Parameterized Algorithms for Module Map Problems[☆]

Frank Sommer[1], Christian Komusiewicz

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Abstract

Motivated by applications in the analysis of genetic networks, we introduce and study the NP-hard Module Map problem which has as input a graph $G = (V, E)$ with red and blue edges and an integer k and asks to transform G by at most k edge modifications into a graph G' which has the following properties: the vertex set of G' can be partitioned into so-called clusters such that inside a cluster every pair of vertices is connected by a blue edge and for two distinct clusters A and B either all vertices $u \in A$ and $v \in B$ are connected by a red edge or there is no edge between A and B. We show that Module Map can be solved in $\mathcal{O}(3^k \cdot (|V|+|E|))$ time and $\mathcal{O}(2^k \cdot |V|^3)$ time, respectively. Furthermore, we show that Module Map admits a kernel with $\mathcal{O}(k^2)$ vertices.

1. Introduction

Graphs are a useful tool for many tasks in data analysis such as graph-based data clustering or the identification of important agents and connections in social networks. In graph-based data clustering, the edges in the graph indicate similarity between the objects that are represented by the vertices. The goal is to obtain a partition of the vertex set into clusters such that the objects inside each cluster are similar to each other and objects from different clusters are dissimilar. One of the central problems in this area is called Cluster Editing [5], also known as Correlation Clustering [24]. The decision version of Cluster Editing is formulated as follows.

Cluster Editing
Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.
Question: Can we transform G into a *cluster graph*, that is, a disjoint union of cliques, by deleting or adding at most k edges?

[☆]An extended abstract of this work appeared in the proceedings of the 5th International Symposium on Combinatorial Optimization (ISCO '18) held in Marrakesh, Morocco. Fundamental parts of this work also appeared in the first author's master thesis [F. Sommer, Parameterized Algorithms for the Module Map Problem, Masterarbeit, Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany, 2017].

Email addresses: fsommer@informatik.uni-marburg.de (Frank Sommer), komusiewicz@informatik.uni-marburg.de (Christian Komusiewicz)

[1]Frank Sommer was supported by the DFG, project MAGZ (KO 3669/4-1).

In Cluster Editing, we essentially view clustering as a graph modification problem: If we can transform G into a cluster graph G' by at most k edge modifications, then the connected components of G' define a partition of V into clusters such that at most k vertex-pairs of G contradict this partition; these are exactly the deleted edges and the inserted edges.

In the input graph for Cluster Editing there are two possible states for each pair of objects: similarity or dissimilarity. To model observed data more precisely, one may aim to distinguish more than two states. This can be done by using graphs with multiple edge types, also called multilayer graphs [9, 20]. In this work, we study Module Map, a generalization of Cluster Editing in graphs with two types of edges.

Module maps. The Module Map problem arises in the construction of so-called module maps in computational biology [4, 25]. The input is an edge-bicolored graph $G = (V, E_b, E_r)$ with a set E_b of *blue* edges and a set E_r of *red* edges where E_b and E_r are disjoint. In the following, we will refer to these objects simply as graphs. The vertices of G represent genes of an organism, the blue edges represent physical interactions between the proteins that are built from these genes, and the red edges represent negative genetic interactions between the genes. These are inferred from knockout experiments. In these experiments, it is measured for two genes a and b, whether the fitness of organisms in which a and b are defunct is lower than the fitness of organisms in which only a or only b is defunct [4]. If this is the case, then the interaction between a and b is called a negative interaction. In the biological application, the task is to find modules which are groups of genes that have a common function in the organism.

According to Amar and Shamir [4], the following properties are desirable for these modules: First, each module should be highly connected with respect to the physical protein interactions. In other words, within each module there should be many blue edges. Second, there should be few physical interactions and, thus, few blue edges between different modules. Third, two different modules A and B may have a *link* between them. If they have a link, then there are many negative genetic interactions and, thus, many red edges between them; otherwise, there are few negative genetic interactions and, thus, few red edges between them. A partition of the genes into modules and the set of links between modules then defines a module map. Amar and Shamir [4] discuss different objective functions for obtaining a module map that take these properties into account.

Inspired by Cluster Editing, where one views clustering as a graph modification problem, we study the problem of obtaining module maps from a graph modification point of view. That is, we first define formally the set of *module graphs* which are the graphs with a perfect module map. Then, the computational problem is to find a module graph that can be obtained from the input graph by few edge modifications.

Module graphs. According to the properties described above, each module is ideally a blue clique and there are no blue edges between different modules.

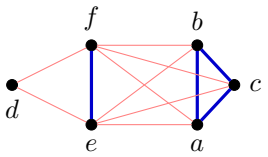


Figure 1: A module graph with the clusters $\{a, b, c\}$, $\{d\}$, and $\{e, f\}$. Here, blue edges are dark and bold, and red edges are bright.

In other words, the *blue subgraph* $G_b := (V, E_b)$ obtained by discarding all red edges is a cluster graph. Each connected component of G_b is called a *cluster*, and we say that a graph G where G_b is a cluster graph fulfills the *cluster property*. Moreover, for each pair of different clusters A and B there are ideally either no edges between $u \in A$ and $v \in B$ or each $u \in A$ and each $v \in B$ are connected by a red edge. In other words, the graph $G_r[A \cup B]$ is either edgeless or complete bipartite with parts A and B, where $G_r := (V, E_r)$ is the *red subgraph* obtained by discarding all blue edges. This property is called *link property*, and the red bicliques are called *links*. The link property is only defined for graphs that fulfill the cluster property. A graph has a perfect module map if it satisfies both properties.

Definition 1. *A graph $G = (V, E_b, E_r)$ is a* module graph *if G satisfies the cluster property and the link property.*

A module graph is shown in Figure 1. Clearly, not every graph is a module graph. For example, a graph G with three vertices u, v, and w where the edges $\{u, v\}$ and $\{u, w\}$ are blue and the edge $\{v, w\}$ is red violates the cluster property. Our aim is to find a module graph which can be obtained from the input graph G by as few edge modifications as possible.[2]

Module Map
Input: A graph $G = (V, E_b, E_r)$ and a nonnegative integer k.
Question: Can we transform G into a module graph by deleting or adding at most k red and blue edges?

Herein, to transform a blue edge into a red edge, we first have to delete the blue edge and in a second step we may insert the red edge. Thus, transforming a blue edge into a red edge has cost two and vice versa.

As in the case of Cluster Editing, a module graph that is obtained by at most k edge modifications directly implies a module map such that at most k vertex pairs in this map contradict the input vertex pairs. Here, a contradiction is a red edge or a non-edge inside a cluster, a blue edge between different clusters,

[2]To make the correctness proof for the algorithms more concise, we focus on the decision version of Module Map. The algorithms can be easily modified to output a solution and an optimal solution can be found by considering increasing k starting with $k = 0$; the running time for the first yes-instance dominates the total running time of all previous calls.

a non-edge between different clusters that have a link, or a red edge between different clusters that have no link. Our problem formulation is thus related to previous ones [4, 25] but more simplistic: for example it does not use statistically defined p-values to determine whether a link between modules should be present or not. As observed previously [4, 25], most formulations of the construction problem for module maps contain Cluster Editing as a special case. This is also true for Module Map: if the input has no red edges, then it is not necessary to add red edges, and thus Module Map is the same as Cluster Editing in this case.

As a consequence, hardness results for Cluster Editing transfer directly to Module Map. Since Cluster Editing is NP-complete [22] and cannot be solved in $2^{o(|V|+|E|)}$ time under the Exponential-Time Hypothesis (ETH) [15, 21], we observe the following.

Proposition 1. Module Map *is NP-complete. If the ETH is true, then* Module Map *cannot be solved in* $2^{o(|V|+|E_b|+|E_r|)}$ *time.*

Because of this algorithmic hardness, heuristic approaches are used in practice [4, 25]. In this work, we are interested in exact algorithms for Module Map. In particular, we are interested in fixed-parameter algorithms. These are algorithms that have a running time of $f(k) \cdot n^{O(1)}$ for a problem-specific parameter k on inputs of size n. If k has moderate values and f grows not too fast, then these algorithms solve the problem efficiently [11–13, 23]. Motivated by the practical success of fixed-parameter algorithms with the natural parameter number k of edge modifications for Cluster Editing [8, 18], we focus on fixed-parameter algorithms for Module Map with the same parameter. We find that viewing Module Map as a graph modification problem facilitates the algorithmic study of the problem.

A weighted problem variant. In practice, for some edges we may have a higher confidence in the observed edge type. For this reason it is useful to consider edge-weighted versions of the problem, where the input includes a weight function $g : \binom{V}{2} \to \mathbb{N}^+$ on vertex pairs. The higher the weight, the more confidence we have in the observed edge type. Let $\omega := g(\{u, v\})$ be the weight of the vertex pair $\{u, v\}$. If $\{u, v\}$ is a red edge or a blue edge, then deleting $\{u, v\}$ costs ω; if $\{u, v\}$ is not an edge in G, then inserting a blue edge $\{u, v\}$ or a red edge $\{u, v\}$ costs ω and, finally, replacing a blue edge $\{u, v\}$ by a red edge $\{u, v\}$ or vice versa costs 2ω. In other words, each single modification of a pair $\{u, v\}$ produces cost $g(\{u, v\})$. This gives the following problem.

Weighted Module Map
Input: A graph $G = (V, E_b, E_r)$ with vertex-pair weights $g : \binom{V}{2} \to \mathbb{N}^+$ and a nonnegative integer k.
Question: Can we transform G into a module graph by edge modifications of total cost at most k?

Our results. In Section 3, we present a characterization of module graphs by three forbidden induced subgraphs and show how to determine whether a graph G

contains one of them in linear time. Afterwards, we present a simple linear-time fixed-parameter algorithm for Module Map with running time $\mathcal{O}(3^k \cdot (|V| + |E_b| + |E_r|))$. We further obtain an algorithm for Weighted Module Map with running time $\mathcal{O}(3^k \cdot |V|^2)$. This is also a linear-time fixed-parameter algorithm since the weight function g has size $\Theta(|V|^2)$.

In Section 4, we present an improved (in terms of the exponential running-time part) fixed-parameter algorithm for Weighted Module Map with running time $\mathcal{O}(2^k \cdot |V|^3)$. This algorithm is an extension of a previous algorithm for Weighted Cluster Editing [7]. In order to transfer the technique to Weighted Module Map, we solve a more general variant of Weighted Module Map that uses a condensed view of the modification costs of an edge in terms of cost vectors. Here, each possible type of a vertex pair (being a blue edge, a red edge, or a non-edge) corresponds to one component of the cost vector. We believe that this view can be useful for other graph modification problems with multiple edge types.

In Section 5, we show that Weighted Module Map admits a kernel with a quadratic number of vertices. More precisely, we show that given an instance of Weighted Module Map, we can compute in $\mathcal{O}(|V|^3 + k \cdot |V|^2)$ time an equivalent instance that has $\mathcal{O}(k^2)$ vertices. As a corollary, we can solve Weighted Module Map in $\mathcal{O}(2^k \cdot k^6 + |V|^3)$ time by first applying the reduction rules leading to the kernelization and then using the search tree algorithm.

Finally, in Section 6 we consider for two of the three forbidden induced subgraphs of module graphs, the problem of finding them efficiently in a bicolored graph. We show that any linear-time algorithm for finding these forbidden induced subgraphs would result in improved algorithms for the problem of detecting triangles in undirected simple graphs without edge colors.

Related work. Compared to the study of graphs with only one edge type, there has been little work on algorithms for graphs with multiple edge types which may be referred to as multilayer graphs [20] or edge-colored (multi)graphs. In the following, we point to work that studies edge modification problems in such graphs.

Chen et al. [10] introduced Multi-Layer Cluster Editing, a variant of Cluster Editing. In this problem, the input is a multi-layer graph and one asks to transform all layers into cluster graphs which differ only slightly. Here, a layer is the subgraph containing only the edges of one type. Roughly speaking, the task is to find one cluster graph such that each layer can be transformed into this cluster graph by at most k edge modifications. Chen et al. [10] show fixed-parameter algorithms and hardness results for different parameter combinations. The problem differs from Module Map in the sense that all layers, which correspond to edge colors in our formulation, play the same role in the problem definition and that layers are evaluated independently. In contrast, in Module Map the aim is to obtain a graph with blue and red edges that fulfills different properties for the blue and red edges and where the property of the red edges depends on the blue edges.

A further edge modification problem on multilayer graphs is Simultaneous

Feedback Edge Set [2] where the aim is to delete at most k edges in a multilayer graph with ℓ layers such that each layer is acyclic. Simultaneous Feedback Edge Set is polynomial-time solvable if $\ell = 2$, NP-hard for all $\ell > 3$, and fixed-parameter tractable for the parameter $k + \ell$ [2].

2. Preliminaries

Graph notation. A simple and undirected graph $G = (V, E)$ consists of a finite set V of *vertices* and a finite set E of *edges*, which are unordered pairs of vertices of V. For a graph $G = (V, E)$, the *subgraph of G induced by a set $V' \subseteq V$* is denoted by $G[V'] := (V', \{\{u, v\} \in E \mid u, v \in V'\})$. A graph G is a *cluster* graph if each connected component of G is a complete graph. A graph $G = (V, E)$ is *bipartite* if V can be partitioned into U and W such that each edge in E has exactly one endpoint in U and exactly one in W. A bipartite graph with partite sets U and W and $|E| = |U| \cdot |W|$ is called *biclique*.

For a color set C, an *edge-$|C|$-coloring* of a graph is an assignment of exactly one color $c \in C$ to each edge in the graph. It follows from this definition that the edge sets of distinct colors are pairwise disjoint; we use E_c to denote the edges of color c. With $G_c := (V, E_c)$ we denote the subgraph of G *induced by the edges with color c*. In this work, we consider an edge-bicoloring with the colors blue and red, that is, the edge-bicolored graph $G = (V, E_b, E_r)$ has a set E_b of blue edges and a set E_r of red edges. We let $E := E_b \cup E_r$ denote the union of the blue and red edges. We denote the *blue subgraph* (V, E_b) by G_b and the *red subgraph* (V, E_r) by G_r. The *open neighborhood with color c* of a vertex v is defined as $N_c(v) := \{u \mid \{u, v\} \in E_c\}$. Furthermore, $N_c[v] := N_c(v) \cup \{v\}$ is the *closed neighborhood with color c* of v.

