourth vertex, but due to symmetry this is the same as being the second or first vertex, respectively.

Reduction Rule 4.3. *Given $k+1$ restricted P_4's that each contain the vertex v, but are otherwise vertex-disjoint, then delete v and decrease k by 1 or ∞ if v is permanent.*

Lemma 4.4. *Reduction Rule 4.3 is safe.*

Proof. If G contains $k+1$ restricted P_4's that overlap only in v, then if v is not deleted, then one vertex from each of the $k+1$ restricted P_4's has to be deleted. This is not possible with a budget of k. □

In Section 4.4 we will further improve Reduction Rule 4.3 by exploiting permanent vertices.

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

One could use Observation 4.5 as a simple optimality test. 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 of the search tree to some branch. The test can be applied to S' in the same way 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 4.4. *Let S' be a partial 2-club vertex deletion set of G constructed at some stage of the branching. If for any $v \in S' : |N(v) \setminus S'| = 1$, then mark the unique vertex $x \in N(v) \setminus S'$ as permanent.*

Lemma 4.6. *Reduction Rule 4.4 is safe.*

Proof. Let S' and x be as above. Any 2-club vertex deletion set S with $S \cap (S' \subseteq \{x\})$ clearly fails the optimality test from Observation 4.5, which means x cannot be in any optimal solution. □

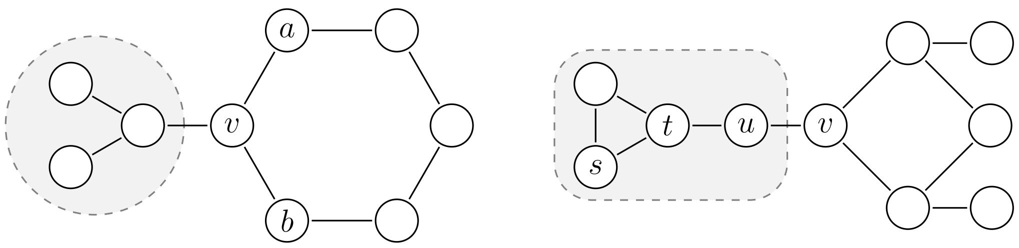


Figure 4.2: Examples of graphs where Reduction Rule 4.5 can be applied on the non-bridge vertex v. The gray areas contain vertices that can be marked as permanent. Each gray area is a 2-club that is isolated from the rest of the graph after deleting v. Notice that in the first graph the removal of v increases the distance of a and b to five and no induced and therefore restricted P_4 exists between a and b. Notice also how in the second graph the restricted P_4 $stuv$ contains three permanent vertices. Reduction Rule 4.2 will allow us to delete v in such a case.

The intuition for the next rule is that we would like to have something similar to the simple Degree-One-Rule from Vertex Cover, that is, if there is a vertex with degree one, then its neighbor can be safely included in the vertex cover [Nie06]. For Cstr2CVD we would like to replace the degree one vertex with a 2-club, but still require that this 2-club can be isolated from the rest of the graph by deleting only a single vertex.

However, this proved to be not quite so simple. The following rule does not state that given some conditions a vertex v can be deleted, but rather that the vertices in the 2-club that is isolated by removing v can be safely excluded from the solution (by marking them as permanent)—all under the premise that if any vertex from that 2-club was included in the solution it could be replaced by v to yield another solution of the same size. In the Vertex Cover analogy we could rephrase the Degree-One-Rule to say that a degree one vertex can be safely excluded from a minimum vertex cover (unless its neighbor also has degree one and is already excluded), because if it was it could be replaced by its neighbor to yield another minimum vertex cover.

This is all because even if deleting v creates an isolated 2-club C we cannot be sure that deleting v is the right choice. For example, the graph could have no restricted P_4's and deleting any vertex would be unnecessary.

Recall that a bridge vertex b is a vertex such that for $s, v \in N(b)$ there exists an induced P_4 $stuv$. Recall also that only the deletion of a bridge vertex b can decrease robustness and therefore create restricted P_4's. For an example of the application of the following rule see Figure 4.2.

Reduction Rule 4.5. *Given a cut vertex v in G that is not a bridge vertex and not permanent. For any component C in $G - v$, if C is a 2-club, then mark the vertices in C as permanent.*

Lemma 4.7. *Reduction Rule 4.5 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

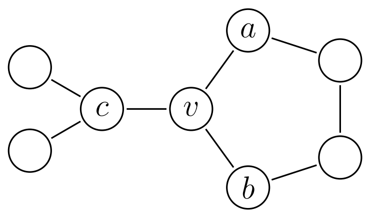


Figure 4.3: An example where Reduction Rule 4.5 cannot be applied on the vertex v, which bridges a and b. Deleting v would increase the distance of a and b to three, and thus create a restricted P_4. This means that deleting v would force us to delete a second vertex. However, deleting c eliminates all restricted P_4's in the graph, and is the only optimal solution. Reduction Rule 4.5 can, however, be applied to c, and all vertices except c can be marked as permanent.

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. That is because v eliminates the same P_4's as F and because v is not a bridge vertex, it follows that removing v cannot contribute to the creation of a restricted P_4. □

For an example why we require v to be a non-bridge vertex in Reduction Rule 4.5 see Figure 4.3.

A slight generalization of Reduction Rule 4.5 can be made. Recall that robust(a, b) tells us how many vertices need to be removed before there can exist a restricted P_4 between a and b. In Reduction Rule 4.5 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 ourselves 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 under the same premise that v can be deleted instead of any vertex in those 2-clubs. 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 4.6. *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 $a, b \in N_G(v) \cap V(H)$ robust$_G(a, b) > k$, then mark all vertices in $C_1, \ldots, C_\ell$ as permanent.*

Lemma 4.8. *Reduction Rule 4.6 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 $S' = (S \setminus F) \cup \{v\}$ 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 must start and end in two neighbors of v. Deleting v cuts off the 2-clubs $C_1, \dots, C_\ell$ 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 at least $k+1$ vertices 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 that to happen, replacing F by the single vertex v to obtain S' results in another optimal solution. □

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

Reduction Rule 4.7. *Let C be a permanent 2-club that can be isolated by removing the vertex v as in Reduction Rule 4.6. 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 $d+1$.*

Lemma 4.9. *Reduction Rule 4.7 is safe.*

Proof. Let C and v be as in Reduction Rule 4.5. Since the vertices in the 2-club are all marked permanent and if they are part of any restricted P_4, then the vertex v must also be part of the restricted P_4. This means that only the distance of the vertices in C to 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. □

4.3 Lower bounds

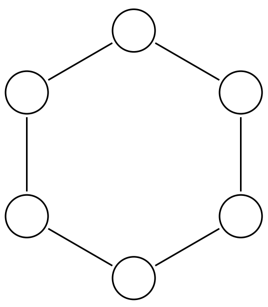
We have developed multiple data reduction rules, but another way to reduce the size of the search tree are *lower bounds*. A lower bound can be thought of as a function $\ell(G)$ of the graph G such that $\ell(G) \leq |S|$, where S is an optimal 2-club vertex deletion set for G. Lower bounds are a very practical way to reduce the size of the search tree, because if $k < \ell(G)$, then we know that there is no solution.

Lower Bound 4.1. *Let $\mathcal{P} = \{p_1, \dots, p_\ell\}$ be a set of vertex-disjoint restricted P_4's in G. Then a minimum 2-club vertex deletion set for G has size at least ℓ.*

Proof. As long as the restricted P_4's in $\mathcal{P}$ exist, the graph cannot be a cluster graph. Because no two paths in $\mathcal{P}$ have a common vertex, at least one vertex from each P_4 has to be deleted before the graph can become a 2-club cluster graph. □

The number of vertex-disjoint P_4's cannot be greater than $\frac{n}{4}$. While we have not been able to prove any *upper bounds* for 2-Club Cluster Vertex Deletion better than a trivial $n-3$ upper bound, this value is a factor of 2 away from the 2-club vertex deletion size $\frac{n}{2} - 1$ for some graphs we found (see Figure 4.4). These graphs had a minimum vertex cut of size $\frac{n}{2} - 1$.

