Cluster Editing in Multi-Layer and Temporal Graphs

Jiehua Chen

Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland
jiehua.chen2@gmail.com

Hendrik Molter

Algorithmics and Computational Complexity, Faculty IV, TU Berlin, Berlin, Germany
h.molter@tu-berlin.de

Manuel Sorge

Faculty of Mathematics, Informatics, and Mechanics, University of Warsaw, Poland
manuel.sorge@mimuw.edu.pl

Ondřej Suchý

Department of Theoretical Computer Science, Faculty of Information Technology,
Czech Technical University in Prague, Prague, Czech Republic
ondrej.suchy@fit.cvut.cz

Abstract

Motivated by the recent rapid growth of research for algorithms to cluster multi-layer and temporal graphs, we study extensions of the classical Cluster Editing problem. In Multi-Layer Cluster Editing we receive a set of graphs on the same vertex set, called *layers* and aim to transform all layers into cluster graphs (disjoint unions of cliques) that differ only slightly. More specifically, we want to mark at most d vertices and to transform each layer into a cluster graph using at most k edge additions or deletions per layer so that, if we remove the marked vertices, we obtain the same cluster graph in all layers. In Temporal Cluster Editing we receive a *sequence* of layers and we want to transform each layer into a cluster graph so that consecutive layers differ only slightly. That is, we want to transform each layer into a cluster graph with at most k edge additions or deletions and to mark a distinct set of d vertices in each layer so that each two consecutive layers are the same after removing the vertices marked in the first of the two layers. We study the combinatorial structure of the two problems via their parameterized complexity with respect to the parameters d and k, among others. Despite the similar definition, the two problems behave quite differently: In particular, Multi-Layer Cluster Editing is fixed-parameter tractable with running time $k^{O(k+d)}s^{O(1)}$ for inputs of size s, whereas Temporal Cluster Editing is W[1]-hard with respect to k even if $d = 3$.

2012 ACM Subject Classification Theory of computation → Fixed parameter tractability

Keywords and phrases Cluster Editing, Temporal Graphs, Multi-Layer Graphs, Fixed-Parameter Algorithms, Polynomial Kernels, Parameterized Complexity

Digital Object Identifier 10.4230/LIPIcs.ISAAC.2018.24

Related Version Preprint of the full version of this paper is available from ArXiv [7], `https://arxiv.org/abs/1709.09100`.

Funding JC and MS supported by the People Programme (Marie Curie Actions) of the European Union's Seventh Framework Programme (FP7/2007-2013) under REA grant agreement number 631163.11, the Israel Science Foundation (grant no. 551145/14), and by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme under grant agreement numbers 677651 (JC) and 714704 (MS). HM supported by the DFG, project MATE (NI 369/17). OS supported by grant 17-20065S of the Czech Science Foundation.

Acknowledgements Work initiated at the annual research retreat of the TU Berlin Algorithmics and Computational Complexity group held in Boiensdorf (Baltic Sea), in April 2017. Main work of JC and MS done while with Ben-Gurion University of the Negev, Beer Sheva, Israel.

1 Introduction

Cluster Editing and its weighted form Correlation Clustering are two important and well-studied models of graph clustering [1, 4, 6, 12, 18]. In the former, we are given a graph and we aim to edit (that is, add or delete) the fewest number of edges in order to obtain a *cluster graph*, a graph in which each connected component is a clique. Cluster Editing has attracted a lot of attention from a parameterized-algorithms point of view (e.g. [4, 6, 12, 18]) and the resulting contributions have found their way back into practice [4].

Meanwhile, additional information is now available and used in clustering methods. In particular, research on clustering so-called multi-layer and temporal graphs grows rapidly (e.g. [16, 22, 23, 24, 25]). A *multi-layer graph* is a set of graphs, called *layers*, on the same vertex set [5, 16, 17]. In social networks, a layer can represent social interactions, geographic closeness, common interests or activities [16].[1] A *temporal graph* is a multi-layer graph in which the layers are ordered linearly [14, 15, 19, 20, 24, 25]. Temporal graphs naturally model the evolution of relationships of individuals over time or their set of time-stamped interactions.

The goals in clustering multi-layer and temporal graphs are, respectively, to find a clustering that is consistent with all layers [16, 17, 22, 23] or a clustering that slowly evolves over time consistently with the graph [24, 25]. The methods used herein are often heuristic and beyond observing NP-hardness, to the best of our knowledge, there is no deeper analysis of the complexity of the general underlying computational problems that are attacked in this way. Hence, there is also a lack of knowledge about the possible avenues for algorithmic tractability. We initiate this research here.

We analyze the combinatorial structure behind cluster editing for multi-layer and temporal graphs, defined formally below, via studying their parameterized complexity with respect to the most basic parameters, such as the number of edits. That is, we aim to find *fixed-parameter algorithms*, which have running time $f(p) \cdot \ell^{O(1)}$ where p is the parameter and ℓ the input length, or to show W[1]-hardness, which indicates that there cannot be such algorithms.

As we will see, both problems offer rich interactions between the layers on top of the structure inherited from Cluster Editing. Our main contributions are an intricate fixed-parameter algorithm for multi-layer cluster editing, whose underlying techniques should be applicable to a broader range of multi-layer problems, and a hardness result for temporal cluster editing, which shows that certain non-local structures harbor algorithmic intractability.

[1] When considering the activity in different communities, we typically obtain a large number of layers [21].

Temporal Cluster Editing (TCE). Berger-Wolf and Tantipathananandh [25] were motivated by cluster detection problems from practice to study the following problem. Given a temporal graph, edit each layer into a cluster graph, that is, add or remove edges such that the layer becomes a disjoint union of cliques, while minimizing the total number of edits and the number of vertices moving between different clusters in two consecutive layers. TCE is a variant of this problem where we instead minimize the layer-wise maxima of the number of edits and moving vertices, respectively. The problem can be formalized as follows.