For two sets A and B, the *symmetric difference* $A \triangle B := (A \cup B) \setminus (A \cap B)$ is the set of elements which are in exactly one of the two sets. A *solution* $\mathcal{S}$ for an instance of Module Map is a tuple of edge modifications (E'_b, E'_r) of size at most k such that the transformed graph $G' = (V, E_b \triangle E'_b, E_r \triangle E'_r)$ is a module graph. Herein, the *size* of a solution (E'_b, E'_r) is $|E'_b| + |E'_r|$. The graph G' obtained from applying a solution $\mathcal{S}$ is called *target graph*. A solution $\mathcal{S}$ is *optimal* if every other solution is at least as large as $\mathcal{S}$.

Parameterized algorithmics. A *parameterized problem* L is a set of instances of the form (I, k) where $I \in \Sigma^*$ for a finite alphabet Σ, and $k \in \mathbb{N}_0$ is the parameter. A parameterized problem L is *fixed-parameter tractable* (FPT) with respect to k if it can be determined in $f(k)|I|^{\mathcal{O}(1)}$ time whether $(I, k) \in L$, where f is a computable function not depending on I.

Let L be a parameterized problem. A *kernelization* is a polynomial-time algorithm that replaces any instance (I, k) by an instance (I', k'), called *kernel*, such that

- $|I'| + k' \leq g(k)$ for some function g which depends only on k, and
- $(I, k) \in L$ if and only if $(I', k') \in L$.

The function $g(k)$ is called the *size* of the kernel. For more information on parameterized complexity, we refer to the literature [11–13, 23].

Kernels are often presented via *reduction rules*. A reduction rule for a parameterized problem L is a computable function that maps an instance (I, k) of L to an instance (I', k') of L. A reduction rule is called *safe* if (I, k) and (I', k') are equivalent, that is, $(I, k) \in L$ if and only if $(I', k') \in L$. An instance is *reduced exhaustively* with respect to a reduction rule if an application of the rule does not change the instance. A *branching rule* transforms an instance (I, k) of a parameterized problem into ℓ instances $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ of the same problem such that $k_i < k$ for each i, $1 \leq i \leq \ell$. A branching rule is *safe* if $(I, k) \in L$ if and only if $(I_i, k_i) \in L$ for at least one i. A standard tool in the analysis of search tree algorithms are branching vectors; for further background on the analysis of branching vectors, refer to the monograph of Fomin and Kratsch [14].

3. Basic Observations

Next, we present a forbidden subgraph characterization for Module Map. Afterwards, we prove that a forbidden subgraph of Module Map can be detected in $\mathcal{O}(|V| + |E_r| + |E_b|)$ time and present an $\mathcal{O}(3^k)$ search tree algorithm for Module Map.

Recall that an uncolored graph G is a cluster graph if and only if G contains no P_3 as an induced subgraph: If a connected component of G contains a P_3, then there exist two vertices in this component which are not adjacent and, hence, G is not a cluster graph. Conversely, if a connected component C of G contains no induced P_3, then each triple of vertices of C has three edges between them. Hence, C is a clique. Moreover, it is well-known that using the following algorithm it can be checked in linear time whether G contains an induced P_3. For each connected component C of G do the following until a P_3 is found: First, check in linear time whether $|N(v)| = |C| - 1$ for all $v \in C$. If yes, C is a clique. Otherwise, any vertex v with $|N(v)| < |C| - 1$ is an endpoint of a P_3 and with a breadth-first search starting in v, a P_3 can be detected. Altogether, this gives the following well-known fact.

Proposition 2. *Let $G = (V, E)$ be a graph. In $\mathcal{O}(|V| + |E|)$ time, one can decide whether G is a cluster graph and find an induced P_3 in G if this is not the case.*

In the following we present a forbidden subgraph characterization for the property of being a module graph. To this end, we define the following three graphs which are shown in Figure 2: a *blue* P_3 is a path on three vertices consisting of two blue edges, an *almost-blue* K_3 is a clique of size three, where one edge is red and the other two are blue, and a *bicolored* P_3 is a path on three vertices with exactly one blue and one red edge.

Theorem 1. *A bicolored graph G is a module graph if and only if G has no blue P_3, no almost-blue K_3, and no bicolored P_3 as induced subgraph.*

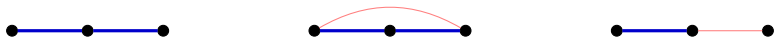


Figure 2: The forbidden induced subgraphs for module graphs. From left to right: a blue P_3, consisting of two (dark and bold) blue edges, an almost-blue K_3, consisting of two blue and one (bright) red edge, and a bicolored P_3, consisting of one blue and one red edge.

To prove Theorem 1 we first show that the blue subgraph G_b is a cluster graph if and only if G contains no blue P_3 and no almost-blue K_3.

Lemma 1. *A graph G fulfills the cluster property if and only if G contains neither a blue P_3 nor an almost-blue K_3 as induced subgraph.*

Proof. ($\Rightarrow$) If G_b is a cluster graph, then for all blue edges $\{u, v\} \in E_b$ and all $w \in V \setminus \{u, v\}$ either $\{u, w\}$ and $\{v, w\}$ are both contained in E_b or none of them is contained in E_b. In either case, the subgraph induced by u, v, and w is neither a blue P_3 nor an almost-blue K_3. Hence, no blue edge is contained in one of these two forbidden induced subgraphs and consequently G does not contain such an induced subgraph.

($\Leftarrow$) We prove the statement by contraposition. That is, we show that if a graph G does not fulfill the cluster property, then it contains either a blue P_3 or an almost-blue K_3 as induced subgraph. If G does not fulfill the cluster property, then G_b is not a cluster graph which means that G_b contains a P_3 as induced subgraph. Let $\{u, v\}$ and $\{v, w\}$ be the two edges of this P_3. The edges $\{u, v\}$ and $\{v, w\}$ are blue edges of G and $\{u, w\}$ is not a blue edge of G since it is not an edge of G_b. Hence, $\{u, w\}$ is either a non-edge or a red edge. In the first case, G contains a blue P_3 as induced subgraph; in the second case G contains an almost-blue K_3 as induced subgraph. □

Lemma 1 implies that the blue P_3 and the almost-blue K_3 are forbidden induced subgraphs for module graphs. We are now ready to present the complete forbidden subgraph characterization.

Proof (of Theorem 1). ($\Rightarrow$) Let G be a module graph. Because G fulfills the cluster property, the graph G_b induced by the blue edges is a cluster graph. By Lemma 1, G contains neither a blue P_3 nor an almost-blue K_3. Now observe that any bicolored P_3 contains exactly two vertices from a cluster K and one vertex from a cluster $L \neq K$ of G_b. Since G fulfills the link property, we have for each pair of clusters K and L in G_b that $G_r[K \cup L]$ is either edgeless or a biclique. In both cases, $G[K \cup L]$ contains no bicolored P_3. Hence, G contains no bicolored P_3.

($\Leftarrow$) Let G be a graph which contains no blue P_3, no almost-blue K_3, and no bicolored P_3 as induced subgraph. By Lemma 1, G fulfills the cluster property. We now show by contraposition that G fulfills the link property. That is, we show that if G does not fulfill the link property then it contains a bicolored P_3. Thus, assume that there is a pair of clusters K and L in G_b such that some but not all edges between K and L are red. Let $\{u, v\}$ be a red edge between K and L and let $\{u', v'\}$ be a non-edge between K and L. Without

loss of generality, assume $\{u, u'\} \subseteq K$, $\{v, v'\} \subseteq L$, and $v \neq v'$. If $u = u'$, then $G[\{u, v, v'\}]$ is a bicolored P_3 because $\{v, v'\}$ is a blue edge. If $u \neq u'$ and $\{u', v\}$ is a non-edge in G, then $G[\{u, u', v\}]$ is a bicolored P_3. Otherwise, $G[\{u', v, v'\}]$ is a bicolored P_3: $\{v, v'\}$ is blue, $\{u', v\}$ is red, and $\{u', v'\}$ is a non-edge. Hence, G contains a bicolored P_3 if it does not fulfill the link property. □

From the forbidden induced subgraph characterization of a module graph we can immediately observe the following.

Proposition 3. *Let $u, v \in V$ be two vertices that are connected by a blue edge in a module graph. Then $N_b[u] = N_b[v]$ and $N_r(u) = N_r(v)$, that is, the closed blue neighborhood and the open red neighborhood of u and v are identical.*

We now show a simple linear-time fixed-parameter algorithm for Module Map and Weighted Module Map. The algorithm uses the standard approach to branch on the graphs of the forbidden subgraph characterization presented in Theorem 1. The main point is to obtain a linear running time for fixed k. To this end, we show that we can determine in $\mathcal{O}(|V| + |E|)$ time if a graph contains any of the three forbidden subgraphs.

We start by determining if the blue subgraph G_b of the input graph $G = (V, E_b, E_r)$ is a cluster graph. According to Lemma 1, we have to determine if G has a blue P_3 or an almost-blue K_3.

Lemma 2. *Given an edge-bicolored graph $G = (V, E_b, E_r)$, we can find in $\mathcal{O}(|V| + |E|)$ time a blue P_3 or an almost-blue K_3, or decide that G contains no blue P_3 and no almost-blue K_3 as induced subgraph.*

Proof. If G contains a blue P_3 or an almost-blue K_3, then by Lemma 1, G_b is not a cluster graph. Hence, the graph G_b contains a P_3. According to Proposition 2, a P_3 in G_b can be detected in $\mathcal{O}(|V| + |E|)$ time. If we do not find such a P_3, we conclude that G contains no blue P_3 and no almost-blue K_3. Otherwise, let $\{u, v, w\}$ denote the vertex set of the P_3 and assume that u and w are not adjacent in G_b. Then $\{u, w\}$ is either a non-edge in G, making $G[\{u, v, w\}]$ a blue P_3, or $\{u, w\}$ is a red edge in G, making $G[\{u, v, w\}]$ an almost-blue K_3. Thus, returning $\{u, v, w\}$ yields the desired forbidden induced subgraph. □

Now we show how to find a bicolored P_3 in time $\mathcal{O}(|V| + |E|)$ in a graph $G = (V, E_b, E_r)$ when we assume that G contains no blue P_3 and no almost-blue K_3. The idea is to find two clusters of G_b such that at least one but not all edges between these clusters are red.

Lemma 3. *Let $G = (V, E_b, E_r)$ be a graph which contains no blue P_3 and no almost-blue K_3 as induced subgraph. Then, in $\mathcal{O}(|V| + |E|)$ time we can find a bicolored P_3 in G or decide that G contains no bicolored P_3 as induced subgraph.*

Proof. According to Lemma 1, we can assume that the graph G_b is a cluster graph. In a first step, determine in $\mathcal{O}(|V| + |E|)$ time the sizes $c_1, \ldots, c_\ell$ of the

clusters $C_1, \ldots, C_\ell$ in G_b. Now create an array A of size ℓ. The ith entry of A will be used to count the number of red neighbors of a vertex v in the cluster C_i.

Now do the following for each vertex $v \in V$: Create a list L_v consisting of the red neighbors of the vertex v. For each vertex u in L_v determine the cluster C_r that contains u and increase $A[r]$ by one. Simultaneously, save the indices of the entries of the array A which are different from zero in a set I. Afterwards, check for each $i \in I$ whether $A[i] = c_i$. If this is the case for all $i \in I$, set $A[i] = 0$ for all $i \in I$, set $I = \emptyset$, and investigate the next vertex with this algorithm. If this is not the case, then v has at least one but less than c_i red neighbors in some cluster C_i. Now do the following. First, color each vertex $u \in C_i$ white. Next, color each vertex $w \in C_i$ which has a red edge to v black. This can be done by a linear scan over L_v. Now determine a white vertex x and a black vertex y in the cluster C_i by a linear scan over all vertices in C_i. The vertex set $\{v, x, y\}$ forms a bicolored P_3, since $\{v, x\}$ is a non-edge, $\{v, y\}$ is red, and $\{x, y\}$ is blue. Thus, we return $\{v, x, y\}$.

If the algorithm does not return any bicolored P_3, then for each vertex v in any cluster C_i and each cluster C_j, $i \neq j$, either v forms a red edge with every vertex of C_j or there are no edges between v and C_j. Consequently, no vertex is contained in a bicolored P_3.

It remains to bound the running time of the algorithm: To compute for a vertex v the number of red neighbors of v in every other cluster C_i and storing it in $A[i]$, we scan once through the adjacency list of v which takes $\mathcal{O}(|N_r(v)|)$ time. Comparing $A[i]$ with c_i for all $i \in I$ takes $\mathcal{O}(|N_r(v)|)$ time as well. Hence, this part of the algorithm takes $\mathcal{O}(\sum_{v \in V} |N_r(v)|) = \mathcal{O}(|V|+|E|)$ time. If G contains a bicolored P_3, at some point the algorithm will choose a vertex v which has at least one but less than c_i red neighbors in some cluster C_i. For this vertex v, the algorithm performs one extra scan of the list L_v and of the cluster C_i. This can be done in $\mathcal{O}(|V| + |E|)$ time. Hence, the overall running time is $\mathcal{O}(|V| + |E|)$. □

According to Theorem 1, a bicolored graph G is a module graph if and only if G has no blue P_3, no almost-blue K_3, and no bicolored P_3 as induced subgraphs. With Lemmas 2 and 3 at hand, it can be determined in $\mathcal{O}(|V|+|E|)$ time if a graph $G = (V, E_b, E_r)$ contains one of those three forbidden subgraphs and, thus, also whether G is a module graph. A simple fixed-parameter algorithm for Module Map now works as follows: Check whether G is a module graph. If this is the case, then return 'yes'. Otherwise, check whether $k = 0$. If this is the case, return 'no'. Otherwise, find one of the three forbidden subgraphs and branch on the possibilities to destroy it by an edge modification as follows.

Branching Rule 1. *Let (G, k) be an instance of* Module Map. *If G contains three vertices u, v, and w such that $G[\{u, v, w\}]$ is either a blue P_3 with non-edge $\{u, w\}$, or an almost-blue K_3 with red edge $\{u, w\}$, or a bicolored P_3 with blue edge $\{u, v\}$ and red edge $\{v, w\}$, then branch into three cases:*

Case 1: Transform $\{u, v\}$ into a non-edge and decrease k by 1.

Case 2: Transform $\{v, w\}$ into a non-edge and decrease k by 1.

Case 3: If $G[\{u,v,w\}]$ is a blue P_3, transform $\{u,w\}$ into a blue edge and decrease k by 1. If $G[\{u,v,w\}]$ is an almost-blue K_3, transform $\{u,w\}$ into a blue edge and decrease k by 2. If $G[\{u,v,w\}]$ is a bicolored P_3, transform $\{u,w\}$ into a red edge and decrease k by 1.

Lemma 4. *Branching Rule 1 is safe.*

PROOF. Clearly, if one of the three instances created by the branching rule is a yes-instance, then so is (G,k).