The size of a minimum vertex cut can also be used as a lower bound if we know that an optimal 2-club vertex deletion set S for G splits it into multiple 2-clubs, which means that S must be a vertex cut set. However, an optimal 2-club vertex deletion set is not always a vertex cut set. The following lower bound overcomes this problem:


(a) $k = 2, i = 0, n = 6$

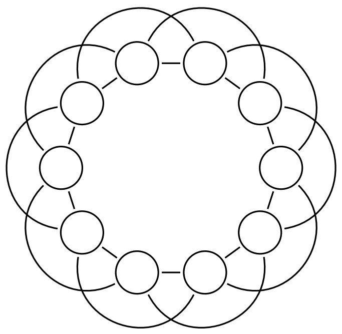

(b) $k = 4, i = 1, n = 10$

Figure 4.4: Graphs where a minimum 2-club vertex deletion set has a size of $k = \frac{n}{2} - 1$, which can be found by picking any vertex and deleting all its neighbors. There appears to be a family of graphs with this property. For $i \in \mathbb{N}$: create a circle with $n = 6 + 4i$ vertices such that each vertex has an edge to the first $w = 1 + i$ vertices to its left and right side.

Lower Bound 4.2. *Let G be a connected graph, t the size of the largest 2-club in G, and c the minimum vertex cut of G. Then a minimum 2-club vertex deletion set for G has size of at least* $\min(n - t, c)$*.*

Proof. Let S be an optimal 2-club vertex deletion set and $G' = G[V \setminus S]$ the resulting 2-club cluster graph. If G' contains at least two components, then S is a vertex cut set for G and $|S| \geq c$. If G' has only one component, then $V \setminus S = V(G')$ is the largest 2-club in G and $|S| = n - t$. □

4.4 Introducing weights

Weights allow to further control the type of solution that will be found. We will allow vertices to have weights. Some vertices might be more important than others, which is why one might consider giving them a higher weight to avoid their deletion. However, we will only use weights to allow for more flexibility while designing data reduction rules and lower bounds. Similarly, Böcker et al. [Böc+09] were able to develop an elegant $\mathcal{O}^*(1.82^k)$ algorithm for Cluster Editing through a branching rule that allowed vertex merging thanks to edge weights. We will develop a data reduction rule which will allow us to merge twin vertices.

We now consider a weighted variant of 2-Club Cluster Vertex Deletion, where along with G and k we are given a positive weight function $w : V \to \mathbb{N}^+ \cup \{\infty\}$. For a set $A \subseteq V$ we define $w(A) := \sum_{v \in A} w(v)$.

Weighted 2-Club Cluster Vertex Deletion (Weighted 2CVD)

Input: An undirected graph G and an integer $k \in \mathbb{N}$ and a weight function $w : V \to \mathbb{N}^+ \cup \{\infty\}$.

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

We can clearly convert an unweighted 2-Club Cluster Vertex Deletion instance to a weighted one by assigning each vertex a weight of one. Furthermore for Cstr2CVD permanent vertices can be modeled by assigning them an infinite weight, or if strictly integer weights are sought a weight of $k + 1$. We will see that most of our results for Cstr2CVD also apply to the weighted variant, due to a simple way to convert a weighted instance to an unweighted one. Some data reduction rules can be made more powerful thanks to weights. A problem with this weighted variant is that given a binary encoding of the weights the minimum 2-club vertex deletion set cost k can be exponentially larger than the size of the encoding, meaning that a FPT approach might not be practically feasible. However, because we convert unweighted instances to weighted ones this is not a big concern for us.

We want to show a surprising connection between Weighted 2-Club Cluster Vertex Deletion and its unweighted variant. For this we require a simple observation about the structure of 2-clubs with twins. Recall that twins are vertices u and v which have the same neighborhood and may be adjacent, i.e., $N(u) = N(V)$ or $N[u] = N[v]$.

Observation 4.10. *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.*

The following data reduction rule allows us to “compress” a graph by “merging” twins, because, as we will see, they are always in the same 2-club in an optimal solution. Two vertices u and v in the graph G can be merged by creating a new vertex x such that in the transformed graph G' it holds that $N_{G'}(x) = (N_G(u) \cup N_G(v)) \setminus \{u, v\}$ and $w(x) = w(u) + w(v)$. This can be simplified in the case of twins as seen below.

Reduction Rule 4.8. *Given two vertices $u, v \in V$ such that either $N[u] = N[v]$ or $N(u) = N(v)$, i.e., u, v are twins, then v can be merged into u as follows:*

- *delete v and*
- *set $w(u)$ to $w(u) + w(v)$.*

Lemma 4.11. *Reduction Rule 4.8 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.

“$\Leftarrow$”: 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 4.10 S' is also a 2-club vertex deletion set for G.

“$\Rightarrow$”: 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 4.10 $S' = S \setminus \{u\}$ 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 a contradiction to the optimality of S. □

Equivalence of weighted and unweighted instances There is a simple way to convert an instance (G, w, k) with positive integer weights into an equivalent unweighted instance (G', k). Each vertex v in the weighted instance can be replaced by $w(v)$ many twins of v each with a weight of one. After this is done for all vertices the vertices in the resulting graph all have a weight of one and the minimum 2-club vertex deletion set cost of both graphs is the same. The correctness follows immediately from Reduction Rule 4.8. Because the vertices all have a weight of one, we can discard weights and are left with an equivalent unweighted instance.

Due to this duality of the weighted and unweighted instance, the constraints, data reduction rules, and lower bounds for 2-Club Cluster Vertex Deletion can be easily adapted for the weighted variant. This is what we did for our solver. For brevity, however, we only describe a few improved rules which make use of weights. We only note that all previous branching and data reduction rules and lower bounds still apply without restrictions, except for Reduction Rule 4.5 and Reduction Rule 4.6 where the vertex v must have a weight of one. Additionally, while deleting a vertex we have to decrease k by its weight rather than one.

An important idea that we used repeatedly for Lower Bound 4.1 is to find many vertex-disjoint restricted P_4's, and in the case of Reduction Rule 4.3 many restricted P_4's that overlap in a single vertex, but are otherwise vertex-disjoint. One difficulty with this is that finding a maximum set of vertex-disjoint P_4's is NP-hard [IPS82], thus finding such a set of restricted P_4's is likely also NP-hard. A practical implementation would then employ heuristics. Separately from that another challenge might be that the maximum number of such vertex-disjoint paths is small, due to each P_4 containing four vertices. We exploit permanent vertices by switching from using vertex-disjoint paths to paths that may overlap multiple times in vertices whose weight is greater than one, in particular we allow restricted P_4's to overlap infinitely many times in permanent vertices

The following is a generalization of Reduction Rule 4.3 to weighted graphs. We can relax the restrictions and allow restricted P_4's to overlap in more than just the vertex v, if the other vertices they overlap in have a weight of more than 1. This is particularly the case for permanent vertices. Clearly, we cannot afford to remove a vertex with infinite weight, so we can allow multiple restricted P_4's to contain this vertex and still consider them "disjoint".

Reduction Rule 4.9. *Let $\mathcal{P}$ be a multiset of restricted P_4's that each contain the vertex v and such that each vertex $a \in V$ other than v is contained in at most $w(a)$ many restricted P_4's in $\mathcal{P}$. If $|\mathcal{P}| \geq k + 1$, then delete v and decrease k by $w(v)$.*

Lemma 4.12. *Reduction Rule 4.9 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 ℓ the number of P_4's in $\mathcal{P}$ that contain u, which means deleting u eliminates ℓ P_4's in $\mathcal{P}$. The cost of deleting u is $w(u) \geq \ell$. This means eliminating all P_4's in $\mathcal{P}$ without deleting v has a cost of at least $|\mathcal{P}| \geq k + 1$, which is not possible with a budget of k. □

A perhaps more intuitive proof for the correctness of Reduction Rule 4.9 is to use Reduction Rule 4.8 "in reverse" (see Figure 4.5) to split vertices which appear in multiple

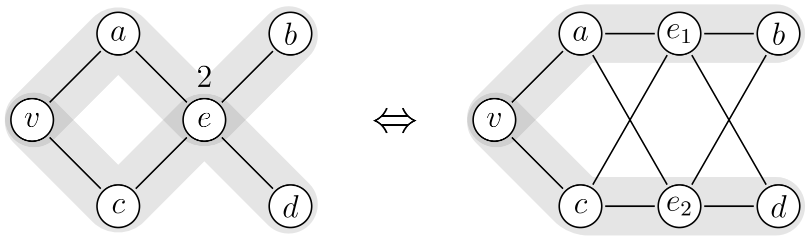


Figure 4.5: Example of graphs where Reduction Rule 4.9 can be applied on the vertex v when $k = 1$. If a vertex's weight is not one, then the number above the vertex is its weight. The gray paths outline restricted P_4's in the multiset $\mathcal{P}$. For the left graph $\mathcal{P} = \{vaeb, vced\}$. The right graph can be transformed into the left one by merging e_1 and e_2 to create the vertex e using Reduction Rule 4.8. Notice how in the right graph the multiset $\mathcal{P}' = \{vae_1b, vce_2d\}$ has the desired property from Reduction Rule 4.3 of containing only restricted P_4's that overlap only in v. This means that merging twins has no impact on the applicability of Reduction Rule 4.9.

restricted P_4's into twins (while distributing the weight of the vertex arbitrarily, but ensuring that each vertex has a weight of at least one). That way we can transform the graph in such a way that one can translate the multiset $\mathcal{P}$ into $\mathcal{P}'$ such that $\mathcal{P}'$ is a set of vertex-disjoint restricted P_4's in the transformed graph that overlap only in v and then use the idea from Reduction Rule 4.3 to argue that v must be deleted.