Let $\mathcal{G} = (G_i)_{i\in[\ell]}$ be a temporal graph with vertex set V, that is, G_i is the ith layer. Let $G_i = (V, E_i)$. An edge modification set for a graph $G = (V, E)$ is a set of pairs of vertices from V. A *clustering* for $\mathcal{G}$ is a sequence $\mathcal{M} = (M_i)_{i\in[\ell]}$ of edge modification sets such that each layer G_i is turned into a cluster graph $G'_i = (V, E_i \oplus M_i)$.[2] (Throughout this work, G'_i denotes the modified ith layer of the temporal or multi-layer graph and the corresponding clustering understood from the context.) Intuitively, sets M_i contain the data that we need to disregard in order to cluster our input and hence we want to minimize their sizes [24, 25]. For that, we say that $\mathcal{M}$ is *k-bounded* for some integer $k \in \mathbb{N}$ if $|M_i| \leq k$ for each $i \in \ell$.

A fundamental property of clusterings of temporal graphs is their evolution over time. In practice, these clusterings evolve only slowly as measured by the number of vertices switching between clusters from one layer to another [24, 25]. This requirement can be formalized as follows. Let $d \in \mathbb{N}$. Clustering $\mathcal{M}$ for $\mathcal{G}$ (as above) is *temporally d-consistent* if there exists a sequence $(D_i)_{i\in[\ell-1]}$ of vertex sets such that $G'_i[V \setminus D_i] = G'_{i+1}[V \setminus D_i]$ for each $i \in [\ell - 1]$. Hence, the sets D_i contain the vertices changing clusters. We arrive at the following.

Temporal Cluster Editing (TCE)
Input: A temporal graph $\mathcal{G}$ and two integers k, d.
Question: Is there a temporally d-consistent k-bounded clustering for $\mathcal{G}$?

We also say that the corresponding sets $D_i \subseteq V$ and $M_i \subseteq \binom{V}{2}$ as above form a *solution* and the vertices in D_i are *marked*.

The most natural parameters are the "number k of edge modifications per layer", the "number d of marked vertices", the "number ℓ of layers", and the "number $n = |V|$ of vertices". An overview on our results is shown in Figure 1. (Note that, within these parameters, we have $d \leq n$ and $k \leq n^2$.) A straightforward reduction yields that TCE is NP-complete even if both $d = 0$ and $\ell = 1$ $(\star)$[3]. On the positive side, we obtain an algorithm for TCE with running time $n^{O(k)}\ell$: The basic idea is to check whether any two possible cluster editing sets for two consecutive layers allow for a small number of marked vertices by matching techniques. As it turns out, even for $d = 3$, we cannot obtain an improved running time on the order of $(n\ell)^{o(k)}$ unless the Exponential Time Hypothesis (ETH) fails. The reason is an obstruction represented by small clusters which may have to be joined or split throughout many layers, to be able to form clusters in some later layer. Finally, we give a polynomial kernel with respect to the parameter combination (d, k, ℓ) and show that the problem does not admit a polynomial kernel for parameter "number n of vertices" unless NP $\subseteq$ coNP/poly.

Multi-Layer Cluster Editing (MLCE). For clusterings of multi-layer graphs we typically have to consider the tradeoff between closely matching individual layers and getting an

[2] Herein, $\oplus$ denotes the symmetric difference: $A \oplus B = (A \setminus B) \cup (B \setminus A)$ and $[\ell]$ denotes the set $\{1, \ldots, \ell\}$ for $\ell \in \mathbb{N}$.

[3] The proofs of results marked by $(\star)$ and proofs of correctness and safeness of reduction rules and branching rules marked by $(\star)$ are omitted due to space constraints and deferred to a full version [7].

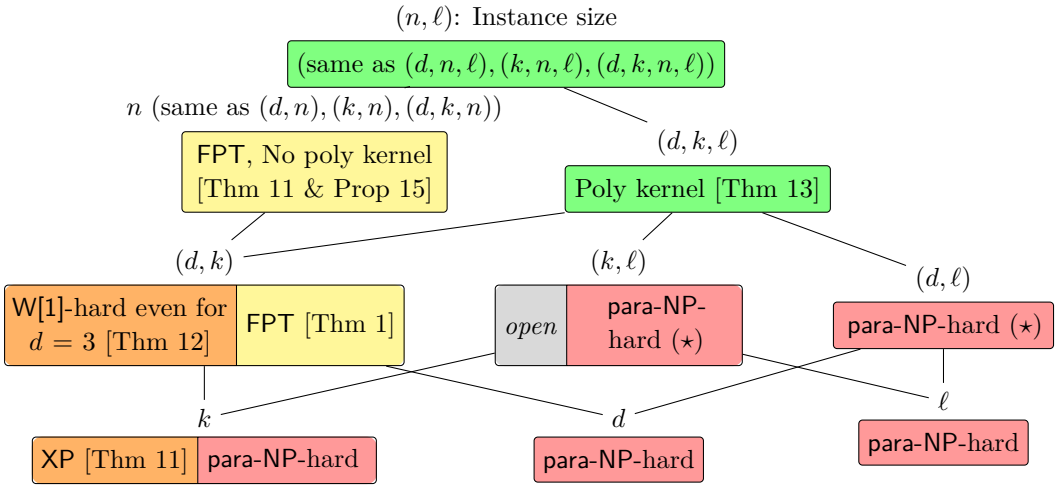

Figure 1 Our results for TCE and MLCE in a Hasse diagram of the upper-boundedness relation between the parameters the "number k of edge modifications per layer", the "number d of marked vertices", the "number ℓ of layers", and the "number $n = |V|$ of vertices" and all of their combinations. A node is split into two parts if the complexity results differ; the left part shows the result for TCE, the right part for MLCE. Red entries mean that the corresponding parameterized problem is para-NP-hard. Orange entries mean that the corresponding parameterized problem is W[1]-hard while contained in XP. It is in FPT for all parameter combinations colored yellow or green and admits a polynomial kernel for all parameter combinations colored green. It does not admit a polynomial kernel for all parameter combinations that are colored yellow unless NP $\subseteq$ coNP/poly. A tight parameterized complexity classification for the gray colored parameter combination is open.

overall sufficient fit [22, 23]. A local upper bound on the number of allowed edits per layer and a global set of marked vertices allow us to study the influence of this tradeoff on the complexity of multi-layer cluster editing. Formally, a clustering $\mathcal{M} = (M_i)_{i \in [\ell]}$ for a multi-layer graph $\{G_i \mid i \in [\ell]\}$ is defined in the same way as for temporal graphs. Clustering $\mathcal{M}$ is *totally d-consistent* if there is a single subset D of vertices such that $G'_i[V \setminus D] = G'_j[V \setminus D]$ for all $i, j \in [\ell]$.[4] In Multi-Layer Cluster Editing (MLCE) the input is a multi-layer graph $\mathcal{G}$ and two integers k and d and we ask for a totally d-consistent k-bounded clustering for $\mathcal{G}$.