Conversely, assume (G,k) is a yes-instance for MODULE MAP. Let $\mathcal{S}$ be a solution for (G,k) and let H be the corresponding target graph. First, assume that the type of $\{u,v\}$ in H is different from the type of $\{u,v\}$ in G. Then, $\mathcal{S}$ deletes the blue edge $\{u,v\}$ and thus the instance constructed in the first branch is a yes-instance. In the following, assume that the type of $\{u,v\}$ in G and H are identical. Next, assume that the type of $\{v,w\}$ in H is different from the type of $\{v,w\}$ in G. Then, $\mathcal{S}$ deletes the edge $\{v,w\}$ and thus the instance constructed in the second branch is a yes-instance. Finally, consider the case that $\{v,w\}$ has the same type in G and H. Now we apply Theorem 1 to $G[\{u,v,w\}]$. If $G[\{u,v,w\}]$ is a blue P_3 or an almost-blue K_3 and $\{u,v\}$ and $\{v,w\}$ are blue in H, then $\{u,w\}$ is blue in H too. This means that if $G[\{u,v,w\}]$ is a blue P_3, then the solution inserts the blue edge $\{u,w\}$, and if $G[\{u,v,w\}]$ is an almost blue K_3, then the solution deletes the red edge $\{u,w\}$ and inserts the blue edge $\{u,w\}$. If $G[\{u,v,w\}]$ is a bicolored P_3 and $\{u,v\}$ is blue and $\{v,w\}$ is red in H, then $\{u,w\}$ is red in H, too. Then, the solution inserts the red edge $\{u,w\}$. In all cases, the instance created in the third branch is a yes-instance. □

The algorithm branches into three cases and decreases k by at least 1 in each case. This leads to a branching vector of $(1,1,1)$. Since branching is performed only while $k > 0$, the overall search tree size is $\mathcal{O}(3^k)$; the steps of each search tree node can be performed in $\mathcal{O}(|V|+|E|)$ time by Lemmas 2 and 3. Altogether, we obtain the following.

Proposition 4. MODULE MAP *can be solved in $\mathcal{O}(3^k \cdot (|V|+|E|))$ time.*

For WEIGHTED MODULE MAP, we can use the same algorithm: since the edge weights are positive integers, the parameter decrease is again at least 1 in each created branch of the search tree algorithm. A subtle difference is that, due to the edge weight function g, the overall instance size is $\Theta(|V|^2)$. Hence, the following running time is still linear for each fixed k.

Proposition 5. WEIGHTED MODULE MAP *can be solved in $\mathcal{O}(3^k \cdot |V|^2)$ time.*

4. An Improved Search Tree Algorithm

To improve the running time, we adapt a branching strategy for CLUSTER EDITING [7]. To apply this strategy, we first introduce a generalization of WEIGHTED MODULE MAP. Then, we explain our branching strategy which

consists of two branching rules. Finally, we show that the instances to which these rules do not apply can be solved in polynomial time. Altogether, this leads to an $\mathcal{O}(2^k \cdot |V|^3)$-time search tree algorithm.

4.1. A More Flexible Cost Function

To describe our algorithm for Weighted Module Map, we introduce a more general problem since during branching, we will *merge* some vertices. To represent the adjacencies of the merged vertices, we generalize the concept of edge weights: Recall that in Weighted Module Map, transforming a blue edge with weight ω into a non-edge costs ω and transforming it into a red edge costs 2ω. Hence, the two modification costs are related. From now on, we allow independent modification costs for the different possibilities. To this end, we introduce a *vertex-pair cost function* $s : \binom{V}{2} \to \mathbb{R}^3$ for all pairs of vertices $\{u, v\}$ of a given graph G where $s(u, v) := (b_{u,v}, n_{u,v}, r_{u,v})$. The vector $(b_{u,v}, n_{u,v}, r_{u,v})$ is called *cost vector*. Herein, $b_{u,v}$ is the cost of making $\{u, v\}$ blue, $n_{u,v}$ is the cost of making $\{u, v\}$ a non-edge, and $r_{u,v}$ is the cost of making $\{u, v\}$ red. For a short form of the cost vector we write $(b, n, r)_{u,v}$. If there is no danger of confusion, we omit the index of the associated vertices u and v. Consider for example a blue edge $\{u, v\}$ in an instance of Weighted Module Map with weight ω. The corresponding modification cost would be represented by the cost vector $(0, \omega, 2\omega)$. Given a vertex-pair cost function s, we define the cost of a module graph as follows.

Definition 2. *Let $s : \binom{V}{2} \to \mathbb{R}^3$ with $s(u, v) := (b_{u,v}, n_{u,v}, r_{u,v})$ be a vertex-pair cost function. The* cost of a graph $G' = (V, E'_b, E'_r)$ *on the vertex set V (with respect to s) is*

$$\sum_{\{u,v\} \in E'_b} b_{u,v} + \sum_{\{u,v\} \in E'_r} r_{u,v} + \sum_{\{u,v\} \in \binom{V}{2} \setminus (E'_b \cup E'_r)} n_{u,v}.$$

We do not allow arbitrary cost vectors but demand the following four properties. Herein, a number x is called *half-integral* if and only if $x = p + 1/2$ for some nonnegative integer p.

Property 1. *In each cost vector $(b, n, r)_{u,v}$ either all components are nonnegative integers or all components are half-integral.*

A cost vector $(b, n, r)_{u,v}$ where all three components are half-integral is called *half-integral* as well. All other cost vectors are called *integral*. Half-integral cost vectors will be introduced during the algorithm for technical reasons. The second property restricts the relation between the three costs.

Property 2. *For each cost vector $(b, n, r)_{u,v}$ we have $b + r \geq 2n$.*

Property 2 is essentially a relaxed version of the property that transforming a blue edge into a red edge is at least as expensive as transforming the blue edge first into a non-edge and subsequently into a red edge. The third property guarantees that for every given cost function, there is at most one graph that has cost zero.

Property 3. *Each integral cost vector $(b, n, r)_{u,v}$ contains exactly one component which is equal to zero.*

The final property demands that at least two components of a half-integral cost vector are small.

Property 4. *Each half-integral cost vector $(b, n, r)_{u,v}$ contains at least two components equal to $1/2$.*

With these definitions at hand, we may now state Module Map with Cost Function (MMC).

MMC
Input: A vertex set V with a vertex-pair cost function $s : \binom{V}{2} \to \mathbb{R}^3$ which fulfills Properties 1–4 and an integer k.
Question: Is there a module graph $G' = (V, E'_b, E'_r)$ of cost at most k?

We call a module graph G' of cost at most k a *target graph* for s. Properties 1–4 are fulfilled by the cost function obtained from Weighted Module Map instances. Thus, Weighted Module Map is a special case of MMC and, hence, MMC is also NP-complete. Our aim is to show the following.

Theorem 2. MMC *can be solved in $\mathcal{O}(2^k \cdot |V|^3)$ time.*

Before we describe the algorithm, we provide further details for the cost function and for the rationale behind Properties 1–4. First, observe that due to Properties 3 and 4, we may put each vertex pair $\{u, v\}$ into exactly one of the following four categories: We call a vertex pair $\{u, v\}$ with cost vector (b, n, r) *blue* if $b = 0$, *neutral* if $n = 0$, *red* if $r = 0$, and *half-integral* otherwise.

Moreover, we can observe the following fact which will be crucial in several correctness proofs.

Proposition 6. *Let $(b, n, r)_{u,v}$ be a cost vector fulfilling Properties 1–4.*

- *If $\{u, v\}$ is blue, then $n \geq 1$ and $r \geq 2n$.*
- *If $\{u, v\}$ is red, then $n \geq 1$ and $b \geq 2n$.*
- *If $\{u, v\}$ is half-integral, then $n = 1/2$.*

Proof. The first two claims follow from the fact that (b, n, r) has only integer components in these cases, that only one component is zero, and that $b + r \geq 2n$ due to Property 2. The third claim can be seen as follows. By Property 1, all three components are half-integral and at least two of them are equal to $1/2$. By Property 2, $b + r \geq 2n$. If $n > 1/2$, then $b = 1/2 = r$ according to Property 4 and hence Property 2 is violated. Thus, $n = 1/2$. □

4.2. Merge-Based Branching

As for unweighted graphs, the overall strategy is to branch on small substructures whose presence implies that there is no module graph with cost 0. To this end, recall that by Proposition 3, the endpoints of a blue edge in a module graph have the same closed blue neighborhood and the same open red neighborhood. Thus, in the unweighted case every blue edge whose endpoints have different blue or different red neighbors is contained in a forbidden subgraph on which we may branch. In MMC, we follow a similar strategy. A subtle difference is that now costs may also be caused by half-integral cost vectors. We define the configurations on which we branch as follows.

Definition 3. *A blue pair $\{u, v\}$ forms a* conflict triple *with a vertex w if the vertex pairs $\{u, w\}$ and $\{v, w\}$ are not both blue, not both neutral, or not both red.*

In other words, the vertices u, v, and w form a conflict triple if $\{u, w\}$ and $\{v, w\}$ have different type or if $\{u, w\}$ and $\{v, w\}$ are both half-integral.

The branching on a blue pair $\{u, v\}$ follows the approach that Böcker et al. [7] used for Cluster Editing: In one case we increase $b_{u,v}$ in such a way that $\{u, v\}$ does not become a blue edge in any target graph. In the other case, we keep the blue pair and use Proposition 3 which states that vertices which are connected by a blue edge have the same neighborhood in a module graph. This allows us to *merge* the vertex pair $\{u, v\}$ into a new vertex u' in this case.

Definition 4. *Let (V, s, k) be an instance of* MMC. Merging *two vertices u and v is the following operation: Remove u and v from V and add a new vertex u'. For all vertices $w \in V \setminus \{u, v\}$ set $s(u', w) := s(u, w) + s(v, w)$.*

We call $s(u', w)$ the *join* of $s(u, w)$ and $s(v, w)$.

In both cases of the branching, we *reduce* the modified cost vectors by applying what we call a *restoration rule*. More precisely, we say that the restoration rule *reduces a cost vector by x* if the value of each entry is decreased by x. There are two reasons for introducing this rule. First, the modified cost vectors may not fulfill Properties 1–4. Second, increasing $b_{u,v}$ for a vertex pair $\{u, v\}$ or merging the two vertices u and v does *not* decrease the parameter k. The restoration rule will thus decrease the bound k and, simultaneously, reduce the new cost vectors in such a way that they fulfill Properties 1–4.

The overall strategy is now as follows: If $k < 0$, then the instance (V, s, k) has no solution. Else, find a blue pair $\{u, v\}$ which is either contained in at least two conflict triples or there exists a vertex w such that merging u and v into v' results in reducing the joint cost vector $s(u', w)$ by at least 1. If there is no such blue pair, then solve the remaining instance of MMC in polynomial time. The pseudocode of this algorithm is shown in Algorithm 1.

Let $x = \min\{b, n, r\}$ be a minimal value of the cost vector (b, n, r). If x is unique, then we can reduce (b, n, r) by x, since afterwards exactly one component of the cost vector is equal to zero. Otherwise, we cannot reduce the cost vector by x, since afterwards at least two components have value zero, a contradiction to Property 3. Clearly, we could reduce the vector by $x - 1$, but this would not

Algorithm 1 The $\mathcal{O}(2^k \cdot |V|^3)$ algorithm for MMC.

Input: An instance (V, s, k) of MMC
1: **if** $k < 0$ **then**
2: reject
3: **else if** (V, s, k) contains blue pair $\{u, v\}$ that is in at least two conflicts **then**
4: apply Branching Rule 2
5: **else if** (V, s, k) contains blue pair $\{u, v\}$ and a vertex $w \in V \setminus \{u, v\}$ such that merging u and v into u' decreases $s(u', w)$ by at least 1 **then**
6: apply Branching Rule 3
7: **else**
8: solve (V, s, k) in $\mathcal{O}(|V|^2)$ time (Theorem 3)
9: **end if**

give a parameter decrease for vectors such as $(1, 1, 3)$. Using the bookkeeping trick introduced in [7], in such a case, we reduce this vector by $x - 1/2$ to circumvent the above problem.

Restoration Rule 1. *Let $\{u, v\}$ be a vertex pair with the cost vector (b, n, r). If (b, n, r) has a unique minimum component, then decrease each component of (b, n, r) and parameter k by $\min\{b, n, r\}$. Otherwise, decrease each component of (b, n, r) and parameter k by $\min\{b, n, r\} - 1/2$.*

For example, Restoration Rule 1 reduces the vector $(1, 1, 3)$ by $1/2$ giving the vector $(1/2, 1/2, 5/2)$. We may now formulate the first branching rule.

Branching Rule 2. *Let (V, s, k) be an instance of* MMC*. If (V, s, k) contains a blue pair $\{u, v\}$ and two distinct vertices w and w' that form a conflict triple with $\{u, v\}$, then branch into two cases:*

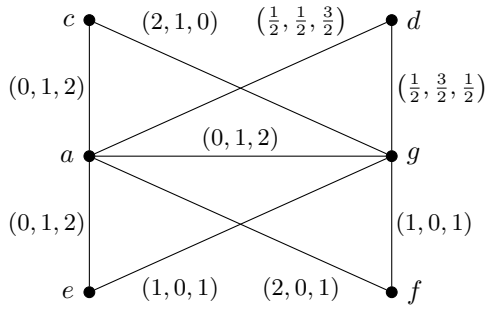
Case 1: Set $b_{u,v} := k + 1$. Afterwards, apply Restoration Rule 1 to the vertex pair $\{u, v\}$.

Case 2: Merge the vertices u and v into a new vertex u'. Afterwards, for each vertex $z \in V \setminus \{u'\}$ apply Restoration Rule 1 to the vertex pair $\{u', z\}$.

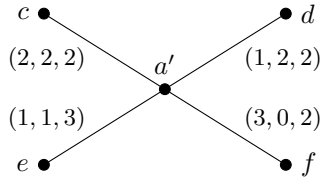
An example of Case 2 of Branching Rule 2 is shown in Figure 3. Now we prove that Properties 1–4 are maintained in both cases of the branching rule.