We use exactly the same idea as with Reduction Rule 4.9 to improve Lower Bound 4.1.

Lower Bound 4.3. *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 a size of at least ℓ.*

Proof. Let $\mathcal{P}$ be as above and $v \in V$ be any vertex. Denote by ℓ the number of P_4's in $\mathcal{P}$ that contain v, which means deleting v eliminates ℓ P_4's in $\mathcal{P}$. The cost of deleting v is $w(v) \geq \ell$. This means eliminating all 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. □

Closing discussion We presented a sophisticated search tree algorithm with multiple data reduction rules and lower bounds. We started with the plain 2-Club Cluster Vertex Deletion problem and switched to the constrained variant Cstr2CVD in which we could formally embed permanent vertices. The generalized variant Weighted 2-Club Cluster Vertex Deletion allowed for even more flexibility while designing data reduction rules and lower bounds. In particular it allowed us to merge twin vertices. It also allowed us to represent permanent vertices as vertices with infinite weight.

Unfortunately, our results were not enough to guarantee a polynomial problem kernel. Due to the non-hereditary property of 2-clubs, we had to take particular care while designing data reduction rules to not accidentally create more restricted P_4's.

5 2-Club Cluster Edge Deletion

We want to show that 2-Club Cluster Vertex Deletion shares many properties with 2-Club Cluster Editing and 2-Club Cluster Vertex Deletion. Similarly to 2-Club Cluster Vertex Deletion we will use a search tree approach and include permanent and forbidden edges as constraints. We will use the observations we made for 2-Club Cluster Editing that deleting an edge ab from the graph makes the edge forbidden and through a simple constraint we can enforce that a and b must be in different 2-clubs. Recall that a 2-club to which an edge is inserted remains a 2-club, which is why if an edge ab is deleted, then this can only be optimal if a and b are to be separated into different 2-clubs.

We formally capture permanent edges in the following problem definition:

Constrained 2-Club Cluster Edge Deletion (Cstr2CED)

Input: An undirected graph $G = (V, E)$, an integer $k \in \mathbb{N}$ and a set $E_p \subseteq E$ of permanent edges.

Question: Is there an $F \subseteq E$ with $|F| \leq k$ and $F \cap E_p = \emptyset$ such that $G' = G[E \setminus F]$ is a 2-club cluster graph?

We will also include a set E_f of forbidden edges as input to remember edges which have been deleted. At the start E_f is empty. We did not include it in the above problem definition, due to problems with formality. We will rather treat forbidden edges as an optional "hint" than a strict constraint of the problem.

An instance (G, k) of 2-Club Cluster Edge Deletion is clearly equivalent to the instance $(G, k, \emptyset, \emptyset)$ of Cstr2CED.

While we cannot insert edges into the graph and mark them as permanent, as we could for 2-Club Cluster Editing, there are other ways of generating permanent edges. Consider the following branching rule inspired by Branching Rule 4.2. For any restricted P_4 $stuv$ at least one of the edges st, tu, uv need to be deleted, but if a solution F deletes two or more, then this solution would be found in two or more branches. The following branching rule reduces this overlap.

Branching Rule 5.1. *Let $\mathcal{I} = (G, k, E_p, E_f)$ be an instance of* Cstr2CED. *If G is not a 2-club cluster graph, then find a restricted P_4 $stuv$ and split $\mathcal{I}$ into three smaller instances $\mathcal{I}_{st}, \mathcal{I}_{tu}, \mathcal{I}_{uv}$ as follows:*

- $\mathcal{I}_{st} = (G - st, k - 1, E_p, E_f \cup \{st\})$,
- $\mathcal{I}_{tu} = (G - tu, k - 1, E_p \cup \{st\}, E_f \cup \{tu\})$,
- $\mathcal{I}_{uv} = (G - uv, k - 1, E_p \cup \{st, tu\}, E_f \cup \{uv\})$.

If an instance $\mathcal{I}_x$ was derived by removing a permanent edge $x \in \{st, tu, uv\} \cap E_p$ then skip $\mathcal{I}_x$.

Lemma 5.1. *Branching Rule 5.1 is correct.*

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

"$\Leftarrow$": Let F' be a 2-club edge deletion for one of the three instances where the edge $x \in \{st, tu, uv\}$ was deleted. Because all three instances have a set of permanent and forbidden edges that is a superset of E_p and E_f, respectively, the set $F = F' \cup \{x\}$ is a 2-club edge 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 edges st, tu or uv or has to be deleted. Let F be a 2-club edge deletion set for G with $|F| \leq k$ and $F \cap E_p = \emptyset$, and such that for any edge $ab \in E_f$ the vertices a and b are in different components of $G[E \setminus F]$. If $st \in F$, then $\mathcal{I}_{st}$ is clearly a yes instance; otherwise if $tu \in F$, then $\mathcal{I}_{tu}$ is a yes-instance; otherwise if $uv \in S$, then $\mathcal{I}_{uv}$ is a yes-instance. □

A very important property of permanent edges is transitivity. We capture this in the following data reduction rule.

Reduction Rule 5.1. *For any edges $ab, bc, ac \in E$ if the first two are permanent, then mark ac as permanent.*

Lemma 5.2. *Reduction Rule 5.1 is safe.*

Proof. Because ab and bc are permanent that means that a, b and c must be in the same 2-club of $G[E \setminus F]$ for any solution F. Because no optimal solution F would delete the edge ac it can be marked as permanent. □

Permanent edges are not the only thing that necessitates that two vertices must be in the same 2-club. The budget k can sometimes not be enough to separate the two vertices.

Reduction Rule 5.2. *If two adjacent vertices $a, b \in V$ with $|N(a) \cap N(b)| \geq k$ exist, then mark the edge ab as permanent.*

Lemma 5.3. *Reduction Rule 5.2 is safe.*

Proof. Because a and b are adjacent and have k common neighbors, then a minimum edge cut set that separates a and b has to delete at least $k+1$ edges. This is not possible with a budget of k which means a and b must be in the same 2-club and the edge ab can be marked as permanent. □

We are also able to devise a simple equivalent of Reduction Rule 4.5 for CSTR2CED.

Reduction Rule 5.3. *Given a non-permanent edge uv whose deletion isolates a 2-club C from the graph, then mark all edges in C as permanent.*

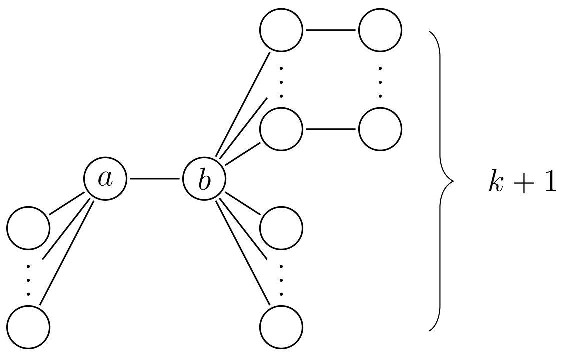


Figure 5.1: An example of a graph that has $k + 1$ P_4's that overlap only in the edge ab. Here we also illustrate the two types of restricted P_4's that can contain ab: either ab is at the beginning or it in the middle of the restricted P_4.

Lemma 5.4. *Reduction Rule 5.3 is safe.*

Proof. Assume there exists a solution F that deletes an edge ab in the 2-club C. Then $F' = (F \setminus \{ab\}) \cup \{uv\}$ would be another solution of the same size, because any restricted P_4's that contain edges from C must also contain the edge uv. □

So far we only used forbidden edges to remember deleted edges. The next rule will take advantage of the opposite function of permanent and forbidden edges.