To briefly summarize our results for MLCE: While strong overall fit (small parameter d) or closely matched layers (small parameter k) alone do not lead to fixed-parameter tractability, jointly they do. Indeed, we obtain an $k^{O(k+d)} \cdot n^3 \cdot \ell$-time algorithm, in contrast to TCE. At first glance, this is surprising because in the temporal case, we only need to satisfy the consistency condition "locally". This requires less interaction among layers and thus, seemed to be easier to tackle than the multi-layer case. The algorithm uses a novel method that allows us make decisions over a large number of layers at once. It can be compared with greedy localization [9] in that some of the decisions are greedy and transient, meaning that they seem intuitively favorable and can be reversed in individual layers if they later turn out to be wrong. However, the application of this method is not straightforward, requires new techniques to deal with the interaction between layers and consequently intricately tuned branching and reduction rules.

[4] Below we drop the qualifiers "temporally" and "totally" if they are clear from the context.

Algorithm 1: MLCE.

Input:
- A set of graphs $G_1, \ldots, G_\ell = (V, E_1), \ldots, (V, E_\ell)$ two integers k and d.
- A set of marked vertices D, edge modification sets $M_1, \ldots, M_\ell$.
- A set $B \subseteq \binom{V \setminus D}{2}$ of permanent vertex pairs.

1 **if** $|D| > d$ *or there is an* $i \in [\ell]$ *such that* $|M_i \cap B| > k$ **then return** `false`
2 Apply the first applicable rule in the following ordered list: 1, Greedy Rule, 2, Clean-up Rule, 3, and 4.
3 **return** `true`

We in fact completely classify MLCE in terms of fixed-parameter tractability and existence of polynomial-size problem kernels with respect to the parameters k, d, ℓ, and n, and all of their combinations, see Figure 1 for an overview. MLCE is para-NP-hard for all parameter combinations which are smaller or incomparable to $k + d$. Straightforward reductions yield NP-completeness even if both $d = 0$ and $\ell = 1$ or both $k = 0$ and $\ell = 3$; the problem is polynomial-time solvable if $k = 0$ and $\ell \leq 2$ ($\star$). Finally, the kernelization results for TCE also hold for MLCE, that is, the problem admits a polynomial kernel with respect to (d, k, ℓ) and does not admit a polynomial kernel for the "number n of vertices" unless NP $\subseteq$ coNP/poly.

Related Work. We are not aware of studies of the fundamental algorithmic properties of multilayer and temporal graph clustering. In terms of parameterized algorithms, only the indirect approach of aggregating clusterings into one has been studied for multilayer [3, 11] and temporal graphs [24]. These approaches are less accurate, however [2, 25]. The approximability of temporal versions of k-means clustering and its variants was studied by Dey et al. [10].

2 Multi-Layer Cluster Editing (MLCE)

In this section, we show that MLCE can be solved efficiently for small k and d.

▶ **Theorem 1.** MLCE *is FPT with respect to the number k of edge modifications per layer and number d of marked vertices combined. It can be solved in $k^{O(k+d)} \cdot n^3 \cdot \ell$ time.*

We describe a recursive search-tree algorithm (see algorithm 1) for the following input:
- An instance I of MLCE consisting of a multi-layer graph $G_1, \ldots, G_\ell = (V, E_1), \ldots, (V, E_\ell)$ and two integers k and d.
- A *constraint* $P = (D, (M_i)_{i \in [\ell]}, B)$, consisting of a set of *marked* vertices $D \subseteq V$, edge modification sets $M_1, \ldots, M_\ell \subseteq \binom{V}{2}$, and a set $B \subseteq \binom{V \setminus D}{2}$ of *permanent* vertex pairs.

The algorithm follows the greedy localization approach [9] in which we make some decisions greedily, which we possibly revert through branching later on. The greedy decisions herein give us some structure that we can exploit to keep the search-tree size small. The edge modification sets M_i represent both the greedy decisions and those that we made through branching. The set B contains only those made by branching.

Throughout the algorithm, we try to maintain a property that the constraint at hand is good which intuitively means that the constraint can be turned into a solution (if one exists).

▶ **Definition 2** (Good Constraint). Let I be an instance of MLCE. A constraint $P = (D, M_1, \ldots, M_\ell, B)$ is *good* for I if there is a solution $S = (M_1^\star, \ldots, M_\ell^\star, D^\star)$ such that (i) $D \subseteq D^\star$, (ii) there is no $\{u, v\} \in B$ such that $u \in D^\star$, and (iii) for all $i \in [\ell]$ we have $M_i \cap B = M_i^\star \cap B$. We also say that S *witnesses* that P is good.

Furthermore, it is easy to see that an "empty" constraint is good.

▶ **Observation 3.** *For any yes-instance $I = (G_1, \ldots, G_\ell, d, k)$ of* MLCE*, we have that $P_0 = (D = \emptyset, M_1 = \emptyset, \ldots, M_\ell = \emptyset, B = \emptyset)$ is a good constraint.*

We also call the above constraint P_0 *trivial*. The initial call of our algorithm is with the input instance of MLCE together with the trivial constraint P_0.

