Parameterized Algorithms for Module Map Problems

Frank Sommer and Christian Komusiewicz

Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Marburg, Germany
{fsommer,komusiewicz}@informatik.uni-marburg.de

Abstract. We introduce and study the NP-hard Module Map problem which has as input a graph G with red and blue edges and asks to transform G by at most k edge modifications into a graph which does not contain a two-colored K_3, that is, a triangle with two blue edges and one red edge, a blue P_3, that is, a path on three vertices with two blue edges, and a two-colored P_3, that is, a path on three vertices with one blue and one red edge, as induced subgraph. We show that Module Map can be solved in $\mathcal{O}(2^k \cdot n^3)$ time on n-vertex graphs and present a problem kernelization 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 should be similar to each other and objects between different clusters should be dissimilar. One of the central problems in this area is called Cluster Editing [3], also known as Correlation Clustering [18].

Cluster Editing
Input: An undirected graph $G = (V, E)$ and a non-negative 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?

Here, we essentially view the clustering problem 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 edges of G contradict this partition; these are exactly the deleted and inserted edges. In recent years, there has been an increased focus to model the observed data more precisely by incorporating different edge types. As a consequence, many data analysis tasks are now carried out on graphs with multiple edge types [7, 15]. In this work, we study a generalization of Cluster Editing in graphs with two types of edges.

To appear in *Proceedings of the 5th International Symposium on Combinatorial Optimization (ISCO '18)*, Marrakesh, Morocco, April 2018.

Module maps. The problem arises in the construction of so-called module maps in computational biology [2, 19]. Here, the input is a two-edge-colored graph $G = (V, E_b, E_r)$ with a set E_b of *blue* edges and a set E_r of *red* edges. 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 genetic interactions between the genes. These may be inferred, for example from a high correlation of expression levels of the genes [2]. 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 [2], 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 genetic interactions and, thus, many red edges between them; otherwise, there are few genetic interactions and, thus, few red edges between them. Amar and Shamir [2] discuss different objective functions for obtaining a module map that take these properties into account.

We study the problem of obtaining module maps from a graph modification point of view in the same spirit as Cluster Editing is a canonical graph modification problem for graph clustering. 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. By the above, each module is ideally a blue clique and there are no blue edges between different modules. 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, ideally for each pair of different clusters A and B there are 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 Fig. 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

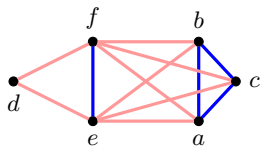

Fig. 1: A module graph with the clusters $\{a, b, c\}$, $\{d\}$, and $\{e, f\}$.

aim is to find a module graph which can be obtained from the input graph G by as few edge transformations as possible.

> Module Map
> **Input**: A graph $G = (V, E_b, E_r)$ and a non-negative 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, the module graph that is obtained by at most k edge modifications directly implies a partitioning of the input vertex set into clusters such that at most k vertex pairs 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, or a non-edge between different clusters that have a link and a red edge between different clusters that have no link. Our problem formulation is thus related to previous ones [2, 19] 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 [2, 19] 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.

As a consequence, hardness results for Cluster Editing transfer directly to Module Map. Since Cluster Editing is NP-complete [17] and cannot be solved in $2^{o(|V|+|E|)}$ time under a standard complexity-theoretic assumption [11, 16] we observe the following.

Proposition 1. Module Map *is NP-complete and cannot be solved in* $2^{o(|V|+|E|)}$ *time unless the Exponential-Time Hypothesis (ETH) fails.*

Because of this algorithmic hardness, heuristic approaches are used in practice [2, 19]. In this work, we are interested in exact algorithms for Module Map. In particular, we are interested in fixed-parameter algorithms that have a running time of $f(k) \cdot n^{o(1)}$ for a problem-specific parameter k. If k has moderate values and f grows not too fast, then these algorithms solve the problem efficiently [9]. Motivated by the practical success of fixed-parameter algorithms with the natural parameter number k of edge transformations for Cluster Editing [6, 13], 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, 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. To obtain the cost of a set of edge deletions and additions, we multiply each edge modification $\{u, v\}$ with the weight $g(\{u, v\})$. For example, a blue edge $\{u, v\}$ with weight ω can be transformed into a non-edge with cost ω and into a red edge with cost 2ω. This gives the following problem:

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

Our results. In Section 2, we present a characterization of module graphs by three forbidden induced subgraphs and show how to determine whether a graph G contains one of these in linear time. This implies a simple linear-time fixed-parameter algorithm for Module Map with running time $\mathcal{O}(3^k \cdot (|V| + |E|))$, where $|E| = |E_b| + |E_r|$.

In Section 3, 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 [5]. 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 (blue edge, red edge, or 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.

Finally, in Section 4 we show that Weighted Module Map admits a problem 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 kernelization and then using the search tree algorithm.