Reduction Rule 5.4. *Let $A, B \subseteq V$ be two different components in $G[E_p]$. If for some $a \in A$ and $b \in B$ the edge ab is forbidden, then delete all edges between A and B from G and decrease k by the number of deleted edges or ∞ if this would delete a permanent edge.*

Lemma 5.5. *Reduction Rule 5.4 is safe.*

Proof. The permanent edges E_p enforce that the vertices in A must be in the same 2-club, as do the vertices in B. Because the edge ab was deleted this means that a and b must be in different 2-clubs, which means that the vertices in A and B must be in two different 2-clubs and all edges between them need to be deleted. □

With 2-Club Cluster Edge Deletion we are also interested in disjoint P_4's, but unlike 2-Club Cluster Vertex Deletion they do not need to be vertex-disjoint, but rather edge-disjoint which is much less restrictive. We can use these edge-disjoint P_4's for data reduction rules and lower bounds. The following data reduction rule is an equivalent of Reduction Rule 4.3. For an example see Figure 5.1.

Reduction Rule 5.5. *Given $k + 1$ restricted P_4's that overlap only in the edge ab, but are otherwise edge-disjoint, then delete ab mark it as forbidden and decrease k by one or ∞ if $ab \in E_p$.*

Lemma 5.6. *Reduction Rule 5.5 is safe.*

Proof. If G contains $k + 1$ restricted P_4's that overlap only in ab, then if ab is not deleted, then one edge from each of the $k + 1$ restricted P_4's has to be deleted. This is not possible with a budget of k. Deleting this edge is only optimal if a and b are to be separated into different 2-clubs, which means we can mark the edge as forbidden. □

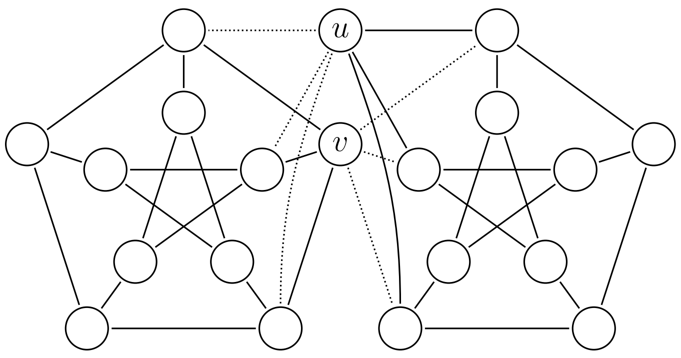


Figure 5.2: An example for a graph with twins u, v which are separated into different 2-club clusters in an optimal solution. The dotted edges indicate edges that are deleted in an optimal solution.

Similarly we create an equivalent of Lower Bound 4.1 for 2-Club Cluster Edge Deletion.

Lower Bound 5.1. *Let $\mathcal{P} = \{p_1, \dots, p_\ell\}$ be a set of edge-disjoint restricted P_4's in G. Then a minimum 2-club edge deletion set for G has a size of at least ℓ.*

Proof. While the restricted P_4's in $\mathcal{P}$ exist the graph cannot be a cluster graph. Because no two paths in $\mathcal{P}$ have a common edge, at least one edge from each P_4 has to be deleted before the graph becomes a 2-club cluster graph. □

We could extend Reduction Rule 5.5 and Lower Bound 5.1 to also incorporate permanent edges as we did for 2-Club Cluster Vertex Deletion in Reduction Rule 4.9 and Lower Bound 4.3 to allow the restricted P_4's to overlap in permanent edges.

However, unlike 2-Club Cluster Vertex Deletion, 2-Club Cluster Edge Deletion does not allow us to treat twins exactly the same. For an example of a graph where this is the case see Figure 5.2. Notice how that graph has two overlapping Petersen graphs, and that the vertex v they overlap in has a twin u. We can separate the two Petersen graph with six edge deletions by "assigning" each twin u and v to a different Petersen graph. However, if we wanted to find a solution in which the vertices u and v are in the same 2-club, then we would have to delete at the very least six edges from each Petersen graph.

The intuition for why this happens is because the following. A 2-club edge deletion set F partitions the vertices in a graph into 2-clubs. When deciding to which 2-club a vertex should be assigned we may be faced with multiple choices. For example consider again the graph from Figure 5.2, but without the vertex u. We could assign the vertex v to the left Petersen graph, however, this would force us to delete many edges from the right Petersen graph, because the vertex v is very "useful". However, because u and v are twins we can assign v to the left and u to the right Petersen graph.

6 Implementation and Experiments

In this section we want to cover results from an implementation of solvers. The deletion of a vertex is more "powerful" than the deletion of an edge and therefore it is probable that 2-Club Cluster Vertex Deletion is easier to solve than 2-Club Cluster Edge Deletion. For this reason we only try to solve 2-Club Cluster Vertex Deletion. For this we implemented our Weighted 2-Club Cluster Vertex Deletion solver and we assign each vertex from a unweighted instance a weight of one. Recall that twin merging required vertex weights and that weights were useful for explicitly modeling permanent vertices.

In Section 6.1 we describe what configurations of our solver we will be using for our experiments, whereas in Section 6.2 we discuss some important aspects of the implementation. Then finally in Section 6.3 we describe our experiments on a biological dataset.

6.1 Setup and solver configurations

We implemented our search tree with all data reduction rules and lower bounds for Weighted 2-Club Cluster Vertex Deletion. This solver computes a 2-club vertex deletion set size of minimum cost and outputs the solution set. Recall that we used Weighted 2-Club Cluster Vertex Deletion to make it possible to merge twins and explicitly model permanent vertices as vertices with infinite weight. The input of our solver expects an unweighted graph and then assigns each vertex a weight of one. The solver could be easily adapted to accept weighted instances, but we only focus on unweighted ones.

Solver configurations To later test the impact of the different combinations of our data reduction rules and lower bounds we introduce the following configurations of our solver:

- solver ALL (almost everything) - which includes Branching Rule 4.2, Reduction Rules 4.1, 4.2, 4.4 and 4.6 to 4.9, and Lower Bound 4.3. These also also improve or generalize Reduction Rules 4.3 and 4.5 and Lower Bound 4.1
- solver NPV (no permanent vertices) - implements Branching Rule 4.1, Reduction Rules 4.1, 4.8 and 4.9, and Lower Bound 4.3 which also improve or generalize Reduction Rule 4.3 and Lower Bound 4.1
- solver NTM (no twin merging) - implements everything in solver ALL except Reduction Rule 4.8

Solver ALL is only missing Lower Bound 4.2, which we implemented, but disabled for reason we describe in Section 6.2. Note also that for the implementation of some data reduction rules and lower bounds we use heuristics; see Section 6.2 for details. All of these configurations apply all their data reduction rules at each recursive step. These data reduction rules are not applied exhaustively, but rather we try to apply them once to each vertex.

CPLEX We will later compare the performance of our solver against the ILP formulation from the beginning of Chapter 4 solved using the commercial solver CPLEX.

We used a recent version of CPLEX, 12.8, for our experiments. We use mostly default parameters and only set `mip tolerances mipgap` and `absmipgap` to zero and enabled `emphasis numerical`. These are parameters that Akiba and Iwata [AI16] used for finding minimum vertex covers in order to force CPLEX to find optimal solutions (although they report that even with these parameters some solutions were not optimal). While we did not have problems with CPLEX not returning optimal solutions we did not want CPLEX perhaps cheating by using heuristics that do not guarantee optimal solutions. However, this had only a small impact on the running time, with all instances being solved $\pm 10\%$ slower or faster, except one which was twice as slow with these parameters.

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 a small advantage of CPLEX, because our solver was only written to use a single thread.

6.2 Implementation

The implementation was written in C++ and uses only the C++ standard library. The source code is available at: `https://fpt.akt.tu-berlin.de/software/two-club-editing/two-club-vertex-deletion.zip` We would now like to discuss some important details of the implementation.

Some aspects of our algorithm from Weighted 2-Club Cluster Vertex Deletion have been intentionally left open. For example we do not say how to compute a multiset of restricted P_4's that can overlap in complex ways. It is clear that ideally we would like to find such a multiset whose size is maximum. This is likely an NP-hard problem (refer to Itai, Perl, and Shiloach [IPS82]), 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 aspects of heuristics involved or other challenges that could be of interest.