Our algorithm uses various different branching rules to search for a solution to an MLCE input instance: A *branching rule* takes as input an instance I of MLCE and a constraint P and returns a set of constraints $P^{(1)}, \ldots, P^{(x)}$. When a branching rule is applied, the algorithm invokes a recursive call for each constraint returned by the branching rule and returns `true` if at least one of the recursive calls returns `true`; otherwise, it returns `false`. For that to be correct, whenever a branching rule is invoked with a good constraint, at least one of the constraints returned by the branching rule has to be a good constraint as well. In this case we say that a branching rule is *safe*.

In the following, we introduce the branching rules used by the algorithm and prove that each of them is safe. This together with Theorem 3 will allow us to prove by induction that the algorithm eventually finds a solution for the input instance of MLCE if it is a yes-instance. To make the description of the branching rules more readable, we introduce four types of non-marked vertex pairs. Say that a vertex pair $\{u, v\} \in \binom{V \setminus D}{2}$ is

- *settled* if $\{u, v\} \in E_i \oplus M_i$ for all $i \in \ell$ or $\{u, v\} \notin E_i \oplus M_i$ for all $i \in [\ell]$ (edge always present or never present),
- *frequent* if $|\{i \mid \{u, v\} \in E_i \oplus M_i\}| \geq \frac{2\ell}{3}$ (edge almost always present),
- *scarce* if $|\{i \mid \{u, v\} \in E_i \oplus M_i\}| \leq \frac{\ell}{3}$ (edge almost never present), and
- *unsettled* otherwise, that is, $\frac{\ell}{3} < |\{i \mid \{u, v\} \in E_i \oplus M_i\}| < \frac{2\ell}{3}$ (edge sometimes present).

Note that, if a vertex pair $\{u, v\}$ falls in one of the above categories, both u, v are not marked.

Our aim with the first two rules is to settle all pairs in $\binom{V \setminus D}{2}$. In order to achieve our running time bound, we can only afford to exhaustively search through all unsettled vertex pairs:

▶ **Branching Rule 1 ($\star$).** If there is an unsettled vertex pair $\{u, v\} \in \binom{V \setminus D}{2}$, then output the following up to four constraints:

1. For all $i \in [\ell]$, put $M_i^{(1)} = M_i \cup (\{\{u, v\}\} \setminus E_i)$, $D^{(1)} = D$, and $B^{(1)} = B \cup \{\{u, v\}\}$.
2. For all $i \in [\ell]$, put $M_i^{(2)} = M_i \cup (\{\{u, v\}\} \cap E_i)$, $D^{(2)} = D$, and $B^{(2)} = B \cup \{\{u, v\}\}$.
3. If there is no $x \in V \setminus D$ with $\{u, x\} \in B$, then $D^{(3)} = D \cup \{u\}$, the rest stays the same.
4. If there is no $x \in V \setminus D$ with $\{v, x\} \in B$, then $D^{(4)} = D \cup \{v\}$, the rest stays the same.

The following Greedy Rule deals with all frequent and scarce vertex pairs. It only produces one constraint and hence no branching occurs in that sense. For formal reasons it is nevertheless useful to treat the Greedy Rule as a special case of a branching rule. Note that the algorithm also invokes a recursive call with the output constraint of this rule. The rule greedily adds the edge corresponding to a frequent vertex pair in all layers where it is not present and removes edges corresponding to scarce vertex pairs in all layers where it is present. Intuitively, the Greedy Rule is safe, because all of its decisions can be reverted later.

▶ **Greedy Rule ($\star$).** If there is a *frequent* or a *scarce* vertex pair $\{u, v\} \in \binom{V \setminus D}{2}$, then return one of the following two constraints:

- Frequent: for all $i \in [\ell]$ put $M_i^{(1)} = M_i \cup (\{\{u,v\}\} \setminus E_i)$, the rest stays the same.
- Scarce: for all $i \in [\ell]$ put $M_i^{(1)} = M_i \cup (\{\{u,v\}\} \cap E_i)$, the rest stays the same.

After the above two rules have been applied exhaustively, all pairs in $\binom{V \setminus D}{2}$ are settled. With the following rule we edit the subgraphs induced by all non-marked vertices into cluster graphs. This branching rule represents a well-known rule from the classical Cluster Editing with the addition that we also branch on marking vertices.

▶ **Branching Rule 2 (⋆).** If there is an induced $P_3 = (\{u,v\},\{v,w\})$ in $G_i'[V \setminus D]$ for some $i \in [\ell]$, where $G_i' = (V, E_i \oplus M_i)$, then return the following up to six constraints:

1. If $\{u,v\} \notin B$: for all $i \in [\ell]$ put $M_i^{(1)} = M_i \oplus \{\{u,v\}\}$, $D^{(1)} = D$, and $B^{(1)} = B \cup \{\{u,v\}\}$.
2. If $\{v,w\} \notin B$: for all $i \in [\ell]$ put $M_i^{(2)} = M_i \oplus \{\{v,w\}\}$, $D^{(2)} = D$, and $B^{(2)} = B \cup \{\{v,w\}\}$.
3. If $\{u,w\} \notin B$: for all $i \in [\ell]$ put $M_i^{(3)} = M_i \oplus \{\{u,w\}\}$, $D^{(3)} = D$, and $B^{(3)} = B \cup \{\{u,w\}\}$.
4. For each $x \in \{u,v,w\}$: If there is no $y \in V \setminus D$ such that $\{x,y\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{x\}$, the rest stays the same.

If none of the above possibilities apply, then reject the current branch.[5]

The next rule keeps the sets of edge modifications M_i free of marked vertices. Pairs in M_i can become marked if vertices of vertex pairs processed by the Greedy Rule are marked by other branching rules further down the search tree. Like the Greedy Rule, it only produces one constraint and hence no branching occurs, so it is also a degenerate branching rule. Note that the algorithm also invokes a recursive call with the output constraint of this rule.

▶ **Clean-up Rule (⋆).** If there is an $i \in [\ell]$ such that there is a $\{u,v\} \in M_i$ with $u \in D$, then return a constraint with $M_i^{(1)} = M_i \setminus \{\{u,v\}\}$, the rest stays the same.