Lemma 5. *Let $\{u, v\}$ be a blue pair with cost vector (b, n, r) in an instance (V, s, k) of* MMC*, and let (V', s', k') be obtained by an application of Branching Rule 2 for $\{u, v\}$. Then, (V', s', k') is an instance of* MMC*. In particular, s' fulfills Properties 1–4.*

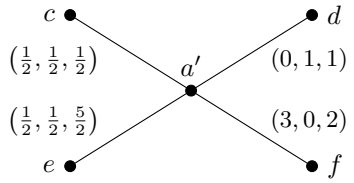
Proof. First, we show that for a blue pair $\{u, v\}$, setting $b := k + 1$ and applying Restoration Rule 1 afterwards, maintains Properties 1–4: Observe that since the vertex pair $\{u, v\}$ is blue, we have that each component of $s(u, v)$ is integral and $b = 0$. Since n and r are not changed, Property 2 remains true



(a) The initial instance (V, s, k) of MMC.



(b) The instance (V, s, k) after merging a and g into a'.



(c) The instance (V', s', k') of MMC after applying Restoration Rule 1. We have: $k' = k - 3$.

Figure 3: Example for merging two vertices a and g, and for subsequent application of Restoration Rule 1.

after setting $b := k + 1$. Furthermore, each component is integral and, hence, Property 1 remains true. Property 4 is fulfilled, since $\{u, v\}$ is not half-integral. Property 3, however, is violated since each component has value at least 1.

Applying Restoration Rule 1 decreases each of the three components by the same value. Hence, Property 2 remains true. Recall that according to Proposition 6, $r \geq 2n$. Let (b', n', r') denote the reduced cost vector. If $n = k + 1$, applying Restoration Rule 1 reduces this cost vector by $k + 1/2$ and in the reduced cost vector (b', n', r') the components b' and n' are equal to $1/2$. Furthermore, r' is half-integral. If $n > k + 1$, then Restoration Rule 1 reduces this cost vector by $k + 1$ and only b' is equal to zero. Furthermore, n' and r' are integral. If $n < k + 1$, then Restoration Rule 1 will reduce the cost vector by n and afterwards only n' is equal to zero. Furthermore, b' and r' are integral. Hence, Properties 1–4 remain true.

Second, we prove that merging the vertices u and v into a new vertex u' and applying Restoration Rule 1 afterwards, maintains Properties 1–4: After merging, we compute for each $z \in V \backslash \{u, v\}$ the joint cost vector $s(u', z)$. According to the requirements, the cost vectors $s(u, z)$ and $s(v, z)$ fulfill Properties 1–4. Adding both inequalities implied by Property 2 gives $b_{u,z} + b_{v,z} + r_{u,z} + r_{v,z} \geq 2n_{u,z} + 2n_{v,z}$. Using the definition of the new cost vector $s(u', z)$, we conclude that $b_{u',z} + r_{u',z} \geq 2n_{u',z}$. Applying Restoration Rule 1 decreases each compo-

nent of $s(u', z)$ by the same value. Hence, Property 2 remains true.

To verify Properties 1, 3, and 4, observe first that joining two vectors which fulfill Properties 1, 3, and 4 gives a joint cost vector that is either integral or half-integral: If either both terms of the sum are nonnegative integers or half-integral, then the resulting cost vector is integral. If one cost vector is half-integral and the other consists of nonnegative integers, then the resulting cost vector is half-integral. Since applying Restoration Rule 1 to a cost vector fulfilling Property 1 does not violate Property 1, Property 1 remains true. Moreover, applying Restoration Rule 1 restores Properties 3 and 4: First, assume there is exactly one component of $(b, n, r)_{u',z}$ which has value $x = \min\{b_{u',z}, n_{u',z}, r_{u',z}\}$. Then we can reduce $(b, n, r)_{u',z}$ by the value x of the minimal component. Observe that it is not relevant whether $(b, n, r)_{u',z}$ is integral or half-integral. After reducing $(b, n, r)_{u',z}$ by x, the resulting cost vector $(b', n', r')_{u',z}$ consist of integers and exactly one component is equal to zero. Hence, Properties 3 and 4 are fulfilled. Second, assume there are at least two components of $(b, n, r)_{u',z}$ which have value x. In this case, we can reduce $(b, n, r)_{u',z}$ only by $x - 1/2$. Observe that it is not relevant whether $(b, n, r)_{u',z}$ is integral or half-integral. After reducing $(b, n, r)_{u',z}$ by $x - 1/2$, the resulting cost vector is half-integral and since at least two components have value x, the reduced cost vector has at least two components with value $1/2$. Hence, Properties 3 and 4 are fulfilled. We conclude: merging vertices u and v into a new vertex u' and applying Restoration Rule 1 afterwards maintains Properties 1–4. □

With this lemma at hand, we can prove the correctness of Branching Rule 2.

Lemma 6. *Branching Rule 2 is safe.*

Proof. By Lemma 5, Branching Rule 2 produces an instance of MMC. It remains to show that the original instance is a yes-instance if and only if one of the two new instances is a yes-instance.

First, assume (V, s, k) is a yes-instance and let H denote a target graph for (V, s, k). If $\{u, v\}$ is either a non-edge or a red edge in the target graph H, then H is also a target graph of the instance constructed in Case 1.

Second, assume the vertex pair $\{u, v\}$ is a blue edge in the target graph H. Hence, the vertices u and v are in the same cluster in H_b. According to Proposition 3, $N_b[u] = N_b[v]$ and $N_r(u) = N_r(v)$ in H. We may thus obtain a target graph H' for the instance constructed in Case 2, by assigning the vertex pair $\{u', w\}$ in H' the same type as the vertex pair $\{u, w\}$ for each $w \in V \setminus \{u, v\}$ in H. The cost of H' is the same as the cost of H since $s(u', w) := s(u, w) + s(v, w)$.

Conversely, assume that one of the two instances created in Case 1 and 2 is a yes-instance. First, any target graph H' of the instance constructed in Case 1 is a target graph of the same cost for (V, s, k) since $\{u, v\}$ is not a blue edge in H'. Second, any target graph H' for the instance created in Case 2 implies a target graph H of the same cost for (V, s, k): Replace u' by u and v, make $\{u, v\}$ a blue edge, and for each $w \in V \setminus \{u, v\}$, assign both edges $\{u, w\}$ and $\{v, w\}$ the same type as edge $\{u', w\}$. This step does not introduce new conflict triples

since u and v have the same neighborhood and hence we obtain a solution for the original instance (V, s, k). Moreover, the cost of H is the same as the cost of H' since $s(u', w) := s(u, w) + s(v, w)$ for all $w \in V \setminus \{u, v\}$. □

We now show that if we merge a blue pair $\{u, v\}$ into a new vertex u' where the vertices u and v form a conflict triple with some vertex w, then Restoration Rule 1 reduces the joint cost vector $(b, n, r)_{u',w}$ by at least 1/2.

Lemma 7. *Let $s(u', w)$ be the join of two cost vectors $s(u, w)$ and $s(v, w)$, where the vertices u, v, and w form a conflict triple. Then, Restoration Rule 1 applied to $s(u', w)$ decreases k by at least 1/2.*

Proof. We give a complete case distinction.

Case 1: $s(u, w)$ and $s(v, w)$ are half-integral. Then, each component of $s(u, w)$ and $s(v, w)$ has value at least 1/2. Consequently, the minimum component of $s(u', w)$ has value at least 1. Thus, $s(u', w)$ can be reduced by at least 1/2.

Case 2: $s(u, w)$ and $s(v, w)$ are integral. Since vertices u, v, and w form a conflict triple, the vertex pairs $s(u, w)$ and $s(v, w)$ have different type. Hence, there is no component that has value 0 in both vectors. Consequently, the minimum component of $s(u', w)$ has value at least 1. Thus, this cost vector can be reduced by at least 1/2.

Case 3: The cost vector $s(u, w)$ is half-integral and $s(v, w)$ is integral. Then, a minimum component of $s(u, w)$ has value exactly 1/2 and there is exactly one component of $s(v, w)$ that has value 0. If the minimum component of $s(u', w)$ is unique, then $s(u', w)$ can be reduced by at least 1/2. Otherwise, the minimum components have value at least 3/2 because one of them has value at least $1/2 + 1$. In this case, the parameter can be decreased by at least 1. □

Now we show that increasing b for blue pairs decreases k by at least 1.

Lemma 8. *Let $\{u, v\}$ be a blue pair and let (b^*, n, r) be the cost vector that results from (b, n, r) by setting $b := k + 1$. Then, applying Restoration Rule 1 to (b^*, n, r) decreases k by at least 1.*

Proof. Since $\{u, v\}$ is a blue pair, we have $n \geq 1$ and $r \geq 2n$ according to Proposition 6. Thus, applying Restoration Rule 1 to (b^*, n, r) reduces this vector and k by $\min\{b^*, n, r\} \geq 1$. □

Consider the instances obtained by an application of Branching Rule 2. In Case 1, the new parameter is at most $k - 1$ due to Lemma 8 since we may immediately reject the no-instances. In Case 2, the new parameter is also at most $k - 1$ because $\{u, v\}$ is in two conflict triples. By Lemma 7 this means that we create two cost vectors which are both reduced by at least 1/2.

Corollary 1. *Branching Rule 2 has a branching vector of $(1, 1)$ or better.*

We now introduce a second branching rule that also has a branching vector of $(1, 1)$ or better. This rule deals with blue pairs $\{u, v\}$ which are contained in exactly one conflict triple with some vertex w such that $s(u, w)$ and $s(v, w)$ give a join that can be reduced by at least 1.

Branching Rule 3. *Let (V, s, k) be an instance of* MMC. *If (V, s, k) contains a blue pair $\{u, v\}$ and a vertex w such that u, v, and w form a conflict triple and the joint cost vector $s(u', w) := s(u, w) + s(v, w)$ can be reduced by at least 1, then branch into the following two cases:*

Case 1: Set $b_{u,v} := k + 1$. Afterwards, apply Restoration Rule 1 to $\{u, v\}$.

Case 2: Merge *the vertices u and v into a new vertex u'. Afterwards, for each vertex $z \in V \setminus \{u'\}$ apply Restoration Rule 1 to $\{u', z\}$.*

Lemma 9. *Branching Rule 3 is correct and has branching vector $(1, 1)$ or better.*

Proof. The proof of correctness of this branching rule is the same as the proof of correctness of Branching Rule 2.

According to Lemma 8, the parameter of the instance created in Case 1 is at most $k - 1$. The parameter of the instance created in Case 2 is at most $k - 1$ according to the condition of Branching Rule 3. Hence, Branching Rule 3 has a branching vector of $(1, 1)$ or better. □

4.3. Solving The Remaining Instances in Polynomial Time

We now show that instances (V, s, k) of MMC to which Branching Rules 2 and 3 do not apply can be solved efficiently.

Theorem 3. *Let (V, s, k) be an instance of* MMC. *If Branching Rules 2 and 3 do not apply, then (V, s, k) can be solved in $\mathcal{O}(|V|^2)$ time.*

The polynomial-time algorithm behind the theorem consists of a series of data reduction rules. To formulate the reduction rules, we define the graph M_b of an instance (V, s, k) of MMC to be the graph with vertex set V and an edge for each blue pair in (V, s, k). In the rules, we make use of two properties of optimal solutions. The first property is the following.

Proposition 7. *Let (V, s, k) be an instance of* MMC. *Then, there exists a target graph H which does not contain any blue edges between vertices of different connected components of M_b.*

Proof. Let H be a target graph that has a minimum number t of blue edges between different connected components of M_b. If $t \geq 1$, then there is a cluster D in H_b such that D contains vertices from two different connected components K and L of M_b. In the following, we show that there is a target graph H' with two clusters $D_L := D \cap L$ and $D_K := D \setminus D_L$ in H'_b. More precisely, the graph H' is the same as H with the only difference that there are no blue edges with one

endpoint in D_L and one in D_K. Hence, H' has less blue pairs between different connected components of M_b.

First, we show that H' is a module graph. Observe that H' differs from H only with respect to edges with one endpoint in D_L and one in D_K. Hence, any forbidden induced subgraph in H' has to contain at least one vertex $u \in D_L$ and one vertex $v \in D_K$. If the third vertex w is from D_L or D_K, then $H'[\{u, v, w\}]$ has at least two non-edges and therefore u, v, and w do not form a forbidden induced subgraph. If w is from $V \setminus D$, then $\{u, w\}$ and $\{v, w\}$ are either both non-edges or both are red. Consequently, u, v, and w do not form a forbidden induced subgraph. Hence, H' is a module graph.

We now show that the cost of H' is not higher than the cost of H. Consider any pair $\{u, v\}$ with $u \in D_L$, $v \in D_K$ and let (b, n, r) be the cost vector of $\{u, v\}$. We show $b \geq n$. By assumption, $\{u, v\}$ is not blue. Thus, $\{u, v\}$ is neutral, red, or half-integral. If $\{u, v\}$ is neutral, then $n = 0$ which implies $b \geq 1$. If $\{u, v\}$ is red, then $r = 0$ which implies $b \geq 2n$, due to Proposition 6. If $\{u, v\}$ is half-integral, then $n = 1/2$, due to Proposition 6. According to Property 1, all components are half-integral. Hence, $b \geq 1/2 = n$. Altogether, the cost of H' is not larger than the cost of H. □

Second, we observe the following fact about instances to which Branching Rule 2 does not apply. The same fact was shown by Böcker et al. [7] for Cluster Editing; since the proof is completely analogous for MMC, we omit it.

Proposition 8. *Let K be a connected component of the graph M_b of an instance (V, s, k) of* MMC. *If each blue edge $\{u, v\}$ of K is contained in at most one conflict triple in (V, s, k), then K is either a cluster or a cluster minus exactly one blue pair.*

Proposition 8 states that when neither Branching Rule 2 nor 3 can be applied to an instance (V, s, k) of MMC, there are two cases for the connected components in M_b. To prove Theorem 3 we thus introduce reduction rules which can be exhaustively applied in $\mathcal{O}(|V|^2)$ time and resolve these two cases.

In a first step, we resolve all connected components of M_b which are clusters minus exactly one blue pair. Reduction Rule 1 treats the subcase $|K| = 3$ and Reduction Rule 2 treats the subcase $|K| \geq 4$. Before describing the rules, we observe the following facts about clusters minus exactly one blue pair.

Lemma 10. *Let (V, s, k) be an instance of* MMC *to which Branching Rule 2 does not apply, and let K be a cluster minus exactly one blue pair $\{v, w\}$ in M_b. Then, each vertex in $V \setminus K$ has either only neutral pairs or only red pairs with the vertices in K.*

Proof. Recall that for a blue pair which is contained in at least two conflict triples, Branching Rule 2 applies. First, observe that for each vertex $y \in K \setminus \{v, w\}$, every red neighbor x of v is also a red neighbor of y because otherwise the pair $\{v, y\}$ is contained in the two conflicts v, w, y and v, x, y. Similarly, every neutral neighbor x of v is also a neutral neighbor of y. By a symmetric

argument, y and w have the same neutral neighbors and the same red neighbors. Moreover, neither v nor w form half-integral vertex pairs with vertices $x \in V \setminus K$ because otherwise the pair $\{v, y\}$ is again in two conflicts. Altogether, this implies that every vertex $x \in V \setminus K$ is either a red neighbor of all vertices in K or a neutral neighbor of all vertices in K. □

Next, we prove that the vertices of a cluster K minus exactly one blue pair in M_b do not form conflict triples with vertices of other connected components of M_b.

