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

Algorithmic Approaches to Cluster Editing on Multipartite Hypergraphs

Lara Glessen

Thesis submitted in fulfillment of the requirements for the degree
"Master of Science" (M. Sc.) in the field of Mathematics

February 2022

Supervisor and first reviewer:	Prof. Dr. Rolf Niedermeier
Second reviewer:	Prof. Dr. Martin Skutella
Co-Supervisor:	Dr. André Nichterlein

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 eigenhändige Ausfertigung versichert an Eides statt

Berlin, den _____ _____

Datum Unterschrift

Zusammenfassung

Diese Arbeit führt eine Verallgemeinerung des Problems Bicluster Editing ein, um auch Beziehungen höherer Ordnung zwischen Datenpunkten beim Clustering dieser berücksichtigen zu können. Die Eingabe ist ein ℓ-partiter ℓ-uniformer Hypergraph G und eine natürliche Zahl k, und die Frage ist, ob G durch höchstens k Modifikationen (Kanteneinfügungen + Kantenlöschungen) in einen ℓ-Cluster-Graphen umgewandelt werden kann, d.h. in jeder Zusammenhangskomponente sind alle möglichen Kanten innerhalb dieser vorhanden. Dieses Problem wird ℓ-P ℓ-U Cluster Editing genannt und es wird gezeigt, dass es für $\ell \geq 3$ NP-schwer ist. Dies motiviert die parametrisierte Komplexität zu untersuchen, im Zuge dessen wird eine Charakterisierung eines ℓ-Cluster-Graphen durch seine verbotenen Subgraphen angegeben. Mithilfe dieser werden zwei Suchbaumalgorithmen hergeleitet, die zeigen, dass ℓ-P ℓ-U Cluster Editing in $\mathcal{O}(3^k \cdot n^{2\ell-1})$ Zeit lösbar ist, wobei n die Anzahl der Knoten im Eingabegraphen ist. Außerdem wird diese Charakterisierung genutzt, um zu zeigen, dass ℓ-P ℓ-U Cluster Editing einen Problemkern mit $\mathcal{O}(k^2)$ Knoten für alle $\ell \geq 3$ besitzt. Als nächstes wird $\ell = 3$ fixiert, um einige der bisherigen Ergebnisse für diesen Spezialfall zu verbessern. Zuletzt wird analysiert, warum der Fall $\ell = 2$, also Bicluster Editing, gesondert betrachtet werden muss und es werden einige algorithmische Ansätze, die bereits für $\ell \geq 3$ vorgestellt wurden, auch für diesen Fall diskutiert.

Abstract

In this work, we introduce a generalization of the problem Bicluster Editing in order to handle relations of higher order between data points when clustering these. In this problem, we are given an ℓ-partite ℓ-uniform hypergraph G and an integer k, and the question is whether we can transform G by at most k modifications (edge insertions + edge deletions) into an ℓ-cluster graph, that is, in each connected component C all possible edges within C are present. We call this problem ℓ-P ℓ-U Cluster Editing and show its NP-hardness for $\ell \geq 3$. This motivates to study its parameterized complexity. Thus, we give a characterization of an ℓ-cluster graph by its forbidden subgraphs that allows us to derive two branching algorithms, yielding that ℓ-P ℓ-U Cluster Editing is solvable in $\mathcal{O}(3^k \cdot n^{2\ell-1})$ time, where n is the number of vertices in the input graph. Further, we use this characterization to show that ℓ-P ℓ-U Cluster Editing admits a problem kernel of $\mathcal{O}(k^2)$ vertices for all $\ell \geq 3$. Next, we fix $\ell = 3$ to improve some of the previous results for this special case. Last, we analyze why we have to study the case $\ell = 2$, that is, Bicluster Editing, separately and discuss some algorithmic approaches, that we already presented for $\ell \geq 3$, also for this case.

Contents

Chapter 1

Introduction

In many fields, data clustering became a fundamental task. Hence, many formulations try to capture what constitutes a good clustering [KN11; LZZ12; NG04; RB08]. In the setting we study, the goal is to partition a set of data points into clusters such that the points within the same cluster are similar, while between points of different clusters there is less similarity. This task is often modeled by a graph where a vertex corresponds to a data point and an edge between two vertices indicates that the corresponding data points are similar. If the data set is perfectly clustered, then this translates into a cluster graph, that is, each connected component is a clique. However, real-world data sets usually do not translate into disjoint cliques, for example due to noise in the data. In response, different approaches using graph models were introduced to cluster real-world data. In this thesis, we ask whether it is possible to transform a given graph into a cluster graph by at most k modifications for some integer k. Thereby, the allowed modifications are edge deletions and edge insertions, yielding the following combinatorial problem:

Cluster Editing
Input: An undirected graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq \binom{V}{2}$ of vertex pairs with $|S| \leq k$ to delete or add such that the resulting graph is a cluster graph, that is, each connected component is a clique?

For an example, see Figure 1.1. It shows a graph and how it can be transformed into a cluster graph by a minimum number of modifications. However, be aware that we formulate Cluster Editing as a decision problem, and not as an optimization problem.

Cluster Editing is NP-hard [KM86] and has been widely studied from different perspectives, including heuristics [Blo+08; Wit+10], approximation algorithms [ACN08; CGW05] and parameterized algorithms [Gra+05; LPS21]. Subsequently, we focus on the latter approach. In parameterized algorithmics, we parameterize an NP-hard problem with a parameter p and try to come up with algorithms that solve instances, where p is rather small, efficiently. That is, a parameterized algorithm with respect to p computes an optimal solution in $f(p) \cdot n^{\mathcal{O}(1)}$ time where n is the input size and f any computable function exclusively depending on p. Hence, we can ensure that the exponential blow-up of the running time is restricted to the parameter p and, thus, there is hope that parameterized algorithms perform well for applications where such a parameter is expected to be small.

(a) (b)

Figure 1.1: (a) depicts a graph G and (b) an optimum cluster graph $\widetilde{G}$ for G, that is, $\widetilde{G}$ results from G by a minimum number of modifications. These are indicated in red: a solid, red line indicates that the edge was added whereas a dashed, red line indicates that the edge was deleted.

To return to Cluster Editing, we hope that the given data set is somehow well preclustered. That is the case, following the context above, when not too much noise is contained in the data. Translated to our setting, this means that we assume that the minimum number of edge modifications k needed to transfer a graph into a cluster graph is rather small. Therefore, it seems reasonable to study Cluster Editing parameterized by the solution size k. As usual in the context of parameterized algorithms with respect to the parameter k, several branching algorithms were introduced for Cluster Editing, see Section 1.1 for an overview. However, this clustering model is sometimes not satisfactory. In some applications, we have two inherently different sets of data points and the relationships between points from different sets are of interest. Thus, clustering data points in this setting translates into finding subsets in which the elements of one subset have similar relationships regarding the second one. This can be modeled as a bipartite graph G with two disjoint vertex sets V_1 and V_2, and the goal is to transfer G by a minimum number of modifications into a bicluster graph, that is, each connected component is a complete bipartite graph. Formalizing this approach yields the following problem:

Bicluster Editing
Input: A bipartite graph $G = \big(V = V_1 \dot\cup V_2, E \subseteq V_1 \times V_2\big)$[1] and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V_1 \times V_2$ of vertex tuples with $|S| \leq k$ to delete or add such that the resulting graph is a bicluster graph?

Clearly, a limitation of Bicluster Editing is that only pairwise relationships are allowed. Since pairwise relations are not sufficient in some applications, it seems natural to develop algorithms that can handle higher-order relationships among data points. Motivated by an application in detecting radioactive sources [Sch02], we study a generalization of Bicluster Editing where we have ℓ different types of data points and a relation involves exactly ℓ points, each of them of a different type. This setting can be modeled on hypergraphs by modeling a relation as a hyperedge, motivating ℓ-P ℓ-U Cluster Editing:

[1] To avoid confusion, we emphasize that we consider undirected graphs. In this work, we denote multipartite graphs instead by this notation.

ℓ-P ℓ-U Cluster Editing
Input: An ℓ-partite ℓ-uniform hypergraph $G = \big(V = V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell\big)$ and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V_1 \times \cdots \times V_\ell$ of vertex tuples with $|S| \leq k$ to delete or add such that the resulting graph is an ℓ-cluster graph, that is, each connected component is an ℓ-clique?

We want to explain roughly how a weighted version of ℓ-P ℓ-U Cluster Editing can be used to detect radioactive sources. Each vertex tuple, that is, each edge and each non-edge, is assigned a weight, determining the cost of deleting resp. inserting the edge. The integer ℓ corresponds to the number of detectors we use to measure the radiation and a vertex in V_i corresponds to an event measured by detector i. We use some function g to calculate for each vertex tuple $(v_1, \ldots, v_\ell) \in V_1 \times \cdots \times V_\ell$ whether the measured radiation at the sensors $v_1, \ldots, v_\ell$ is from a common point of origin. In theory, this applies if $g(v_1, \ldots, v_\ell) = 0$. Due to errors in practice, we assume that the radiation is from the same origin if $|g(v_1, \ldots, v_\ell)| \leq t$ for some threshold t. In this case, we interpret $(v_1, \ldots, v_\ell)$ as an edge with weight $t - |g(v_1, \ldots, v_\ell)|$. Otherwise, we interpret $(v_1, \ldots, v_\ell)$ as a non-edge with weight $|g(v_1, \ldots, v_\ell)| - t$. For more details on the modeling and the function g, we refer to Schopper [Sch02].

When modeling a data set by a hypergraph G in the above way and computing an optimal solution S for G, we interpret an ℓ-clique in G after applying S as one radioactive source and the modifications we had to perform as noise, e.g. as background radiation and missed events. Hence, if we assume that there is not too much noise in the data set, then we expect the parameter k to be rather small. This motivates to study the parameterized complexity of ℓ-P ℓ-U Cluster Editing with respect to the parameter k.

Clustering problems on ℓ-partite ℓ-uniform hypergraphs have been studied for community detection before [NO09; NO10]. However, to the best of our knowledge, ℓ-P ℓ-U Cluster Editing has not been introduced yet. Therefore, we first provide several general results for ℓ-P ℓ-U Cluster Editing for $\ell \geq 3$, and second, we consider it from a parameterized perspective with parameter k in Chapter 3. In doing so, we are guided by ideas and techniques that were successfully applied for Cluster Editing or Bicluster Editing. In addition, we focus on the special case $\ell = 3$ in Chapter 4, and finally discuss the case $\ell = 2$, which we already know under the name Bicluster Editing introduced above, in Chapter 5.

1.1 Related Work

As our approaches are based on results for Cluster Editing or Bicluster Editing, we shall focus on these two problems in this section.

Cluster Editing. We start off by giving an overview on what is known for Cluster Editing, in particular in the field of parameterized complexity. Its NP-hardness was proven by Křivánek and Morávek [KM86] and more proofs of this fact have been published subsequently [BBC04; SST04]. The parameterized complexity of Cluster Editing with respect to the parameter k, which is referred to as solution size, has

been widely studied. Parameterized algorithms for Cluster Editing with parameter k are mostly branching algorithms that are based on the observation that a graph is a cluster graph if and only if it has no P_3, that is, a path on three vertices, as an induced subgraph. Then, in each branching step such a P_3 is resolved. A first algorithm with running time $\mathcal{O}^*(2.27^k)$[2] by Gramm et al. [Gra+05] was improved to a running time of $\mathcal{O}^*(1.92^k)$ by making an extensive, automated case distinction [Gra+04]. Later, Böcker et al. [Böc+09] introduced a simple branching algorithm of time $\mathcal{O}^*(1.82^k)$ for Cluster Editing. By combining this branching strategy, which we refer to as merge branching, with a more thorough case analysis they improved the running time to $\mathcal{O}^*(1.62^k)$, being the currently best known for the problem [Böc12].

Gramm et al. [Gra+05] gave a first kernelization result for Cluster Editing, showing that it has a vertex kernel quadratic in k. The basic idea of kernelization is to replace the original instance by a (usually) smaller but equivalent one, the kernel, by preprocessing. Subsequently, several kernelization algorithms have been devised that produce a linear-vertex kernel [CC12; CM12; Fel+07; Guo09] among which the smallest kernel has $2k$ vertices [CC12; CM12].

For Cluster Editing, not only fixed-parameter tractability with respect to the parameter k has been studied. Since the number of modifications is often rather large in practice, smaller parameters are desirable. One approach is to obtain smaller parameters by a so called 'parameterization above guaranteed values' [Cyg+13; GP16; Lok+14; MR99]. The idea is to think of lower bounds on the solution size and once given such a lower bound h, we parameterize by $k - h$ instead of k. For Cluster Editing, van Bevern, Froese, and Komusiewicz [vFK18] showed fixed-parameter tractability with respect to the parameter $k - h_v$, where h_v is the lower bound obtained by a set of vertex-disjoint P_3s. In their conclusion, they asked whether Cluster Editing parameterized above the size of a set of edge-disjoint P_3s is still fixed-parameter tractable. This question was answered negatively by Li, Pilipczuk, and Sorge [LPS21], showing that Cluster Editing is NP-hard even when restricted to instances where k equals the size of a set of edge-disjoint P_3s.

Shamir, Sharan, and Tsur [SST04] showed that Cluster Editing remains NP-hard even when the solution may contain at most two clusters. Komusiewicz and Uhlmann [KU11] showed that it remains NP-hard on graphs with maximum degree six, implying that there is no hope to show fixed-parameter tractability by the parameter maximum degree, unless P = NP. A direct consequence is that Cluster Editing is presumably also not fixed-parameter tractable with respect to the parameter t, where t is the maximum number of modified edges incident to any vertex in an optimal solution. However, Komusiewicz and Uhlmann showed that a restricted version of Cluster Editing is fixed-parameter tractable when combining t with the parameter d, denoting the number of cliques in the resulting cluster graph. Further, they showed fixed-parameter tractability with respect to the parameter 'cluster vertex deletion number'. Moreover, Xin [Xin11] presented an FPT algorithm that takes treewidth as a parameter and has a running time linear in the number of vertices.

We close the literature review on Cluster Editing with the remark that several

[2]For a function f and a constant c, the notation $\mathcal{O}^*(f) = \mathcal{O}(f \cdot n^c)$ is used to hide polynomial factors in the input size n.

Table 1.1: A summary of our results and previously known results for Cluster Editing, Bicluster Editing and ℓ-P ℓ-U Cluster Editing. Due to space constraints, we abbreviate 'Cluster Editing' by 'CE'.

	CE	BiCE	ℓ-P ℓ-U CE
NP-hardness	[KM86]	[Ami04]	Theorem 3.3
ILP formulation	[GW89]	[Ami04]	Corollary 3.7
FPT wrt. k			
unweighted	$\mathcal{O}^*(1.62^k)$ [Böc12]	$\mathcal{O}^*(2.64^k)$ [Tsu21]	$\mathcal{O}^*(3^k)$ Prop. 3.11
weighted	$\mathcal{O}^*(1.62^k)$ [Böc12]	$\mathcal{O}^*(3^k)$ Theorem 5.5	$\mathcal{O}^*(3^k)$ Corollary 3.12
Vertex kernel size	$2k$ [CC12; CM12]	$5k$ [Laf20]	$\mathcal{O}(k^2)$ Theorem 3.18
FPT wrt. $k - h_v$	$\mathcal{O}^*(4^{k-h_v})$ [vFK18]	?	?

experimental studies have been published that demonstrate that fixed-parameter algorithms can indeed be applied to solve real-world instances of the problem [BBK11; Deh+06; Kel+21].

Bicluster Editing. Next, we give an overview of parameterized complexity results for the sister problem Bicluster Editing. Its NP-hardness has been proven by Amit [Ami04] and it is known that it remains NP-hard on dense graphs [SGB14]. While in the literature it is often stated that Drange et al. [Dra+15] showed that Bicluster Editing, as defined above, is NP-hard on subcubic graphs, we emphasize that Drange et al. [Dra+15] defined Bicluster Editing differently. That is, the input graph does not need to be bipartite and, hence, their polynomial-time many-one reduction used to show NP-hardness produces indeed instances with maximum degree three that are not bipartite in general.

By observing that graphs of disjoint biclusters coincide with bipartite graphs without an induced P_4, Protti, Silva, and Szwarcfiter [PSS06] first presented a simple $\mathcal{O}^*(4^k)$ time algorithm that finds a P_4 and branches over four possible ways to resolve it. They also showed that Bicluster Editing admits a quadratic-vertex kernel. Later, Guo et al. [Guo+08] refined this approach by branching on a somehow good P_4, decreasing the branching factor to 3.24. Lafond [Laf20] made use of the kernelization technique by Guo [Guo09] for Cluster Editing to show that Bicluster Editing also admits a linear-vertex kernel and presented a branching algorithm with a running time of $\mathcal{O}^*(2.695^k)$. Instead of resolving a P_4 in each branching step, two vertices, that prevent the graph from being a bicluster graph, are chosen and every conflict they are part of is eliminated. Recently, [Tsu21] stated another branching algorithm of running time $\mathcal{O}^*(2.636^k)$ that is in theory the currently fastest one for Bicluster Editing with respect to the parameter k. However, it does not seem simple to implement as it needs tedious case distinctions.

1.2 Our Contributions

For an overview of our results, see Table 1.1. The results are divided into three chapters, starting with Chapter 3 which is devoted to the problem ℓ-P ℓ-U Cluster Editing

for $\ell \geq 3$. In Section 3.1, we prove that ℓ-P ℓ-U Cluster Editing is NP-hard by a many-one reduction from ℓ-Partite Perfect Matching (Theorem 3.3). The reduction allows us to conclude that NP-hardness also holds when only edge deletions—instead of edge deletions *and* insertions—are allowed (Corollary 3.4). In Section 3.2, we characterize an ℓ-cluster graph by forbidding a certain substructure that consists of two edges and one non-edge (Theorem 3.5). Next, we state an ILP formulation of ℓ-P ℓ-U Cluster Editing (Corollary 3.7), that is a direct consequence of Theorem 3.5. In Section 3.4, we introduce a trivial branching algorithm, yielding that ℓ-P ℓ-U Cluster Editing is solvable in $\mathcal{O}^*(3^k)$ time (Proposition 3.11). Subsequently, we discuss two branching algorithms following the merge branching by Böcker et al. [Böc+09] (Theorem 3.14). In Section 3.5, we present a data reduction rule which is then used to show that ℓ-P ℓ-U Cluster Editing admits a vertex kernel quadratic in k (Reduction Rule 3.5.2 and Theorem 3.18). In the last section of the third chapter, we try to transfer the parameterization above lower bounds for Cluster Editing, due to van Bevern, Froese, and Komusiewicz [vFK18], to ℓ-P ℓ-U Cluster Editing. However, this approach relies on a data reduction rule and we show that its natural generalization to ℓ-P ℓ-U Cluster Editing is not correct by giving a counterexample (Lemma 3.25).

In Chapter 4, we fix $\ell = 3$, that is, we study the problem 3-P 3-U Cluster Editing. In Section 4.1, we refine one of the merge branchings introduced in Section 3.4 for $\ell = 3$ and show that its branching factor is at most three (Theorem 4.4). Subsequently, we show that the preceding running time analysis of the refined merge branching is tight (Theorem 4.6 and Lemma 4.7). In Section 4.2, we adapt a data reduction rule that produces a vertex kernel linear in k for Cluster Editing, see Guo [Guo09], to 3-P 3-U Cluster Editing (Reduction Rule 4.2.1). However, we could only show that Reduction Rule 4.2.1 can be used to achieve a quadratic-vertex kernel for 3-P 3-U Cluster Editing (Theorem 4.10). Therefore, we give a high-level idea why we achieve a linear-vertex kernel for Cluster Editing, but only a quadratic one for 3-P 3-U Cluster Editing by this and several other approaches.

In Chapter 5, we first discuss why ℓ-P ℓ-U Cluster Editing behaves differently for $\ell = 2$ and $\ell \geq 3$. Then, we introduce the merge branching for Bicluster Editing and refine it as well, yielding a branching factor of 3 (Theorem 5.5). Last, we try to transfer the parameterization above lower bounds by van Bevern, Froese, and Komusiewicz [vFK18] to Bicluster Editing and discuss what prevents us in this case from obtaining fixed-parameter tractability above a lower bound.

Chapter 2

Preliminaries

In this chapter, we introduce definitions and notations that we use throughout this thesis.

2.1 Graph Theory

Most of the following notation is based on Diestel [Die17].

If S is a set and $k \in \mathbb{N}$, then $\binom{S}{k}$ is the set of all subsets of S containing exactly k elements. A *graph* $G = (V, E)$ consists of a finite set V and $E \subseteq \binom{V}{2}$. The elements of V are called *vertices* and those of E *edges*. If $e = \{v, w\} \notin E$, then we say that e is a *non-edge*. Sometimes, we call a graph an *ordinary graph* to avoid confusion with the subsequent class of graphs.