The next rule tries to repair any budget violations that might occur. Since with the Greedy Rule we greedily make decisions and do not exhaustively search through the whole search space, we expect that some of the choices were not correct. This rule will then revert these choices. Also, to have a correct estimate of the sizes of the current edge modification sets, this rule requires that the Clean-up Rule is not applicable. For technical reasons, it also requires that 1 and the Greedy Rule are not applicable.

▶ **Branching Rule 3 (⋆).** If there is an M_i for some $i \in [\ell]$ with $|M_i| > k$, then if $|M_i \setminus B| \leq k+1$, let $M_i' = M_i \setminus B$, otherwise, take any $M_i' \subseteq M_i \setminus B$ with $|M_i'| = k+1$ and return the following constraints:

1. For each $\{u,v\} \in M_i'$ return a constraint in which for all $j \in [\ell]$ we put $M_j^{(\cdot)} = M_j \oplus \{\{u,v\}\}$, $D^{(\cdot)} = D$, and $B^{(\cdot)} = B \cup \{\{u,v\}\}$.
2. For each $\{u,v\} \in M_i'$:
   - If there is no $x \in V \setminus D$ such that $\{u,x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{u\}$, $B^{(\cdot)} = B$, and $1 \leq j \leq \ell$: $M_j^{(\cdot)} = M_j \setminus \{\{u,v\}\}$.
   - If there is no $x \in V \setminus D$ such that $\{v,x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{v\}$, $B^{(\cdot)} = B$ and $1 \leq j \leq \ell$: $M_j^{(\cdot)} = M_j \setminus \{\{u,v\}\}$.

If $M_i' = \emptyset$, then reject the current branch.

[5] This technically does not fit the definition of a branching rule but we can achieve the same effect by returning trivially unsatisfiable constraints such as a constraint with $|D^{(\cdot)}| > d$.

The last rule, 4, requires that all other rules are not applicable. In this case the non-marked vertices induce the same cluster graph in every layer. 4 checks whether in every layer it is possible to turn the whole layer (including the marked vertices) into a cluster graph such that the cluster graph induced by the non-marked vertices stays the same and the edge modification budget is not violated in any layer. If this is not the case for a layer i, the rule checks whether it is necessary to revert a greedy decision or whether there is an induced P_3 where exactly one vertex is marked and it is necessary to modify the vertex pair not containing the marked vertex. To achieve the latter we introduce a modified version of a known kernelization algorithm [13] for classic Cluster Editing. We call the algorithm K and it takes as input a tuple (G_i, k, D, M_i, B) and either outputs a distinct failure symbol or two sets R and C, where R contains all unmarked vertex pairs modified by K and C contains unmarked vertex pairs of the produced kernel that are part of induced P_3s. The formal description is as follows.

▶ **Modified Kernelization Algorithm K.** Given an input (G_i, k, D, M_i, B). First, set all vertex pairs in $M_i \cup B$ to *obligatory* and exhaustively apply the following modified versions of standard data reduction rules for Cluster Editing to $G_i' = (V, E_i \oplus M_i)$. Let $k_i = k - |M_i|$ and $R = \emptyset$.

- If $k_i < 0$ or there is an induced P_3 where all vertex pairs are obligatory, then abort and output a failure symbol.
- If a vertex pair $\{u, v\}$ is involved in $k_i + 1$ induced P_3's, then, if it is obligatory, abort and output a failure symbol, otherwise modify it, set it to obligatory, and reduce k_i by one. If $u \notin D$ and $v \notin D$, then add $\{u, v\}$ to R.
- If there is an isolated clique, then remove it.

Let $G_i^{(R)}$ be the reduced version of G_i. If the number of vertices in $G_i^{(R)}$ is larger than $k_i^2 + 2k_i$, then abort and output a failure symbol. Otherwise, let C be the set of all vertex pairs of *unmarked* (not contained in D) vertices that are part of an induced P_3 in $G_i^{(R)}$. Output R and C.

▶ **Branching Rule 4 ($\star$).** For all $1 \leq i \leq \ell$ we use $\mathcal{M}_i$ to denote the set of all possible edge modifications where each edge is incident to at least one marked vertex, that turn $G_i' = (V, E_i \oplus M_i)$ into a cluster graph. More specifically, we have

$$\mathcal{M}_i = \{M \subseteq \tbinom{V}{2} \mid \forall e \in M : e \cap D \neq \emptyset \ \wedge \ G_i'' = (V, E_i \oplus (M_i \cup M)) \text{ is a cluster graph}\}.$$

If there is an $1 \leq i \leq \ell$ such that $\min_{M \in \mathcal{M}_i} |M| > k - |M_i|$ then let $M_i' = M_i \setminus B$ and invoke the modified kernelization algorithm K on (G_i, k, D, M_i, B). If it outputs a failure symbol and $M_i' = \emptyset$, then reject the current branch. Otherwise let R and C be the sets output by K or $R = C = \emptyset$ if K output a failure symbol, and return the following constraints:

1. For each $\{u, v\} \in M_i'$:
   - Return a constraint in which for all $j \in [\ell]$ we put $M_j^{(\cdot)} = M_j \oplus \{\{u, v\}\}$, $D^{(\cdot)} = D$, and $B^{(\cdot)} = B \cup \{\{u, v\}\}$.
   - If there is no $x \in V \setminus D$ such that $\{u, x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{u\}$, $B^{(\cdot)} = B$, and $1 \leq j \leq \ell$: $M_j^{(\cdot)} = M_j \setminus \{\{u, v\}\}$.
   - If there is no $x \in V \setminus D$ such that $\{v, x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{v\}$, $B^{(\cdot)} = B$ and $1 \leq j \leq \ell$: $M_j^{(\cdot)} = M_j \setminus \{\{u, v\}\}$.
2. For each $u \in V \setminus D$ such that $\{u, v\} \in R$ for some $v \in V$: If there is no $x \in V \setminus D$ such that $\{u, x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{u\}$, $B^{(\cdot)} = B$, and $1 \leq j \leq \ell$: $M_j^{(\cdot)} = M_j$. If $R \neq \emptyset$, then output a constraint with $D^{(\cdot)} = D$, $B^{(\cdot)} = B \cup M_i \cup R$, and $1 \leq j \leq \ell$: $M_j^{(\cdot)} = M_j \oplus R$.