Lemma 11. *Let (V, s, k) be an instance of* MMC *to which Branching Rule 2 does not apply, and let K be a cluster minus exactly one blue pair $\{v, w\}$ in M_b. Then, there exists no vertex $x \notin K$ which forms conflict triples with the vertices of K.*

Proof. According to Lemma 10, each vertex $x \notin K$ forms either a neutral pair with each vertex in K or x forms a red pair with each vertex in K. Hence, it remains to show that for every connected component $L \neq K$ of M_b, either each vertex of L forms a red pair with each vertex in K or each vertex of L forms a neutral pair with each vertex in K. Assume otherwise. Then, there is a blue pair $\{x, y\}$ where $x \in L$ and $y \in L$ and a vertex $z \in K$ such that $\{y, z\}$ is red and $\{x, z\}$ is neutral. By Lemma 10, this implies that each vertex of K forms a red pair with y and a neutral pair with x. Thus, the blue pair $\{x, y\}$ is contained in at least three conflict triples since $|K| \geq 3$. This contradicts the fact that Branching Rule 2 does not apply. □

Now we are ready to present the reduction rules.

Reduction Rule 1. *Let (V, s, k) be an instance of* MMC *to which Branching Rule 2 does not apply. If M_b has a connected component $\{u, v, w\}$ such that $\{u, w\}$ is not blue, then compute the value t of an optimal solution for the restriction of (V, s, k) to $\{u, v, w\}$, decrease k by t, and remove $\{u, v, w\}$ from (V, s, k).*

Lemma 12. *Reduction Rule 1 is safe and can be applied exhaustively in $\mathcal{O}(|V|)$ time.*

Proof. Let $(V', s', k' := k - t)$ denote the instance obtained from the application of the rule. If (V, s, k) is a yes-instance, then so is (V', s', k'): Any target graph H has cost at least t for pairs between u, v, and w and, therefore, $H' = H[V \setminus \{u, v, w\}]$ is a module graph with cost at most k' for (V', s', k').

For the converse, let H' be a target graph for (V', s', k'), thus H' has cost at most $k' = k - t$. According to Proposition 7, we may assume that H' does not contain blue edges with endpoints in different connected components of M_b'. Let H^* be a solution for (V, s, k) created by combining the graph H' with a target graph H'' for the restriction of (V, s, k) to $\{u, v, w\}$ as follows: Take the union of H' and H''. Then, for each pair $\{x, y\}$ with $x \in \{u, v, w\}$ and $y \in V \setminus \{u, v, w\}$ add a red edge in H^* if $\{x, y\}$ is a red pair and do not add an edge otherwise. Since H' has cost at most $k - t$ and H'' has cost at most t,

the cost of H^* is at most k. It remains to show that H^* is indeed a module graph.

By construction, H^* has no conflicts containing only vertices from $\{u, v, w\}$ and no conflicts containing only vertices from $V \setminus \{u, v, w\}$. Thus, it is sufficient to show that for each remaining connected component K in M_b^*, there are no conflicts between $\{u, v, w\}$ and vertices of K. According to Lemma 11, the vertex pairs $\{u, v\}$ and $\{v, w\}$ form no conflict triples with any vertex $z \neq w$ and thus, either all pairs between K and $\{u, v, w\}$ are neutral or red. According to Proposition 7, in the graph H^*, the connected component K is partitioned into clusters $C_1, \ldots, C_\ell$ that do not contain vertices from $V \setminus K$. Since H^* has cost 0 for all vertex pairs between K and $\{u, v, w\}$, for each cluster C_i there are in H^* either no edges between $\{u, v, w\}$ and C_i or every vertex pair between $\{u, v, w\}$ and C_i is red. Thus, H^* has no conflict containing vertices of K and $\{u, v, w\}$. Hence, H^* is a module graph of cost at most k for G.

The running time can be seen as follows. In $\mathcal{O}(|V|^2)$ time, we can determine all connected components of M_b which are blue or bicolored P_3s. For each such P_3 with vertex set $\{u, v, w\}$ we can compute the best solution for the restriction of (V, s, k) to $\{u, v, w\}$ in $\mathcal{O}(1)$ time. The overall running time follows. □

Reduction Rule 2. *Let (V, s, k) be an instance of* MMC *to which Branching Rules 2 and 3 do not apply. If M_b has a connected component K of size at least four, such that exactly one vertex pair $\{v, w\}$ of K is not blue, then transform $\{v, w\}$ into a blue pair, decrease k accordingly, and remove K from G.*

Lemma 13. *Reduction Rule 2 is safe and can be applied exhaustively in $\mathcal{O}(|V|)$ time.*

Proof. Let (V', s', k') denote the instance obtained from the application of the rule. We show that (V, s, k) is a yes-instance if and only if (V', s', k') is a yes-instance.

First, assume that (V, s, k) is a yes-instance. Assume $b_{v,w} > 1$. Since $|K| \geq 4$, there is a vertex $u \in K$ such that $\{u, v\}$ and $\{u, w\}$ are blue. Then merging the blue pair $\{u, w\}$ results in joining $s(u, v)$ and $s(v, w)$ into the joint cost vector (b^*, n^*, r^*) with $b^* \geq 3/2$, $n^* \geq 1$, and $r^* \geq 2$. Applying Restoration Rule 1 reduces this vector by at least 1. Hence, Branching Rule 3 applies to $\{u, w\}$, a contradiction. Consequently, $b_{v,w} \leq 1$. Since $|K| \geq 4$, it is optimal to transform K into a cluster by transforming $\{v, w\}$ into a blue pair: otherwise, the cost spent on vertex pairs from K in (V, s, k) is at least two because any cut in $M_b[K]$ contains at least two blue edges and transforming each of them into a neutral pair or a red pair has total cost at least 1. Thus, every module graph for the restriction of (V, s, k) to K has cost at least $b_{v,w} = k - k'$. Consequently, since (V, s, k) has a module graph of cost k, there is a module graph on $V \setminus K$ that has cost at most k' for the vertex pairs of $V \setminus K$.

Conversely, assume that $(V' = V \setminus K, s', k')$ has a module graph H' of cost at most k'. By Proposition 7, we may assume that H' does not have blue edges with endpoints in different connected components of M_b. We show that

extending H' by adding a clique of blue edges on the vertex set K and setting all edges between K and $V \setminus K$ in such a way that they produce no cost gives a module graph H^* of cost at most k for (V, s, k). The cost bound for H^* is obvious. It remains to show that H^* is a module graph. According to Lemma 11, the vertices of K form no conflicts with any vertex $z \notin K$ and thus for each connected component L, $L \neq K$, of M_b, either all vertex pairs between K and L are neutral or all are red. By the above assumption on H', in the graph H^* each L is partitioned into clusters $C_1, \ldots, C_\ell$. Since H^* has no cost on vertex pairs between K and L, there is thus no conflict containing vertices from K and L. Hence, H^* is a module graph of cost at most k for G.

By searching for a P_3 in each connected component of M_b, we can determine all connected components of M_b which are clusters minus exactly one edge in $\mathcal{O}(|V|^2)$ time. Let K be a cluster minus exactly one edge $\{v, w\}$ in M_b. Then we can find the edge $\{u, v\}$ by searching for a P_3 in M_b. This needs $\mathcal{O}(|K|)$ time. In $\mathcal{O}(1)$ time we can make $\{v, w\}$ blue. The overall running time follows. □

After applying these reduction rules exhaustively, every connected component of M_b is a cluster. The following two reduction rules will resolve all conflict triples between a cluster K of size at least two and another cluster L. Reduction Rule 3 treats the case $|L| \geq 2$ and Reduction Rule 4 treats the case $|L| = 1$. Recall that since conflict triples contain at least one blue pair, each conflict triple contains vertices from at most two connected components of M_b. Since M_b is a cluster graph, each conflict triple thus contains vertices of exactly two clusters.

We first observe the following restriction on the structure of conflict triples between clusters in the remaining graph.

Lemma 14. *Let (V, s, k) be an instance of* MMC *to which Branching Rule 2 does not apply, and let K be a cluster of size at least two in M_b. Then, there exists at most one vertex $x \notin K$ which is in conflict triples with blue pairs of K and each vertex $w \in W := V \setminus (K \cup \{x\})$ forms either only neutral pairs or only red pairs with the vertices of K.*

Proof. Assume there exists at least one vertex $x \notin K$ which forms conflict triples with the blue pairs of cluster K. Hence, there are vertices $y, z \in K$ such that the vertices x, y, z form a conflict triple. Similar to the proof of Lemma 10, for each vertex $w \in W$, the vertex pairs $\{w, y\}$ and $\{w, z\}$ are either both red or both neutral, as otherwise Branching Rule 2 applies. Hence, $W = N \uplus R$ where N is the set of vertices in W which form neutral pairs with y and z, and R is the set of vertices in W which form red pairs with y and z. Now we will prove that each vertex $u \in K \setminus \{y, z\}$ forms only red pairs with vertices in R and only neutral pairs with vertices in N. Assume that there is a vertex $r \in R$ such that $\{r, u\}$ is not red (the case that a vertex pair $\{s, u\}$ for a vertex $s \in N$ is not neutral follows by similar arguments). Then, r, u, y and r, u, z are conflict triples. We show that either $\{u, y\}$ or $\{u, z\}$ is contained in two conflict triples which implies that Branching Rule 2 applies, a contradiction to the premise of the lemma.

Consider the conflict triple x, y, z. If $\{x, y\}$ and $\{x, z\}$ are both half-integral, then the vertices u, x, y and u, x, z are conflict triples. Hence, the blue pair $\{u, y\}$ is contained in two conflict triples u, x, y and r, u, y, and Branching Rule 2 applies. If the vertex pairs $\{x, y\}$ and $\{x, z\}$ are not both half-integral, at most one of them has the same type as $\{u, x\}$, since not both of them can be red or neutral because x, y, z is a conflict triple. Assume without loss of generality that $\{x, y\}$ and $\{u, x\}$ have a different type. Then the blue pair $\{u, y\}$ is contained in two conflict triples r, u, y and u, x, y. Hence, Branching Rule 2 applies, a contradiction. Altogether this implies that all vertex pairs between K and R are red and all vertex pairs between K and N are neutral and x is the only vertex which forms conflict triples with blue pairs of K. □

Lemma 15. *Let (V, s, k) be an instance of* MMC *to which Branching Rule 2 does not apply, and let K be a cluster of size at least two in M_b. Then, there is at most one cluster L of M_b such that the vertices from K and L form conflict triples.*

Proof. According to Lemma 14, there is at most one vertex $x \notin K$ which forms conflict triples with blue pairs in K and the set $V \setminus (K \cup \{x\})$ can be partitioned into N, the set of vertices which form neutral pairs with the vertices in K and R, the set of vertices which form red pairs with the vertices in K. Let L denote the cluster containing x and assume that there is a further cluster M such that the vertices of K and M form at least one conflict triple. Consider the partition of M into $N_M := M \cap N$ and $R_M := M \cap R$. Since the vertices of M and K form at least one conflict triple, there exist vertices $s \in N_M$ and $r \in R_M$ such that $\{s, r\}$ is the blue pair of this conflict triple. By the definition of N, R, and M and since $|K| \geq 2$, there are vertices $y, z \in K$ such that $\{r, y\}$ and $\{r, z\}$ are red, and $\{s, y\}$ and $\{s, z\}$ are neutral. Hence, the blue pair $\{r, s\}$ is contained in two conflict triples and Branching Rule 2 applies, a contradiction. Consequently, M does not form conflict triples with K. □

Consider a cluster L which is the unique cluster of M_b forming conflicts with vertices of cluster K. Next, we prove that in the case $|L| \geq 2$, there is exactly one vertex pair between K and L such that after transforming this vertex pair, either all vertex pairs with one endpoint in K and the other in L are red or all of them are neutral. In other words, after transforming this vertex pair, the link property is established between clusters K and L.

Lemma 16. *Let (V, s, k) be an instance of* MMC *to which Branching Rule 2 does not apply and let K and L be two clusters of size at least two in M_b such that there exists a vertex $x \in L$, a vertex $y \in K$, and a vertex $z \in K \cup L$ such that $\{x, y, z\}$ forms a conflict triple. Then either all except one vertex pair between K and L are neutral or all except one vertex pair between K and L are red.*

Proof. Assume without loss of generality that two vertices of K form a conflict with some vertex $x \in L$. According to Lemmas 14 and 15, cluster K has only

conflicts with cluster L and each vertex $w \in W := L \setminus \{x\}$ forms either only neutral pairs with the vertices of cluster K or only red pairs with the vertices of cluster K. Hence, $W = N \uplus R$ where N is the set of vertices in W which have neutral pairs to K, and R is the set of vertices in W which have red pairs to K. Analogously to the proof of Lemma 15, one can show that if $|N| \geq 1$ and $|R| \geq 1$, then Branching Rule 2 applies to any blue pair with one vertex in N and one in R. Hence, either $|N| = 0$ or $|R| = 0$. Assume $|N| = 0$; the case $|R| = 0$ follows by similar arguments. Hence, for each vertex $v \in K$ and each vertex $w \in W$ the vertex pair $\{v, w\}$ is red. Since x forms conflict triples with the vertices of cluster K, there exists at least one vertex $y \in K$ such that the vertex pair $\{y, x\}$ is not red.

For a vertex $w \in W$ consider the blue pair $\{w, x\}$ in cluster L. Since $\{x, y\}$ is not red and $\{w, y\}$ is red, the vertices w, x, y form a conflict triple. Since for each vertex $v \in K$ the vertex pair $\{v, w\}$ is red, the vertex pair $\{v, x\}$ has to be red as well, otherwise Branching Rule 2 applies. Hence, all vertex pairs between K and L are red except for $\{x, y\}$. □

For two clusters K and L with $|K| \geq 2$ and $|L| \geq 2$ of the graph M_b, we call the one vertex pair $\{x, y\}$ that has a different type from all other vertex pairs between both clusters K and L the *dissenting* vertex pair. We now resolve all conflict triples between two clusters of M_b of size at least two by transforming the dissenting pair.