A graph is *ℓ-partite* if one can partition its vertex set $V = V_1 \dot\cup \ldots \dot\cup V_\ell$ such that for all $i \in \{1, \ldots, \ell\}$ there is no edge that contains two vertices of V_i. A *hypergraph* is a generalization of a graph in which an edge can contain any number of vertices. A hypergraph is *ℓ-uniform* if every edge consists of exactly ℓ vertices. In this thesis, we usually denote an ℓ-partite ℓ-uniform hypergraph by $G = (V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell)$. The elements of V are called *vertices* and those of E *hyperedges*. If it is clear from the context, then we also just say *graph* instead of hypergraph, and *edge* instead of hyperedge. We say that $e = (v_1, \ldots, v_\ell)$ is a *non-edge* if $(v_1, \ldots, v_\ell) \notin E$. Note that we denote a hyperedge by an ℓ-tuple in this setting, whereas an edge is denoted by a set for ordinary graphs. By abuse of notation, we sometimes interpret a hyperedge e as the set of its vertices and write $v \in e$ for a vertex v. Moreover, we also denote a hyperedge $(v_1, \ldots, v_\ell)$ by $v_1 \ldots v_\ell$ for convenience. If not stated otherwise, we implicitly assume that the index i of a vertex indicates that it lies in V_i.

We sometimes denote the vertices of G by $V(G)$ and the edges by $E(G)$ to make clear to which graph we refer. If not stated otherwise, then we assume $n = |V|$ and $m = |E|$. Two vertices $u, v \in V$ are *adjacent*, or *neighbors*, if there is an edge e such that $\{u, v\} \subseteq e$. We say that two edges $e_1, e_2 \in E$ are *incident* if $e_1 \cap e_2 \neq \emptyset$. A vertex $v \in V$ is *incident* with an edge e if $v \in e$. For a vertex $v \in V$, we define its *open neighborhood* by $N(v) = \{u \in V \mid \exists e \in E \text{ with } \{u, v\} \subseteq e\}$ and its *closed neighborhood* by $N[v] = N(v) \cup \{v\}$. For a set $V' \subseteq V$, we define its *open neighborhood* by $N(V') = \bigcup_{v \in V'} N(v)$ and its *closed neighborhood* by $N[V'] = \bigcup_{v \in V'} N[v]$. When considering an ℓ-partite ℓ-

uniform hypergraph, the *tuple neighborhood* of a vertex $v_i \in V_i$ is defined by $N^t(v_i) = \{v_1 \dots v_{i-1}v_{i+1} \dots v_\ell \mid v_1 \dots v_{i-1}v_iv_{i+1} \dots v_\ell \in E\}$.

The *degree* of a vertex $v \in V$ is $\deg(v) = |N(v)|$, the size of its open neighborhood. A vertex v is *isolated* if $deg(v) = 0$. A *path of length* $r-1$ in G is a sequence of vertices $P = v_1 - \dots - v_r$ such that there exists an edge e_i with $\{v_i, v_{i+1}\} \subseteq e_i$ and $v_i \neq v_{i+1}$ for all $i \in \{1, \dots, r-1\}$. The path P is an (s,t)-*path* if $v_1 = s$ and $v_r = t$. Two vertices $u, v \in V$ have *distance* j, denoted by $d_G(u,v) = j$, if the shortest (u,v)-path in G has length j. We say that a vertex $u \in V$ is in the *second neighborhood* of a vertex v, if $d_G(u,v) = 2$.

For an edge set $E' \subseteq E$, we denote by $G - E'$ the deletion of E' in G with the vertex set V and the edge set $E \setminus E'$. For a set $S \subseteq \binom{V}{2}$ resp. $S \subseteq V_1 \times \dots \times V_\ell$, we denote by $G \triangle S$ the graph defined by the vertex set V and the edge set $(E \setminus S) \cup (S \setminus E)$. For a vertex set $V' \subseteq V$, we denote by $G - V'$ the deletion of V' in G with the vertex set $V \setminus V'$ and the edge set $\{e \in E \mid e \cap V' = \emptyset\}$. For a vertex set $X \subseteq V$, we denote by $G[X]$ the *induced subgraph* $G - (V \setminus X)$ of G and we say that $G[X]$ is *contained* in G. We say that a graph $G' = (V', E')$ is a *subgraph* of G if $V' \subseteq V(G)$ and $E' \subseteq E(G)$. We say that the subgraph $G[X]$ is *connected* if for each vertex pair $u \neq v \in X$ there is a (u,v)-path in $G[X]$. We say that $G[X]$ is a *connected component* if $G[X]$ is connected and X is maximal under this property. An (s,t)-*cut* in a graph G is a set of edges E' such that the vertices s and t lie in different connected components in $G - E'$. To *cutoff* an induced subgraph $H = (W, F)$ from G means that we delete all edges that have at least one vertex in W and at least one vertex in $G - W$.

Two subgraphs G_1, G_2 of G are *vertex-disjoint* if $V(G_1) \cap V(G_2) = \emptyset$. For an ordinary graph we say G_1, G_2 are *(non-)edge disjoint* if $|V(G_1) \cap V(G_2)| \leq 1$, and for an ℓ-partite ℓ-uniform hypergraph we say G_1, G_2 are *(non-)edge disjoint* if there is $i \in \{1, \dots, \ell\}$ such that $V_i \cap V(G_1) \cap V(G_2) = \emptyset$.

Two graphs $G = (V, E)$ and $G' = (V', E')$ are *isomorphic* if there is a bijective map $\phi : V \to V'$ such that $e = (v_1, \dots, v_\ell)$ is an edge in G if and only if there is an edge in E' consisting of the vertices $\{\phi(v_1), \dots, \phi(v_\ell)\}$ for all $e \in E$. A set $V' \subseteq V$ is an *independent set* in G if $G[V'] = (V', \emptyset)$. A set $E' \subseteq E$ of edges is a *matching* if $e_i \cap e_j = \emptyset$ for all $e_i \neq e_j \in E'$. An ordinary graph $G = (V, E)$ is a *clique* if $E = \binom{V}{2}$ and a *cluster graph* if each connected component is a clique. An ℓ-partite ℓ-uniform hypergraph is an ℓ-*clique* if $E = V_1 \times \dots \times V_\ell$ and an ℓ-*cluster graph* if every connected component is an ℓ-clique.

The *path on n vertices* is the graph $(\{v_1, \dots, v_n\}, \{\{v_i, v_{i+1}\} \mid 1 \leq i \leq n-1\})$. A P_j in a graph G is an induced subgraph that is isomorphic to the path on j vertices.

2.2 Parameterized Algorithmics

In the following, we introduce some basic notation and concepts used in parameterized algorithmics [Cyg+13; DF13; Nie06].

A *(decision) problem* is a formal language $L \subseteq \Gamma^*$ where Γ is a finite alphabet. An instance $I \in \Gamma^*$ of a problem L is a *Yes-instance* if $I \in L$, and otherwise it is a *No-instance*. An instance (I, p) of a *parameterized problem* consists of the actual instance I and of an integer p, referred to as the *parameter*. A parameterized problem is

called *fixed-parameter tractable* if there is an algorithm that solves each instance (I, p) in $f(p) \cdot |I|^{\mathcal{O}(1)}$ time, where f is any computable function exclusively depending on the parameter p. We call such an algorithm a *fixed-parameter algorithm* and say that the parameterized problem lies in the complexity class *FPT*.

We say that two instances (I, p) and (I', p') of a parameterized problem L are *equivalent* if (I, p) is a Yes-instance for L if and only if (I', p') is a Yes-instance for L. A *kernelization* is an algorithm that, given an instance (I, p) of L, computes in polynomial time an equivalent instance (I', p') of L such that $|I'| + p' \leq f(p)$ for some computable function f only depending on p. We call (I', p') the *problem kernel*. We say that L admits a polynomial kernel if f is a polynomial function in p, a quadratic kernel if f is a quadratic function in p, and a linear kernel if f is a linear function in p.

A *data reduction rule* is an algorithm that takes as an input an instance I of a problem L, and outputs another instance I' of L. We say that a data reduction rule is correct if the new instance I' is equivalent to I. We say that a data reduction rule RR has been *exhaustively applied* to an instance, or an instance is *reduced with respect to* RR, if further application of RR does not change the instance.

Branching Algorithms. In the following, we introduce the concept of branching algorithms with respect to the solution size, which is denoted by k throughout this thesis.

A branching algorithm has a *branching rule* (sometimes even more than one) that computes for an instance (I, k) a finite number of instances $(I_1, k_1), \ldots, (I_r, k_r)$ such that $k_i < k$ for all $i \in \{1, \ldots, r\}$ and at least one (I_i, k_i) is equivalent to (I, k). These are then solved recursively, resulting in a tree structure which is called *search tree*. The root of the search tree represents the original instance and other nodes represent smaller instances obtained by the use of the branching rule. The number of nodes in the search tree is called *search tree size*. At the latest, we may halt if $k = 0$ as decreasing k further always yields a No-instance. The time spent in one node of the search tree is usually required to be polynomial in the input size. Thus, the overall running time can be upper-bounded by $\alpha^k \cdot |I|^{\mathcal{O}(1)}$ for some constant $\alpha > 1$.

To compute α, we assume that for an instance (I, k) a branching rule B calls $r \geq 2$ smaller instances in which k is reduced to $k - t_1, \ldots, k - t_{r-1}$ resp. $k - t_r$ such that $t_i > 0$ for all $i \in \{1, \ldots, r\}$. Then, $(t_1, \ldots, t_r)$ is called the *branching vector* of B. It is known that the size of the thereby created search tree can be upper-bounded by α^k, where α is the unique positive real root of the polynomial

$$x^k - x^{k-t_1} - \cdots - x^{k-t_r}.$$

We call α the *branching factor* of the branching rule B.

Chapter 3

ℓ-Partite ℓ-Uniform Cluster Editing

In this chapter, we study the (parameterized) complexity of ℓ-P ℓ-U Cluster Editing. Therefore, we first recap its definition from Chapter 1.

Problem 1: ℓ-P ℓ-U Cluster Editing

Input: An ℓ-partite ℓ-uniform hypergraph $G = \left(V = V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell\right)$ and an integer $k \in \mathbb{N}$.

Question: Is there a set $S \subseteq V_1 \times \cdots \times V_\ell$ of vertex tuples with $|S| \leq k$ to delete or add such that the resulting graph is an ℓ-cluster graph, that is, each connected component is an ℓ-clique?

We start off by proving its NP-hardness for $\ell \geq 3$ in Section 3.1.

Before we study its parameterized complexity, we want to introduce a broader result for graph editing problems first. A property of graphs is said to be *hereditary* if whenever a graph G has the property, then every induced subgraph H of G has the property as well. A graph editing problem is a problem that asks whether a given graph G can be transformed by at most p vertex deletions, q edge deletions, and r edge insertions into another graph, fulfilling certain hereditary properties. Cai [Cai96] showed that this problem is fixed-parameter tractable with respect to the budget $k = p + q + r$ if the hereditary graph property can be characterized by a finite family $\mathcal{F}$ of forbidden induced subgraphs. That is, a graph fulfills such a property if and only if it does not contain a graph F from the property-specific family $\mathcal{F}$ as an induced subgraph.

Therefore, we continue this chapter by implicitly giving such a property-specific family $\mathcal{F}$ for an ℓ-cluster graph in Section 3.2, yielding an ILP formulation for ℓ-P ℓ-U Cluster Editing as a direct consequence, see Section 3.3.

Clearly, the property, that every connected component is an ℓ-clique, is a hereditary property. Hence, if we knew that the above mentioned result by Cai [Cai96] also holds for hypergraphs, then we could conclude that ℓ-P ℓ-U Cluster Editing is fixed-parameter tractable by choosing $p = 0$ and observing that one may iterate over all possible combinations of q and r such that $q + r = k$.

Instead, we provide in Section 3.4 direct algorithms for ℓ-P ℓ-U Cluster Editing that branch on a forbidden subgraph in each branching step and thereby obtain

fixed-parameter tractability with respect to k. In Section 3.5, we again make use of this characterization to derive a vertex kernel that is quadratic in k. Last, we try to show fixed-parameter tractability for ℓ-P ℓ-U Cluster Editing above a lower bound obtained by a set of vertex-disjoint forbidden subgraphs following the approach of van Bevern, Froese, and Komusiewicz [vFK18]. However, this try was not successful and we discuss what prevents us from obtaining fixed-parameter tractability.

3.1 NP-hardness

Bicluster Editing is known to be NP-hard [Ami04]. We show that hardness holds also for all $\ell \geq 3$. To this end, we reduce from ℓ-Partite Perfect Matching, which is NP-hard for $\ell \geq 3$ [PS98].

Problem 2: ℓ-Partite Perfect Matching
Input: An ℓ-partite ℓ-uniform hypergraph $G = \big(V = V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell\big)$ such that $|V_i| = k$ for all $i \in \{1, \ldots, \ell\}$.
Question: Is there a set $S \subseteq E$ of pairwise disjoint edges with cardinality k?

We introduce the concept of critical independent sets to exploit features of the structure of an optimal solution for ℓ-P ℓ-U Cluster Editing.

Definition 3.1. A non-empty set $I \subseteq V_i$ of vertices is called a *critical independent set* if I together with its neighborhood, that is $N[I]$, is an ℓ-clique and I is maximal under this property. In this case, any two vertices $v, w \in I$ are called *strong twins*.

Lemma 3.2. *There is an optimal solution of* ℓ-P ℓ-U Cluster Editing *in which no critical independent set is split.*

The proof of the above lemma is based on a similar one for critical independent sets in the Bicluster Editing setting, see Lemma 1 by Lafond [Laf20].

Proof. Let G be an instance of ℓ-P ℓ-U Cluster Editing. Consider a set S of minimum cardinality such that $\widetilde{G} = G \triangle S$ is an ℓ-cluster graph. Assume that there exist strong twins $v, w \in V_i$ that belong to distinct ℓ-cliques, say C_v and C_w, in $\widetilde{G}$. Let $E_v \subseteq S$ resp. $E_w \subseteq S$ be the set of modifications that are incident to v resp. w. Assume without loss of generality that $|E_v| \leq |E_w|$. By removing w from C_w and putting it into C_v, we obtain an alternative solution. We want to argue why this one cannot be worse than the former solution. The modified edges E_w are not needed anymore. Instead, we need $|E_v|$ edge modifications for w as the vertices v and w are in the same critical independent set. Note that no other modification is required since only w changed its ℓ-clique. We may repeat the above swap until the whole independent set is contained in the same ℓ-clique. Finally, note that we can do this procedure for every independent set consecutively to guarantee that none is split. □

We now have all ingredients to show NP-hardness.

Theorem 3.3. ℓ-P ℓ-U Cluster Editing *is NP-hard for* $\ell \geq 3$.

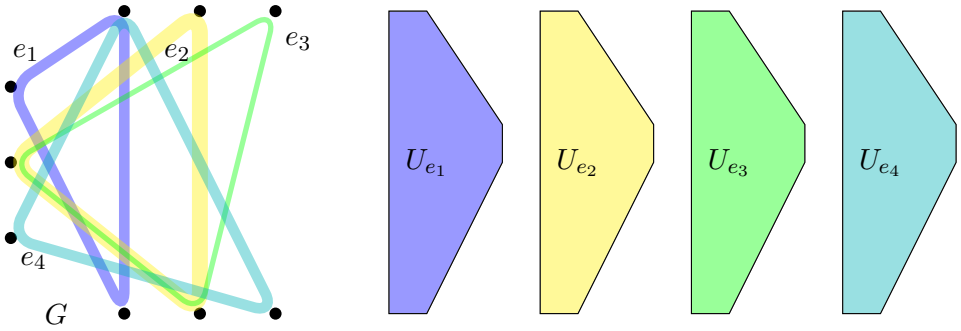


Figure 3.1: The hypergraph G' that is constructed from the exemplary, 3-partite 3-uniform hypergraph G on the left. We indicate a hyperedge by the shape of a triangle. The filled polygons U_{e_i} on the right side represent 'heavy' balanced tricliques. In the construction, all possible edges between $V(e_i)$ and $V(U_{e_i})$ are present. For the sake of clarity, we omit to draw these here.

The construction of the polynomial-time many-one reduction we present subsequently is derived from the construction Amit [Ami04] designed to show that Bicluster Editing is NP-hard. However, the proof itself is structured differently. We briefly sketch the construction, compare to Figure 3.1, and give an intuition how an optimal solution looks like: given an instance $G = (V, E)$ of ℓ-Partite Perfect Matching, we construct an instance G' of ℓ-P ℓ-U Cluster Editing by copying G and building around each edge a 'heavy' balanced ℓ-clique that is only connected to the vertices in the edge, yielding that in an optimal solution neither one of these artificial ℓ-cliques is split nor any two of them end up in the same ℓ-clique. Then, we show that it is best to assign the copied vertices of V to the artificial ℓ-cliques such that as many copied edges as possible can remain which then corresponds to a matching in G.

Proof. Given an instance $(G = (V, E), k)$ of ℓ-Partite Perfect Matching. We define $m = |E|$ and $x = 2^\ell \ell \cdot m$. We construct an instance $(G' = (V', E'), k')$ of ℓ-P ℓ-U Cluster Editing as follows: copy G into G' and add for each edge e a balanced ℓ-clique U_e consisting of $x \cdot \ell$ vertices and all possible edges between vertices in e and U_e. We call U_e the edge-gadget of the edge e. We show that (G, k) is a Yes-instance for ℓ-Partite Perfect Matching if and only if (G', k') with $k' = (m-k) \cdot ((x+1)^\ell - x^\ell)$ is a Yes-instance for ℓ-P ℓ-U Cluster Editing.

Let us assume that G has a perfect matching M of cardinality k. We can construct a solution for ℓ-P ℓ-U Cluster Editing as follows: if an edge e is not contained in M, then we delete the corresponding edge and the edges that connect it with U_e in G'. In doing so, we have to pay $(x+1)^\ell - x^\ell$ for each edge that did not get chosen to be in M as we delete all edges in an ℓ-clique consisting of $x+1$ vertices from each V_i, except of the edges within the edge-gadget. The overall cost sums up to $(m-k) \cdot ((x+1)^\ell - x^\ell)$ and the resulting graph is clearly an ℓ-cluster graph as M is a matching.

Next, let us assume that $G' = (V', E')$ has a cluster editing set S such that $|S| \leq k'$ holds

and let S be of minimum cardinality. Let $\widetilde{G'}$ be the resulting ℓ-cluster graph after application of the editing set S and let $C_1, \ldots, C_p$ be its ℓ-cliques. Subsequently, we show that for all $i \in \{1, \ldots, p\}$ there exists an edge e such that $V(C_i) = V(U_e)$ or $V(C_i) = V(U_e) \cup V(e)$ holds. Given that this is proven, we can construct a matching of size k for G by choosing the edges which correspond to an ℓ-clique of the form $V(C_i) = V(U_e) \cup V(e)$ in $\widetilde{G'}$. We proceed by dividing the proof into three steps.

First, we claim that there is an optimal solution in which no edge-gadget U_e is split, i.e. for all edges e there exists $i \in \{1, \ldots, p\}$ such that $U_e \subseteq C_i$. To this end, we observe that U_e consists of ℓ critical independent sets $I_1, \ldots, I_\ell$. By Lemma 3.2 we know that there is an optimal solution in which none of these is split. We claim that there is also an optimal solution in which none of these independent sets is separated from each other. Thus, assume that I_i and I_j end up in different ℓ-cliques in an optimal solution. In this case, all edges inside U_e are deleted which is of cost x^ℓ. In comparison, cutting U_e off from G has cost $(x+1)^\ell - x^\ell - 1$ which is more favorable by the binomial theorem. Hence, we have shown that there is an optimal solution in which none of the edge-gadgets is split.

Second, we prove that for all $i \in \{1, \ldots, p\}$ there is an edge e such that $V(C_i) \subseteq V(U_e) \cup V(e)$. Let us assume that vertices $v_1, \ldots, v_r$ not contained in e also end up in C_i. Observe that inserting all missing edges between two edge-gadgets has cost $(2x)^\ell - 2x^\ell$ which is strictly greater than k' and, hence, each edge-gadget has its own ℓ-clique in an optimal solution. Thus, the vertices $v_1, \ldots, v_r$ correspond to vertices in G and each of these needed at least $x^{\ell-1}$ insertions, namely to the vertices in U_e. However, by removing the vertices $v_1, \ldots, v_r$ from C_i and putting them into a separate ℓ-clique, we save at least $r \cdot x^{\ell-1}$ edge insertions and only have to pay for at most m deletions. This means we found a solution which needs strictly less modifications, contradicting that S is optimal. Next, assume that there is an ℓ-clique C_i of the form $C_i = \{v_1, \ldots, v_r\}$ in an optimal solution. But in this case, we can find a better solution by assigning each vertex v_i of $\{v_1, \ldots, v_r\}$ to an edge-gadget U_e such that $v_i \in e$. This swap saves at least $r \cdot x^{\ell-1}$ deletions and costs at most m additional deletions. Therefore, in an optimal solution we only have ℓ-cliques that correspond to U_e plus a subset of the vertices of e yielding that it consists of exactly m cliques. Note that at this point we already know that no edge is inserted in an optimal solution since for all edges e the vertices $V(U_e) \cup V(e)$ already induce an ℓ-clique in G'.

Third, we want to show that the vertices in an ℓ-clique in $\widetilde{G'}$ either correspond to $V(U_e)$ or $V(U_e) \cup V(e)$. Since (G', k') is a Yes-instance, we know that there is an optimal solution in which at most $(m-k) \cdot ((x+1)^\ell - x^\ell)$ edges get deleted. Observe that $(x+1)^\ell - x^\ell$ is the cost we have to pay to isolate an edge-gadget from its edge plus deleting the edge itself. Thus, in total k' can be interpreted as the cost we have to pay for isolating $m-k$ edges from their gadgets and deleting these edges themselves. We only have to argue why distributing the $\ell \cdot k$ vertices differently over the m edge-gadgets is more expensive. Let p_e be the number of vertices that end up in the ℓ-clique of U_e besides the vertices of U_e itself. Then, the total deletion cost sums up to

$$\sum_{e \in E} (x+1)^\ell - ((x+1)^{p_e} \cdot x^{\ell - p_e}).$$

Taking into account the constraint $\sum_{e \in E} p_e = \ell \cdot k$, one can check that the function

reaches its unique minimum when exactly k of the variables p_e have value ℓ. This is due to the observation that if a non-trivial subset $\emptyset \neq W_e \subsetneq V(e)$ ends up in the ℓ-clique of U_e, then the edge e itself has to be deleted. In addition, no other edge, that is derived from an edge in G and at the same time incident to vertices in W_e, can remain in the final cluster graph, yielding a strictly greater total deletion cost.