3. For each $\{u, v\} \in C$:
   - If there is no $x \in V \setminus D$ such that $\{u, x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{u\}$, and the rest stays the same. If there is no $x \in V \setminus D$ such that $\{v, x\} \in B$, then return a constraint with $D^{(\cdot)} = D \cup \{v\}$, and the rest stays the same.
   - Return a constraint with $D^{(\cdot)} = D$, $B^{(\cdot)} = B \cup M_i \cup R \cup \{\{u, v\}\}$, and $1 \leq j \leq \ell$: $M_j^{(\cdot)} = M_j \oplus R \oplus \{\{u, v\}\}$.

We now prove the correctness of the algorithm. By a straightforward analysis of the branching rules it follows that, if none is applicable, then the current constraint P yields a solution. Hence, whenever the algorithm outputs `true`, then the input is indeed a yes-instance.

▶ **Lemma 4** (⋆). *Given an instance I of* MLCE*, if algorithm 1 outputs* `true` *on input I and the trivial partial solution P_0, then I is a yes-instance.*

To show that, whenever the input instance I of the algorithm is a yes-instance, then the algorithm outputs `true`, we define the *quality* of a good constraint and show that the algorithm increases the quality until it eventually finds a solution.

▶ **Definition 5** (Quality of a constraint). Let $I = (G_1, \ldots, G_\ell, k, d)$ be an instance of MLCE. The *quality* $\gamma_I(P)$ of a constraint $P = (D, M_1, \ldots, M_\ell, B)$ for I is $\gamma_I(P) = |D| + |B| - |\{\{u, v\} \in \binom{V \setminus D}{2} \mid \{u, v\} \text{ is frequent or scarce}\}|$.

▶ **Lemma 6** (⋆). *Let P be a good constraint for a yes-instance of* MLCE*. If applicable, each of the Greedy Rule and Branching Rules 1, 2, 3, and 4 return a good constraint with strictly increased quality in comparison to P.*

Next we show that the notion of quality of a good constraint is indeed a measure that allows us to argue that the algorithm eventually produces a solution (if it exists).

▶ **Lemma 7** (⋆). *Let I be a yes-instance of* MLCE*, then there is a constant $c_I \geq 0$ such that for every good constraint P we have that $\gamma_I(P) \leq c_I$ and there is at least one good constraint P_{max} with $\gamma_I(P_{max}) = c_I$. Furthermore, for any good constraint P'_{max} with $\gamma_I(P'_{max}) = c_I$, we have that algorithm 1 outputs* `true` *on input I and P'_{max}.*

We can now show the correctness of algorithm 1. Theorem 4 ensures that we only output `true` if the input is actually a yes-instance and Lemmas 6 and 7 together with the safeness of all branching rules ensures that if the input is a yes-instance, the algorithm outputs `true`.

▶ **Corollary 8** (Correctness of algorithm 1) (⋆). *Given a* MLCE *instance I, algorithm 1 outputs* `true` *on input I and the trivial good constraint P_0 if and only if I is a yes-instance.*

It remains to show that algorithm 1 has the claimed running time upper-bound. We can check that all branching rules create at most $O(k^4)$ recursive calls. The differentiation between unsettled, frequent and scarce vertex pairs ensures that the edge modification sets in sufficiently many layers increase for the search tree to have depth of at most $O(k + d)$. The time needed to apply a branching rule is dominated by 4, where we essentially have to solve classical Cluster Editing in every layer.

▶ **Lemma 9** (⋆). *The running time of algorithm 1 is in $k^{O(k+d)} \cdot O(n^3 \cdot \ell)$.*

3 Temporal Cluster Editing (TCE)

In this section we provide an algorithm for TCE with a running time $n^{O(k)}\ell$ and show that the running time cannot substantially be improved unless the Exponential Time Hypothesis (ETH) fails. The algorithm uses the following algorithm for the two-layer case as a subroutine. The algorithm uses similar ideas as Exercise 4.5 and its hint in Cygan et al. [8].

▶ **Proposition 10** (⋆). *If $k = 0$ and $\ell = 2$, then* TCE *and* MLCE *can be solved in $O(n^2 \log n)$ time, where n denotes the number of vertices.*

▶ **Theorem 11.** TCE *can be solved in $O(\ell \cdot n^{4k+2} \log n)$ time.*

Proof. Given an instance $((G_i)_{i\in[\ell]}, k, d)$ of TCE, build an ℓ-partite graph $\mathcal{G}$ as follows: For each possible cluster editing set of G_i of size at most k, add a vertex to the i^{th} part of $\mathcal{G}$. Note that $\mathcal{G}$ contains $O(n^{2k}\ell)$ vertices since each part contains $O(n^{2k})$ vertices. For each i, $1 \leq i \leq n-1$, and each pair of vertices u, v in $\mathcal{G}$ such that u is in part i and v is in part $(i+1)$ add to G the edge $\{u, v\}$ if the algorithm of Theorem 10 accepts on input of the following instance of MLCE. Let M_u, M_v be the cluster editing sets corresponding to u and v, respectively. The MLCE instance consists of a multi-layer graph with the two layers $(V, E_i \oplus M_u)$ and $(V, E_i \oplus M_v)$, edit budget equal to zero, and marking budget equal to d. For each pair of vertices u, v, constructing the corresponding MLCE instance and solving it takes $O(n^2 \log n)$ time, amounting to overall $O(\ell \cdot n^{4k+2} \log n)$ time, because there are at most $n^{4k}\ell$ pairs of vertices to consider. Finally, we test whether there is a path from a vertex in the first part to a vertex in the last part in $\mathcal{G}$. As there are at most $n^{4k}\ell$ edges in $\mathcal{G}$, this takes $O(n^{4k}\ell)$ time. Hence, overall the running time is $O(\ell \cdot n^{4k+3} \log n)$. The correctness is deferred to a full version [7]. ◀

Theorem 11 implies that TCE is fixed-parameter tractable when parameterized by the number n of vertices. At first glance, it seems wasteful to iterate over all possible cluster editing sets for each layer. Rather, the interaction between two consecutive layers seems to be limited by k and d, since the necessary edits are local to induced P_3, and the necessary markings are local to incongruent clusters (perhaps resulting from destroying P_3s). However, to our surprise, when the number of layers grows, this interaction spirals out of control. As the reduction of the following hardness result implies, we have to take into account splitting up small clusters in an early layer (even though locally they were already cliques), so as to be able to form cluster graphs a large number of layers later on. This behavior stands in stark contrast to MLCE, where the combinatorial explosion is limited to k and d.