Finding a minimum 2-club vertex deletion set size In order to find a 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.

Branching Rule 4.2 This is the branching rule that allows us to mark some vertices as permanent and skip branches in which a permanent vertex would have been deleted. It

Algorithm 1 Finding a minimum 2-club vertex deletion set

```
Input: A graph $G = (V, E)$
Output: The minimum 2-club vertex deletion set size of $G$
1: initialize the weight function $w$ with $w(v) = 1$ for each $v \in V$
2: $(G', \infty, w') \leftarrow$ apply data reduction rules to $(G, \infty, w)$
3: $k \leftarrow$ LOWERBOUND$(G', w')$
4:
5: while $(G', k, w')$ is a no-instance of WEIGHTED 2CVD
6:         $k \leftarrow k + 1$
7:
8: return k
```

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, we select the one in which the average weight of non-permanent vertices is the highest, because then on average k is decremented by a larger value in the branches and thus also making the search tree smaller.

Handling multiple components Each component can be solved separately. 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. While this only gives an improvement by a constant factor of at most 4, the effect is much more noticeable when the graph repeatedly decomposes into one large component and a few smaller ones. If from the root of a search tree to some leaf this happens i many times, then we have a speedup of up to 4^i along those branches of the search tree.

Reduction Rule 4.9 This rule allows us to delete the 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 multiset $\mathcal{P}$ 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 multiset.

We focus on finding a maximum multiset that only contains such restricted P_4's that start in the vertex v. This means we do not try to find P_4's where v might be in the "middle" (see Figure 4.1). This can then be modeled as a maximum flow problem. The algorithm is described in Algorithm 2. 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

Algorithm 2 Heuristic for Reduction Rule 4.9

```
Input: A graph $G = (V, E)$, with a weight function $w$, a vertex $v$ and $k \in \mathbb{N}$
Output: true if Reduction Rule 4.9 can be applied to remove $v$ from $G$
 1: Create a directed graph $G_f$ containing only the vertex $s$ and $t$
 2: for each $i \in \{1, 2, 3\}$                    // Split vertices into layers based on distance
 3:     $D_i \leftarrow$ all vertices with distance $i$ to $v$ in $G$
 4: for each $u \in D_1 \cup D_2 \cup D_3$                        // Limit flow through a vertex
 5:     add the vertex $u_{\text{in}}$ and $u_{\text{out}}$ to $G_f$
 6:     add the edge $(u_{\text{in}}, u_{\text{out}})$ with a capacity of $w(u)$ to $G_f$
 7: for each $u \in D_1$                                        // Connect layer 1 to source
 8:     add the edge $(s, u_{\text{in}})$ with infinite capacity to $G_f$
 9: for each $i \in \{1, 2\}$                                          // Connect the layers
10:     for each $u \in D_i$
11:         for each $x \in D_{i+1}$ with $\{u, x\} \in E$
12:             add the edge $(u_{\text{in}}, x_{\text{out}})$ with infinite capacity to $G_f$
13: for each $u \in D_3$                                          // Connect layer 3 to sink
14:     add the edge $(u_{\text{out}}, t)$ with infinite capacity to $G_f$
15:
16: $f \leftarrow$ maximum $s - t$ flow in $G_f$
17: if $f > k$ then return true
18: return false
```

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 that path means adding the restricted P_4 $vu_1u_2u_3$ to $\mathcal{P}$. The final, maximum flow in G_f does not uniquely identify a multiset $\mathcal{P}$, however the value of the maximum flow tells us the size of all maximum $\mathcal{P}$'s. Because we are only interested in the size of the multiset $\mathcal{P}$ this is all we need.

The difficulty of finding restricted P_4's where v is in the "middle" comes from the need to partition v's neighbors, as for a restricted P_4 $avbc$, both a and b are neighbors of v, but the role of a and b is slightly different, because b needs to be adjacent to c. Additionally, if we were able to overcome this partitioning problem we might have to solve not a maximum flow, but rather maximum 4 or 5-Length-Bounded Flow. The maximum L-Lenght-Bounded Flow problem is to find a maximum flow which can be decomposed into flows along paths of length at most L. For $L = 5$ it is NP-hard [IPS82] while NP-hardness for $L = 4$ is an open problem.

Lower Bound 4.3 Here we use a multiset of restricted P_4's $\mathcal{P}$ such that each vertex v is present in at most $w(v)$ many restricted P_4's. The size of this multiset is then the lower bound. We compute the multiset $\mathcal{P}$ 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

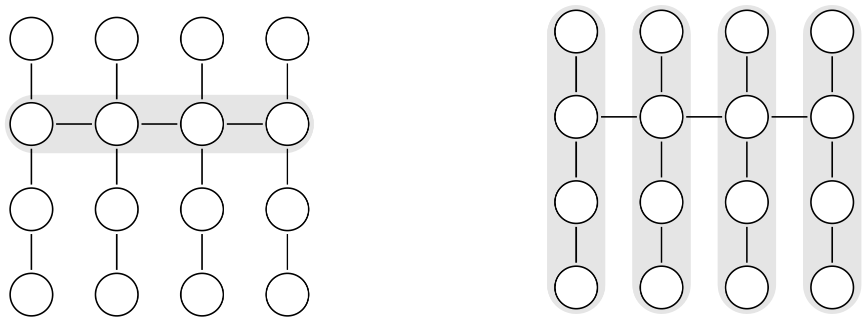


Figure 6.1: Two maximal sets of disjoint restricted P_4's in the same graph. Each P_4 is represented by a gray path. The restricted P_4 on the left contains the highest degree vertices, each of which could have been in their own disjoint restricted P_4 like on the right. Our heuristic for Lower Bound 4.3 prevents such a bad case.

of this P_4 their counter is positive. The restricted P_4 is not chosen randomly, but rather we select such 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 we repeat this step and search for another P_4.

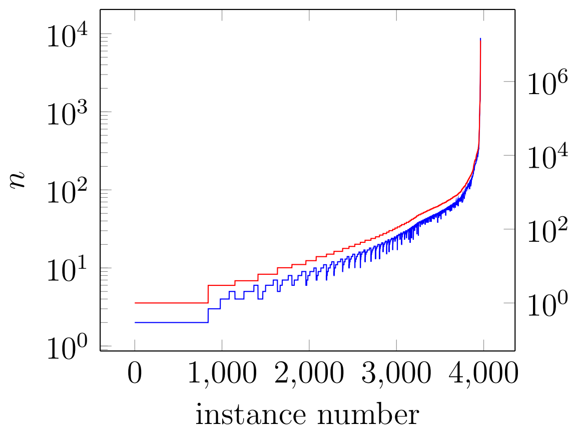
The reason for minimizing the sum of the degrees is that selecting a P_4 which contains many high degree vertices might overlap with and therefore likely exclude many other restricted P_4's (see Figure 6.1).

Lower Bound 4.2 The lower bound is $\min(n-t, c)$ where t is the largest 2-club and c is the minimum vertex cut of G. Computing t involves solving 2-CLUB, which is NP-complete. It would suffice to use an upper bound for t, however we are not aware of any good upper bounds. A very practical solver from Hartung, Komusiewicz, and Nichterlein [HKN15] is able to compute this value even for large real-world graphs. This solver is invoked as a separate process each time this lower bound is computed. However, invoking the process and writing the graph to a file in each recursive step has a very large overhead so this lower bound was disabled in our experiments.

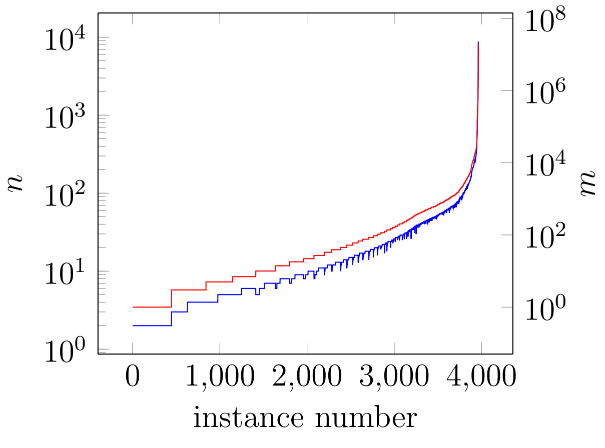
The vertex connectivity c can be computed by solving $\mathcal{O}(n+\delta(G)^2)$ maximum flow problems [EH84]. Our implementation runs in $\mathcal{O}((n+\delta(G)^2)nm)$.