Thus, in an optimal ℓ-cluster graph we only have ℓ-cliques that either consist of the vertices $V(U_e)$, or of the vertices $V(U_e) \cup V(e)$ for some edge-gadget U_e. In combination with the assumption that we needed at most $k' = (m-k) \cdot ((x+1)^\ell - x^\ell)$ modifications, we have shown that G is a Yes-instance of ℓ-Partite Perfect Matching. □

Note that in the above construction—in both directions—only edge deletions are performed. Hence, we can conclude that even if the single allowed operation is edge deletion, the problem remains NP-hard. We thereby motivate the following problem.

Problem 3: ℓ-P ℓ-U Cluster Edge Deletion
Input: An ℓ-partite ℓ-uniform hypergraph $G = \big(V = V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell\big)$ and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V_1 \times \cdots \times V_\ell$ of vertex tuples with $|S| \leq k$ to delete such that the resulting graph is an ℓ-cluster graph, that is, each connected component is an ℓ-clique?

Corollary 3.4. *ℓ-P ℓ-U Cluster Edge Deletion is NP-hard for $\ell \geq 3$.*

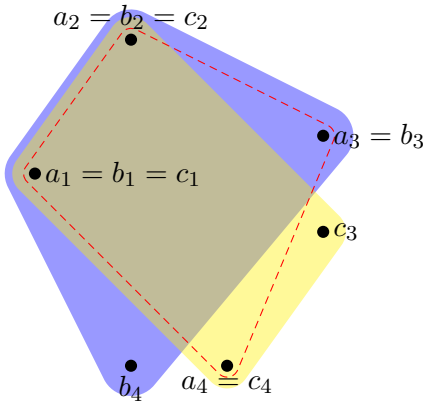
For the sake of completeness, we briefly discuss why we can solve the version, that only allows edge insertions, in polynomial time. Clearly, vertices that lie in the same connected component cannot end up in different ℓ-cliques. Therefore, an optimal solution inserts all missing edges within a connected component and no edge that would connect distinct ones.

Be aware that we cannot omit the assumption $\ell \geq 3$ in the proof above as the matching problem is solvable in polynomial time on ordinary graphs and in particular on bipartite graphs [Edm65]. Thus, the above reduction does not allow us to conclude that ℓ-P ℓ-U Cluster Editing is NP-hard for $\ell = 2$. However, we already know that NP-hardness holds for $\ell = 2$ as well [Ami04].

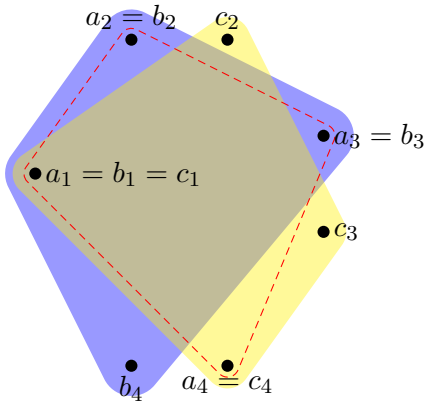
3.2 Forbidden Substructures

As discussed in the introduction of this section, we would like to give a characterization of an ℓ-cluster graph by listing its forbidden induced subgraphs. Indeed, this does not seem possible in a somehow compact way. Instead, we characterize an ℓ-cluster graph by stating only one forbidden substructure—having at most $2\ell - 1$ vertices—that can be used to derive a complete, finite family of forbidden induced subgraphs when ℓ is fixed.

Theorem 3.5. *An ℓ-partite ℓ-uniform hypergraph $G = \big(V = V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell\big)$ with $\ell \geq 3$ is an ℓ-cluster graph if and only if it contains no non-edge $a_1 \ldots a_\ell$ for which there are two edges $b_1 \ldots b_\ell$ and $c_1 \ldots c_\ell$ such that $\{a_1, \ldots, a_\ell\} \subseteq \{b_1, c_1, \ldots, b_\ell, c_\ell\}$ and $\{b_1, \ldots, b_\ell\} \cap \{c_1, \ldots, c_\ell\} \neq \emptyset$.*



(a) Two edges b and c, intersecting in two vertices, cover the non-edge a. This results in a conflict of exactly six vertices. Note that there are four (non-)edges between these vertices of which we specified three.



(b) Two edges b and c, intersecting in one vertex, cover the non-edge a. This results in a conflict of exactly seven vertices. Note that there are eight (non-)edges between these vertices of which we specified three.

Figure 3.2: The forbidden subgraphs of 4-P 4-U Cluster Editing projected onto the plane. We indicate the edges $b = b_1b_2b_3b_4$ and $c = c_1c_2c_3c_4$ by filled quadrangles and the non-edge $a_1a_2a_3a_4$ by a red, dashed line.

We first want to give a notion how one can visualize the statement of the theorem and afterwards prove its correctness.

We do not characterize an ℓ-cluster graph by its forbidden induced subgraphs at this point due to the advantage of a somehow compact formulation. One may visualize the forbidden substructure by imagining the vertices of the two edges as a cover of the vertices of the non-edge. In the following, we also say that a non-edge is covered by two intersecting edges. We say that the (non-)edges $a_1 \dots a_\ell, b_1 \dots b_\ell, c_1 \dots c_\ell$ are in a conflict and call this substructure itself a *conflict*. Further, note that $b_1, \dots, b_\ell$ and $c_1, \dots, c_\ell$ have to differ in at least two positions.

As we study the forbidden induced subgraphs of 3-P 3-U Cluster Editing in greater detail in the next chapter, we have a look at the conflicts of 4-P 4-U Cluster Editing now. First, observe that the edges $b_1b_2b_3b_4$ and $c_1c_2c_3c_4$ intersect in at least one vertex and in at most two vertices. Hence, a conflict has either six or seven vertices, compare to Figure 3.2. Note that we do not specify for all edges whether they are present or missing in the figure. Hence, we depict the forbidden subgraphs, but not the forbidden induced subgraphs.

We prove the theorem in two steps. To this end, we show the following lemma at first.

Lemma 3.6. *If G is not an ℓ-cluster graph and $\ell \geq 3$, then there is a non-edge $a_1 \dots a_\ell$ such that $d_G(a_i, a_j) = 1$ for some $j \in \{1, \dots, \ell\}$ and all $i \neq j$.*

Proof. Consider a connected component that is missing at least one edge. For all non-edges $a_1 \dots a_\ell$, compute all pairwise distances $d_G(a_i, a_j)$ for $i < j$. Choose a permuta-

tion π of $\{a_1, \ldots, a_\ell\}$ such that the distance vector

$$\big(d_G(\pi(a_1), \pi(a_2)), \ldots, d_G(\pi(a_1), \pi(a_\ell))\big)$$

is lexicographically smallest among all permutations. Then, choose a non-edge such that its corresponding distance vector is lexicographically smallest among all missing edges. Without loss of generality, we assume that π is the identity function in the following. From now on we assume $d_G(a_1, a_\ell) > 1$, otherwise our lemma follows immediately. Consider a shortest (a_1, a_ℓ)-path $a_1 - b - c - \ldots - a_\ell$ in G. We distinguish two cases for the vertex b:

First, let $b \notin V_\ell$. Without loss of generality, we may assume $b \in V_2$. Vertex a_1 neighbors b, that is, there exist vertices $b_3, \ldots, b_\ell$ such that $a_1 b b_3 \ldots b_\ell$ is an edge. We claim that $f = a_1 b b_3 \ldots b_{\ell-1} a_\ell$ is a non-edge and a better choice than $a_1 \ldots a_\ell$ is and, thus, arrive at a contradiction. Clearly, f must be a non-edge as $d_G(a_1, a_\ell) > 1$. Further, its distance vector is lexicographically smaller since $d_G(b, a_\ell) < d_G(a_1, a_\ell)$ and $d_G(b, a_1) = d_G(b, b_i) = 1$ for $i \in \{3, \ldots, \ell - 1\}$.

Second, let $b \in V_\ell$. If $a_1 \ldots a_{\ell-1} b$ is a non-edge, then it is clearly a better choice. Otherwise, consider the vertex c. If c lies in $V_k \neq V_1$, then we know that $f_2 = a_1 \ldots a_{k-1} c a_{k+1} \ldots a_{\ell-1} b$ is a non-edge. Moreover, all distances from b to other vertices in f_2 are one, i.e. the non-edge f_2 has a smaller distance vector than the one we chose. Last, let $c \in V_1$. If f_2 is still a non-edge, then we have found again a non-edge with a lexicographically smaller distance vector by the same argument as above. Otherwise, we choose the non-edge $f_3 = c a_2 \ldots a_\ell$. Since $d_G(c, a_i) = 1$ for all $i \in \{2, \ldots, \ell - 1\}$ and $d_G(c, a_\ell) < d_G(a_1, a_\ell)$, we may conclude the correctness of the lemma. □

The lemma implies that if a graph is not an ℓ-cluster graph, then we can find a non-edge $a_1 \ldots a_\ell$ and $\ell - 1$ edges e_i that cover the non-edge, i.e. $\{a_1, \ldots, a_\ell\} \subseteq \bigcup_{i=2,\ldots,\ell} V(e_i)$. To prove the theorem, we show next that we can reduce the number of needed edges from $\ell - 1$ to only two edges.

Proof of Theorem 3.5. Due to the previous lemma, without loss of generality, we may assume there is a non-edge $a_1 \ldots a_\ell$ in G such that $d_G(a_1, a_i) = 1$ for all $i \in \{2, \ldots, \ell\}$. We want to show the following claim: if there is one edge containing the vertices $\{a_1, \ldots, a_i\}$, but the vertices $\{a_1, \ldots, a_{i+1}\}$ are not contained in any edge, then there are two edges $a_1 b_2 \ldots b_\ell$ and $a_1 c_2 \ldots c_\ell$ covering $\{a_1, \ldots, a_{i+1}\}$.

Thus, assume $\{a_1, \ldots, a_i\}$ are contained in the edge $e_1 = a_1 \ldots a_i b_{i+1} \ldots b_\ell$ but there is no edge containing the vertices $\{a_1, \ldots, a_{i+1}\}$, in particular $f = a_1 \ldots a_{i+1} b_{i+2} \ldots b_\ell$ is a non-edge. As $d_G(a_1, a_{i+1}) = 1$, there is an edge $e_2 = a_1 c_2 \ldots c_i a_{i+1} c_{i+2} \ldots c_\ell$ containing these two vertices and together with e_1, they cover the non-edge f.

As $\{a_1, a_2\}$ are contained in at least one edge and, clearly, $\{a_1, \ldots, a_\ell\}$ are not contained in any, there is an i such that the vertices $\{a_1, \ldots, a_i\}$ are contained altogether in one edge, but the vertices $\{a_1, \ldots, a_{i+1}\}$ are not contained in any edge.

For the sake of completeness, we argue now why an ℓ-uniform ℓ-partite hypergraph is not an ℓ-cluster graph in case it has a non-edge $a = a_1 \ldots a_\ell$ for which there are two intersecting edges $b = b_1 \ldots b_\ell$ and $c = c_1 \ldots c_\ell$ that cover a. As the edges b and c intersect, all vertices lie in the same connected component and hence, we found vertices, namely $\{a_1, \ldots, a_\ell\}$, that lie in the same connected component but share a non-edge. □

Unless otherwise stated, we assume $\ell \geq 3$ from now on. In Chapter 5, we come back to the special case $\ell = 2$, known as Bicluster Editing, and discuss why we have to exclude this case for the moment.

3.3 ILP Formulation

In this section, we state an Integer Linear Program (ILP) for ℓ-P ℓ-U Cluster Editing. For an introduction to the theory of ILPs, we refer to Schrijver [Sch98].

When designing an algorithm for an NP-hard problem—what we will do in Section 3.4—we often want to compare its practical performance with the performance of existing ILP solvers when giving an ILP formulation of this particular problem as an input. Thus, we present an ILP formulation of ℓ-P ℓ-U Cluster Editing in this section. This formulation is a direct consequence of Theorem 3.5 and based on the ILP formulation for Cluster Editing due to Grötschel and Wakabayashi [GW89].

Corollary 3.7. *The below ILP is an ILP formulation of* ℓ-P ℓ-U Cluster Editing.

$$\textbf{minimize} \quad \sum_{(a_1,\ldots,a_\ell)\in V_1\times\cdots\times V_\ell} \begin{cases} 1 - x_{a_1\ldots a_\ell} & \text{if } a_1\ldots a_\ell \in E \\ x_{a_1\ldots a_\ell} & \text{if } a_1\ldots a_\ell \notin E \end{cases}$$

subject to for all $(a_1,\ldots,a_\ell),(b_1,\ldots,b_\ell),(c_1,\ldots,c_\ell) \in V_1\times\cdots\times V_\ell$ such that $\{a_1,\ldots,a_\ell\} \subseteq \{b_1,c_1,\ldots,b_\ell,c_\ell\}$ and $\{b_1,\ldots,b_\ell\} \cap \{c_1,\ldots,c_\ell\} \neq \emptyset$:

$$x_{b_1\ldots b_\ell} + x_{c_1\ldots c_\ell} - x_{a_1\ldots a_\ell} \leq 1$$

for all $(a_1,\ldots,a_\ell) \in V_1\times\cdots\times V_\ell$:

$$x_{a_1\ldots a_\ell} \in \{0,1\}$$

Clearly, the performance of an ILP solver depends—among other things—on the number of variables and constraints given.

Observation 3.8. *The above ILP formulation has* $\mathcal{O}(n^\ell)$ *variables and* $\mathcal{O}(n^{2\ell-1})$ *constraints.*

Proof. We have at most $2\ell - 1$ different vertices for each constraint that correspond to the vertices in the tuples $(b_1,\ldots,b_\ell)$ and $(c_1,\ldots,c_\ell)$ since we consider intersecting tuples, that is, there is $j \in \{1,\ldots,\ell\}$ such that $b_j = c_j$. For each such pair of tuples, we get at most $2^{\ell-1}$ constraints as we have at most two options to choose from, namely b_i or c_i, for every a_i in the tuple $(a_1,\ldots,a_\ell)$ and only one option for a_j resulting in at most $2^{\ell-1}\cdot n^{2\ell-1}$ constraints in total. □

3.4 Branchings

We know from Section 3.1 that ℓ-P ℓ-U Cluster Editing is NP-hard and, thus, if we want so solve ℓ-P ℓ-U Cluster Editing to optimality, then there is no way around to

think of algorithms that have an exponential running time in the worst case, but might be nevertheless useful in practice. In the previous section, we have suggested one way to do so by encoding an instance of ℓ-P ℓ-U Cluster Editing as an ILP and then hand it over to an existing ILP solver. However, it is often more promising to exploit combinatorial features of a specific problem and directly design an algorithm for that problem. Therefore, we consider ℓ-P ℓ-U Cluster Editing from a parameterized perspective in this section. We discuss branching algorithms for ℓ-P ℓ-U Cluster Editing that are fixed-parameter tractable with respect to the parameter k. These branching algorithms have in common that they solve at least one conflict—as characterized in Section 3.2—in each branching step.

3.4.1 Trivial Branching

We give a simple branching algorithm that is guaranteed to find an optimal solution for ℓ-P ℓ-U Cluster Editing, analogous to the trivial branching for Cluster Editing by Gramm et al. [Gra+05].

Search for a non-edge $f = a_1 \dots a_\ell$ that is covered by two intersecting edges $e_1 = b_1 \dots b_\ell$ and $e_2 = c_1 \dots c_\ell$ and branch into three cases:

1. delete the edge e_1;
2. delete the edge e_2;
3. insert all missing edges between the vertices $b_1, c_1, \dots, b_\ell, c_\ell$.

At this point we want to emphasize that this branching might not solve all conflicts among the vertices $b_1, c_1, \dots, b_\ell, c_\ell$ within one branching step. In the first and second case, some conflicts might remain and even new ones might arise. If we want to guarantee that all conflicts get solved in one step, then we need a more thorough case distinction regarding present and missing edges. Moreover, we observe that the performance of the branching depends on the third branching case, i.e. how many edges are missing. Thereby, we motivate the following lemma.

Lemma 3.9. *Given two conflicting edges $b_1 \dots b_\ell$ and $c_1 \dots c_\ell$ that intersect in $i \geq 1$ vertices and let j edges be present among the vertices $b_1, c_1, \dots, b_\ell, c_\ell$. Then, the trivial branching has a branching vector of $(1, 1, 2^{\ell-i} - j)$ for this conflict.*

Proof. If the vertices $b_1 \dots b_\ell$ and $c_1 \dots c_\ell$ intersect in i vertices, then we have $2^{\ell-i}$ possible edges among the vertices $b_1, c_1, \dots, b_\ell, c_\ell$. Assuming that j edges are already present leaves $2^{\ell-i} - j$ edges to insert. □

Lemma 3.10. *The above branching strategy finds an optimal solution for ℓ-P ℓ-U Cluster Editing and has a search tree of size $\mathcal{O}(3^k)$.*

Proof. Since we have a complete case distinction and resolve a conflict in each step, an algorithm based on this branching strategy terminates with an optimal solution. Subsequently, we analyze the size of the search tree: we can decrease the parameter k by one for each edge we add or delete. This results in a branching vector of size $(1, 1, 1)$ in case f is the only missing edge among the vertices $b_1, c_1, \dots, b_\ell, c_\ell$ and a strictly better

branching vector if more than one edge is missing. Thus, we can guarantee a search tree size of $\mathcal{O}(3^k)$ as a whole. □

Regarding the running time, on the one hand, a larger ℓ has the potential to scale down the branching factor in practice. On the other hand, conflicts can consist of more vertices, namely $2\ell - 1$, and therefore, searching for conflicting edges to branch on is more time-consuming for larger ℓ.

Proposition 3.11. *ℓ-P ℓ-U Cluster Editing can be solved in $\mathcal{O}(3^k \cdot n^{2\ell-1})$ time for $\ell \geq 3$.*

Proof. It is sufficient to show that each branching step can be executed in $\mathcal{O}(n^{2\ell-1})$ time. As a conflict consists of at most $2\ell - 1$ vertices, we can iterate over all sets with at most $2\ell - 1$ vertices and check for each set whether it is indeed a conflict. Clearly, the latter can be done in constant time as ℓ is fixed. □

3.4.2 Merge Branching

In the following, we adapt a branching introduced for Cluster Editing by Böcker et al. [Böc+09] to ℓ-P ℓ-U Cluster Editing, hoping to achieve a better branching vector than the trivial branching yields. In the following, we call this branching strategy *merge branching*.

As for Cluster Editing, we consider a weighted version to handle integer edge costs which arise from merging. Recall that we motivated this problem in Chapter 1 as algorithms for it can be used to detect radioactive sources.

Problem 4: Weighted ℓ-P ℓ-U Cluster Editing

Input: $G = (V = V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell)$, a cost function $w : V_1 \times \cdots \times V_\ell \to \mathbb{N}$ and an integer $k \in \mathbb{N}$.

Question: Is there a set $S \subseteq V_1 \times \cdots \times V_\ell$ of vertex tuples to delete or add with $\sum_{e \in S} w(e) \leq k$ such that the resulting graph is an ℓ-cluster graph?

Note that we may derive the NP-hardness of Weighted ℓ-P ℓ-U Cluster Editing from Theorem 3.3 as ℓ-P ℓ-U Cluster Editing is a special case of Weighted ℓ-P ℓ-U Cluster Editing. Moreover, observe that the trivial branching is correct for Weighted ℓ-P ℓ-U Cluster Editing as well and yields the same search tree size if we do not allow (non-)edges of cost zero.

Corollary 3.12. *An instance (G, w, k) of Weighted ℓ-P ℓ-U Cluster Editing can be solved in $\mathcal{O}(3^k \cdot n^{2\ell-1})$ time if it has no (non-)edges of cost zero.*

Recall that in the Cluster Editing setting, merging an edge means that this edge is present in the final cluster graph. In contrast, if we merge an edge in an ℓ-partite ℓ-uniform hypergraph, then the resulting graph is in general neither ℓ-partite nor ℓ-uniform anymore. Clearly, destroying the structure of the graph is to be avoided as we aim at designing a recursive algorithm. However, we can instead think of merging two vertices from the same V_i.

Definition 3.13. The *modified cost function* $w^* : V_1 \times \cdots \times V_\ell \to \mathbb{Z}$ is defined by $w^*(v_1 \dots v_\ell) = w(v_1 \dots v_\ell)$ if $v_1 \dots v_\ell$ is an edge, and by $w^*(v_1 \dots v_\ell) = -w(v_1 \dots v_\ell)$ otherwise.