Reduction Rule 3. *Let (V, s, k) be an instance of* MMC *to which Branching Rules 2 and 3 do not apply. If M_b contains two clusters K and L such that $\{x, y\}$, with $y \in K$ and $x \in L$, is a dissenting pair, then transform $\{x, y\}$ into the same type as the other vertex pairs between K and L and decrease the parameter k accordingly. Afterwards, remove K and L from G.*

Lemma 17. *Reduction Rule 3 is safe and can be applied exhaustively in $\mathcal{O}(|V|^2)$ time.*

Proof. According to Lemma 16, all vertex pairs between K and L are either all red or neutral, except the dissenting vertex pair $\{x, y\}$. Assume that all these vertex pairs are red; the other case follows by similar arguments. Consider another vertex $z \in K$. Let $s(x, z) = (b', n', r')$. Observe that since $\{x, z\}$ is red, we have $b' \geq 2, n' \geq 1$, and $r' = 0$. Merging the blue pair $\{y, z\}$ results in joining $s(x, y)$ and $s(x, z)$ into the joint cost vector (b^*, n^*, r^*). We distinguish whether the dissenting vertex pair is half-integral or integral.

First, assume $\{x, y\}$ is half-integral. If $r_{x,y} \geq 3/2$, then $b^* \geq 5/2$, $n^* \geq 3/2$, and $r^* \geq 3/2$. Thus, applying Restoration Rule 1 reduces (b^*, n^*, r^*) by at least 1, and hence, Branching Rule 3 applies. Consequently, we can assume that $r_{x,y} = 1/2$. We have cost at least $1/2$ to resolve the half-integral vertex pair and transforming $\{x, y\}$ into a red pair has cost $1/2$. Afterwards, there are no conflict triples with vertices of clusters K and L. Moreover, for each further cluster M the vertex pairs between K and M are either all red or all neutral. Similarly, the vertex pairs between L and M are either all red or all neutral. Hence, combining a module graph with minimum cost for the vertex

set $V \setminus (K \cup L)$ and the solution for clusters K and L described above results in a module graph of optimal cost for (V, s, k).

Second, assume $\{x, y\}$ is neutral. If $r_{x,y} \geq 2$, then $b^* \geq 3$, $n^* \geq 1$, and $r^* \geq 2$ and again Branching Rule 3 applies in this case. Hence, we can assume that $r_{x,y} = 1$. Now observe that the minimum cost to resolve all conflict triples is 1 since no pair between two vertices of $K \cup L$ is half-integral. Thus, making $\{x, y\}$ a red edge and setting all other edges such that they have cost 0 is a minimum-cost module graph for the vertex set $K \cup L$ since the vertices of $K \cup L$ are not part of any further conflict triples. As in the first case, we can expand this graph to a minimum-cost module graph for (V, s, k).

In $\mathcal{O}(|V|^2)$ time we can determine all clusters of M_b to which Reduction Rule 3 applies. For each pair K and L of such clusters, we can determine the dissenting vertex pair, change the type of this vertex pair, and remove K and L from (V, s, k) in $\mathcal{O}(|K| \cdot |L|)$ time. The total running time is thus $\mathcal{O}(|V|^2)$. □

After applying Reduction Rule 3, all remaining conflict triples are between clusters of size at least two and size-one clusters.

Reduction Rule 4. *Let (V, s, k) be an instance of* MMC *to which Branching Rules 2 and 3 do not apply. If M_b contains a cluster $K \supseteq \{y, z\}$ and a cluster $L = \{x\}$ such that $\{x, y, z\}$ forms a conflict triple, then compute the cost $\tilde{n}$ to transform all vertex pairs between K and L into non-edges and the cost $\tilde{r}$ to transform all vertex pairs between K and L into red edges. Decrease the parameter k by $\min\{\tilde{n}, \tilde{r}\}$ and remove K and L from G.*

Lemma 18. *Reduction Rule 4 is safe and can be applied exhaustively in $\mathcal{O}(|V|^2)$ time.*

Proof. By Lemma 14, the vertices of cluster K form only conflict triples with the vertex x of cluster L. We prove that the minimum of $\tilde{n}$ and $\tilde{r}$ is not higher than the cost of partitioning cluster K of the graph M_b into at least two smaller clusters. Observe that partitioning K into at least two clusters causes cost of at least $|K| - 1$, since the minimum edge cut in a clique of size $|K|$ has size at least $|K| - 1$ and transforming a blue pair into a non-edge costs at least 1. In the proof we distinguish four cases and show in each case that $\tilde{n} \leq |K| - 1$ or $\tilde{r} \leq |K| - 1$.

Case 1: there is a vertex $y \in K$ such that $\{x, y\}$ is neutral and $r_{x,y} \geq 2$. If there is another vertex $z \in K$ such that $\{x, z\}$ is red, merging the blue pair $\{y, z\}$ results in joining $s(x, y)$ and $s(x, z)$ into the joint cost vector (b', n', r') where $b' \geq 3$, $n' \geq 1$, and $r' \geq 2$. Restoration Rule 1 reduces (b', n', r') by at least 1 and, hence, Branching Rule 3 applies, a contradiction. We conclude that each vertex pair between K and x is either half-integral or a neutral. Transforming each half-integral vertex pair into a non-edge causes cost of exactly 1/2. There are at most $|K| - 1$ half-integral vertex pairs between K and x. Hence, we have cost at most $(|K| - 1)/2 < |K| - 1$ to transform all half-integral vertex pairs between clusters K and L into non-edges.

Case 2: there is a vertex $y \in K$ such that $\{x, y\}$ is half-integral and $r_{x,y} \geq 3/2$. If there is another vertex $z \in K$ such that $\{x, z\}$ is red,

merging the blue pair $\{y, z\}$ results in joining $s(x, y)$ and $s(x, z)$ into (b', n', r') where $b' \geq 5/2$, $n' \geq 3/2$, and $r' \geq 3/2$. Restoration Rule 1 reduces (b', n', r') by at least 1 and, hence, Branching Rule 3 applies, a contradiction. By the same argumentation as in the first case, transforming all vertex pairs between clusters K and L into non-edges has cost at most $|K|/2 \leq |K| - 1$.

Case 3: there is a vertex $y \in K$ such that $\{x, y\}$ is red and $n_{x,y} \geq 2$. If there is another vertex $z \in K$ such that $\{x, z\}$ is neutral, merging the blue pair $\{y, z\}$ results in joining $s(x, y)$ and $s(x, z)$ into cost vector (b', n', r') with $b' \geq 5$, $n' \geq 2$, and $r' \geq 1$. Restoration Rule 1 reduces (b', n', r') by at least 1 and, hence, Branching Rule 3 applies, a contradiction. Since Case 2 does not apply, we conclude that transforming all vertex pairs between K and L into red edges has cost at most $|K|/2 \leq |K| - 1$.

Case 4: otherwise. By excluding the cases above, we may assume that for each neutral pair $\{x, y\}$ we have $r_{x,y} = 1$, for each red pair $\{x, y\}$ we have $n_{x,y} = 1$ and for each half-integral pair $\{x, y\}$ we have $r_{x,y} = 1/2$. Consequently, $\sum_{y \in K} n_{y,x} + r_{y,x} = |K|$. Thus, $\tilde{n} = \sum_{y \in K} n_{y,x} \leq |K|/2 \leq |K| - 1$ or $\tilde{r} = \sum_{y \in K} r_{y,x} \leq |K|/2 \leq |K| - 1$.

This proves the correctness of the rule. The running time can be seen as follows. In $\mathcal{O}(|V|^2)$ time we can determine all clusters of M_b to which Reduction Rule 4 applies. Let K and L be two such clusters. In $\mathcal{O}(|K|)$ time we can examine all vertex pairs between K and L, decide if $\tilde{n} \leq \tilde{r}$, and remove K and L. The overall running time is thus $\mathcal{O}(|V|^2)$. □

Recall that a conflict triple contains at least one blue pair. After applying Reduction Rule 4 exhaustively, the instance (V, s, k) of MMC contains no conflict triple. The instance (V, s, k) is, however, not necessarily equivalent to a module graph: there can be clusters of size one in M_b which have a half-integral vertex pair between them. Nevertheless, we can solve (V, s, k) in polynomial time.

Lemma 19. *An instance (V, s, k) of* MMC *can be solved in $\mathcal{O}(|V|^2)$ time if (V, s, k) contains no conflict triple.*

Proof. If the number of half-integral vertex pairs is larger than $2k$, then we may return 'no'. Otherwise, we may return 'yes'. If (V, s, k) has a module graph with cost 0 this is clearly correct. If this is not the case, then, since G contains no conflict triple, all half-integral vertex pairs are between clusters of size one. Thus, transforming all half-integral vertex pairs into non-edges and choosing the cost-0 edge type for all other vertex pairs yields a module graph H. By the check above, the number of half-integral vertex pairs is at most $2k$ and thus H has cost at most k. The running time follows from the fact we can count the number of half-integral vertex-pairs in $\mathcal{O}(|V|^2)$ time. □

Now we can prove Theorem 3, that is, any instance of MMC can be solved in $\mathcal{O}(|V|^2)$ time if Branching Rules 2 and 3 do not apply.

Proof (of Theorem 3). Since Branching Rule 2 does not apply to this instance, each connected component K of M_b is either a cluster or a cluster minus

exactly one blue pair, according to Proposition 8. Applying Reduction Rules 1 and 2 resolves all connected components of M_b which are clusters minus exactly one blue pair. Hence, all remaining connected components of M_b are clusters. Reduction Rule 3 resolves all conflict triples between two clusters of size at least two. Reduction Rule 4 resolves all conflict triples between a cluster of size one and a cluster of size at least two.

Applying those reduction rules exhaustively needs $\mathcal{O}(|V|^2)$ time. Afterwards, no conflict triples remain. Now Lemma 19 applies and (V, s, k) can be solved in $\mathcal{O}(|V|^2)$ time. □

Finally, we prove Theorem 2, that is, MMC can be solved in $\mathcal{O}(2^k \cdot |V|^3)$ time.

Proof (of Theorem 2). First, check for each blue pair $\{u, v\}$ if Branching Rule 2 or 3 applies. This needs $\mathcal{O}(|V|^3)$ time. If this is the case, we branch on $\{u, v\}$. According to Corollary 1 and Lemma 9, Branching Rules 2 and 3 have a branching vector of $(1, 1)$ or better. This implies a search tree size of $\mathcal{O}(2^k)$ because we only branch as long as $k > 0$. In one case, we set $b_{u,v} := k+1$, which can be done in constant time. In the other case, we merge u and v. That is, we remove the vertices u and v from the graph and replace them by a new vertex u' and join all $\mathcal{O}(|V|)$ incident vertex pairs. We can compute the cost vector for each new joint pair in $\mathcal{O}(|V|)$ time. Hence, we need $\mathcal{O}(|V|^3)$ time for each inner node of the search tree. By Theorem 3, MMC can be solved in $\mathcal{O}(|V|^2)$ time if Branching Rules 2 and 3 do not apply, that is, for each leaf of the search tree. Altogether, we obtain an $\mathcal{O}(2^k \cdot |V|^3)$-time algorithm for MMC. □

5. A Polynomial Kernel

We now present a kernelization for Weighted Module Map that yields a kernel with $\mathcal{O}(k^2)$ vertices. In the description of the kernelization, it will be convenient to use the cost vector notation introduced in Section 4. To this end, recall that each instance (G, g, k) of Weighted Module Map is also an instance of MMC. In the terminology of MMC each vertex pair in G is either blue, red, or neutral. The basic idea is the following: Let $\{u, v\}$ be a vertex pair of an instance $(G = (V, E_b, E_r), g, k)$ of Weighted Module Map. We investigate if it is possible that the vertex pair $\{u, v\}$ can be a blue, a non-, or red edge in any target graph of a size-k solution. To this end, we estimate for each edge type the *induced costs* of transforming $\{u, v\}$ into this type; this approach was also used for Cluster Editing [16] and Weighted Cluster Editing [7].

If, for example, $\{u, v\}$ is a blue edge in a target graph, then for each other vertex w in this graph, $\{u, w\}$ and $\{v, w\}$ are either both red or both non-edges. Thus, if $\{u, w\}$ and $\{v, w\}$ have a different type in G, at least one of them has to be transformed. To formally define the cost estimation, we make use of the vertex-pair cost function introduced in Section 4. As discussed in Section 4, we can easily translate the weight and the edge type for each vertex pair $\{u, v\}$ into

a cost vector $(b, n, r)_{u,v}$. The value b describes the cost of transforming $\{u, v\}$ into a blue edge, the value n is the cost to transform $\{u, v\}$ into a non-edge and the value r is the cost to transform $\{u, v\}$ into a red edge. Recall that the edge weight function g assigns only positive integers ω and that the resulting cost vectors have the form $(0, \omega, 2\omega)$, $(\omega, 0, \omega)$, or $(2\omega, \omega, 0)$.

We first define how to estimate the three types of induced costs: $\operatorname{icb}(u, v)$ is a lower bound for the cost of $\{u, v\}$ being a blue edge in the target graph of an optimal solution, $\operatorname{icn}(u, v)$ is a lower bound for the cost of $\{u, v\}$ being a non-edge in the target graph of an optimal solution, and $\operatorname{icr}(u, v)$ is a lower bound for the cost of $\{u, v\}$ being a red edge in the target graph of an optimal solution.

Definition 5. *Let $(G = (V, E_b, E_r), g, k)$ be an instance of* Weighted Module Map *and let u and v be vertices in G. Furthermore, let $U := \{u, v\}$ and $W := V \backslash U$. Then the induced costs of transforming $\{u, v\}$ are defined as*

$$\operatorname{icb}(u, v) := b_{u,v} + \sum_{w \in W} \min(b_{u,w} + b_{v,w}, n_{u,w} + n_{v,w}, r_{u,w} + r_{v,w}),$$

$$\begin{aligned}\operatorname{icn}(u, v) :=& n_{u,v} + \sum_{w \in W:\ b_{u,w} = 0 = b_{v,w}} \min(n_{u,w}, n_{v,w}) \\ &+ \sum_{w \in W, x, y \in U, x \neq y:\ b_{x,w} = 0 = r_{y,w}} \min(n_{x,w}, n_{y,w}),\end{aligned}$$

$$\begin{aligned}\operatorname{icr}(u, v) :=& r_{u,v} + \sum_{w \in W:\ b_{u,w} = 0 = b_{v,w}} \min(r_{u,w}, r_{v,w}, n_{u,w} + n_{v,w}) \\ &+ \sum_{w \in W, x, y \in U, x \neq y:\ b_{x,w} = 0 = n_{y,w}} \min(n_{x,w}, r_{y,w}).\end{aligned}$$

The definition of $\operatorname{icb}(u, v)$, $\operatorname{icn}(u, v)$, and $\operatorname{icr}(u, v)$ can be explained as follows.