6.3 Experiments

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. Our solver only needs very little memory ($\mathcal{O}(n+m)$), but we have seen CPLEX use even 30GB of RAM. For the running time measurements of our solver and CPLEX we used wallclock time. We will be testing different configurations of our solver from Section 6.1 and an ILP formulation solved using CPLEX which we will simply refer to as CPLEX from now on. For running time measurements of CPLEX we excluded the time it takes to build the ILP model, which



(a) Bio33



(b) Bio50

Figure 6.2: Two graphs showing the number of vertices n in blue and the number of edges m in red in the bio33 and bio50 instances. The instance number was 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.

can have $\mathcal{O}(n^4)$ constraints. For instances with 250 vertices this process can take 20 seconds, and sometimes even 60. However, in a vast majority of cases, the build time was at most 30% of the total running time.

6.3.1 Dataset

For our analysis we used a real-world biological dataset[1] that has been used for the evaluation of Weighted Cluster Editing solvers [Böc+09; Rah+07]. The vertices in the graphs represent protein sequences and between each vertex there is an edge whose weight represents some sort of similarity of the proteins. The edge weights can be positive or negative.

A graph with weighted edges does not match the input of 2-Club Cluster Vertex Deletion. Hartung and Hoos [HH15] 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 [HH15] used the values $c = 33$, $c = 50$ and $c = 66$, which is also what we did, and 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 Figure 6.2 for the number of vertices and edges in the instances. The "noise" in the number of vertices is a results of us deleting degree zero vertices from the graphs. From Figure 6.2 we can see that these datasets contain many instances with less than 50 vertices and a few with

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

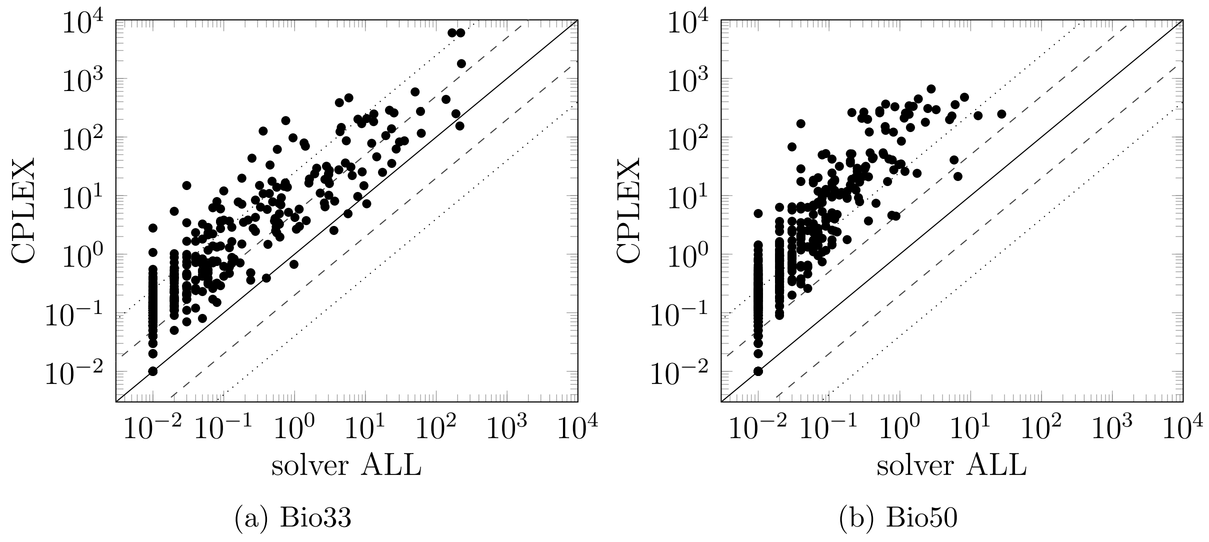


Figure 6.3: Running time comparison (in seconds) of our solver and the ILP solved by CPLEX. The diagonal lines mark running time factors of 1 (solid), 5 (dashed) and 25 (dotted). A running time of 6000s means that no solution was found in the 30 minute time limit.

around 8000 vertices. Our results show that the vast majority of instances with less than 100 vertices can be solved under a second. For this reason we want to focus on the harder instances. From each dataset we only kept instances with 50–250 vertices. From now on when referring to bio33 and bio50 we will mean these filtered datasets. After this filtering bio33 contains 430 instances, whereas bio50 446 instances.

6.3.2 Results

From these results in Figure 6.3 we can see 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. For bio33 it appears that for harder instances CPLEX is not much slower than our solver, with CPLEX potentially outperforming us on even larger instances. On bio50, CPLEX does very poorly compared to our solver. This is likely due to the minimum 2-club vertex deletion set size on these graphs being smaller than for bio33 (see Figure 6.4). The majority of bio33 instances has a minimum 2-club vertex deletion set size of up to 30, whereas bio50 up to 20. Our strategy for trying increasing values for k is then well suited for a dataset such as this. From Figure 6.4 we also see that usually at most a quarter of vertices are deleted, meaning that a large portion of the graph remains in the 2-club cluster graphs.

The process for building the ILP model for CPLEX is usually fairly short, 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 1.6GB (uncompressed) and includes 5.8 million constraints, whereas the original graph only takes up 72kB stored in an edge list format.

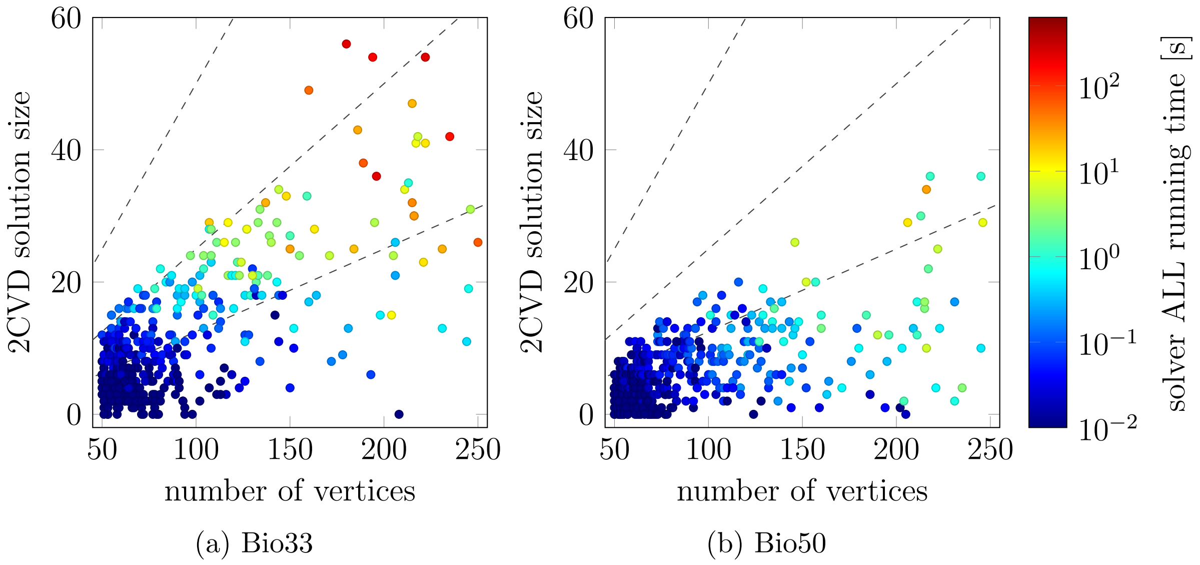


Figure 6.4: Comparison of the size of a minimum 2-club vertex deletion set and the number of vertices in our dataset. The dashed lines are $y = 0.5x$, $y = 0.25x$ and $y = 0.125x$.

In Figure 6.5 we compare the sizes of a minimum cluster vertex deletion set (CVD) size and the 2-club vertex deletion set (2CVD) size 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 impact of different configurations of our solver. We first tested a configuration that disables the marking of permanent vertices (solver NPV). From Figure 6.6 is clear that for the harder bio33 instances permanent vertices had a large improvement on the running time. For bio50 permanent vertices mostly slowed us down, very likely due to Reduction Rule 4.6 being an expensive data reduction rule. The vertex connectivty on the more dense bio50 instances is likely higher than for bio33 and it is therefore less likely that we can apply Reduction Rule 4.6, because there needs to be a cut vertex.