The *merge operation* applied to two vertices b_i, c_i replaces these by a single vertex, say d_i, and adjusts the modified cost in the following way:

$$\text{for all } (a_1, \dots, a_{i-1}, a_{i+1}, \dots, a_\ell) \in V_1 \times \cdots \times V_{i-1} \times V_{i+1} \times \cdots \times V_\ell$$

set $w^*(a_1, \dots, a_{i-1}, d_i, a_{i+1}, \dots, a_\ell)$ to

$$w^*(a_1, \dots, a_{i-1}, b_i, a_{i+1}, \dots, a_\ell) + w^*(a_1, \dots, a_{i-1}, b_i, a_{i+1}, \dots, a_\ell).$$

Note that we may gather from a modified cost function w^* not only the corresponding cost function w, but also the set of edges, assuming that we interpret zero-edges always as non-edges. Thus, we may omit to adjust the edge set when merging.

We now give two simple merge branching strategies and analyze their running times. Search for two conflicting edges $e_1 = b_1 \dots b_\ell$ and $e_2 = c_1 \dots c_\ell$. Without loss of generality, we assume the edges intersect in $V_1, \dots, V_j$ and do not intersect in $V_{j+1}, \dots, V_\ell$, that is, $b_i = c_i$ for $i \leq j$ and $b_i \neq c_i$ for $i > j$.

Then, the first merge branching branches into three cases:

1. merge (b_ℓ, c_ℓ);
2. delete all edges containing $\{b_i, b_\ell\}$ for all $i \leq j$;
3. delete all edges containing $\{b_i, c_\ell\}$ for all $i \leq j$.

We say that we branch on the pair (b_ℓ, c_ℓ) in the above branching. In contrast, in the second branching we favor the merge operation to the deletion operation:

1. merge the vertices $(b_{j+1}, c_{j+1}), \dots, (b_\ell, c_\ell)$ pairwise;
2. delete the edge e_1;
3. delete the edge e_2.

Theorem 3.14. *For an instance* (G, w, k) *of* Weighted ℓ-P ℓ-U Cluster Editing *with* $\ell \geq 3$ *and cost function* $w : V_1 \times \cdots \times V_\ell \to \mathbb{N} \setminus \{0\}$*, both merge branchings are correct and yield a search tree of size* $\mathcal{O}(3.08^k)$ *resp.* $\mathcal{O}(4^k)$*.*

Proof. The correctness of the branchings follows from the fact that the algorithms do exhaustive branching and terminate on every input. Regarding the running time, we emphasize that we interpret zero-edges—that result from merging—as non-edges which is why k is decreased by at least one through each edge deletion. When merging two vertices b_ℓ, c_ℓ, we join for each $(a_1, \dots, a_{\ell-1}) \in V_1 \times \cdots \times V_{\ell-1}$ the (non-)edges $e_1 = a_1 \dots a_{\ell-1} b_\ell$ and $e_2 = a_1 \dots a_{\ell-1} c_\ell$ into a single (non-)edge with modified cost $w^*(a_1 \dots a_{\ell-1} b_\ell) + w^*(a_1 \dots a_{\ell-1} c_\ell)$ and decrease k each time by

$$\min\big\{w(a_1 \dots a_{\ell-1} b_\ell), w(a_1 \dots a_{\ell-1} c_\ell)\big\}$$

if either e_1 or e_2 is an edge and neither a zero-edge is joined nor results from merging. To handle resulting zero-edges, we apply a bookkeeping trick by Böcker et al. [Böc+09]: we reduce k only by

$$\min\{w(a_1 \dots a_{\ell-1}b_\ell), w(a_1 \dots a_{\ell-1}c_\ell)\} - \frac{1}{2}$$

when joining an edge with a non-edge produces a zero-edge and decrease k by the remaining $\frac{1}{2}$ when joining the resulting zero-edge at a later point.

To upper-bound the search tree size of the first branching we make a case distinction. Recall that $1 \leq j \leq \ell - 2$. If e_1 and e_2 are the only edges present among their vertices, then we merge at least twice an edge with a non-edge, namely $b_1 \dots b_\ell$ with $b_1 \dots b_{\ell-1}c_\ell$ and $c_1 \dots c_\ell$ with $c_1 \dots c_{\ell-1}b_\ell$, resulting in a branching vector of at least $(1, 1, 1)$. If there is another edge present, then, in the second or third branching case, at least two edges get deleted. Hence, we have a branching vector of at least $(\frac{1}{2}, 2, 1)$ in this case, yielding a branching factor of at most 3.08 overall.

As we cannot guarantee that more than once an edge is joined with a non-edge, we can only guarantee that we may decrease k by $\frac{1}{2}$ when merging in the second branching. Thus, the second branching has a branching vector of at least $(\frac{1}{2}, 1, 1)$, resulting in a branching factor of 4. □

Although the merge branchings do not yield a better branching vector than the trivial branching in theory, we expect the merge branching to be more efficient in practice. First, merging vertices shrinks an instance. Second, we can decrease the parameter k by more than our worst-case analysis yields if the affected (non-)edges have weights strictly greater than one. We expect that this is somehow 'likely' due to the fact that we increase (non-)edge weights through merging. Third, the merge branching proved successful for the sister problem Cluster Editing in the Parameterized Algorithms and Computational Experiments Challenge (PACE) 2021. For more details, we refer to Kellerhals et al. [Kel+21].

In theory, the first merge branching outperforms the second one. Nevertheless, we suggest to consider the second branching in practice as well since it favors the merge operation which has the above discussed advantages. We also remark that it seems that—in both branchings—the bottleneck appears when branching on sparse or dense conflicts. Therefore, it might be of interest to have a closer look at which hypergraphs only have sparse conflicts and whether data reduction rules or a different branching strategy could be used to circumvent these.

3.5 Kernelization

In this section, we consider a kernelization technique for Cluster Editing that produces a vertex kernel quadratic in k, see Gramm et al. [Gra+05], and adapt it to our setting, yielding a quadratic kernel for ℓ-P ℓ-U Cluster Editing as well. We start by presenting two data reduction rules.

Reduction Rule 3.5.1. *Remove all connected components that are ℓ-cliques from the graph.*

Lemma 3.15. *Reduction Rule 3.5.1 is correct and can be applied in polynomial time.*

Proof. For the correctness, it is sufficient to show that in an optimal solution no vertex whose connected component is already an ℓ-clique is incident to a modification. Assume the contrary, that is, there is a connected component C that is an ℓ-clique and an optimal solution S such that $S' = \{e \in S \mid V(e) \cap V(C) \neq \emptyset\}$ is non-empty. Then, we observe that $S \setminus S'$ is also a solution and is clearly better than S, yielding a contradiction.

Regarding the running time, we remark that computing the connected components of a hypergraph can be done in linear time using depth-first search [CKP18]. Checking whether a connected component is an ℓ-clique can also be done in linear time by just counting the number of present edges. □

In the subsequently presented data reduction rule, we set edges to *permanent* resp. *forbidden*. This means we store for each edge whether it has been assigned the attribute *permanent* and whether it has been assigned the attribute *forbidden*. Once assigned an attribute to an edge, it will not be reset.

Reduction Rule 3.5.2. *For all* $(a_1, \ldots, a_\ell) \in V_1 \times \cdots \times V_\ell$:

1. *If* $(a_1, \ldots, a_\ell)$ *is in strictly more than* k *pairwise (non-)edge-disjoint conflicts in* $G - a_1 \ldots a_\ell$*, then set* $a_1 \ldots a_\ell$ *to permanent and if applicable, add it to* E*.*
2. *If* $(a_1, \ldots, a_\ell)$ *is in strictly more than* k *pairwise (non-)edge-disjoint conflicts in* $G + a_1 \ldots a_\ell$*, then set* $a_1 \ldots a_\ell$ *to forbidden and if applicable, delete it from* E*.*
3. *If* $a_1 \ldots a_\ell$ *is set to permanent and forbidden, then the given instance is a No-instance.*

When an edge e is set to *permanent*, one may think of it as e must be present in the final ℓ-cluster graph. Analogously, when an edge e is set to *forbidden*, one may think of it as e is not allowed to be present in the final ℓ-cluster graph.

Lemma 3.16. *Reduction Rule 3.5.2 is correct.*

Proof. We prove the correctness of all three cases by contradiction.

Case 1: Assume an instance $G = (V, E)$ of ℓ-P ℓ-U Cluster Editing has a solution S of size k such that $a_1, \ldots, a_\ell$ is a non-edge in $G \triangle S$. We have at least $k + 1$ conflicts in $G - a_1, \ldots, a_\ell$ which are disjoint except from the vertices $a_1, \ldots, a_\ell$ yielding that each of them needs a modification involving at least one vertex not contained in $\{a_1, \ldots, a_\ell\}$ and thus, S must be of size at least $k+1$ which contradicts our assumption. We conclude that if G has a solution of size at most k, then $a_1 \ldots a_\ell$ is an edge in the resulting ℓ-cluster graph.

Case 2: Assume an instance $G = (V, E)$ of ℓ-P ℓ-U Cluster Editing has a solution S of size k such that $a_1, \ldots, a_\ell$ is an edge in $G \triangle S$. We have at least $k+1$ conflicts in $G + a_1, \ldots, a_\ell$ which are disjoint except from the vertices $a_1, \ldots, a_\ell$ yielding that each of them needs a modification involving at least one vertex not contained in $\{a_1, \ldots, a_\ell\}$ and thus, S must be of size at least $k+1$ which contradicts our assumption. We conclude that if G has a solution of size at most k, then $a_1 \ldots a_\ell$ is a non-edge in the resulting ℓ-cluster graph.

Case 3: From the proofs for Case 1 and Case 2 follows that a solution would require at least $k + 1$ edge modifications in both scenarios, namely if $a_1 \ldots a_\ell$ is an edge as well as if $a_1 \ldots a_\ell$ is a non-edge in the resulting ℓ-cluster graph. □

Lemma 3.17. *An instance of* ℓ-P ℓ-U Cluster Editing *can be reduced with respect to Reduction Rule 3.5.2 in polynomial time.*

Proof. We briefly sketch the algorithmic idea for exhaustive application of Reduction Rule 3.5.2. For a precise description of the data structures that enable the subsequent procedure, we refer to Gramm et al. [Gra+05]. Herein, the authors explain in detail the data structures the algorithm uses for the Cluster Editing setting and one can observe that they are transferable to our setting in a natural way.

Note that checking whether Case 1 or Case 2 applies to a vertex tuple $(a_1, \dots, a_\ell) \in V_1 \times \cdots \times V_\ell$ can be done in $\mathcal{O}(n^{\ell-1})$ time. Clearly, we have to check this for at most $|V_1 \times \cdots \times V_\ell|$ tuples but since Reduction Rule 3.5.2 performs changes within the graph, it is not obvious how often we have to recheck whether Case 1 or Case 2 is applicable. Hence, it is sufficient to argue why we do not have to do this 'too often'. Therefore, we make two observations: first, if during an iteration—that is, checking for all vertex tuples whether the data reduction rule is applicable—neither Case 1 nor Case 2 is at least once for the first time true, then we can stop. Second, we have at most $|V_1 \times \cdots \times V_\ell|$ iterations since once a (non-)edge is set to *permanent* resp. *forbidden*, it will not be reset. In case a second attribute is assigned in a later iteration, we may immediately stop as Case 3 is applicable and hence, the given instance is a No-instance. Altogether, this results in a running time of $\mathcal{O}(n^{3\ell-1})$. □

Theorem 3.18. ℓ-P ℓ-U Cluster Editing *has a problem kernel which contains* $\mathcal{O}(k^2)$ *vertices and* $\mathcal{O}(k^{2\ell})$ *edges for* $\ell \geq 3$.

Proof. Let G be an ℓ-partite ℓ-uniform hypergraph which is reduced with respect to Reduction Rules 3.5.1 and 3.5.2. Without loss of generality, we assume that G is connected as we can solve each connected component separately. Find a maximal set S of (non-) edge-disjoint conflicts. If S has cardinality strictly greater than k, then the given instance is a No-instance as every conflict needs at least one modification and they do not affect each other. For every (non-)edge $a_1 \dots a_\ell$ in one of these conflicts, we know that it can be involved in at most k other conflicts due to Reduction Rule 3.5.2. Recall that any conflict in S consists of at most $2\ell - 1$ vertices implying that there are at most $2^{\ell-1}$ (non-)edges among these vertices. Summing over S, we conclude that there are at most

$$((\ell - 1) \cdot 2^{\ell-1} \cdot k + 2\ell - 1) \cdot k$$

vertices that are contained in at least one conflict and thus, assuming that every vertex is contained in at least one conflict, the correctness of the theorem follows.

We now prove by contradiction that every vertex is indeed contained in a conflict. Hence, assume there is a vertex $a_1 \in V_1$ which is in no conflict. Clearly, $N[a_1]$ has to be an ℓ-clique. It is also easy to see that $N(N(a_1)) \subseteq N(a_1) \cup V_1$ holds. Hence, assume there is a vertex $b_1 \in V_1$ that shares an edge with $b_2 \dots b_\ell$ but not with $c_2 \dots c_\ell$ where both tuples lie in $V_2 \cap N(a_1) \times \cdots \times V_\ell \cap N(a_1)$. Then, a conflict is induced among the vertices $a_1, b_1, b_2, c_2, \dots, b_\ell, c_\ell$: if $b_1 c_2 b_3 \dots b_\ell$ is an edge, then the vertices $a_1, b_1, c_2, b_3, c_3, \dots, b_\ell, c_\ell$ form a conflict. Otherwise, the vertices a_1, b_1, b_2, c_2, $b_3, \dots, b_\ell$ form a conflict. Observe that in both cases the vertex a_1 is involved, leading to a contradiction. Similarly, one can see that any vertex $b_1 \in N(N(a_1))$ cannot have

neighbors different from $N(a_1)$. Altogether, we have shown that the connected component of a_1 is an ℓ-clique. However, any connected component that is an ℓ-clique was removed due to Reduction Rule 3.5.1, yielding a contradiction. Thus, every vertex is involved in at least one conflict. □

We conjecture that the given upper bound of $\mathcal{O}(k^{2\ell})$ on the number of edges is not tight and could be improved by a more thorough analysis as done for Cluster Editing by Gramm et al. [Gra+05]. In contrast, we have no idea whether a subquadratic-vertex kernel exists. For a more detailed discussion of that question, we refer to Section 4.2.2 where we discuss some—unfortunately unsuccessful—approaches to achieve a linear-vertex kernel for 3-P 3-U Cluster Editing.

3.6 Parameterization Above Lower Bounds

In this section, we try to adapt an idea for Cluster Editing by van Bevern, Froese, and Komusiewicz [vFK18] to our setting. We try to show that ℓ-P ℓ-U Cluster Editing is fixed-parameter tractable with respect to the parameter $k - h_v$, where h_v is a lower bound on k given by a set of vertex-disjoint conflicts. One of their key ingredients is a data reduction rule which allows them to upper-bound k by $c \cdot (k - h_v)$ for some constant c after exhaustive application of the data reduction rule. Unfortunately, the straightforward generalization of the rule for ℓ-P ℓ-U Cluster Editing—that would at the same time allow us to stick to their technique—turned out to be wrong. Hence, we could not show that ℓ-P ℓ-U Cluster Editing is fixed-parameter tractable with respect to $k - h_v$. In the following, we describe these observations in detail. First, let us introduce some definitions and lemmata.

Definition 3.19. A *vertex-disjoint packing* of induced subgraphs of a graph G is a set $\mathcal{H} = \{H_1, \ldots, H_t\}$ such that each H_i is an induced subgraph of G and the H_is are pairwise vertex-disjoint.

In the following, we denote by $\tau(H)$ the minimum number of edge modifications required to transform a graph H into an ℓ-cluster graph. Recall that we gave in Section 3.2 a characterization of an ℓ-cluster graph based on forbidden subgraphs. Therefore, to obtain a lower bound on k for ℓ-P ℓ-U Cluster Editing, we use a vertex-disjoint packing of such forbidden subgraphs in the style of the lower bound based on vertex-disjoint P_3s for Cluster Editing.

Lemma 3.20. *Let G be an instance of ℓ-P ℓ-U Cluster Editing, $\mathcal{H}$ a packing of vertex-disjoint forbidden subgraphs and S a solution for G. Then, $\sum_{H \in \mathcal{H}} \tau(H) \leq |S|$.*

Proof. As $\mathcal{H}$ is a packing of vertex-disjoint conflicts, these conflicts do not intersect and therefore, solving one conflict $H \in \mathcal{H}$ does not affect the other ones. By definition, H needs at least $\tau(H)$ modifications and summing up over all conflicts yields the inequality. □

At this point, we sketch the idea behind the proof that Cluster Editing is fixed-parameter tractable with respect to the parameter $k - h_v$. The authors van Bevern,

Froese, and Komusiewicz [vFK18] showed that for each induced subgraph H of G in a given packing $\mathcal{H}$, we may face two situations. If there is an optimal solution for H that is a subset of an optimal solution for G, then we can apply a data reduction rule. Otherwise, we find a specific structure in $N[H]$ that shows that H itself needs to be solved suboptimally or that a (non-)edge lying in the cut of H and $G \setminus H$ needs to be modified in an optimal solution for G. For the latter one, we use the following terminology.

Definition 3.21. A vertex tuple $(v_1, \ldots, v_\ell)$ is an *external* tuple for a packing graph $H \in \mathcal{H}$ if $\emptyset \neq \{v_1, \ldots, v_\ell\} \cap V(H) \subsetneq V(H)$.

Clearly, we can resolve a P_3—that is, the forbidden induced subgraph for Cluster Editing—with only one modification. However, conflicts of ℓ-P ℓ-U Cluster Editing vary in the number of involved vertices and, moreover, in the number of present edges. Thus, it is not that clear how many modifications we need at least to solve a conflict. Nevertheless, for any forbidden subgraph H, we can upper-bound the number $\tau(H)$ of required edge modifications by a constant.

Lemma 3.22. *Let H be a forbidden subgraph for ℓ-P ℓ-U Cluster Editing according to Theorem 3.5. Then, $\tau(H) \leq 2^{\ell-2}$.*

Proof. Theorem 3.5 implies that H has at most $2\ell-1$ vertices. Thus, the number of edges plus the number of non-edges equals $2^{\ell-1}$, implying that there are at most $2^{\ell-2}$ edges or at most $2^{\ell-2}$ non-edges. In the first case, we can delete all of them to transform H into an ℓ-cluster graph, and in the latter one, we can add all missing edges. □

We are now ready to adapt Lemma 3.2 from van Bevern, Froese, and Komusiewicz [vFK18] to our setting and show its correctness.

Lemma 3.23. *Let G be an instance of ℓ-P ℓ-U Cluster Editing, $\mathcal{H}$ a packing of vertex-disjoint conflicts and $r = k - \sum_{H \in \mathcal{H}} \tau(H)$. Let S be a solution of size k that contains, for each $H \in \mathcal{H}$,*

1. *at least $\tau(H) + 1$ vertex tuples within H or*
2. *at least one external vertex tuple for H.*

Then, $|\mathcal{H}| \leq \ell \cdot r$ and thus, $k \leq (\ell \cdot 2^{\ell-2} + 1)r$.

Proof. Denote by $\mathcal{H}_1 \subseteq \mathcal{H}$ the set of all graphs that fulfill the first property and let $\mathcal{H}_2$ denote the set containing the remaining packings from $\mathcal{H}$. According to prerequisites of the lemma, they fulfill the second property. Further, let $h_1 = \sum_{H \in \mathcal{H}_1} \tau(H)$ be the lower bound obtained by the graphs in $\mathcal{H}_1$ and let $h_2 = \sum_{H \in \mathcal{H}_2} \tau(H)$ be the lower bound obtained by the remaining graphs. The packing graphs in $\mathcal{H}_1$ cause at least $h_1 + |\mathcal{H}_1|$ modifications inside of them. Similarly, the packing graphs in $\mathcal{H}_2$ cause at least h_2 modifications inside of them and each packing graph $H \in \mathcal{H}_2$ has a vertex incident to at least one external modification for H. Since every vertex tuple is for at most ℓ different packing graphs an external vertex tuple, at least $h_2 + |\mathcal{H}_2|/\ell$ edge modifications are caused by the graphs in $\mathcal{H}_2$, yielding

$$\begin{aligned} & k \geq h_1 + |\mathcal{H}_1| + h_2 + |\mathcal{H}_2|/\ell \\ \Leftrightarrow \quad & k - h_1 - h_2 \geq |\mathcal{H}_1| + |\mathcal{H}_2|/\ell \\ \Leftrightarrow \quad & \ell \cdot r \geq \ell \cdot |\mathcal{H}_1| + |\mathcal{H}_2| \geq |\mathcal{H}|. \end{aligned}$$

Combining the above inequality with Lemma 3.22 yields

$$k = r + h_1 + h_2 \leq r + 2^{\ell-2} \cdot |\mathcal{H}| \leq r + 2^{\ell-2}\ell \cdot r = (1 + 2^{\ell-2}\ell)r.$$

□

Thus, if we could somehow simplify every instance G together with a packing of vertex-disjoint conflicts $\mathcal{H}$ such that they fulfill the prerequisites of the preceding lemma, then we would know that ℓ-P ℓ-U Cluster Editing is fixed-parameter tractable with respect to r. This motivates us to present the following attempt towards a data reduction rule that, if it was correct, would guarantee that $(G, \mathcal{H})$ satisfies the prerequisites of Lemma 3.23 after exhaustive application of the data reduction rule.