▶ **Theorem 12** (⋆). TCE *is W[1]-hard with respect to k, even if $d = 3$. Moreover, it does not admit an $f(k)(n\ell)^{o(k)}$-time algorithm unless the ETH fails.*

4 Kernelization for MLCE and TCE

In this section we investigate the kernelizability of MLCE and TCE for different combinations of the four parameters as introduced in section 1. More specifically, we identify the parameter combinations for which MLCE and TCE admit polynomial kernels, and then we identify the parameter combinations for which no polynomial kernels exist, unless $\mathsf{NP} \subseteq \mathsf{coNP/poly}$.

We first present a polynomial kernel for MLCE for the parameter combination (k, d, ℓ) and then argue that essentially the same reduction rules give a polynomial kernel for TCE.

▶ **Theorem 13.** MLCE *admits a kernel of size $O(\ell^3 \cdot (k+d)^4)$ and* TCE *admits a kernel of size $O(\ell^3 \cdot (k + d \cdot \ell)^4)$. Both kernels can be computed in $O(\ell \cdot n^3)$ time.*

We provide several reduction rules that subsequently modify the instance and we assume that if a particular rule is to be applied, then the instance is reduced with respect to all previous rules, that is, all previous rules were already exhaustively applied. We introduce Multi-Layer Cluster Editing with Separate Budgets (MLCEwSB) which differs from MLCE only in that, instead of a global upper bound k on the number of edits, we receive ℓ individual budgets k_i, $i \in [\ell]$, and we require that $|M_i| \leq k_i$.

We first transform the input instance of MLCE to an equivalent instance of MLCEwSB by letting $k_i = k$ for every $i \in [\ell]$. Then we apply all our reduction rules to MLCEwSB. Finally, we transform the resulting instance of MLCEwSB to an equivalent instance of MLCE with just a small increase of the vertex set. Through the presentation, let $(G_1 = (V, E_1), \ldots, G_\ell = (V, E_\ell), k_1, \ldots, k_\ell, d)$ be the current instance and $k = \max\{k_i \mid i \in [\ell]\}$.

Next, we apply slightly modified versions of well known rules for classical Cluster Editing [13] and apply them on each layer individually ($\star$). These rules are known to produce a kernel of size $k^2 + 2k$. Notably, we leave out a rule that removes isolated cliques. Hence, after the application of these rules we either conclude that we face a no-instance or every layer i consists of a set $R_i \subseteq V$, that contains the vertices v that appear in some induced P_3 in G_i, and a number of isolated cliques. Furthermore, let $R = \bigcup_{i=1}^{\ell} R_i$.

As a major difference to Cluster Editing for a single layer, we cannot simply remove the vertices that are not involved in any P_3 since we require the cluster graphs in individual layers not to differ too much. Only vertices in the clusters that do not change can be removed.

▶ **Reduction Rule 1** ($\star$)**.** If there is a subset $A \subseteq V \setminus R$ such that for each layer $i \in [\ell]$, the subset A is the vertex set of a connected component of G_i, then remove A (and the corresponding edges) from every G_i.

The next rule allows us to reduce vertices that appear in exactly the same clusters.

▶ **Reduction Rule 2** ($\star$)**.** If there is a set $A \subseteq V \setminus R$ with $|A| \geq k + d + 3$ such that for every layer $i \in [\ell]$ it holds that all vertices of A are in the same connected component of G_i, then select an arbitrary $v \in A$ and remove v from every G_i.

The next rule shows that the remaining clusters in a yes-instance cannot be too large.

▶ **Reduction Rule 3** ($\star$)**.** If there is a layer $i \in [\ell]$ and a connected component A of G_i with $|A \setminus R| \geq k + 2d + 3$, then answer NO.

Now we introduce our final rule bounding the number of vertices in the instance.

▶ **Reduction Rule 4** ($\star$)**.** If $|V| > \ell \cdot (k^2 + 2k + d \cdot (k + 2d + 2) + 2k)$, then answer NO.

After bounding the size of the instance through 4 it remains to transform the resulting instance of MLCEwSB to an equivalent instance of MLCE. To this end we introduce new vertex set A of size exactly $2k + 2$ to V and to each E_i introduce all edges from $\binom{A}{2}$. Then, for each $i \in \{1, \ldots, \ell\}$ we remove $k - k_i$ arbitrary edges between vertices of A from E_i and set $k_i = k$. It is straightforward to show that this produces an equivalent instance, which can be turned into an equivalent instance of MLCE in an obvious way.

Since no rule increases k, d, or ℓ, $|V| = O(\ell \cdot (k + d)^2)$, the resulting instance can be described using $O(\ell^3 \cdot (k + d)^4)$ bits and it is equivalent to the original instance, it remains to show that the kernelization is computable in polynomial time.

▶ **Lemma 14** ($\star$)**.** *The kernelization can be done in $O(\ell \cdot n^3)$ time.*

Lastly, we argue that slightly modified reduction rules can be applied to TCE (with individual edge-modification budgets, where the resulting instance can be transformed back). Intuitively this follows from the following observations: The reduction rules do not mark vertices, and the union of all marked vertices of a TCE solution together with the edge modification sets forms a solution for a MLCE instance, where the maximal number of marked vertices is $d \cdot \ell$. Hence, replacing d with $d \cdot \ell$ in the description of all reductions rules yields a set of rules that produce a kernel of size $O(\ell^3 \cdot (k + d \cdot \ell)^4)$ for TCE ($\star$).