We further tested the impact of twin merging from Reduction Rule 4.8. For this we disabled merging of twins (solver NTM) and compared it to solver ALL. In Figure 6.7 we can see that twin merging gives us a modest average speedup of up to five for bio33 and a speedup of up to two for bio50. This difference is likely because in the less dense bio33 instances the average vertex degree is lower and, therefore, it is more likely that there are more twins.

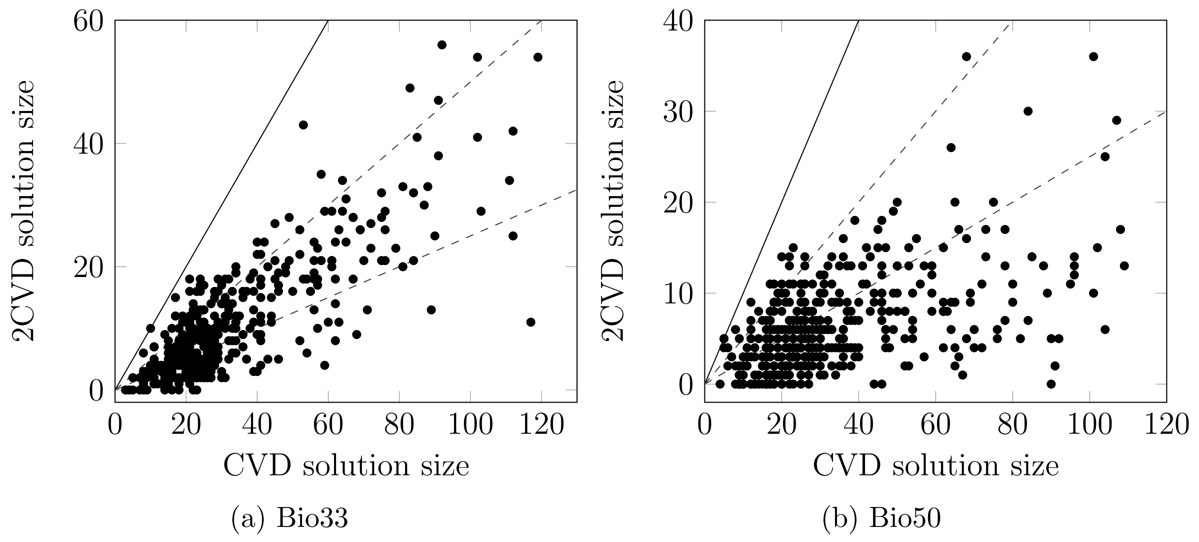


Figure 6.5: Comparison of the size of a minimum cluster vertex deletion set and a 2-club vertex deletion set on our dataset. The solid line is $y = x$, and the dashed ones are $y = 0.5x$ and $y = 0.25x$

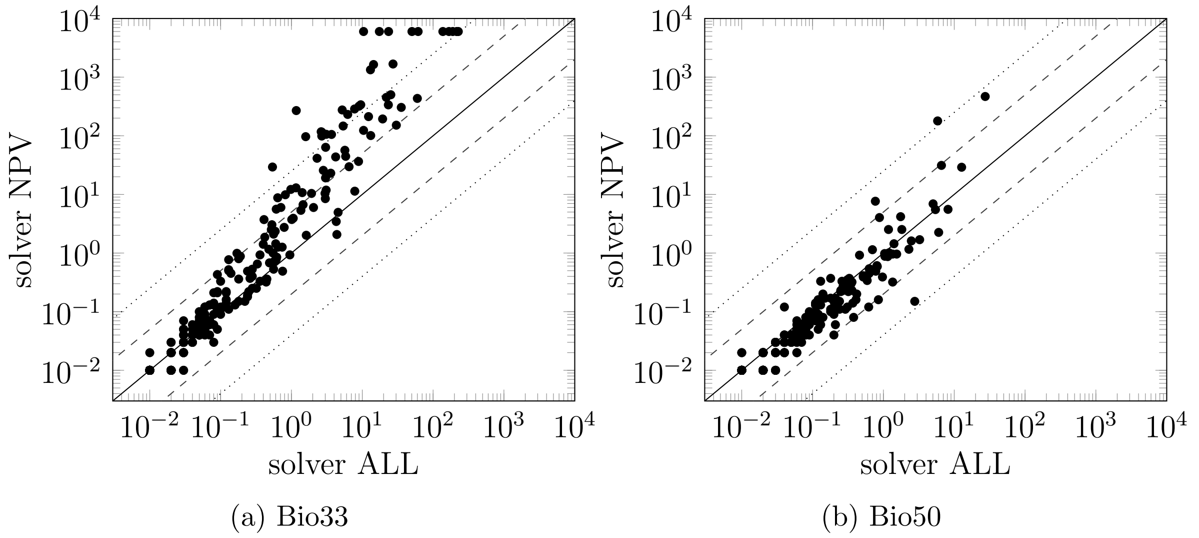
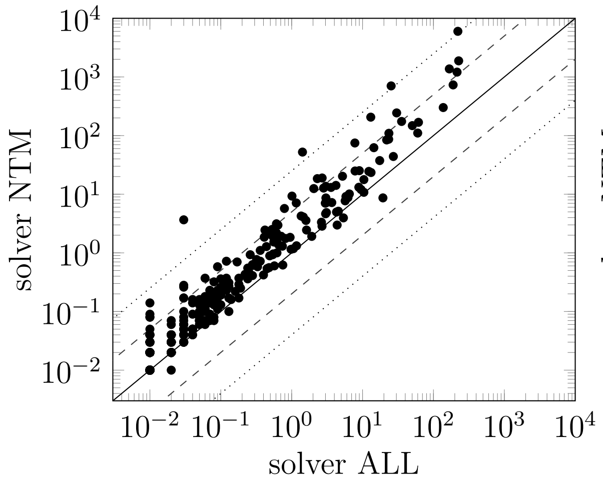
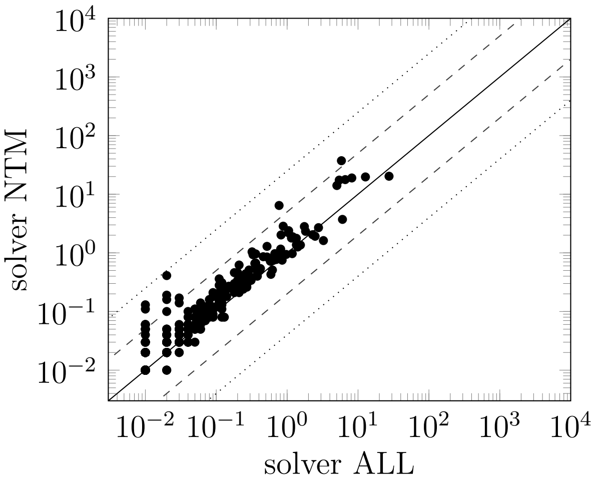


Figure 6.6: Running time comparison of different configurations of our solver—solver ALL versus solver NPV.



(a) Bio33



(b) Bio50

Figure 6.7: Running time comparison of different configurations of our solver—solver ALL versus solver NTM.

7 Conclusion

We investigated the problem of modifying graphs into 2-club cluster graphs. To this end, we considered three problem variants: 2-Club Cluster Editing, 2-Club Cluster Vertex Deletion and 2-Club Cluster Edge Deletion. We have shown that 2-Club Cluster Editing is W[2]-hard for the parameter solution size k, which means that it is likely not fixed-parameter tractable with respect to k. Furthermore, we developed sophisticated search tree algorithms for the fixed-parameter tractable 2-Club Cluster Vertex Deletion and 2-Club Cluster Edge Deletion problems and employed multiple data reduction rules and lower bounds. We also investigated the close connection between Weighted 2-Club Cluster Vertex Deletion and 2-Club Cluster Vertex Deletion and provide a simple way to translate weighted instances into unweighted ones. We provide an implementation of our 2-Club Cluster Vertex Deletion solver, which can be easily adapted to also accept weighted instances. Our experiments on a biological dataset show that we can solve problem instances with 250 vertices and 15,000 edges in usually well below 4 minutes. Our results also show that our techniques involving permanent vertices and merging of twins yielded significant speedups. Compared to an ILP formulation solved using CPLEX our solver was 5–25 times faster in most cases.