Reduction Rule Attempt 3.6.1. *If $G = (V = V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell)$ contains a forbidden induced subgraph $H = (W = W_1 \dot\cup \ldots \dot\cup W_\ell, F)$ having an optimal solution S of size $\tau(H)$ such that*

- *for all $i \in \{1, \ldots, \ell\}$ no two vertices $u, v \in W_i$ are contained in a conflict in $G \triangle S$ and*
- *no vertex tuple $(v_1, \ldots, v_\ell) \in W_1 \times \cdots \times W_\ell$ is contained in a conflict in $G \triangle S$,*

then replace G by $G \triangle S$, remove H from $\mathcal{H}$ and, decrease k by $\tau(H)$.

Lemma 3.24. *Let G be an instance of ℓ-P ℓ-U Cluster Editing and $\mathcal{H}$ a packing of vertex-disjoint conflicts. After exhaustive application of Reduction Rule Attempt 3.6.1, the reduced graph and its packing fulfill the prerequisites of Lemma 3.23.*

Proof. We know that for each conflict $H \in \mathcal{H}$ and each optimal solution S for H, two vertices $u, v \in V_i$ are in a conflict in $G \triangle S$ or there is a vertex tuple $(v_1, \ldots, v_\ell)$ such that the vertices $\{v_1, \ldots, v_\ell\}$ are contained in a conflict in $G \triangle S$. In the first case, at least one vertex of $\{u, v\}$ needs a modification besides S or any optimal solution for G needs more than $\tau(H)$ modifications inside S. The same is true if $\{v_1, \ldots, v_\ell\}$ are contained in a conflict in $G \triangle S$: recall that a conflict consists of at most $2\ell - 1$ vertices and, hence, when solving this conflict at least one of the vertices $\{v_1, \ldots, v_\ell\}$ is incident to a modification. □

At this point, we shortly remark that for the above lemma it is fundamental that we assume $\ell \geq 3$ as a forbidden subgraph of Bicluster Editing consists of exactly $2 \cdot \ell = 4$ vertices. We will come back to this observation in Section 5.2.

Reduction Rule Attempt 3.6.1 is a straightforward adaption of the data reduction rule introduced by van Bevern, Froese, and Komusiewicz [vFK18] to show that Cluster Editing is fixed-parameter tractable above a lower bound. Therefore, we assumed that proving its correctness could be done in a somehow similar way. However, in the meantime it turned out that Reduction Rule Attempt 3.6.1 is not correct, which we prove in the following.

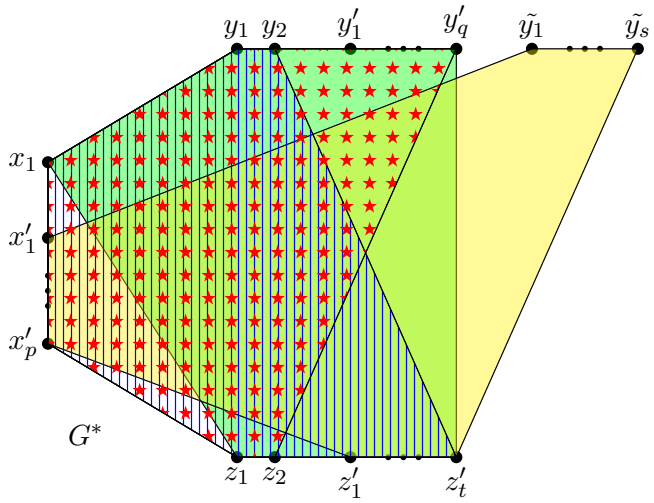


Figure 3.3: We depict a 3-partite 3-uniform hypergraph G^* after Reduction Rule Attempt 3.6.1 inserted the edge $x_1y_1z_1$. The filled resp. patterned polygons indicate that the bordered vertices induce a triclique.

Lemma 3.25. *Reduction Rule Attempt 3.6.1 is not correct.*

Proof. We want to emphasize that, for the moment, the index of a vertex does not indicate of which element of the partition it is from. Instead, we assume $V = X \dot\cup Y \dot\cup Z$ and denote a vertex from X, Y resp. Z by x_i, y_i resp. z_i. We now show Lemma 3.25 by giving a counterexample. To this end, consider the graph $G = G^* - x_1y_1z_1$ where G^* is the graph shown in Figure 3.3 and choose $p = q = t = 10$ and $s = 100$. We first show that Reduction Rule Attempt 3.6.1 inserts the edge $e_1 = x_1y_1z_1$ in G and, second, that it cannot be present in any optimal solution.

A conflict involving the vertices x_1, y_1, z_1 can only be found within $N[x_1, y_1, z_1]$, but for any $u, v \in N(x_1, y_1, z_1)$, the vertices x_1, y_1, z_1, u, v form a triclique in G^*. Further, note that x_1, x_2 resp. z_1, z_2 are strong twins in G^* and therefore, the prerequisites of Reduction Rule Attempt 3.6.1 are satisfied.

In the following, we first construct a solution S in which the vertices x_1, y_1, z_1 do not all end up in the same triclique and afterwards use S to show that every tricluster graph in which x_1, y_1, z_1 are indeed in the same triclique cannot be optimal. We define S indirectly by giving the resulting tricliques in $G \triangle S$:

$$T_1 = \{x'_1, \dots x'_p, y_1, y_2, \tilde{y_1}, \dots, \tilde{y_s}, z'_1, \dots, z'_t\},$$
$$T_2 = \{x_1, y'_1, \dots, y'_q, z_1, z_2\}.$$

We observe that no edge gets inserted in $G \triangle S$ and we have

$$2 \cdot (t+2) + q \cdot t + p \cdot 2 \cdot 2 + p \cdot q \cdot 2 = 364$$

edge deletions.

Next, we assume that the data reduction rule is correct implying that there is an optimal solution in which the vertices x_1, y_1, z_1 end up in the same triclique, say T_1. As the pairs y_1, y_2 resp. z_1, z_2 are strong twins in $G + e_1$, we may assume that y_2 and z_2 also end up in T_1 by Lemma 3.2. Moreover, each of the sets $X' = \{x'_1, \ldots, x'_p\}$, $Y' = \{y'_1, \ldots, y'_q\}$, $\widetilde{Y} = \{\tilde{y}_1, \ldots, \tilde{y}_s\}$ and $Z' = \{z'_1, \ldots, z'_t\}$ is a critical independent set and thus, we may further assume that none of these is split. Moreover, we know that the vertices of $X', \widetilde{Y}, Z'$ need to be put into the same triclique, say T_2: otherwise, we already have to pay for $10 \cdot 10 \cdot 10 = 1000$ deletions which is strictly larger than S is. Similarly, we can argue that Y' cannot end up in T_2 as this would also cost 1000 edge insertions. Let T_3 be the triclique of Y' in an optimal tricluster graph. Altogether, there are three possibilities left how the vertices can be grouped in an optimal tricluster graph:

1. T_1, T_2, and T_3 are all separate tricliques.
2. $T_1 = T_3 = \{x_1, y_1, y_2, z_1, z_2\} \cup Y'$ and $T_2 = X' \cup \widetilde{Y} \cup Z'$.
3. $T_1 = T_2 = \{x_1, y_1, y_2, z_1, z_2\} \cup X' \cup \widetilde{Y} \cup Z'$ and $T_3 = Y'$.

What is left is to argue why each of the three tricluster graphs needs more modifications than $G \triangle S$ does. To construct the first tricluster graph, we need

$$2 \cdot t + q \cdot (t + 2) + p \cdot 2 \cdot (t + 2) + p \cdot q \cdot 2 = 580$$

deletions. Similarly, we need

$$2 \cdot t + q \cdot t + p \cdot 2 \cdot (t + 2) + p \cdot q \cdot 2 = 560$$

deletions to construct the second solution. Clearly, both cannot yield an optimal solution since S is of smaller cardinality. For the last case, we only need to observe that we already need $p \cdot s \cdot 2 = 2000$ edge insertions for the missing edges in $\{X', \widetilde{Y}, z_1, z_2\}$.

Finally, we may conclude that in any optimal solution, the edge e_1 is missing and hence, Reduction Rule Attempt 3.6.1 is not correct. □

We conjecture that our counterexample can be adapted to show that it is also not correct for any $\ell > 3$.

It seems natural to think about whether Reduction Rule Attempt 3.6.1 can be modified such that it becomes correct and the prerequisites of Lemma 3.23 are still fulfilled after exhaustive application. One idea could be to tighten its prerequisites by requiring that no $j < \ell$ vertices of H are contained in a conflict in $G \triangle S$. But a conflict involving $j < \ell$ vertices could theoretically be solved by a modification within the other $2\ell - 1 - j \geq \ell$ vertices. Thus, even if this adaption is correct, we do not know how to guarantee that in a reduced instance each $H \in \mathcal{H}$ has an external vertex tuple or at least $\tau(H) + 1$ modifications in an optimal solution.

Nevertheless, it was insightful that this approach is not adaptable in the natural way to ℓ-P ℓ-U Cluster Editing and it might be an indication that the multipartite structure makes the problem in some way harder compared to Cluster Editing.

Chapter 4

3-Partite 3-Uniform Cluster Editing

In the previous chapter, we have studied some aspects of parameterized complexity of ℓ-P ℓ-U Cluster Editing. In this chapter, we fix $\ell = 3$ hoping to improve some of the previous results for this special case as this seems to be the simplest one. Therefore, we first look more closely at its forbidden subgraph to state its forbidden induced subgraphs. With the help of these, we refine the merge branching in Section 4.1 and subsequently discuss why the size of the thereby obtained search tree is indeed tight. In Section 4.2, we discuss further kernelization approaches for 3-P 3-U Cluster Editing with the hope to show that a linear-vertex kernel exists. However, we observe that these attempts were not successful and try to give an intuition why this is the case.

Problem 5: 3-P 3-U Cluster Editing

Input: A 3-partite 3-uniform hypergraph $G = \big(V = V_1 \dot\cup V_2 \dot\cup V_3, E \subseteq V_1 \times V_2 \times V_3\big)$ and an integer $k \in \mathbb{N}$.

Question: Is there a set $S \subseteq V_1 \times V_2 \times V_3$ of vertex tuples with $|S| \leq k$ to delete or add such that the resulting graph is a tricluster graph, that is, each connected component is a triclique?

From now on, we denote the vertex partition of a 3-uniform 3-partite hypergraph by $X = \{x_1, \ldots, x_r\}$, $Y = \{y_1, \ldots, y_p\}$, and $Z = \{z_1, \ldots, z_q\}$. In doing so, we implicitly assume that there is an edge-preserving bijection $f : \{V_1, V_2, V_3\} \to \{X, Y, Z\}$.

Theorem 3.5 from the previous chapter allows us to characterize an ℓ-cluster graph by forbidding certain subgraphs. Fixing $\ell = 3$ yields the following characterization of a tricluster graph.

Corollary 4.1. *A 3-partite 3-uniform hypergraph $G = (V, E)$ is a tricluster graph if and only if it contains no five vertices x_1, y_1, y_2, z_1, z_2 such that $x_1y_1z_1, x_1y_2z_2 \in E$ and $x_1y_2z_1 \notin E$.*

The above corollary states the forbidden subgraph for 3-P 3-U Cluster Editing. Next, we move on to its forbidden induced subgraphs to make a more thorough case distinction within the merge branching introduced in Section 3.4.2.

(a) Conflict of type C1 (b) Conflict of type C2

Figure 4.1: An illustration of the forbidden induced subgraphs for 3-P 3-U Cluster Editing. In (a) the only present edges among the vertices x_1, y_1, y_2, z_1, z_2 are the edges $x_1y_1z_1$ and $x_1y_2z_2$ whereas in (b) we have additionally the edge $x_1y_1z_2$.

4.1 Branchings

Note that in a conflict of 3-P 3-U Cluster Editing there are exactly four (non)-edges of which we specified three in Corollary 4.1. This leads to two possible induced subgraphs: if the fourth one, namely $x_1y_1z_2$, is a non-edge, then we say that the five vertices induce a conflict of type C1. Else, we refer to them as a conflict of type C2, compare to Figure 4.1.

We now recap the branchings introduced in Section 3.4 for the special case $\ell = 3$, starting with the trivial branching.

Search for a conflict x_1, y_1, y_2, z_1, z_2 of type C1 or C2 and branch into three cases:

1. delete the edge $x_1y_1z_1$;
2. delete the edge $x_1y_2z_2$;
3. insert all missing edges between x_1, y_1, y_2, z_1, z_2.

By Lemma 3.10, we know that the above branching yields a search tree of size $\mathcal{O}(3^k)$. Unfortunately, exploiting the fact that we fixed $\ell = 3$ has no use at this point as a conflict of type C2 only inserts one edge in the third branching case.

Next, we want to study the first merge branching from Section 3.4.2. As before, we have to consider the weighted version to handle integer edge costs which arise from merging.

Problem 6: Weighted 3-P 3-U Cluster Editing
Input: A 3-partite 3-uniform hypergraph $G = \big(V = V_1 \dot\cup V_2 \dot\cup V_3, E \subseteq V_1 \times V_2 \times V_3\big)$, a cost function $w : V_1 \times V_2 \times V_3 \to \mathbb{N}$, and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V_1 \times V_2 \times V_3$ of vertex tuples to delete or add with $\sum_{e \in S} w(e) \leq k$ such that the resulting graph is a tricluster graph?

Search for a conflict x_1, y_1, y_2, z_1, z_2 of type C1 or C2 and branch into three cases:

1. merge the vertices y_1, y_2;
2. delete all edges containing $\{x_1, y_1\}$;

3. delete all edges containing $\{x_1, y_2\}$.

Recall that the merge branching yields a search tree of size $\mathcal{O}(3.08^k)$, see Theorem 3.14. Again, we say that the above branching strategy branches on the pair y_1, y_2 and call y_1, y_2 a conflicting pair.

4.1.1 Refined Merge Branching

At this point, we want to improve our merge branching by exploiting that we fixed $\ell = 3$. Observe that a conflict of type C1 provides already two pairs of edges and non-edges that are joined when applying the merge branching. For conflicts of type C2, we make use of an idea by Böcker, Briesemeister, and Klau [BBK11] introduced to refine the merge branching for Cluster Editing: in case we only have conflicts of type C2, we choose a somehow 'good' conflicting pair to branch on, i.e. we choose a conflicting pair y_1, y_2 such that as many edges as possible are joined with non-edges when y_1, y_2 are merged. Thereby, we motivate the following lemma.

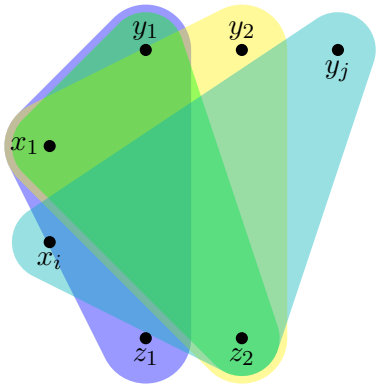
Lemma 4.2. *Given a connected, 3-partite 3-uniform hypergraph $G = (V, E)$. Assume that for every vertex pair from the same V_i at most one edge is joined with a non-edge when merging the pair. Then, G is either a triclique or a triclique minus a single edge.*

Proof. If G contains no conflict of type C1 or C2, then G is a triclique. Moreover, we know that only conflicts of type C2 can exist. Hence, we choose a conflict, x_1, y_1, y_2, z_1, z_2, of type C2 and without loss of generality, we assume that every possible edge between the five vertices is present except of $f = x_1y_2z_1$. We constructively show that G is a triclique minus the edge f.

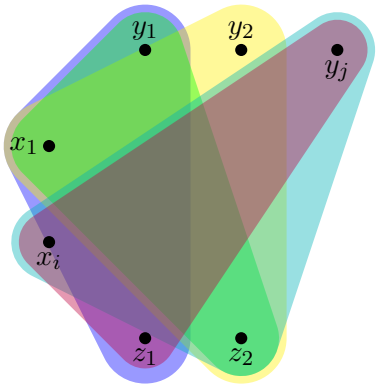
Recall the definition of the *tuple neighborhood* $N^t(v)$ of a vertex v from Chapter 2. If another vertex y_j with $j \neq 1, 2$ is adjacent to x_iz_2 for some $x_i \in X$, then it is also adjacent to x_iz_1 and vice versa: otherwise, $|N^t(z_1) \triangle N^t(z_2)| \geq 2$ holds. In case $i \neq 1$ holds, we still need to check whether all edges, except of f, among the vertices $x_1, x_i, y_1, y_2, y_j, z_1, z_2$ are present. We refer to Figure 4.2 for a better understanding of the subgraph we consider in the following. We know that either $N^t(y_j) = N^t(y_1)$ or $N^t(y_j) = N^t(y_2)$ holds enforcing that $x_1y_jz_2, x_iy_1z_1, x_iy_1z_2, x_iy_2z_1, x_iy_2z_2$ are edges. This in turn implies that $x_1y_jz_2$ is an edge as otherwise $|N^t(x_1) \triangle N^t(x_i)| \geq 2$ would hold. Hence, we are given eleven out of twelve edges among the vertices $x_1, x_i, y_1, y_2, y_j, z_1, z_2$, implying that f is the only missing edge.

Clearly, our observations above are symmetric, i.e. we may replace y_j by some vertex of $Z \setminus \{z_1, z_2\}$ and adapt the proof accordingly.

Next, assume that a_1 is adjacent to y_jz_k with $j, k \notin \{1, 2\}$. Again, we know that either $N^t(y_j) = N^t(y_1)$ or $N^t(y_j) = N^t(y_2)$ holds, implying that $x_1y_jz_2$ is an edge. Symmetrically, we obtain that $x_1y_1z_k$ is an edge. This in return implies that the edges $x_1y_jz_1, x_1y_2z_k$ are also present. Thus, we have shown that eight of nine edges among the vertices $x_1, y_1, y_2, y_j, z_1, z_2, c_k$ are present and conclude that, if there is another edge $x_iy_jz_k$ missing in $G[N[x_1, y_1, y_2, z_1, z_2]]$, then $i \neq 1$ and $j, k \notin \{1, 2\}$. We show that neither such an edge can be missing: on the one hand, we know that y_j is adjacent to x_1z_1, and on the other hand we know that z_k is adjacent to x_iy_2. This yields a contradiction as again $|N^t(y_2) \triangle N^t(y_j)| \geq 2$ would hold.



(a) A conflict of type C2 among the vertices x_1, y_1, y_2, z_1, z_2 and two vertices x_i, y_j sharing an edge with z_2.



(b) We observe that the edge $x_i y_j z_1$ also needs to be present to not contradict $|N^t(z_1) \triangle N^t(z_2)| \leq 1$.

Figure 4.2: An illustration of one of the substructures we encounter in the proof of Lemma 4.2.

Finally, we prove that $N[x_1, y_1, y_2, z_1, z_2] = V(G)$. To this end, we show that the neighborhood of our conflict equals its second neighborhood. Assume the contrary, i.e. there is an $x_i y_j z_k \in E$ with y_j in the second—but not first—neighborhood of y_1 and x_i or z_k lying in the first neighborhood. However, this implies that both, y_1 and y_j, are incident to x_i resp. z_k and $|N^t(y_1) \triangle N^t(y_j)| \geq 2$ which contradicts our assumption. The same argument works when replacing y_j by x_i resp. z_k. From there we have shown that G is a triclique minus the single edge f, proving the claim. □

Clearly, we would like to solve the special case of instances that are missing only one edge somehow efficiently. Fortunately, this special case can indeed be solved in polynomial time, even for general $\ell \geq 3$.

Lemma 4.3. *ℓ-P ℓ-U Cluster Editing can be solved in polynomial time given that the input graph is an ℓ-clique minus a single edge.*

Proof. We briefly sketch how we can solve such an instance in polynomial time by making use of the fact that also in hypergraphs a minimum (s,t)-cut can be computed in polynomial time [Law73]: let f be the missing edge. Then, compute for all sets $\{s,t\} \in \binom{V(f)}{2}$ the cost of a minimum (s,t)-cut. If the cut of minimum cost among the $\binom{\ell}{2}$ cuts, or inserting the missing edge has cost at most k, then it is a Yes-instance, otherwise a No-instance.

The above procedure is correct since in case inserting the edge f is too expensive, that is $w(f) > k$, at least two of its vertices have to end up in different ℓ-cliques. Thus, it is sufficient to compute all $\binom{\ell}{2}$ minimum cuts. □

We are now ready to define the *refined merge branching*: check whether all conflicting pairs y_1, y_2 are involved in at most one conflict, and in this case solve the instance in polynomial time. Otherwise, branch on a conflicting pair y_1, y_2 where $|N^t(y_1) \triangle N^t(y_2)|$ is maximum.

Theorem 4.4. *For an instance (G, w, k) of* Weighted 3-P 3-U Cluster Editing *with $w : V_1 \times V_2 \times V_3 \to \mathbb{N} \setminus \{0\}$, the refined merge branching strategy yields a search tree of size $\mathcal{O}(3^k)$.*

Proof. The correctness follows from Lemmas 4.2 and 4.3 together with the observation that, in case the refined merge branching branches on a conflict of type C2, we can reduce the parameter k by $2 \cdot \frac{1}{2}$ when merging and hence, achieve a branching vector of $(1, 2, 1)$ for conflicts of type C2. □

4.1.2 Limitations of the Merge Branching

One might wonder whether it is also possible to improve the branching vector for conflicts of type C1 in a similar way. However, we will see hereinafter that this requires changes in the branching itself. To this end, we first introduce a graph class of 3-partite 3-uniform hypergraphs that have only conflicts of type C1.