First, consider the cost $\operatorname{icb}(u, v)$. We have to pay $b_{u,v}$ to transform $\{u, v\}$ into a blue edge. For each remaining vertex $w \in W$, the vertex pairs $\{u, w\}$ and $\{v, w\}$ have to be of the same type in the target graph. Otherwise, there is a conflict triple. The definition assumes the best case among the three possibilities.

Second, consider the cost $\operatorname{icn}(u, v)$. We have to pay $n_{u,v}$ to transform the vertex pair $\{u, v\}$ into a non-edge. Then we have to destroy each blue P_3 and each bicolored P_3 which contains the non-edge $\{u, v\}$. In both cases, we have to transform one of the incident blue or red edges. The case that such an edge is transformed into a non-edge is a lower bound for the costs that arise.

Finally, consider $\operatorname{icr}(u, v)$. We have to pay $r_{u,v}$ to transform the vertex pair $\{u, v\}$ into a red edge. Moreover, we have to destroy each resulting almost-blue K_3 and each bicolored P_3 which contains the red edge $\{u, v\}$. For the almost-blue K_3, either both blue edges have to be transformed into a non-edge or one of the two blue edges has to be transformed into a red edge. For the

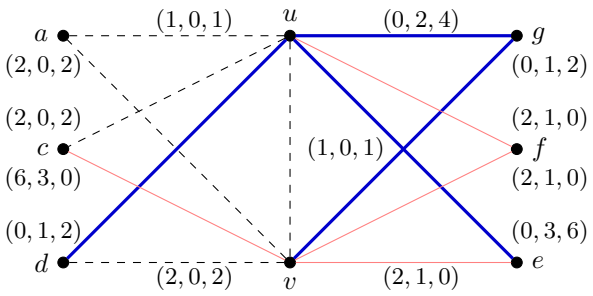


Figure 4: A vertex pair $\{u, v\}$ with $\operatorname{icb}(u, v) = 6$, $\operatorname{icn}(u, v) = 2$, and $\operatorname{icr}(u, v) = 4$. Here, blue edges are dark and bold, red edges are bright and non-edges are dashed.

bicolored P_3, either the blue edge has to be transformed into a non-edge or the non-edge has to be transformed into a red edge (because transforming it into a blue edge gives an almost-blue K_3).

Proposition 9. *Let $(G = (V, E_b, E_r), g, k)$ be an instance of* Weighted Module Map. *We can compute $\operatorname{icb}(u, v)$, $\operatorname{icn}(u, v)$, and $\operatorname{icr}(u, v)$ for all vertex pairs $\{u, v\}$ in $\mathcal{O}(|V|^3)$ time.*

After computing the induced costs, we compare them with the parameter k. If, for example, there is a blue edge $\{u, v\}$ with $\operatorname{icb}(u, v) > k$, then in any solution of cost at most k the edge $\{u, v\}$ has to be transformed into either a non-edge or a red edge. Hence, we may directly transform $\{u, v\}$ into a non-edge and pay $n_{u,v}$. Similarly, we can transform any red edge with $\operatorname{icr}(u, v) > k$ into a non-edge. In contrast, if there is a non-edge $\{u, v\}$ with $\operatorname{icn}(u, v) > k$, it is not immediately clear whether we need to transform this edge into a blue edge or into a red edge.

The following reduction rule will change the type of an edge if the associated induced costs are larger than the parameter k. For the case that $\operatorname{icn}(u, v) > k$, it considers the values of $\operatorname{icb}(u, v)$ and $\operatorname{icr}(u, v)$ to decide which transformation to apply.

Reduction Rule 5. *Let $(G = (V, E_b, E_r), g, k)$ be an instance of* Weighted Module Map.

If G contains a blue edge $\{u, v\}$ such that $\operatorname{icb}(u, v) > k$, then transform $\{u, v\}$ into a non-edge and decrease the parameter by $n_{u,v}$.

If G contains a red edge $\{u, v\}$ such that $\operatorname{icr}(u, v) > k$, then transform $\{u, v\}$ into a non-edge and decrease the parameter by $n_{u,v}$.

If G contains a non-edge $\{u, v\}$ such that $\operatorname{icn}(u, v) > k$ and $\operatorname{icb}(u, v) > k$, then transform $\{u, v\}$ into a red edge and decrease the parameter by $r_{u,v}$.

If G contains a non-edge $\{u, v\}$ such that $\operatorname{icn}(u, v) > k$ and $\operatorname{icr}(u, v) > k$, then transform $\{u, v\}$ into a blue edge and decrease the parameter by $b_{u,v}$.

Consider the example in Figure 4. If $k = 3$, then $\operatorname{icb}(u, v) > k$ and $\operatorname{icr}(u, v) > k$. Hence, $\{u, v\}$ has to be a non-edge in any solution of cost at most k. Since $\{u, v\}$ is already a non-edge, the rule does not apply.

Lemma 20. *Reduction Rule 5 is safe and can be applied exhaustively in $\mathcal{O}(k \cdot |V|^2)$ time when the induced costs of each edge have been calculated.*

Proof. The safeness of the rule can be seen as follows. In the first case, there is no target graph for solutions of cost at most k such that $\{u, v\}$ is a blue edge. Hence, we need to transform $\{u, v\}$ into a non-edge or a red edge, and for both possibilities, it is optimal to transform $\{u, v\}$ into a non-edge first. The safeness of the second case follows by symmetric arguments. The safeness of the third case follows from the fact that transforming $\{u, v\}$ into a blue edge or keeping $\{u, v\}$ a non-edge is impossible in solutions of cost at most k. Hence, we need to transform $\{u, v\}$ into a red edge. Similarly, in the last case, we need to transform $\{u, v\}$ into a blue edge.

Checking the conditions of Reduction Rule 5 can be done in $\mathcal{O}(|V|^2)$ time by considering each vertex pair. Reduction Rule 5 can be applied at most k times because each application decreases the parameter by at least 1. Observe that after every application, we need to recompute the induced costs of each vertex pair containing either u or v. This can be done in $\mathcal{O}(|V|)$ time for each pair. Since there are $\mathcal{O}(|V|)$ such pairs, the update of the induced costs takes $\mathcal{O}(|V|^2)$ time. □

Note that if an instance (G, g, k) of Weighted Module Map contains a vertex pair $\{u, v\}$ such that $\text{icb}(u, v) > k$, $\text{icn}(u, v) > k$, and $\text{icr}(u, v) > k$, then each solution for (G, g, k) has cost at least k and hence (G, g, k) is a no-instance.

Reduction Rule 5 only changes edges. The definition of the induced costs can be further used to mark some edges as *unchangeable* which could be useful in implementations. Consider Figure 4 for an example: Since $\text{icb}\,(u, v) > k$ and $\text{icr}\,(u, v) > k$, the non-edge $\{u, v\}$ can be marked as unchangeable. In the following, we will not further use this marking scheme since transforming edges will be sufficient to provide a kernelization.

To obtain a quadratic-vertex kernel, we have to delete clusters whose vertices are not contained in any conflict triples.

Reduction Rule 6. *Let $(G = (V, E_b, E_r), g, k)$ be an instance of* Weighted Module Map*. If G_b contains a cluster C such that for all other connected components L of G_b the vertex pairs between C and L are either all non-edges or all are red, then delete C from the graph.*

Lemma 21. *Reduction Rule 6 is safe and can be applied exhaustively in $\mathcal{O}(|V|^2)$ time.*

Proof. Since for each other connected component L (not necessarily a cluster) all vertex pairs between C and L are either all red or all are non-edges, G contains no conflict triples with at least two vertices u and v from C. Now, assume towards a contradiction that G contains three vertices u, v, and w which form a conflict triple such that $u \in C$ and $v, w \notin C$. Since each conflict triple contains at least one blue pair, there exists a connected component L such that $v, w \in L$. Since the vertex pairs between C and L are either all red or all non-edges, u, v, and w do not form a conflict triple, a contradiction. Hence, no vertex of cluster C is contained in a conflict triple.

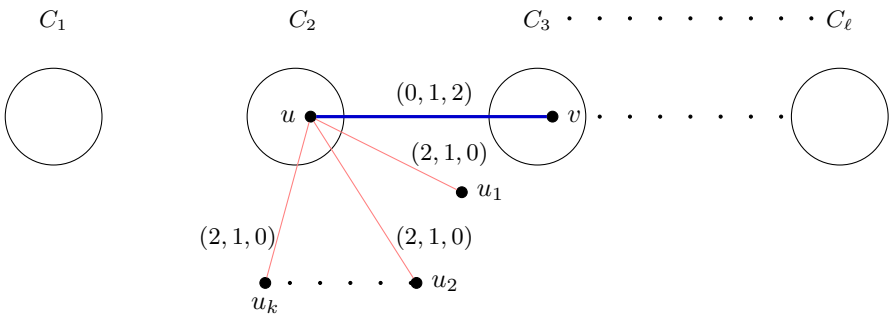


Figure 5: The clusters $C_1, C_2, \ldots, C_\ell$ are modified. Consider a blue edge $\{u, v\}$ between two modified clusters which has to be deleted to obtain the optimal solution $\mathcal{S}$. Then the blue edge $\{u, v\}$ forms conflict triples with at most k vertices from unmodified clusters.

Consider a solution $\mathcal{S}$ for $G[V \setminus C]$. We will prove that $\mathcal{S}$ is also a solution for G. Since instances of Weighted Module Map are also instances of MMC, Proposition 7 applies: in the target graph H that is obtained from solution $\mathcal{S}$ there is no blue edge between vertices that are from different connected components of G_b. In G the edges between C and another connected component L are either all red or non-edges, and since connected components of H_b are subsets of connected components of G_b, each connected component in H_b also forms either only non-edges or red edges with cluster C. Hence, $\mathcal{S}$ is also a solution for G.

Fix a cluster C in G_b. To check if all edges between cluster C and some other connected component L of G_b have the same type, we check $|C| \cdot |L|$ edges. Since the sum of the sizes of all connected components in G_b is $|V|$, checking whether Reduction Rule 6 applies to cluster C needs $|C| \cdot |V|$ time. Since each vertex of G is contained in at most one cluster of G_b, Reduction Rule 6 can be applied exhaustively in $\mathcal{O}(|V|^2)$ time. □

We now obtain a kernelization with a quadratic number of vertices.

Theorem 4. Weighted Module Map *admits a kernelization with $\mathcal{O}(k^2)$ vertices which can be computed in $\mathcal{O}(|V|^3 + k \cdot |V|^2)$ time.*

Proof. The kernelization algorithm is to first compute all induced costs, then to apply Reduction Rule 5 exhaustively, and finally to apply Reduction Rule 6 exhaustively. The running time of $\mathcal{O}(|V|^3 + k \cdot |V|^2)$ follows from Lemmas 20 and 21. Thus, it remains to prove the size bound on the kernel.

Let $(G = (V, E_b, E_r), g, k)$ be an instance of Weighted Module Map which is reduced with respect to Reduction Rules 5 and 6 and which has a solution $\mathcal{S}$ of cost at most k. The target graph H of solution $\mathcal{S}$ is a module graph which can be obtained from the graph G by edge transformations of cost at most k.

We distinguish modified and unmodified clusters. A *modified cluster* is a cluster which contains at least one vertex which is part of a vertex pair trans-

formed by solution $\mathcal{S}$. An *unmodified cluster* is a cluster whose vertices are not part of any transformed vertex pair. In the first step of the proof, we bound the size of any modified cluster in this target graph H. In the second step, we bound the number of modified clusters and in the third step, we bound the overall size of unmodified clusters. In Step 4, we complete the proof of the quadratic number of vertices.

Step 1: Bounding the size of modified clusters. We prove that for each modified cluster C in H, $|C| \leq 2k$. Assume towards a contradiction that H contains a cluster C with at least $2k+1$ vertices. First, assume G contains a blue edge $\{u, v\}$ with $u \in C$ and $v \notin C$. Since $\mathcal{S}$ transforms at most k edges, there are at least $k+1$ vertices in C that are neighbors of u in G and non-neighbors of v in G. Hence, $\operatorname{icb}(u, v) \geq k+1$ and Reduction Rule 5 applies. Second, assume there is a vertex pair $\{u, v\}$ in C which is red or a non-edge. Since $\mathcal{S}$ transforms at most $k-1$ edges with one endpoint in $\{u, v\}$ and one endpoint in $C \setminus \{u, v\}$, the vertices u and v have at least $k+1$ common neighbors in C. Hence, $\operatorname{icn}(u, v) > k$ and $\operatorname{icr}(u, v) > k$ and Reduction Rule 5 applies. We conclude that C is a cluster in G.

Consider a vertex $z \notin C$ in G. Since $|C| \geq 2k+1$, there are at least $k+1$ non-edges between C and z or at least $k+1$ edges between C and z are red. Assume without loss of generality that at least $k+1$ edges between z and C are red. If there is a vertex $u \in C$ such that $\{u, z\}$ is a non-edge, then for each red edge between C and z, the vertices incident with the red edge and vertex u form a bicolored P_3. Hence, $\operatorname{icn}(u, z) > k$ and $\operatorname{icb}(u, z) > k$. This implies that Reduction Rule 5 transforms $\{u, z\}$ into a red edge, a contradiction since G is reduced exhaustively. Consequently, if z has at least $k+1$ red edges to C, then it has only red edges to C. By symmetric arguments, it holds that if z has at least $k+1$ non-edges to C, then z has only non-edges to C. Thus, each vertex $z \notin C$ has only red or non-edges to cluster C. This implies that each connected component $L \neq C$ of G_b can be partitioned into L_n and L_r, where L_n is the set of vertices which have only non-edges with C, and L_r is the set of vertices which have only red edges with C. Consider a vertex $u \in L_n$ and a vertex $v \in L_r$ that are connected in G by a blue edge. Such a pair of vertices exists if L_n and L_r are not empty. Clearly, u and v form a bicolored P_3 with each vertex w in cluster C. Hence, $\operatorname{icb}(u, v) > k$. Thus, Reduction Rule 5 applies and transforms $\{u, v\}$ into a non-edge. This implies that for each connected component $L \neq C$ in G_b, either all edges between C and L are red or all are non-edges. Consequently, Reduction Rule 6 applies to C, a contradiction. Hence, all clusters in H have size at most $2k$.

Step 2: Bounding the number of modified clusters. Since all edge-weights are integers, each edge-transformation has cost at least 1. Hence, the graph H contains at most $2k$ modified clusters since exactly two vertices are incident with a transformed edge.