Future work & open problems It would be interesting to see if our results also generalize for s-clubs with $s \geq 3$, particularly the W[2]-hardness of 2-Club Cluster Editing with respect to k or perhaps even other parameters. An important question that remains is the existence of polynomial kernels for 2-Club Cluster Vertex Deletion and 2-Club Cluster Edge Deletion. For other 2-club related graph modification problems one could perhaps allow overlapping clusters or use stricter 2-club models such as well-connected 2-clubs [Kom+19]. Limiting the number of local manipulations (*locally bounded manipulation* [KU12]) might also lead to a better quality of the clustering, particularly for 2-Club Cluster Editing.

Literature

[Abu10] F. N. Abu-Khzam. “A kernelization algorithm for d-Hitting Set”. In: *Journal of Computer and System Sciences* 76.7 (2010), pp. 524–531. URL: https://doi.org/10.1016/j.jcss.2009.09.002 (cit. on p. 20).

[AI16] T. Akiba and Y. Iwata. “Branch-and-reduce exponential/FPT algorithms in practice: A case study of vertex cover”. In: *Theoretical Computer Science* 609 (2016), pp. 211–225. URL: https://doi.org/10.1016/j.tcs.2015.09.023 (cit. on p. 36).

[BBT05] B. Balasundaram, S. Butenko, and S. Trukhanov. “Novel Approaches for Analyzing Biological Networks”. In: *Journal of Combinatorial Optimization* 10.1 (2005), pp. 23–39. URL: https://doi.org/10.1007/s10878-005-1857-x (cit. on p. 6).

[Bor+16] A. Boral, M. Cygan, T. Kociumaka, and M. Pilipczuk. “A Fast Branching Algorithm for Cluster Vertex Deletion”. In: *Theory of Computing Systems* 58.2 (2016), pp. 357–376. URL: https://doi.org/10.1007/s00224-015-9631-7 (cit. on pp. 5, 6).

[Böc+09] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. “Going weighted: Parameterized algorithms for cluster editing”. In: *Theoretical Computer Science* 410.52 (2009), pp. 5467–5480. URL: https://doi.org/10.1016/j.tcs.2009.05.006 (cit. on pp. 17, 19, 27, 40).

[Böc12] S. Böcker. “A golden ratio parameterized algorithm for Cluster Editing”. In: *Journal of Discrete Algorithms* 16 (2012), pp. 79–89. URL: https://doi.org/10.1016/j.jda.2012.04.005 (cit. on pp. 5, 6).

[Cas+19] M. Castelli, R. Dondi, S. Manzoni, G. Mauri, and I. Zoppis. “Top k 2-Clubs in a Network: A Genetic Algorithm”. In: *Computational Science - ICCS 2019 - 19th International Conference*. Vol. 11540. Lecture Notes in Computer Science. Springer, 2019, pp. 656–663. URL: https://doi.org/10.1007/978-3-030-22750-0_63 (cit. on p. 6).

[CC12] Y. Cao and J. Chen. “Cluster Editing: Kernelization Based on Edge Cuts”. In: *Algorithmica* 64.1 (2012), pp. 152–169. URL: https://doi.org/10.1007/s00453-011-9595-1 (cit. on p. 6).

[Cyg+15] M. Cygan, F. V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-21275-3 (cit. on p. 10).

[DF13] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013. URL: https://doi.org/10.1007/978-1-4471-5559-1 (cit. on pp. 10, 11).

[EH84] A. Esfahanian and S. L. Hakimi. “On computing the connectivities of graphs and digraphs”. In: *Networks* 14.2 (1984), pp. 355–366 (cit. on p. 39).

[GHN13] Y. Gao, D. R. Hare, and J. Nastos. “The parametric complexity of graph diameter augmentation”. In: *Discrete Applied Mathematics* 161.10-11 (2013), pp. 1626–1631. URL: https://doi.org/10.1016/j.dam.2013.01.016 (cit. on p. 14).

[Guo+10] J. Guo, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. “A More Relaxed Model for Graph-Based Data Clustering: s-Plex Cluster Editing”. In: *SIAM Journal on Discrete Mathematics* 24.4 (2010), pp. 1662–1683. URL: https://doi.org/10.1137/090767285 (cit. on pp. 5, 6).

[HH15] S. Hartung and H. H. Hoos. “Programming by Optimisation Meets Parameterised Algorithmics: A Case Study for Cluster Editing”. In: *Learning and Intelligent Optimization - 9th International Conference, LION 2015.* Vol. 8994. Lecture Notes in Computer Science. Springer, 2015, pp. 43–58. URL: https://doi.org/10.1007/978-3-319-19084-6_5 (cit. on p. 40).

[HKN15] S. Hartung, C. Komusiewicz, and A. Nichterlein. “Parameterized Algorithmics and Computational Experiments for Finding 2-Clubs”. In: *Journal of Graph Algorithms and Applications* 19.1 (2015), pp. 155–190. URL: https://doi.org/10.7155/jgaa.00352 (cit. on pp. 6, 39).

[Hüf+10] F. Hüffner, C. Komusiewicz, H. Moser, and R. Niedermeier. “Fixed-Parameter Algorithms for Cluster Vertex Deletion”. In: *Theory of Computing Systems* 47.1 (2010), pp. 196–217. URL: https://doi.org/10.1007/s00224-008-9150-x (cit. on p. 6).

[IPS82] A. Itai, Y. Perl, and Y. Shiloach. “The complexity of finding maximum disjoint paths with length constraints”. In: *Networks* 12.3 (1982), pp. 277–286. URL: https://doi.org/10.1002/net.3230120306 (cit. on pp. 19, 29, 36, 38).

[Kom+19] 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”. In: *European Journal of Operational Research* 275.3 (2019), pp. 846–864. URL: https://doi.org/10.1016/j.ejor.2018.12.006 (cit. on pp. 6, 45).

[KU12] C. Komusiewicz and J. Uhlmann. “Cluster editing with locally bounded modifications”. In: *Discrete Applied Mathematics* 160.15 (2012), pp. 2259–2270. URL: https://doi.org/10.1016/j.dam.2012.05.019 (cit. on pp. 17, 45).

[Lee+10] V. E. Lee, N. Ruan, R. Jin, and C. C. Aggarwal. “A Survey of Algorithms for Dense Subgraph Discovery”. In: *Managing and Mining Graph Data.* Ed. by C. C. Aggarwal and H. Wang. Vol. 40. Advances in Database Systems. Springer, 2010, pp. 303–336. URL: https://doi.org/10.1007/978-1-4419-6045-0_10 (cit. on p. 5).

[Lok+13] D. Lokshtanov, N. Misra, G. Philip, M. S. Ramanujan, and S. Saurabh. “Hardness of r-Dominating Set on Graphs of Diameter $(r+1)$”. In: *Parameterized and Exact Computation*. Vol. 8246. Lecture Notes in Computer Science. Springer, 2013, pp. 255–267. URL: `https://doi.org/10.1007/978-3-319-03898-8_22` (cit. on pp. 14, 16).

[LZZ12] H. Liu, P. Zhang, and D. Zhu. “On Editing Graphs into 2-Club Clusters”. In: *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*. Vol. 7285. Lecture Notes in Computer Science. Springer, 2012, pp. 235–246. URL: `https://doi.org/10.1007/978-3-642-29700-7_22` (cit. on pp. 6, 12, 13, 21).

[Nie06] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006 (cit. on pp. 10, 11, 24).

[PYB12] J. Pattillo, N. Youssef, and S. Butenko. “Clique Relaxation Models in Social Network Analysis”. In: *Handbook of Optimization in Complex Networks: Communication and Social Networks*. New York, NY: Springer New York, 2012, pp. 143–162. URL: `https://doi.org/10.1007/978-1-4614-0857-4_5` (cit. on p. 6).

[Rah+07] S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truss, and S. Böcker. “Exact and heuristic algorithms for weighted cluster editing”. In: *Computational Systems Bioinformatics: (Volume 6)*. World Scientific, 2007, pp. 391–401. URL: `https://doi.org/10.1142/9781860948732_0040` (cit. on p. 40).

[Wit+10] T. Wittkop, D. Emig, S. Lange, S. Rahmann, M. Albrecht, J. H. Morris, S. Böcker, J. Stoye, and J. Baumbach. “Partitioning biological data with transitivity clustering”. In: *Nature methods* 7.6 (2010), pp. 419–420. URL: `https://doi.org/10.1038/nmeth0610-419` (cit. on p. 5).