Definition 4.5. A 3-partite 3-uniform hypergraph $G = (V = X \dot\cup Y \dot\cup Z, E)$ is called a *star hypergraph* if $X = \{x_1\}, Y = \{y_1, \ldots, y_n\}, Z = \{z_1, \ldots, z_n\}$ and $E = \bigcup_{i=1,\ldots,n} x_1 y_i z_i$.

A star hypergraph has only conflicts of type C1 as the only conflict involving the pair y_i, y_j is the conflict induced by x_1, y_i, y_j, z_i, z_j. Thus, also when applying the refined merge branching, we cannot guarantee that we can lower the parameter k by more than one. The following theorem shows that there is also no hope to solve this special case in polynomial time, unless P = NP.

Theorem 4.6. Weighted 3-P 3-U Cluster Editing *is NP-hard even if the underlying unweighted graph is a star hypergraph.*

We prove the proposition by reducing from the problem Clique, which is NP-hard by Karp [Kar72].

Problem 7: Clique
Input: A graph $G = (V, E)$ and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V$ of vertices with $|S| \geq k$ such that all vertices in S are pairwise adjacent in G?

For the sake of clarity, we take an intermediate step. To this end, we introduce the Prize Collecting Clique problem.

Problem 8: Prize Collecting Clique
Input: A set V of vertices, connection costs $c : \binom{V}{2} \to \mathbb{N}$, penalty costs $p : V \to \mathbb{N}$, and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V$ such that the overall cost for making S a clique and isolating $V \setminus S$, that is $\sum_{\{u,v\} \in \binom{S}{2}} c(\{u, v\}) + \sum_{v \in V \setminus S} p(v)$, is at most k?

According to our knowledge, this problem has not been studied before. We chose the name Prize Collecting Clique following the NP-hard problems Prize Collecting Steiner Tree and Prize Collecting TSP [Bie+93]. For each of them the task is to choose a subset S of vertices such that the edge costs we have to pay for, roughly speaking, building a particular graph upon S—namely a clique, a tree or a tour—plus the sum of penalties associated with vertices not contained in S, is minimized.

Lemma 4.7. Prize Collecting Clique *is NP-hard.*

Proof. Let $(G = (V, E), k)$ be an instance of Clique. Then, we construct an instance (V', c', p', k') of Prize Collecting Clique as follows: set $V' = V$, $p'(v) = 1$ for all $v \in V'$, $k' = |V| - k$ and $c'(\{u, v\}) = 0$ if $\{u, v\} \in E$, and $c'(\{u, v\}) = k' + 1$ otherwise.

First, assume that (G, k) is a Yes-instance of Clique, i.e. there is a clique $C \subseteq V$ of size k. We choose $S = C$ and isolate all other vertices. Note that isolating them, i.e. deleting all their incident edges, has zero cost. Hence, we only need to pay penalty costs of amount $|V| - k$. Second, assume that the constructed instance (V', c', p', k') is a Yes-instance of Prize Collecting Clique. As adding one edge already exceeds our budget k', we know that we can split the graph into a clique and isolated vertices by only paying penalty costs of at most $|V| - k$. Thus, there is a clique of size k in G. □

Proof of Theorem 4.6. We show the claim by reducing from Prize Collecting Clique. Given an instance $(V = \{v_1, \dots, v_n\}, c, p, k)$ of Prize Collecting Clique, we construct an instance $(G' = (V' = X \dot\cup Y \dot\cup Z, E'), w', k')$ of Weighted 3-P 3-U Cluster Editing that is a star hypergraph as follows: for every vertex $v_i \in V$ introduce two vertices $y_i \in Y$ and $z_i \in Z$. Recall that $X = \{x_1\}$ and set the edge costs to $w'(x_1 y_i z_i) = 2p(v_i)$, the non-edge costs to $w'(x_1 y_i z_j) = c(\{v_i, v_j\})$ for $i \neq j$ and the solution size to $k' = 2k$.

For the first direction, we assume that we have a Yes-instance of Prize Collecting Clique, i.e. there is $S \subseteq V$ such that making S a clique and isolating $V \setminus S$ has cost at most k. We can turn G' into a tricluster graph by adding the edges $x_1 y_i z_j$ and $x_1 y_j z_i$ for $v_i, v_j \in S$ and $i \neq j$, and deleting the edges $x_1 y_i z_i$ for $v_i \in V \setminus S$. By construction, the combined cost of insertions and deletions is at most $2k$ and hence, the resulting instance is a Yes-instance for Weighted 3-P 3-U Cluster Editing.

For the second direction, we assume that the resulting star hypergraph (G', w', k') is a Yes-instance of Weighted 3-P 3-U Cluster Editing. Note that we may assume without loss of generality that either both vertices y_i and z_i lie in the same connected component with x_1 or both get isolated in the resulting tricluster graph. Thus, if y_i and z_i get isolated, then we also isolate the corresponding vertex $v_i \in V$ in a solution for Prize Collecting Clique at half the cost. Last, observe that if y_i, z_i, y_j, z_j are part of the same triclique, then we can put v_i, v_j into S and pay $w'(x_1 y_i z_j) + w'(x_1 y_j z_i)$. Hence, the original instance is a Yes-instance for Prize Collecting Clique.

Clearly, the two preceding many-one reductions need polynomial time and hence, combining these two proves the theorem. □

Although the merge branching does still not yield a better branching vector than the trivial branching in theory, we emphasize again that we expect the merge branching to be more efficient in practice due to the reasons we discussed in Section 3.4.2.

4.2 Kernelization

There are different kernelization techniques that yield a linear-vertex kernel for Cluster Editing [CC12; CM12; Fel+07; Guo09]. In this section, we want to discuss to what extent they are adaptable to 3-P 3-U Cluster Editing.

4.2.1 Kernelization via Critical Independent Sets

Guo [Guo09] presented a kernelization algorithm that produces a kernel of $\mathcal{O}(k)$ vertices for Cluster Editing. An adapted version of this algorithm was given by [Guo+08] for Bicluster Editing, yielding a linear-vertex kernel for this problem as well. It had a slight inaccuracy which was corrected by Lafond [Laf20]. In this section, we adapt their technique to provide another kernelization algorithm that produces a kernel of $\mathcal{O}(k^2)$ vertices for 3-P 3-U Cluster Editing.

Recall the definition of a critical independent set from Section 3.1. We make again use of this concept to introduce a new data reduction rule that is the key for our kernelization algorithm. Its basic idea is to upper-bound the number of vertices, that are not incident to any modification in the resulting cluster graph, by a function in k. Clearly, vertices from the same V_i that end up in the same triclique and are not incident to any modification, form already a critical independent set in the input graph. Thus, the idea of the data reduction rule is to ensure that there are not 'too many' vertices in a critical independent set by deleting some under certain conditions.

Reduction Rule 4.2.1. *Consider a critical independent set $X' \subseteq X$. Let $Y'' \subseteq Y' \subseteq Y$ and $Z'' \subseteq Z' \subseteq Z$ such that $Y' \cup Z' = N(X')$ and for $y \in Y''$ resp. $z \in Z''$ there is an $(x, z) \in X \setminus X' \times Z \setminus Z'$ resp. $(x, y) \in X \setminus X' \times Y \setminus Y'$ with $xyz \in E$. Further, let $X'' = N(N(X')) \cap X$. If $\max\{|X'' \times Y''|, |X'' \times Z''|\} < |X'|$, then remove any vertex from X'.*

Lemma 4.8. *Reduction Rule 4.2.1 is correct.*

Proof. For the better understanding of the following proof, we hint to Figure 4.3 where we show an exemplary subgraph satisfying the preconditions of Reduction Rule 4.2.1.

We first claim that, as long as $\max\{|X'' \times Y''|, |X'' \times Z''|\} \leq |X'|$, there is an optimal solution constructing a tricluster graph that contains a triclique T with $X' \cup N(X') \subseteq T \subseteq X' \cup N(X') \cup X''$. Hence, no edge incident to vertices in X' is inserted or deleted. Observe that the graphs before and after the application of Reduction Rule 4.2.1 differ only in the size of X' and thus, given that the lemma above is proven, we have shown the correctness of Reduction Rule 4.2.1.

We structure the proof similarly to the NP-hardness proof of ℓ-P ℓ-U Cluster Editing, compare to Section 3.1. First, recall that Lemma 3.2 ensures that X' does not need to be split, that is, there is always an optimal solution such that all vertices in X' end up in the same triclique T. Next, we show by contradiction that no vertex outside of $X' \cup N(X') \cup X''$ is put into T. Hence, assume that there is a vertex $v \notin X' \cup N(X') \cup X''$ ending up in T. First, let $v \in Y \setminus Y'$. Observe that we need at most $|X'' \times Z''|$ edge deletions to isolate v from $X' \cup N(X') \cup X''$ but would need at least $|X'|$ edge insertions to put v into T. Hence, it cannot be better to have v in the triclique T than cutting it off. Note that the same is true for $v \in Z \setminus Z'$. Second, let $v \in X \setminus (X' \cup X'')$, that is, v does not neighbor any vertex in $N(X')$. But we have just seen that no other vertices of Y or Z besides $N(X')$ end up in T and thus, excluding v has cost zero.

Finally, we show that $N(X')$ cannot be outside of T. This is easy to see as excluding any vertex $v \in Y'$ requires at least $|X' \times Z'|$ edge deletions and in contrast, v needs at most $|X'' \times Z'|$ edge insertions in T. Clearly, the same is true for $v \in Z'$. □

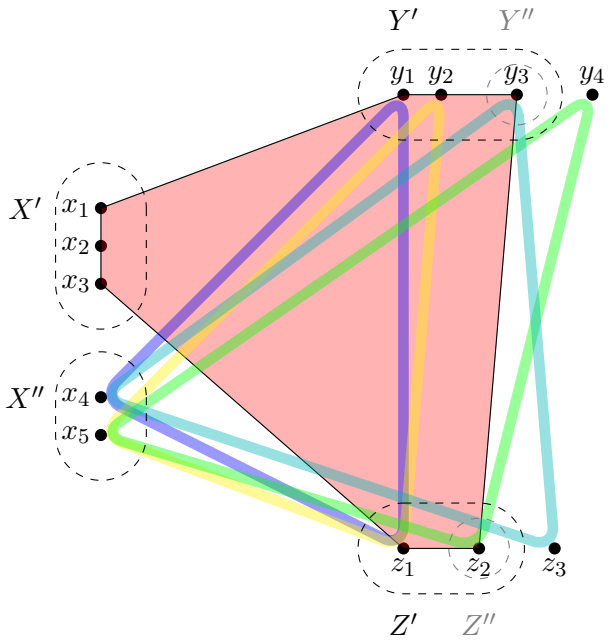


Figure 4.3: A subgraph with a critical independent set X' satisfying the precondition of Reduction Rule 4.2.1. For the sake of clarity, we indicate a 3-dimensional hyperedge only by the shape of a triangle. Whereas the red, filled polygon indicates that the bordered vertices induce a triclique.

Lemma 4.9. *An instance of* 3-P 3-U Cluster Editing *can be reduced with respect to Reduction Rule 4.2.1 in $\mathcal{O}(n^3)$ time.*

Proof. In the following, we sketch an algorithm which has running time $\mathcal{O}(n^3)$ to show the above claim. First, one can compute all critical independent sets in $\mathcal{O}(n^3)$ time by iterating over each vertex $v \in V$, determining its neighborhood and deciding whether $N[v]$ is a triclique. If it is, then we store $(v, N(v))$. This step can clearly be executed in $\mathcal{O}(n^3)$ time. Second, we iterate over the remaining vertices—namely, the ones which form a triclique together with their neighborhoods—and check which vertices have equal neighborhood. As we only need to store $N(v)$—and not $N^t(v)$—this step can also be implemented in $\mathcal{O}(n^3)$. Hence, we found all critical independent sets and their neighborhoods. We denote these by $(A', N(A'))$ in the following.

To check the applicability of Reduction Rule 4.2.1, one iterates over all critical independent sets $(A', N(A'))$, computes $A'' = N(N(A')) \cap V_i$ for $A' \subseteq V_i$ and checks whether the precondition of Reduction Rule 4.2.1 is fulfilled. Note that after one application of Reduction Rule 4.2.1, one only has to adapt the sizes of the critical independent sets whose vertices lie in $N(A')$ and A'' and the sizes of their neighborhoods. Therefore, each application can be carried out in $\mathcal{O}(n^2)$ time and together with the observation that Reduction Rule 4.2.1 can be applied at most n times, the claim is proven. □

Reduction Rule 4.2.2. *Remove all connected components that are tricliques from the graph.*

Recall that we have seen that it is safe to remove any connected component that is an ℓ-clique in Section 3.5. Thus, the correctness of Reduction Rule 4.2.2 follows immediately.

Using these two data reduction rules, we can present another kernelization algorithm for 3-P 3-U Cluster Editing, leading to a quadratic-vertex kernel as well.

Theorem 4.10. *An instance which is reduced with respect to Reduction Rules 4.2.1 and 4.2.2 has $\mathcal{O}(k^2)$ vertices.*

Proof. Let G be a 3-partite 3-uniform hypergraph which is reduced with respect to Reduction Rules 4.2.1 and 4.2.2. Further, let S be a tricluster editing set with $|S| \leq k$ and let $\widetilde{G}$ be the resulting tricluster graph after applying the edge modifications in S to G. We partition the vertices in G into two sets: P containing the endpoints of the edges in S, and Q the rest. Clearly, $|P| \leq 3k$. It remains to upper-bound $|Q|$. Suppose $\widetilde{G}$ consists of f triclusters, $T_1, \ldots, T_f$. First, note that $f \leq 3k$ holds since every connected component needs at least one modification due to Reduction Rule 4.2.2 and every modification increases the number of connected components by at most two. It is easy to see that the unaffected vertices from one element of the partition in T_i must form a critical independent set in G.

We can now upper-bound the number of these vertices by a quadratic function in k. Let X_i' be a critical independent set in T_i and the sets $X_i'', Y_i', Y_i'', Z_i', Z_i''$ defined as in Reduction Rule 4.2.1 accordingly. After exhaustive application we know that $|X_i'| \leq \max\{|X_i'' \times Y_i''|, |X_i'' \times Z_i''|\}$ holds. Clearly, all vertices in X_i'' are in P: some of them are in T_i after gaining some edges between them and the vertices in $N(X_i')$ and the others are in other tricliques after losing all edges between them and $N(X_i')$. Moreover, all vertices of Y_i'' and Z_i'' are in P as we have seen above that in an optimal solution some of their incident edges are deleted. Hence, summing up the critical independent sets of X over all tricliques yields

$$\sum_{i=1}^{f} |X_i'| \leq \sum_{i=1}^{f} \max\{|X_i'' \times Y_i''|, |X_i'' \times Z_i''|\} \leq \sum_{i=1}^{f} \left(|X_i''| \cdot \max_{i=1,\ldots,f} \{|Y_i''|, |Z_i''|\} \right)$$
$$\leq \max_{i=1,\ldots,f} \{|Y_i''|, |Z_i''|\} \cdot \sum_{i=1}^{f} |X_i''| \leq k \cdot \sum_{i=1}^{f} |X_i''| \leq k \cdot 2k = 2k^2.$$

The argument for the last inequality can be verified by a counting argument: every vertex in X_i'' needs at least one modification. However, the X_i''s do not need to be pairwise disjoint, see Figure 4.4. Nevertheless, we observe that if a vertex is contained in r different X_i''s, then this vertex also needs at least $r-1$ modifications. As a modification can only affect one vertex of $\bigcup_{i=1}^{f} X_i''$, the inequality follows.

Together with the observation that the vertices in $T_i \cap Q$ form at most three critical independent sets in G, we may derive the upper bound

$$|Q| \leq 3 \cdot 2k^2 = 6k^2.$$

Summing up the number of vertices in P and Q yields that the reduced instance has at most $6k^2 + 3k$ vertices. □

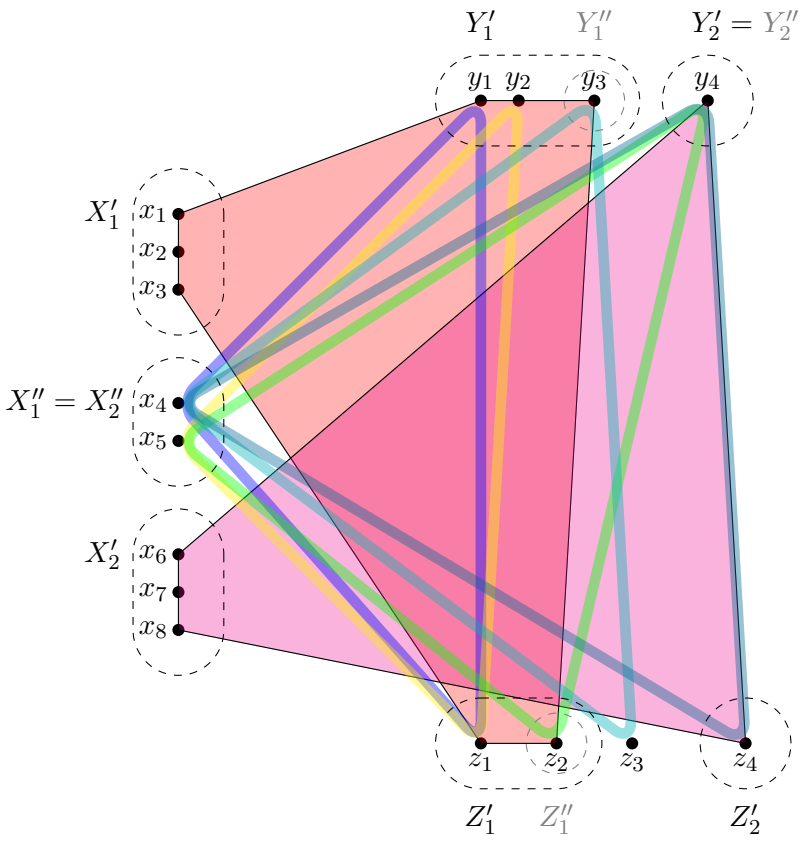


Figure 4.4: An extension of the graph in Figure 4.3. We depict two critical independent sets X_1' and X_2' with equal second neighborhood within X, namely $X_1'' = X_2''$. Again, we indicate a 3-dimensional hyperedge only by the shape of a triangle and a triclique by a reddish, filled polygon.

Originally, we hoped to produce a linear-vertex kernel for 3-P 3-U Cluster Editing with this technique as this was possible for Cluster Editing and Bicluster Editing. However, we were only able to build a quadratic kernel and we doubt that it is possible to reduce instances to a linear kernel based on Reduction Rule 4.2.1. In the next subsection, we explain further why we have these doubts.

To conclude, we would like to compare the two presented kernelization techniques that both produce a quadratic-vertex kernel for 3-P 3-U Cluster Editing. First, we want to point out that the data reduction rules used in the first kernelization depend on the parameter k whereas the ones used in this section do not. Clearly, it is more beneficial to have reduction rules independent of k to avoid recalculations. On the other hand, the kernelization from Section 3.5 produces a quadratic-vertex kernel for arbitrary $\ell \geq 3$ whereas we conjecture that the one above only achieves a kernel of $\Theta(k^{\ell-1})$ vertices. Last, we want to remark that the shown upper bounds on the number of vertices in the kernel do not differ much for $\ell = 3$: we achieved a kernel of at most $8k^2 + 5k$ vertices in Section 3.5, and a kernel of at most $6k^2 + 3k$ vertices in this section.

4.2.2 On a Linear-Vertex Kernel

As mentioned above, we aimed at producing a linear-vertex kernel with the preceding approach as this proved successful for Cluster Editing and Bicluster Editing. For the former one, further algorithms were suggested that produce a linear-vertex kernel, see Cao and Chen [CC12] and Fellows et al. [Fel+07]. They are all based on the following idea: let S be a cluster editing set for G with $|S| \leq k$ and let $\widetilde{G}$ be the resulting cluster graph after applying the edge modifications in S to G. The vertices in G are partitioned into two sets: P containing the endpoints of the edges in S, and Q the rest. As $|P| \leq 3k$, it remains to upper-bound $|Q|$ by a linear function in k which was successful with the help of data reduction rules.

These data reduction rules are applicable if cutting off specific subgraphs is somehow cheap and these subgraphs are chosen such that they contain vertices from Q. If cutting off a subgraph is cheap, then we can prove that there cannot be too many vertices whose edges get cut or, differently speaking, there cannot be too many vertices from P against vertices from Q in it. And conversely: if we cannot cut off the subgraph, then we can conclude that it has too few vertices from Q as a function of $|P|$. Thus, after exhaustive application of the data reduction rules, $|Q|$ could be upper-bounded by a linear function in $|P|$ or directly in k.

At this point we try to explain why we failed at transferring these approaches to 3-P 3-U Cluster Editing. When trying to adapt the approaches presented by Fellows et al. [Fel+07] and Guo et al. [Guo+08], we were confronted with the fact that, broadly speaking, cutting off a subgraph can be more expensive in 3-dimensional hypergraphs since it might need a cubic—instead of quadratic as in ordinary graphs—number of edges. Let us consider the kernelization in Section 4.2.1: we know that cutting off a vertex v from its neighborhood might cost up to $f(v) \in \Theta(|N(v)|^2)$ deletions while cutting off a vertex v from its neighborhood in ordinary graphs only costs $|N(v)|$ deletions. Therefore, we could only prove that it is safe to delete vertices from a critical independent set U' if $f(v) < |U'|$. Thus, we were not able to upper-bound the number of vertices in $U' \subseteq Q$ by a linear function in the cardinality of $N(v) \subseteq X$, but only by its square which is why we have a quadratic-vertex kernel.