Step 3: Bounding the overall size of unmodified clusters. Let L be an unmodified cluster. Since L is unmodified, no vertex of L is incident with a transformed edge. Since Reduction Rule 6 did not apply, the vertices of L are part of conflict triples. Since no vertex of L is incident with a transformed

edge, the only possible forbidden subgraph is a bicolored P_3 where the blue edge $\{u, v\}$ is part of another connected component K in G_b and deleted by the solution. Hence, there exist at least two distinct clusters C_i and C_j in the target graph H which contain u and v, respectively. Clearly, all vertices of the unmodified cluster L have the same set of red neighbors. Assume the blue edge $\{u, v\}$ forms $k + 1$ bicolored P_3s with vertices from unmodified clusters. By definition this implies $\text{icb}(u, v) \geq k + 1$ which contradicts the fact that Reduction Rule 5 does not apply.

By the above, for each blue edge $\{u, v\}$ the number of vertices in unmodified clusters that form a bicolored P_3 with $\{u, v\}$ is at most k. Since we can transform at most k blue edges and each blue edge can be part of at most k conflict triples, all unmodified clusters together can have at most k^2 vertices; see Figure 5 for an illustration.

Step 4: Bounding the overall number of vertices. From Step 1 and Step 2 we conclude that the overall number of vertices in modified clusters is at most $2k \cdot 2k$. From Step 3 we conclude that all unmodified clusters together can have at most k^2 vertices. Hence, G contains at most $5k^2$ vertices.

Since the number of vertices in yes-instances is bounded, we obtain a kernelization for Weighted Module Map: If an instance $(G = (V, E_b, E_r), g, k)$ contains more than $5k^2$ vertices after the exhaustive application of Reduction Rules 5 and 6, then it is a no-instance and can be replaced by a no-instance of constant size. Afterwards, we have achieved the claimed bound on the number of vertices in both cases. □

6. Two Fine-Grained Complexity Lower Bounds

A natural way to find a forbidden subgraph of module graphs would be to search for example first for a blue P_3, then for an almost-blue K_3, and finally for a bicolored P_3. In this section, we show that, under some common assumptions in fine-grained complexity theory, it is impossible to find an almost-blue K_3 in $\mathcal{O}(|V|+|E|)$ time and impossible to find a blue P_3 in $\mathcal{O}(|V|+|E|)$ time. Hence, the linear-time algorithm for detecting forbidden induced subgraphs of module graphs relies crucially on the fact that we search *simultaneously* for blue P_3s and almost-blue K_3s. This side result, although only loosely connected to Module Map, may prove useful for the investigation of further graph problems with two edge colors.

For both problems, we give a reduction from Triangle Detection, defined as follows.

Triangle Detection
Input: A graph $G = (V, E)$.
Question: Does G contain a K_3?

It has been open for a long time whether triangles can be detected in a graph in $\mathcal{O}(|V| + |E|)$ time or in $\mathcal{O}(|V|^2)$ time. Triangle Detection is a special case of the k-Clique problem where the task is to detect a clique of size k in a graph $G = (V, E)$. The current best algorithm for k-Clique has a running

time of $\mathcal{O}(|V|^{k\omega/3})$ [19], where $\omega < 2.373$ is the exponent of the time that is needed to multiply two $n \times n$ matrices [26]. The *k-clique-conjecture* states that this running time is optimal for k-Clique [1] and thus implies that the best running time for any algorithm for Triangle Detection is $\mathcal{O}(|V|^{\omega})$. Hence, an $\mathcal{O}(|V|^2)$-time algorithm for Triangle Detection would either falsify the k-clique-conjecture or imply $\omega = 2$. We state our relative lower bounds in terms of the running times for Triangle Detection that would be implied by fast algorithms for our problems.

The reductions use hashing. For a set U (called universe) of size n, an (n, k)-*perfect hash family* is a set of functions $h : U \to \{1, \ldots, k\}$ such that for each size-k subset S of U, the family contains a function h that maps the elements of S to different values. Given U and k, an (n, k)-*perfect hash family* of size $2^{\mathcal{O}(k)} \log n$ can be constructed in $2^{\mathcal{O}(k)} n \log(n)$ time [3].

First, we show that a linear-time algorithm for detecting an almost-blue K_3 implies an almost linear-time algorithm for Triangle Detection.

Proposition 10. *If we can detect almost-blue K_3s in a graph $G = (V, E_b, E_r)$ in $\mathcal{O}(|V| + |E|)$ time, then we can detect triangles in $\mathcal{O}((|V| + |E|) \log |V|)$ time.*

Proof. We reduce from Triangle Detection as follows. Let $G = (V, E)$ be the graph of the Triangle Detection instance. The universe U of the hash function consists of the edges in G (so $U := E$) and $k = 3$. In other words, we use a $(|E|, 3)$-perfect hash family. This family has size $\mathcal{O}(\log |E|)$ and can be constructed in $\mathcal{O}(|E| \log |E|)$ time. For each function h in the hash family, we create a graph G_h with blue and red edges from G by making all edges e with $h(e) = 1$ or $h(e) = 2$ blue and the edges with $h(e) = 3$ red.

The graph G contains a triangle if and only if G_h contains an almost-blue K_3 for some hash function h of the hash family: If G contains a triangle, then there exists a function h in which the edges of the triangle receive different hash values, since the hash family is $(|E|, 3)$-perfect. For this function, two of the three triangle edges are blue and one is red in G_h. Thus, G_h contains an almost-blue K_3. The converse statement is obviously true since the edges of an almost-blue K_3 in any G_h are also edges of G.

By the above, we can find a triangle in G by searching for an almost-blue K_3 in each G_h. If this can be done in $\mathcal{O}(|V| + |E|)$ time for each G_h, then the overall running time of the algorithm for detecting a triangle in G is $\mathcal{O}((|V| + |E|) \log |E|)$, since the hash family consists of $\mathcal{O}(\log |E|)$ functions. □

With a similar construction, we can show a slightly weaker statement for the problem of detecting blue P_3s in bicolored graphs. More precisely, we show that an algorithm for detecting blue P_3s whose running time is quadratic in $|V|$ gives an algorithm for Triangle Detection whose running time is almost quadratic in $|V|$. Hence, such an algorithm would imply a major speed-up for Triangle Detection. This result is slightly surprising, since detecting P_3s in uncolored graphs can be done in linear time according to Proposition 2.

Proposition 11. *If we can detect blue P_3s in a graph $G = (V, E_b, E_r)$ in $\mathcal{O}(|V|^2)$ time, then we can detect triangles in $\mathcal{O}((|V|^2) \log |V|)$ time.*

PROOF. Similar to the proof of Proposition 10 we give a reduction from TRIANGLE DETECTION using a perfect hash family with universe $U := E$ and $k = 3$. This hash family has size $\mathcal{O}(\log |E|)$ and can be constructed in $\mathcal{O}(|E| \log |E|)$ time. For each function h in the hash family, we construct in $O(|V|^2)$ time the graph $G_h := (V, E_b^h, E_r^h)$ with blue and red edges by adding a red edge for each non-edge of G and adding all edges $e \in E$ with $h(e) = 1$ or $h(e) = 2$ to the set E_b^h of blue edges. Thus, we can think of the edges with $h(e) = 3$ as being deleted.

The graph G contains a triangle if and only if G_h contains a blue P_3 for some hash function h of the hash family: If G contains a triangle, then there exists a function h in which the edges of the triangle receive different hash values, since the hash family is $(|E|, 3)$-perfect. For this function, two of the three triangle edges are blue in G_h and one is a non-edge in G_h. Thus, G_h contains a blue P_3. The converse statement is also true since in any G_h the two blue edges of a blue P_3 and its non-edge are also edges of G because the non-edges of G are red edges of G_h.

By the above, we can find a triangle in G by searching for a blue P_3 in each G_h. If this can be done in $\mathcal{O}(|V|^2)$ time for each G_h, then the overall running time of the algorithm is $\mathcal{O}(|V|^2 \log |V|)$ since the hash family consists of $\mathcal{O}(\log |E|) = \mathcal{O}(\log |V|)$ functions. □

7. Conclusion

We investigated MODULE MAP, a new natural edge modification problem in graphs with two different edge types from a parameterized algorithmic perspective, presenting a nontrivial search tree algorithm and a kernel with a quadratic number of vertices. There are many open questions: Does MODULE MAP admit a kernelization with $\mathcal{O}(k)$ vertices? Can we compute a constant-factor approximation in polynomial time? Is MODULE MAP NP-hard when G_b is a cluster graph? Is MODULE MAP fixed-parameter tractable for smaller parameters, for example when parameterized by some lower bound on the solution size as it was done for CLUSTER EDITING [6]? Moreover, can we determine in $\mathcal{O}(|V| + |E|)$ time whether a bicolored graph contains a bicolored P_3? We find it also very intriguing to study further edge modification problems in graphs with multiple edge types from an algorithmic point of view. For example, in follow-up work we studied BICOLORED P_3 DELETION, where the aim is to destroy in an edge-bicolored graph all induced bicolored P_3s by at most k edge deletions, and showed for example that BICOLORED P_3 DELETION is NP-hard [17].

Concerning the biological application, it would be interesting to investigate how our model compares to the one of Amar and Shamir [4] in terms of biological results. In this context, one should also investigate the case where vertex pairs may have red and blue edges in the input graph and further possibilities to model the transformation of blue into red edges and vice versa.

Acknowledgments. We would like to thank the anonymous reviewers of *Discrete Applied Mathematics* for their many helpful remarks that have substantially

improved the presentation of the results in this paper.

References

[1] Abboud, A., Backurs, A., Williams, V. V., 2015. Tight hardness results for LCS and other sequence similarity measures. In: Proceedings of the 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS '15). IEEE Computer Society, pp. 59–78.

[2] Agrawal, A., Panolan, F., Saurabh, S., Zehavi, M., 2016. Simultaneous feedback edge set: A parameterized perspective. In: Proceedings of the 27th International Symposium on Algorithms and Computation (ISAAC '16). Vol. 64 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 5:1–5:13.

[3] Alon, N., Yuster, R., Zwick, U., 1995. Color-coding. Journal of the ACM 42 (4), 844–856.

[4] Amar, D., Shamir, R., 2014. Constructing module maps for integrated analysis of heterogeneous biological networks. Nucleic Acids Research 42 (7), 4208–4219.

[5] Bansal, N., Blum, A., Chawla, S., 2004. Correlation clustering. Machine Learning 56 (1-3), 89–113.

[6] van Bevern, R., Froese, V., Komusiewicz, C., 2018. Parameterizing edge modification problems above lower bounds. Theory of Computing Systems 62 (3), 739–770.

[7] Böcker, S., Briesemeister, S., Bui, Q. B. A., Truß, A., 2009. Going weighted: Parameterized algorithms for cluster editing. Theoretical Computer Science 410 (52), 5467–5480.

[8] Böcker, S., Briesemeister, S., Klau, G. W., 2011. Exact algorithms for cluster editing: Evaluation and experiments. Algorithmica 60 (2), 316–334.

[9] Bredereck, R., Komusiewicz, C., Kratsch, S., Molter, H., Niedermeier, R., Sorge, M., 2019. Assessing the computational complexity of multilayer subgraph detection. Network Science 7 (2), 215–241.

[10] Chen, J., Molter, H., Sorge, M., Suchý, O., 2018. Cluster editing in multilayer and temporal graphs. In: Proceedings of the 29th International Symposium on Algorithms and Computation, (ISAAC '18). Vol. 123 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 24:1–24:13.

[11] Cygan, M., Fomin, F. V., Kowalik, L., Lokshtanov, D., Marx, D., Pilipczuk, M., Pilipczuk, M., Saurabh, S., 2015. Parameterized Algorithms. Springer.

[12] Downey, R. G., Fellows, M. R., 2013. Fundamentals of Parameterized Complexity. Texts in Computer Science. Springer.

[13] Flum, J., Grohe, M., 2006. Parameterized Complexity Theory. Texts in Theoretical Computer Science. An EATCS Series. Springer.

[14] Fomin, F. V., Kratsch, D., 2010. Exact Exponential Algorithms. Texts in Theoretical Computer Science. An EATCS Series. Springer.

[15] Fomin, F. V., Kratsch, S., Pilipczuk, M., Pilipczuk, M., Villanger, Y., 2013. Tight bounds for parameterized complexity of cluster editing. In: Proceedings of the 30th International Symposion on Theoretical Aspects of Computer Science (STACS '13). Vol. 20 of LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, pp. 32–43.

[16] Gramm, J., Guo, J., Hüffner, F., Niedermeier, R., 2005. Graph-modeled data clustering: Exact algorithms for clique generation. Theory of Computing Systems 38 (4), 373–392.

[17] Grüttemeier, N., Komusiewicz, C., Schestag, J., Sommer, F., 2019. Destroying bicolored P_3s by deleting few edges. In: Proceedings of the 15th Conference on Computability in Europe (CiE '19). Vol. 11558 of LNCS. Springer, pp. 193–204.

[18] Hartung, S., Hoos, H. H., 2015. Programming by optimisation meets parameterised algorithmics: A case study for cluster editing. In: Proceedings of the 9th International Conference on Learning and Intelligent Optimization (LION '15). Vol. 8994 of LNCS. Springer, pp. 43–58.

[19] Jaroslav Nešetřil, S. P., 1985. On the complexity of the subgraph problem. Commentationes Mathematicae Universitatis Carolinae 026 (2), 415–419.

[20] Kivelä, M., Arenas, A., Barthelemy, M., Gleeson, J. P., Moreno, Y., Porter, M. A., 2014. Multilayer networks. Journal of Complex Networks 2 (3), 203–271.

[21] Komusiewicz, C., Uhlmann, J., 2012. Cluster editing with locally bounded modifications. Discrete Applied Mathematics 160 (15), 2259–2270.

[22] Krivánek, M., Morávek, J., 1986. NP-hard problems in hierarchical-tree clustering. Acta Informatica 23 (3), 311–323.

[23] Niedermeier, R., 2006. Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Mathematics and Its Applications. OUP Oxford.

[24] Shamir, R., Sharan, R., Tsur, D., 2004. Cluster graph modification problems. Discrete Applied Mathematics 144 (1-2), 173–182.

[25] Ulitsky, I., Shlomi, T., Kupiec, M., Shamir, R., 2008. From E-maps to module maps: dissecting quantitative genetic interactions using physical interactions. Molecular Systems Biology 4 (1), 209.

[26] Williams, V. V., 2012. Multiplying matrices faster than Coppersmith-Winograd. In: Proceedings of the 44th ACM Symposium on Theory of Computing (STOC '12). ACM, pp. 887–898.