In contrast, we have the following.

▶ **Proposition 15** ($\star$)**.** MLCE *and* TCE *do not admit polynomial kernels with respect to the number n of vertices, unless* NP $\subseteq$ coNP/poly.

5 Conclusion

Our results highlight that TCE and MLCE are much richer in structure than classical Cluster Editing. Techniques for the classical problem seem to only carry over somewhat for kernelization algorithms and otherwise new methods are necessary. In this regard, we contribute our fixed-parameter algorithm for MLCE with respect to the combination of k and d. In contrast, the W[1]-hardness for TCE with respect to k for $d = 3$ highlights the obstacles we need to overcome. Perhaps we can break the temporal non-locality by bounding the number of allowed modifications at one vertex in any interval of layers of some fixed size.

References

1 Nikhil Bansal, Avrim Blum, and Shuchi Chawla. Correlation Clustering. *Machine Learning*, 56:89–113, 2004.

2 Matteo Barigozzi, Giorgio Fagiolo, and Giuseppe Mangioni. Identifying the Community Structure of the International-Trade Multi-Network. *Physica A*, 390(11):2051–2066, 2011.

3 N. Betzler, J. Guo, C. Komusiewicz, and R. Niedermeier. Average Parameterization and Partial Kernelization for Computing Medians. *J. Comput. Syst. Sci.*, 77(4):774–789, 2011.

4 Sebastian Böcker and Jan Baumbach. Cluster Editing. In *Proc. of CiE '13*, volume 7921 of *LNCS*, pages 33–44. Springer, 2013.

5 Leizhen Cai and Junjie Ye. Dual Connectedness of Edge-Bicolored Graphs and Beyond. In *Proc. of MFCS '14*, volume 8635 of *LNCS*, pages 141–152. Springer, 2014.

6 Yixin Cao and Jianer Chen. Cluster Editing: Kernelization Based on Edge Cuts. *Algorithmica*, 64(1):152–169, 2012.

7 Jiehua Chen, Hendrik Molter, Manuel Sorge, and Ondrej Suchý. Cluster Editing in Multi-Layer and Temporal Graphs. *CoRR*, abs/1709.09100, 2018.

8 M. Cygan, F.V. Fomin, Ł. Kowalik, D. Lokshtanov, D. Marx, Ma. Pilipczuk, Mi. Pilipczuk, and S. Saurabh. *Parameterized Algorithms*. Springer, 2015.

9 Frank Dehne, Mike Fellows, Frances Rosamond, and Peter Shaw. Greedy Localization, Iterative Compression, and Modeled Crown Reductions: New FPT Techniques, an Improved Algorithm for Set Splitting, and a Novel 2k Kernelization for Vertex Cover. In *Proc. of IWPEC '04*, volume 3162 of *LNCS*, pages 271–280. Springer, 2004.

10 T.K. Dey, A. Rossi, and A. Sidiropoulos. Temporal Clustering. In *Proc. of ESA '17*, volume 87 of *LIPIcs*, pages 34:1–34:14. Schloss Dagstuhl, 2017.

11 Martin Dörnfelder, Jiong Guo, Christian Komusiewicz, and Mathias Weller. On the Parameterized Complexity of Consensus Clustering. *Theor. Comput. Sci.*, 542:71–82, 2014.

12 Fedor V. Fomin, Stefan Kratsch, Marcin Pilipczuk, Michał Pilipczuk, and Yngve Villanger. Tight Bounds for Parameterized Complexity of Cluster Editing with a Small Number of Clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014.

13 J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-Modeled Data Clustering: Exact Algorithms for Clique Generation. *Theory Comput. Syst.*, 38(4):373–392, 2005.

14 P. Holme. Modern temporal network theory: a colloquium. *Eur. Phys. J. B*, 88(9):234, 2015.

15 Petter Holme and Jari Saramäki. Temporal networks. *Phys. Rep.*, 519(3):97–125, 2012.

16 Jungeun Kim and Jae-Gil Lee. Community Detection in Multi-Layer Graphs: A Survey. *SIGMOD Rec.*, 44(3):37–48, 2015.

17 Mikko Kivelä, Alex Arenas, Marc Barthelemy, James P. Gleeson, Yamir Moreno, and Mason A. Porter. Multilayer networks. *J. Complex Netw.*, 2(3):203–271, 2014.

18 Christian Komusiewicz and Johannes Uhlmann. Cluster Editing with Locally Bounded Modifications. *Discrete Appl. Math.*, 160:2259–2270, 2012.

19 Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *CoRR*, abs/1710.04073, 2017. URL: `http://arxiv.org/abs/1710.04073`.

20 Othon Michail. An introduction to temporal graphs: An algorithmic perspective. *Internet Math.*, 12(4):239–280, 2016.

21 Vincenzo Nicosia and Vito Latora. Measuring and Modeling Correlations in Multiplex Networks. *Phys. Rev. E*, 92(3):032805, 2015.

22 Andrea Tagarelli, Alessia Amelio, and Francesco Gullo. Ensemble-Based Community Detection in Multilayer Networks. *Data Min. Knowl. Discov.*, 31(5):1506–1543, 2017.

23 W. Tang, Z. Lu, and I. S. Dhillon. Clustering with Multiple Graphs. In *Proc. of ICDM '09*, pages 1016–1021. IEEE Computer Society, 2009.

24 C. Tantipathananandh, T. Berger-Wolf, and D. Kempe. A Framework for Community Identification in Dynamic Social Networks. In *Proc. of KDD '07*, pages 717–726. ACM, 2007.

25 C. Tantipathananandh and T. Y. Berger-Wolf. Finding Communities in Dynamic Social Networks. In *Proc. of ICDM '11*, pages 1236–1241. IEEE Computer Society, 2011.