Interestingly, we had intricacies of different nature when trying to adapt the method of Cao and Chen [CC12]. Their data reduction rule leading to a linear-vertex kernel relies on the observation that if for some vertex $v \in Q$ the cost of the minimum cut of $N[v]$, that is $\lambda(N[v])$, is somehow small, then $|N[v]| \in \mathcal{O}(\lambda(N[v]))$. Unfortunately, this property does not hold in multipartite graphs: in Figure 4.5 we observe that $\lambda(N[x_2]) = 1$ holds, but the number of vertices in $N[x_2] \cap Y$ does not need to be bounded.

We suspect that the obstacles we described above also arise when trying to produce a linear-vertex kernel for ℓ-P ℓ-U Cluster Editing for $\ell > 3$ by using the kernelization techniques that yield a linear-vertex kernel for Cluster Editing. Moreover, we would not be surprised if an adaption of Guo et al. [Guo+08] does not even yield a quadratic-vertex kernel—as it does for 3-P 3-U Cluster Editing—since the potential number of edges increases for larger ℓ and therefore, also the potential cost of a cut does.

To conclude, we gather some thoughts on the question whether a linear-vertex kernel exists for ℓ-P ℓ-U Cluster Editing. To this end, we introduce the well-known problem d-Hitting Set.

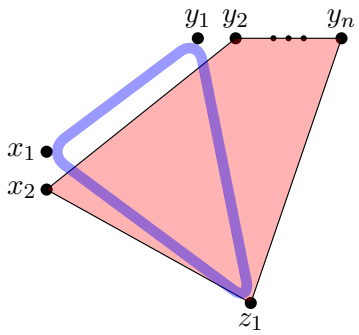


Figure 4.5: The above instance of 3-P 3-U Cluster Editing induces a triclique among the vertices $x_2, y_2, \ldots, y_n, z_1$ and has in addition the edge $x_1y_1z_1$. Observe that there is exactly one optimal solution, that is, deleting the edge $x_1y_1z_1$. Therefore, x_2 is not incident to a modification, i.e. $x_2 \in Q$. But a minimum cut of $N[x_2]$ has cost one whereas $|N[x_2] \cap Y| = n - 1$ and n can be any integer. Thus, $|N[x_2]| \notin \mathcal{O}(\lambda(N[x_2]))$.

Problem 9: 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 $S' \subseteq S$ such that S' contains at least one element from each subset in $\mathcal{C}$ and is of cardinality at most k?

At this time we do not have a conjecture whether a linear-vertex kernel even exists for ℓ-P ℓ-U Cluster Editing. It is also still unknown whether a linear-vertex kernel exists for 3-Hitting Set, see Theorem 15.29 by Cygan et al. [Cyg+15]. One could think of conflicts of ℓ-P ℓ-U Cluster Editing as sets of cardinality three that need to be 'hit' as a conflict consists of exactly three (non-)edges. An open question is whether there is a reduction from 3-P 3-U Cluster Editing to 3-Hitting Set that is suitable to obtain lower bound results on the vertex kernel size.

Chapter 5

Bicluster Editing

Recall that we excluded the case $\ell = 2$ in Chapter 3 since the characterization by forbidden subgraphs introduced therein does not hold for $\ell = 2$. Thus, in this chapter, we have a closer look at this special case which is known as Bicluster Editing.

Problem 10: Bicluster Editing
Input: A bipartite $G = \big(V = V_1 \dot\cup V_2, E \subseteq V_1 \times V_2\big)$ and an integer $k \in \mathbb{N}$.
Question: Is there a set $S \subseteq V_1 \times V_2$ of vertex tuples with $|S| \leq k$ to delete or add such that the resulting graph is a bicluster graph, that is, each connected component is a biclique?

Unlike ℓ-P ℓ-U Cluster Editing for $\ell \geq 3$, the problem Bicluster Editing has been studied from a parameterized perspective before. Amit [Ami04] showed its NP-hardness and gave an ILP formulation. By observing that graphs consisting of disjoint bicliques coincide with bipartite graphs that are P_4-free, Protti, Silva, and Szwarcfiter [PSS06] first devised a simple algorithm, of running time $4^k \cdot n^{\mathcal{O}(1)}$, that finds a P_4, and branches over the four possible ways to remove it. Furthermore, there have also been several kernelization results for Bicluster Editing among which the best is a $5k$ vertex kernel [Laf20].

In this chapter, we discuss the difference between $\ell = 2$ and $\ell \geq 3$ for a deeper insight into why it is indeed reasonable to study this problem separately. Then, we study approaches from Chapter 3 for $\ell = 2$ that have not been studied before for Bicluster Editing to the best of our knowledge. We adapt the merge branching and try to show that Bicluster Editing is fixed-parameter tractable above a lower bound obtained by a set of vertex-disjoint P_4s. However, we could again not transfer the proof from Cluster Editing, surprisingly due to different reasons than in Section 3.6.

Difference between $\ell = 2$ and $\ell \geq 3$. Most of the results from Chapter 3 are built on the characterization by forbidden subgraphs. Thus, let us restate the theorem.

Theorem 3.5. *An ℓ-partite ℓ-uniform hypergraph $G = \big(V = V_1 \dot\cup \ldots \dot\cup V_\ell, E \subseteq V_1 \times \cdots \times V_\ell\big)$ with $\ell \geq 3$ is an ℓ-cluster graph if and only if it contains no non-edge $a_1 \ldots a_\ell$ for which there are two edges $b_1 \ldots b_\ell$ and $c_1 \ldots c_\ell$ such that $\{a_1, \ldots, a_\ell\} \subseteq \{b_1, c_1, \ldots, b_\ell, c_\ell\}$ and $\{b_1, \ldots, b_\ell\} \cap \{c_1, \ldots, c_\ell\} \neq \emptyset$.*

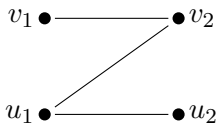


Figure 5.1: An illustration of a P_4 among the vertices v_1, u_1, v_2, u_2.

We first observe that Theorem 3.5 is not true for $\ell = 2$. There is no way that a missing edge a_1a_2 is covered by two edges b_1b_2 and c_1c_2 such that $b_1 = c_1$ or $b_2 = c_2$ holds. Moreover, the weaker statement of Lemma 3.6 neither holds for $\ell = 2$ as the vertices of a non-edge in an ordinary bipartite graph—meaning that they are not from the same element of the partition—cannot have distance one from each other. In contrast, it is possible that two vertices that are contained in a non-edge in a hypergraph have distance one from each other as they might be contained in an edge at the same time.

In bipartite graphs, the vertices of a non-edge have distance at least three. Therefore, we need four (non-)edges to characterize the forbidden subgraph for Bicluster Editing whereas we only need three (non-)edges for ℓ-P ℓ-U Cluster Editing. This also explains why the the trivial branching yields a better branching factor for $\ell \geq 3$ than it does for $\ell = 2$, which seems surprising at first glance.

Last, recall that we proved NP-hardness for ℓ-P ℓ-U Cluster Editing in Section 3.1. We observe that this is the only section from Chapter 3 not relying on Theorem 3.5. Nevertheless, we have seen that not even in this section we may omit the assumption $\ell \geq 3$ as we reduce from ℓ-Partite Perfect Matching which is solvable in polynomial time for $\ell = 2$, that is, the matching problem on bipartite graphs.

5.1 Merge Branching

We adapt the merge branching algorithm for Weighted Cluster Editing by Böcker et al. [Böc+09]. Like in Section 3.4.2 for ℓ-P ℓ-U Cluster Editing, we have to introduce the weighted version of Bicluster Editing.

Problem 11: Weighted Bicluster Editing

Input: A bipartite graph $G = \big(V = V_1 \dot\cup V_2, E \subseteq V_1 \times V_2\big)$, a cost function $w : V_1 \times V_2 \to \mathbb{N}$ and an integer $k \in \mathbb{N}$.

Question: Is there a set $S \subseteq V_1 \times V_2$ of vertex tuples to delete or add with $\sum_{e \in S} w(e) \leq k$ such that the resulting graph is a bicluster graph, that is, each connected component is a biclique?

It is folklore that a bipartite graph is a bicluster graph if and only if it has no P_4 as an induced subgraph [Guo+08; Laf20; PSS06]. Thus, a straightforward adaption of the merge branching aims at resolving a P_4 in each step. Similarly to ℓ-P ℓ-U Cluster Editing, we note that if we merge an edge in a bipartite graph, then the resulting graph is in general not bipartite anymore and thus, we merge again two vertices of the same V_i. To this end, we use the merge operation and the modified cost w^* from Definition 3.13. Like in Chapter 3, the index of a vertex indicates from which element of the partition it is, that is, $v_i \in V_i$ for $i \in \{1, 2\}$.

Figure 5.2: Both possibilities how an induced P_4 among the vertices v_1, u_1, v_2, u_2 can be extended to a fork by another vertex w.

Let the vertices v_1, u_1, v_2, u_2 induce a P_4 in G. Without loss of generality, we assume that v_1u_2 is the missing edge in the P_4, compare to Figure 5.1. To resolve it, we branch into three cases:

1. merge v_1, u_1;
2. delete the edge v_1v_2;
3. delete the edge u_1v_2.

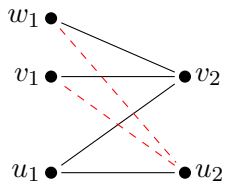
Lemma 5.1. *For an instance (G, w, k) of* Weighted Bicluster Editing *with cost function $w : V_1 \times V_2 \to \mathbb{N} \setminus \{0\}$, the above branching strategy is correct and yields a branching vector of at least $(\frac{1}{2}, 1, 1)$.*

Proof. In the first case, we force v_1 and u_1 to end up in the same biclique. The second and third one are complementing as they destroy the P_4 by preventing v_1 and u_1 from ending up in the same biclique. One can easily check that the above branching stops at some point with an optimal solution since we have a complete case distinction and resolve a P_4 in each step.

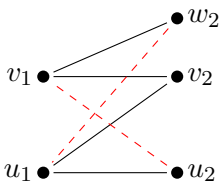
To analyze the size of the search tree, we adapt the proof of Theorem 4.4 in a straightforward way. When deleting an edge e, we decrease the parameter k by $w(e)$. Recall that we interpret zero-edges—that result from merging—as non-edges. This is why k is decreased by at least one in the second and third branching case. Recall as well that, when merging two vertices v_1 and u_1, we join for all vertices $x_2 \in V_2$ the pairs v_1x_2, u_1x_2 into a single pair with modified cost $w^*(v_1x_2) + w^*(u_1x_2)$. Applying the bookkeeping trick—compare to Section 3.4.2—yields a branching vector of at least $(\frac{1}{2}, 1, 1)$. Hence, the merge branching solves Weighted Bicluster Editing in $4^k \cdot |V|^{\mathcal{O}(1)}$ time. □

To improve this branching vector, we choose a somehow 'good' P_4 to branch on. In this way, Böcker et al. [Böc+09] improved the branching vector of the merge branching for Weighted Cluster Editing from $(\frac{1}{2}, 1)$ to $(1, 1)$, and we improved the branching vector for Weighted 3-P 3-U Cluster Editing from $(\frac{1}{2}, 2, 1)$ to $(1, 1, 1)$ in Section 4.1.1.

Definition 5.2. We say that a bipartite graph G has a *fork* if there are five vertices v_1, u_1, v_2, u_2, w such that the only present edges among those vertices are v_1v_2, u_1v_2, u_1u_2 and either wv_2 or u_1w, see Figure 5.2. We call a graph *fork-free* if it has no such induced subgraph.



(a) When merging the vertices v_2, u_2, the edge w_1v_2 is joined with the non-edge w_1u_2, and the edge u_1u_2 is joined with the non-edge v_1u_2.



(b) When merging the vertices v_1, u_1, the edge v_1w_2 is joined with the non-edge u_1w_2, and the edge v_1v_2 is joined with the non-edge v_1u_2.

Figure 5.3: We indicate edges by black, solid lines and non-edges by red, dashed lines. Thus, in (a) we see a fork and in (b) a P_5.

We improve the running time of the merge branching by applying a refined branching strategy on larger induced subgraphs that contain a P_4, namely a fork or a P_5. Thus, we have to study under which circumstances we can guarantee that a bipartite graph contains a fork or a P_5.

Lemma 5.3. *Given a connected, bipartite graph G. If G is fork-free and P_5-free, then G is a biclique or a biclique minus a single edge.*

The correctness of Lemma 5.3 has been proven by Guo et al. [Guo+08] by showing that a connected, bipartite graphs that is fork-free and P_5-free is missing only one edge. Recall that we showed a similar statement for 3-partite 3-uniform hypergraphs in Section 4.1.1.

Lemma 5.4. *Given an instance (G, w, k) of* Weighted Bicluster Editing *with cost function $w : V_1 \times V_2 \to \mathbb{N} \setminus \{0\}$. The merge branching strategy that merges two vertices with minimum branching factor has a branching vector of at least $(1, 1, 1)$.*

Proof. Without loss of generality, we assume that the pair $v_1, u_1 \in V_1$ has minimum branching factor. Let δ be the cost of merging v_1 and u_1. If $\delta \geq 1$, then we are done. Hence, assume $\delta < 1$ in the following. Recall that creating a zero-edge as well as joining a zero-edge reduces k by $\frac{1}{2}$. Thus, when merging v_1, u_1—or any other two vertices—at most one zero-edge can be created or joined. However, this implies that the graph is P_5-free and fork-free. Otherwise, we could choose two vertices in the P_5 resp. fork such that twice an edge is joined with a non-edge when merging them. Clearly, this would result in $\delta \geq 1$, compare to Figure 5.3.

By Lemma 5.3, we know that the connected component containing v_1, u_1 must be a biclique minus a single edge when considering the underlying unweighted graph. As we consider zero-edges as non-edges, we are done if the missing edge is a zero-edge: we can insert the single missing edge without paying any cost. Thus, assume there are two non-adjacent vertices x_1, x_2 and inserting the edge has non-zero cost. We now show that, in this special case, we can omit the bookkeeping trick. We observe that x_1, x_2 form together with any two vertices, say v_1 and v_2, a P_4. We distinguish the cases $w(v_1x_2) \geq w(x_1x_2)$ and $w(v_1x_2) < w(v_1x_2)$. If $w(v_1x_2) \geq w(x_1x_2)$, then the joined edge has cost greater than zero, i.e. the resulting connected component is a biclique. Thus we can reduce the

parameter k by $\min\{w(v_1x_2), w(x_1x_2)\} \geq 1$. In case $w(v_1x_2) < w(x_1x_2)$ holds, the joined pair has weight strictly smaller than zero. Hence, we have not generated a zero-edge and can reduce k by $\min\{w(v_1x_2), w(x_1x_2)\} \geq 1$, again implying that the branching vector is at least $(1, 1, 1)$ as claimed. □

Theorem 5.5. *An instance (G, w, k) of* Weighted Bicluster Editing *with cost function $w : V_1 \times V_2 \to \mathbb{N} \setminus \{0\}$ can be solved in $3^k \cdot n^{\mathcal{O}(1)}$ time.*

To the best of our knowledge, the refined merge branching yields the currently smallest branching factor for Weighted Bicluster Editing. For Bicluster Editing, several branching algorithms were introduced that have a better branching factor, among which the best one is 2.636 due to Tsur [Tsu21]. However, this algorithm needs at least five case distinctions within the branching and due to the reasons we discussed in the end of Section 3.4.2, we think it is worth to consider the (refined) merge branching for the unweighted case in practice as well.

5.2 Parameterization Above Lower Bounds

In Section 3.6, we tried to show that ℓ-P ℓ-U Cluster Editing is fixed-parameter tractable above a lower bound obtained by a set of vertex-disjoint conflicts, following the approach of van Bevern, Froese, and Komusiewicz [vFK18] for Cluster Editing. Unfortunately, we could not transfer one of the key ingredients of the approach, a data reduction rule, as its natural generalization turned out to be incorrect for $\ell = 3$.

Thus, in this section, we discuss whether the approach of van Bevern, Froese, and Komusiewicz is adaptable for Bicluster Editing. We shall see that the data reduction rule is correct for $\ell = 2$, but it seems that it is not 'strong enough' instead to obtain fixed-parameter tractability.

We structure this section similarly to Section 3.6. First, let us state Reduction Rule Attempt 3.6.1 for $\ell = 2$. Consistent with previous notation, we denote by $\mathcal{H}$ a packing of vertex-disjoint P_4s throughout this section.

Reduction Rule 5.2.1. *Let G be a bipartite graph and $\mathcal{H}$ a packing of vertex-disjoint P_4s. If there is an $H \in \mathcal{H}$ having an optimal solution S such that*

- *no two vertices of H are together in a conflict in $G \triangle S$,*

then replace G by $G \triangle S$, delete H from $\mathcal{H}$, and decrease k by one.

Lemma 5.6. *Reduction Rule 5.2.1 is correct.*

Proof. Let $V(H) = \{v_1, u_1, v_2, u_2\}$ and $E(H) = \{v_1v_2, u_1v_2, u_1u_2\}$. We have four different options to resolve H. Since deleting v_1v_2 and u_1u_2 behave symmetrically, it is sufficient to restrict the proof to three cases.

First, assume $S = \{v_1u_2\}$, i.e. the missing edge is inserted. We now show that this implies that the connected component of H is a biclique in $G \triangle S$ and hence, assuming that this is proven, the correctness follows. Let w_1 be a neighbor of v_2 or u_2. As there is no conflict involving v_1 and u_1, the vertex w_1 must neighbor both, v_2 and u_2. Clearly,

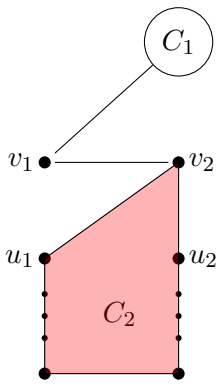


Figure 5.4: The figure outlines the structure of a bipartite graph for which Reduction Rule 5.2.1 deletes the edge v_1v_2. By C_1 we indicate an arbitrary graph that is only connected through the vertex v_1 to the rest of the graph. Again, we indicate by the red polygon that the bordered vertices induce a biclique.

the analogous observation holds for a neighbor w_2 of v_1 or u_1. Assume an edge w_1w_2 within $N(v_1, u_1, v_2, u_2)$ is missing. Then, the vertices v_1, w_1, v_2, w_2 induce a P_4. This yields a contradiction to the assumption that v_1, v_2 are not contained in any conflict in $G \triangle S$. Therefore, we may conclude that $N[v_1, v_2]$ is a biclique. Next, assume there is a vertex x_1 in the second, but not first, neighborhood of $V(H)$. Let $w_2 \in N[v_1, v_2]$ be its neighbor. But again, the two vertices $v_1, v_2 \in H$ are involved in a P_4, namely the one induced by the vertices v_1, x_1, v_2, w_2. Thus, such a vertex x_1 cannot exist, proving our claim.

Second, assume $S = \{u_1v_2\}$. Note that in the proof above, we neither used the vertices u_1 or u_2 to show that $N[v_1, v_2]$ is a biclique, nor to show that there is no vertex in the second, but first, neighborhood of v_1, v_2. Thus, it follows that after the deletion of the edge u_1v_2, the connected components of v_1, v_2 resp. u_1u_2 are both bicliques in $G \triangle S$.

Third, assume $S = \{v_1v_2\}$. Following the line of argument from the first two cases, we may first derive that the connected component of u_1, v_2, u_2 is a biclique, say C_2, in $G \triangle S$, see Figure 5.4. Assume there is an optimal solution S' for G such that $v_1v_2 \notin S'$. We construct an optimal solution S^* from S' such that $S \subseteq S^*$. As H needs to be solved, one of the other three (non)-edges has to be contained in S'. First, let $u_1v_2 \in S'$. Let B_1 be the biclique of v_1, v_2 and $B_2 = C_2 \setminus \{v_2\}$ be the biclique of u_1, u_2 in $G \triangle S'$. We construct S^* implicitly by removing v_2 from B_1 and putting it into B_2. This forces us to delete the edge v_1v_2, but in return we save at least one deletion for the edge u_1v_2. Note that we do not have additional insertions in S^* as C_2 is a biclique, and no further deletions as v_2's only neighbor outside C_2 is v_1. Hence, S^* is not larger than S'. Next, assume that $u_1u_2 \in S'$. Without loss of generality we may assume $v_1v_2, u_1v_2 \notin S'$. By a similar argumentation as the previous one, we can remove the vertices u_1, v_2 from their biclique and put them into the one of u_2. Thereby, we delete again the edge v_1v_2 and save at least one deletion, namely u_1u_2. Last, assume $v_1u_2 \in S'$. Thus, all vertices v_1, v_2, u_1, u_2 end up in the same biclique B. We build another solution by cutting off all vertices from B that do not lie in C_2. As the edge v_1v_2 in the only edge going out from C_2, the

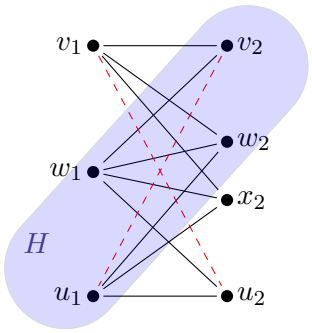


Figure 5.5: A bipartite graph that is reduced with respect to Reduction Rule 5.2.1 but does not fulfill the prerequisites of Lemma 5.7.

cutoff cost equals one. As we also save at least one insertion, namely the edge v_1u_2, the implicitly constructed solution S^* is again at least as good as S' is. We conclude that there is an optimal solution in which the edge v_1v_2 gets deleted and, hence, we have shown that Reduction Rule 5.2.1 is correct. □

As in Section 3.6, we would like to guarantee that an instance reduced with respect to Reduction Rule 5.2.1 satisfies the prerequisites of the following lemma.

Lemma 5.7. *Let G be an instance of* Bicluster Editing*, $\mathcal{H}$ a packing of vertex-disjoint P_4s and $r = k - |\mathcal{H}|$. Let S be a solution of size k that contains, for each $H \in \mathcal{H}$,*

1. *at least two modifications within H, or*
2. *at least one external vertex tuple for H.*

Then, $|\mathcal{H}| \leq 2r$ and thus, $k \leq 3r$.

Note that the above lemma is equivalent to Lemma 3.23 when fixing $\ell = 2$. Indeed, we did not make use of the assumption $\ell \geq 3$ in its proof. Thus, we refer to Section 3.6 for the proof of Lemma 5.7.

As hinted in the beginning of this section, we want to discuss why Reduction Rule 5.2.1 is not 'strong enough', that is, why an instance reduced with respect to Reduction Rule 5.2.1 does not need to satisfy the prerequisites of Lemma 5.7. To this end, consider Figure 5.5. It shows an instance of Bicluster Editing with a maximal packing $\mathcal{H} = \{H\}$. Observe that it is reduced with respect to Reduction Rule 5.2.1. We claim that $S = \{u_1v_2, v_1u_2\}$ is the only optimal solution: the only cuts of cost two exclude the single vertex v_1 resp. u_2 and hence, do not solve all conflicts. But H is neither solved suboptimally, nor is any vertex of H incident to more than one modification, yielding that the prerequisites of Lemma 5.7 are not fulfilled.

Conclusion. Finally, we discuss our observations regarding the differences between $\ell = 2$ and $\ell \geq 3$. To this end, recall that a forbidden subgraph of Bicluster Editing consists of exactly 2ℓ vertices compared to at most $2\ell - 1$ vertices for ℓ-P ℓ-U Cluster Editing.

Broadly speaking, one could say that Reduction Rule 5.2.1 is weaker for $\ell = 2$ as it is only applicable for a conflict H if no $\frac{1}{2} \cdot 2\ell$ vertices of H are contained in a conflict after the modification. In contrast, Reduction Rule Attempt 3.6.1 is already applicable for $\ell \geq 3$ if no $\ell > \frac{1}{2}(2\ell - 1)$ vertices of H are contained in a conflict afterwards. Due to this observation, we believe that one could fix Reduction Rule Attempt 3.6.1 by requiring that no $\ell - 1$ vertices of H are contained in a conflict after its application. However, be aware that even if strengthen the prerequisites of Reduction Rule Attempt 3.6.1 yielded its correctness, then we would still not see how to show that ℓ-P ℓ-U Cluster Editing is fixed-parameter tractable above a lower bound as we would have the same scenario as for Bicluster Editing. That is, we do not think that the prerequisites of Lemma 3.23 are met since a conflict containing $\ell - 1$ vertices of H could theoretically be solved by a modification within the other ℓ vertices.

Lemma 3.23. *Let G be an instance of ℓ-P ℓ-U Cluster Editing, $\mathcal{H}$ a packing of vertex-disjoint conflicts and $r = k - \sum_{H \in \mathcal{H}} \tau(H)$. Let S be a solution of size k that contains, for each $H \in \mathcal{H}$,*

1. *at least $\tau(H) + 1$ vertex tuples within H or*
2. *at least one external vertex tuple for H.*

Then, $|\mathcal{H}| \leq \ell \cdot r$ and thus, $k \leq (\ell \cdot 2^{\ell-2} + 1)r$.

In summary, as soon as we strengthen the prerequisites of the data reduction rule such that it becomes correct, we cannot guarantee that an instance that is reduced with respect to this data reduction rule has the desired structure. For all $\ell \geq 2$ this gap prevented us from showing fixed-parameter tractability with respect to the parameter $k - h_v$, where h_v is a lower bound given by a set of vertex-disjoint conflicts. At the moment, we have no conjecture whether Bicluster Editing or ℓ-P ℓ-U Cluster Editing is fixed-parameter tractable with respect to $k - h_v$ at all. In Chapter 6, we come back to this question to discuss future research opportunities.

Chapter 6

Conclusion

In this work, we introduced the problem ℓ-P ℓ-U Cluster Editing, motivated by an application in physics to detect radioactive sources, and started studying its algorithmic complexity. First, we showed NP-hardness for all $\ell \geq 3$, implying that it is indeed reasonable to come up with algorithms that do not find an optimal solution in polynomial time in general. Thus, we focused on a parameterized approach for ℓ-P ℓ-U Cluster Editing with respect to the solution size k.

We point out that all of our main contributions, see Table 1.1, hold for arbitrary $\ell \geq 3$. This is due to the structure of the forbidden subgraphs for ℓ-P ℓ-U Cluster Editing. First, the number of vertices involved in a forbidden subgraph increases only linearly in ℓ, which allowed us to derive a kernelization algorithm that produces a quadratic-vertex kernel for ℓ-P ℓ-U Cluster Editing for all $\ell \geq 3$. Second, the number of edges plus non-edges needed for the characterization is three and, thus, a constant. This enabled us to deduce for both, the trivial and the merge branching, branching factors not depending on ℓ as long as $\ell \geq 3$ holds. Hence, in the running times of the branching algorithms 'only' the polynomial-time factors increase for increasing ℓ.

In contrast, we had to exclude the case $\ell = 2$ to make the proofs in Chapter 3 work. In Chapter 5 we discussed why we have to treat this special case separately. In short, this was mostly due to the observation that the forbidden induced subgraph for $\ell = 2$ has a different structure when compared to the ones for $\ell \geq 3$. Nevertheless, the algorithmic results do not seem to differ that much so far and therefore, future research concerning possible algorithmic differences would be interesting.

Future Research. We already addressed two questions that were left open in this work. In the following, we restate them and suggest how one could approach answering them. First, we discussed what prevented us from obtaining a linear-vertex kernel for 3-P 3-U Cluster Editing. We encountered two obstacles, remarkably with different approaches, when trying to adapt kernelizations from Cluster Editing that produce a linear-vertex kernel. In short, the first obstacle is due to the fact that we consider hypergraphs, and the second one is due to the multipartite structure. Weak-cross compositions, see Chapter 15.3 by Cygan et al. [Cyg+15], have been successfully applied to prove that d-Hitting Set and ℓ-Partite Perfect Matching do not admit a sublinear-vertex kernel. By our intuition, these two problems have a somehow similar

structure as ℓ-P ℓ-U Cluster Editing. Therefore, we think that future research could use this tool to obtain a similar result for ℓ-P ℓ-U Cluster Editing or look for a suitable reduction from one of the two problems to achieve a lower bound on the size of the vertex kernel. However, following this path we do not expect an answer to the question of a linear-vertex kernel.

Second, we left open whether ℓ-P ℓ-U Cluster Editing is for any $\ell \geq 2$ fixed-parameter tractable with the parameter $k - h_v$, where h_v is a lower bound obtained by a set of vertex-disjoint conflicts. We do not have a strong conjecture whether this is possible, but we tend to approach this question from the other direction, that is, aiming at showing that ℓ-P ℓ-U Cluster Editing combined with the parameter $k - h_v$ is W[1]-hard, i.e. presumably not in FPT. [1]

We want to conclude by giving an outlook for further potential future research.

First, refining the merge branching for Cluster Editing, Bicluster Editing, and 3-P 3-U Cluster Editing by choosing a 'beneficial' conflict to branch on enabled us to improve its branching vector. Thus, we believe that an improved branching vector can be obtained by this technique for general $\ell \geq 3$.

Other parameters. We point out that only few parameters from the parameterized landscape have been studied for Bicluster Editing as well as for ℓ-P ℓ-U Cluster Editing so far. On the one hand, we know that Cluster Editing is NP-hard for graphs with maximum degree six as well as if the resulting cluster graph may consist of at most two cliques. By our intuition, neither Bicluster Editing nor ℓ-P ℓ-U Cluster Editing is 'simpler' than Cluster Editing and, hence, we think that similar results can be obtained for both problems. Recall that we discussed in Section 1.1 that NP-hardness for Bicluster Editing, when restricted to graphs with maximum degree three, has not been shown yet, although stated differently in the literature.

Komusiewicz and Uhlmann [KU11] showed that Cluster Editing is fixed-parameter tractable with respect to the parameter 'cluster vertex deletion number', a usually much smaller parameter than k. We think it might be worth to study whether their corresponding algorithm is adaptable to Bicluster Editing or ℓ-P ℓ-U Cluster Editing, in particular since its basic idea coincides with a previously discussed one, namely of parameterizing above lower bounds: showing fixed-parameter tractability for parameters smaller than k.

Implementation. We presented the merge branching algorithm for all $\ell \geq 2$. As discussed in the according sections, it also seems promising for practical applications. Thus, we are interested in an implementation, and curious whether it proves efficient in practice, in particular for the data generated by the above-mentioned application in detecting radioactive sources. To this end, we suggest to compare its performance with that of established solvers. As searching for a conflict is more time-consuming for Bicluster Editing and ℓ-P ℓ-U Cluster Editing than it is for Cluster Editing, we need to be aware of the resulting polynomial factor in the running time and, hence, it seems natural to think about improvements in theory and practical speedups when implementing.

[1] For a proper definition of the complexity class W[1], see Downey and Fellows [DF13].

Literature

[ACN08] Nir Ailon, Moses Charikar, and Alantha Newman. *Aggregating inconsistent information: ranking and clustering*. In: *Journal of the ACM (JACM)* 55.5 (2008), pp. 1–27 (cit. on p. 9).

[Ami04] Noga Amit. *The Bicluster Graph Editing Problem*. Tel Aviv University, 2004 (cit. on pp. 13, 20, 21, 23, 51).

[BBC04] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. *Correlation clustering*. In: *Machine Learning* 56.1 (2004), pp. 89–113 (cit. on p. 11).

[BBK11] Sebastian Böcker, Sebastian Briesemeister, and Gunnar W Klau. *Exact algorithms for cluster editing: Evaluation and experiments*. In: *Algorithmica* 60.2 (2011), pp. 316–334 (cit. on pp. 13, 41).

[Bie+93] Daniel Bienstock, Michel Goemans, David Simchi-Levi, and David Williamson. *Note on the prize collecting traveling salesman problem*. In: *Math. Program.* 59 (1993), pp. 413–420 (cit. on p. 43).

[Blo+08] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. *Fast unfolding of communities in large networks*. In: *Journal of Statistical Mechanics: Theory and Experiment* 2008.10 (2008), P10008 (cit. on p. 9).

[Böc+09] Sebastian Böcker, Sebastian Briesemeister, Quang Bui, and Anke Truß. *Going weighted: Parameterized algorithms for cluster editing*. In: *Theoretical Computer Science* 410.52 (2009), pp. 5467–5480 (cit. on pp. 12, 14, 28, 29, 52, 53).

[Böc12] Sebastian Böcker. *A golden ratio parameterized algorithm for Cluster Editing*. In: *Journal of Discrete Algorithms* 16 (2012), pp. 79–89 (cit. on pp. 12, 13).

[Cai96] Leizhen Cai. *Fixed-parameter tractability of graph modification problems for hereditary properties*. In: *Information Processing Letters* 58.4 (1996), pp. 171–176 (cit. on p. 19).

[CC12] Yixin Cao and Jianer Chen. *Cluster editing: Kernelization based on edge cuts*. In: *Algorithmica* 64.1 (2012), pp. 152–169 (cit. on pp. 12, 13, 44, 49).

[CGW05] Moses Charikar, Venkatesan Guruswami, and Anthony Wirth. *Clustering with qualitative information*. In: *Journal of Computer and System Sciences* 71.3 (2005), pp. 360–383 (cit. on p. 9).

[CKP18] Oliver Cooley, Mihyun Kang, and Yury Person. *Largest components in random hypergraphs*. In: *Combinatorics, Probability and Computing* 27.5 (2018), pp. 741–762 (cit. on p. 31).

[CM12] Jianer Chen and Jie Meng. *A 2k kernel for the cluster editing problem*. In: *Journal of Computer and System Sciences* 78.1 (2012), pp. 211–220 (cit. on pp. 12, 13, 44).

[Cyg+13] Marek Cygan, Marcin Pilipczuk, Michał Pilipczuk, and Jakub Onufry Wojtaszczyk. *On multiway cut parameterized above lower bounds*. In: *ACM Transactions on Computation Theory (TOCT)* 5.1 (2013), pp. 1–11 (cit. on pp. 12, 16).

[Cyg+15] Marek Cygan, Fedor V Fomin, Łukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michał Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015 (cit. on pp. 50, 59).

[Deh+06] Frank Dehne, Michael A Langston, Xuemei Luo, Sylvain Pitre, Peter Shaw, and Yun Zhang. *The cluster editing problem: Implementations and experiments*. In: *International Workshop on Parameterized and Exact Computation*. Springer. 2006, pp. 13–24 (cit. on p. 13).

[DF13] Rodney G Downey and Michael R Fellows. *Fundamentals of Parameterized Complexity*. Vol. 4. Springer, 2013 (cit. on pp. 16, 60).

[Die17] Reinhard Diestel. *Graph Theory*. Vol. 5. Springer, 2017 (cit. on p. 15).

[Dra+15] Pål Grønås Drange, Felix Reidl, Fernando Sánchez Villaamil, and Somnath Sikdar. *Fast Biclustering by Dual Parameterization*. 2015. arXiv: 1507.08158 [cs.DS] (cit. on p. 13).

[Edm65] Jack Edmonds. *Paths, Trees, and Flowers*. In: *Canadian Journal of Mathematics* 17 (1965), pp. 449–467 (cit. on p. 23).

[Fel+07] Michael Fellows, Michael Langston, Frances Rosamond, and Peter Shaw. *Efficient Parameterized Preprocessing for Cluster Editing*. In: *Fundamentals of Computation Theory*. Springer, 2007, pp. 312–321 (cit. on pp. 12, 44, 49).

[GP16] Shivam Garg and Geevarghese Philip. *Raising the bar for vertex cover: Fixed-parameter tractability above a higher guarantee*. In: *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM. 2016, pp. 1152–1166 (cit. on p. 12).

[Gra+04] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. *Automated Generation of Search Tree Algorithms for Hard Graph Modification Problems*. In: *Algorithmica* 39 (2004), pp. 321–347 (cit. on p. 12).

[Gra+05] Jens Gramm, Jiong Guo, Falk Hüffner, and Rolf Niedermeier. *Graph-Modeled Data Clustering: Exact Algorithms for Clique Generation*. In: *Theory Comput. Syst.* 38 (2005), pp. 373–392 (cit. on pp. 9, 12, 27, 30, 32, 33).

[Guo+08] Jiong Guo, Falk Hüffner, Christian Komusiewicz, and Yong Zhang. *Improved Algorithms for Bicluster Editing*. In: *Theory and Applications of Models of Computation*. Springer, 2008, pp. 445–456 (cit. on pp. 13, 45, 49, 52, 54).

[Guo09] Jiong Guo. *A more effective linear kernelization for cluster editing*. In: *Theoretical Computer Science* 410.8 (2009), pp. 718–726 (cit. on pp. 12–14, 44, 45).

[GW89] Martin Grötschel and Yoshiko Wakabayashi. *A cutting plane algorithm for a clustering problem*. In: *Math. Program.* 45 (1989), pp. 59–96 (cit. on pp. 13, 26).

[Kar72] Richard M Karp. *Reducibility among combinatorial problems*. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103 (cit. on p. 43).

[Kel+21] Leon Kellerhals, Tomohiro Koana, André Nichterlein, and Philipp Zschoche. *The PACE 2021 Parameterized Algorithms and Computational Experiments Challenge: Cluster Editing*. In: *16th International Symposium on Parameterized and Exact Computation (IPEC 2021)*. Vol. 214. 2021, 26:1–26:18 (cit. on pp. 13, 30).

[KM86] Mirko Křivánek and Jaroslav Morávek. *NP-hard problems in hierarchical-tree clustering*. In: *Acta Informatica* 23.3 (1986), pp. 311–323 (cit. on pp. 9, 11, 13).

[KN11] Brian Karrer and Mark EJ Newman. *Stochastic blockmodels and community structure in networks*. In: *Physical Review E* 83.1 (2011), p. 016107 (cit. on p. 9).

[KU11] Christian Komusiewicz and Johannes Uhlmann. *Alternative Parameterizations for Cluster Editing*. In: *SOFSEM 2011: Theory and Practice of Computer Science*. Springer, 2011, pp. 344–355 (cit. on pp. 12, 60).

[Laf20] Manuel Lafond. *Even better fixed-parameter algorithms for bicluster editing*. In: *International Computing and Combinatorics Conference*. Springer. 2020, pp. 578–590 (cit. on pp. 13, 20, 45, 51, 52).

[Law73] Eugene L Lawler. *Cutsets and partitions of hypergraphs*. In: *Networks* 3.3 (1973), pp. 275–285 (cit. on p. 42).

[Lok+14] Daniel Lokshtanov, NS Narayanaswamy, Venkatesh Raman, MS Ramanujan, and Saket Saurabh. *Faster parameterized algorithms using linear programming*. In: *ACM Transactions on Algorithms (TALG)* 11.2 (2014), pp. 1–31 (cit. on p. 12).

[LPS21] Shaohua Li, Marcin Pilipczuk, and Manuel Sorge. *Cluster Editing Parameterized Above Modification-Disjoint P_3-Packings*. In: *38th International Symposium on Theoretical Aspects of Computer Science (STACS 2021)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. 2021 (cit. on pp. 9, 12).

[LZZ12] Hong Liu, Peng Zhang, and Daming Zhu. *On editing graphs into 2-club clusters*. In: *Frontiers in Algorithmics and Algorithmic Aspects in Information and Management*. Springer, 2012, pp. 235–246 (cit. on p. 9).

[MR99] Meena Mahajan and Venkatesh Raman. *Parameterizing above Guaranteed Values: MaxSat and MaxCut*. In: *Journal of Algorithms* 31.2 (1999), pp. 335–354 (cit. on p. 12).

[NG04] Mark EJ Newman and Michelle Girvan. *Finding and evaluating community structure in networks*. In: *Physical Review E* 69.2 (2004), p. 026113 (cit. on p. 9).

[Nie06] Rolf Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford, 2006 (cit. on p. 16).

[NO09] Nicolas Neubauer and Klaus Obermayer. *Towards community detection in k-partite k-uniform hypergraphs*. In: *Proceedings of the NIPS 2009 Workshop on Analyzing Networks and Learning with Graphs*. 2009, pp. 1–9 (cit. on p. 11).

[NO10] Nicolas Neubauer and Klaus Obermayer. *Community detection in tagging-induced hypergraphs*. In: *Workshop on Information in Networks*. New York University NY, USA. 2010, pp. 24–25 (cit. on p. 11).

[PS98] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial Optimization: Algorithms and Complexity*. Courier Corporation, 1998 (cit. on p. 20).

[PSS06] Fábio Protti, Maise Dantas da Silva, and Jayme Luiz Szwarcfiter. *Applying modular decomposition to parameterized bicluster editing*. In: *International Workshop on Parameterized and Exact Computation*. Springer. 2006, pp. 1–12 (cit. on pp. 13, 51, 52).

[RB08] Martin Rosvall and Carl T Bergstrom. *Maps of random walks on complex networks reveal community structure*. In: *Proceedings of the National Academy of Sciences* 105.4 (2008), pp. 1118–1123 (cit. on p. 9).

[Sch02] Florian Schopper. *Entwicklung eines Teleskops zur Abbildung von Gammastrahlung mittels Comptonstoß und Paarerzeugung*. Dissertation. München: Technische Universität München, 2002 (cit. on pp. 10, 11).

[Sch98] Alexander Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998 (cit. on p. 26).

[SGB14] Peng Sun, Jiong Guo, and Jan Baumbach. *Complexity of dense bicluster editing problems*. In: *International Computing and Combinatorics Conference*. Springer. 2014, pp. 154–165 (cit. on p. 13).

[SST04] Ron Shamir, Roded Sharan, and Dekel Tsur. *Cluster graph modification problems*. In: *Discrete Applied Mathematics* 144.1-2 (2004), pp. 173–182 (cit. on pp. 11, 12).

[Tsu21] Dekel Tsur. *Faster parameterized algorithm for Bicluster Editing*. In: *Information Processing Letters* 168 (2021), p. 106095 (cit. on pp. 13, 55).

[vFK18] René van Bevern, Vincent Froese, and Christian Komusiewicz. *Parameterizing edge modification problems above lower bounds*. In: *Theory of Computing Systems* 62.3 (2018), pp. 739–770 (cit. on pp. 12–14, 20, 33–35, 55).

[Wit+10] Tobias Wittkop, Dorothea Emig, Sita Lange, Sven Rahmann, Mario Albrecht, John H Morris, Sebastian Böcker, Jens Stoye, and Jan Baumbach. *Partitioning biological data with transitivity clustering*. In: *Nature Methods* 7.6 (2010), pp. 419–420 (cit. on p. 9).

[Xin11] Xiao Xin. *An FPT Algorithm for the Correlation Clustering Problem*. In: *Advanced Materials and Computer Science*. Vol. 474. Trans Tech Publications Ltd, 2011, pp. 924–927 (cit. on p. 12).