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 [15] or edge-colored (multi)graphs.

Chen et al. [8] introduced Multi-Layer Cluster Editing, a variant of Cluster Editing with multiple edge types. In this problem, 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. [8] show fixed-parameter algorithms and hardness results for different parameter combinations. The problem differs from Module Map in the sense that all edge types play the same role in the problem definition and that layers are evaluated independently whereas in Module Map the aim is to obtain one graph with blue and red edges that fulfills different properties for the blue and red edges. A further problem studied in this context is Simultaneous Feedback Vertex Set [1] where the aim is to delete at most k vertices in a multilayer graph such that each layer is acyclic. Further, Bredereck et al. [7] present several algorithmic and hardness results for a wide range of subgraph problems in multilayer graphs.

Preliminaries. We follow standard notation in graph theory. For a graph $G = (V, E)$ and a set $V' \subseteq V$, the *subgraph of G induced by V'* is denoted by $G[V'] := (V', \{\{u, v\} \in E \mid u, v \in V'\})$. 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 transformations (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 (E'_b, E'_r) is $|E'_b| + |E'_r|$. The graph G' is called *target graph*. A solution $\mathcal{S}$ is *optimal* if every other solution is at least as large as $\mathcal{S}$.

For the basic definitions on parameterized complexity such as fixed-parameter tractability and kernelization, we refer to the literature [9]. We present our kernelization via *reduction rules*. A reduction rule is *safe* if the resulting instance is equivalent. 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 instances $(I_1, k_1), \ldots, (I_\ell, k_\ell)$ of the same problem such that $k_i < k$. A branching rule is *safe* if (I, k) is a yes-instance if and only if there exists a j such that (I_j, k_j) is a yes-instance. A standard tool in the analysis of search tree algorithms are branching vectors; for further background refer to the monograph of Fomin and Kratsch [10].

Due to lack of space, several proofs are deferred to a long version of the article.

2 Basic Observations

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 Fig. 2: a *blue* P_3 is a path on three vertices consisting of two blue edges, a *two-colored* K_3 is a clique of size three, where one edge is red and the other two are blue, and a *two-colored* P_3 is a path on three vertices with exactly one blue and one red edge.

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

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

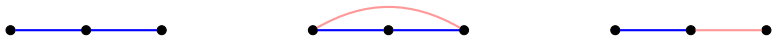

Fig. 2: The forbidden induced subgraphs for module graphs. From left to right: a blue P_3, consisting of two (dark) blue edges, a two-colored K_3, consisting of two blue and one (light) red edge and a two-colored P_3, consisting of one blue and one red edge.

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

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. 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 two-colored 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 a two-colored K_3. We can find blue P_3s in linear time. We also would like to find two-colored K_3s in linear time. As we show in the following, however, under a standard assumption in complexity theory it is impossible to find a two-colored K_3 in $\mathcal{O}(|V| + |E|)$ time.

The current best algorithm to determine if a graph G contains a triangle has a running time of $\mathcal{O}(|V|^\omega)$ time, where $\omega < 2.376$ is the exponent of the time that is needed to multiply two $n \times n$ matrices [14].

Proposition 2. *We cannot find a two-colored K_3 in a graph $G = (V, E_b, E_r)$ in $\mathcal{O}(|V|+|E|)$ time, unless we can detect triangles in $\mathcal{O}((|V|+|E|)\cdot\log|V|)$ time.*

Instead, we obtain a linear-time algorithm by searching in one step for two-colored K_3s *and* for blue P_3s.

Lemma 2. *For a two-colored graph $G = (V, E_b, E_r)$ we can find in $\mathcal{O}(|V|+|E|)$ time a blue P_3 or a two-colored K_3 if G contains either one.*

Now we show how to find a two-colored 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 two-colored K_3.

Lemma 3. *A two-colored P_3 in a graph $G = (V, E_b, E_r)$ which contains no blue P_3 and no two-colored K_3 can be found in $\mathcal{O}(|V| + |E|)$ time if it exists.*

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 a forbidden subgraph 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. If G contains a blue P_3 with

vertex set $\{u, v, w\}$ and non-edge $\{u, w\}$, then transform $\{u, w\}$ into a blue edge in the first case, transform $\{u, v\}$ into a non-edge in the second case, and transform $\{v, w\}$ into a non-edge in the third case. In each case, decrease k by one and solve the resulting instance recursively. The treatment of the other forbidden subgraphs is similar: If G contains a two-colored P_3, transform the blue edge into a non-edge, or transform the red edge into non-edge, or transform the non-edge into a red edge (observe that the case where a non-edge is transformed into blue edge need not be considered since this produces a two-colored K_3). If G contains a two-colored K_3, either transform one of the blue edges into a non-edge or transform the red edge into a blue edge. For each forbidden induced subgraph, the algorithm branches into three cases and decreases k by at least one. This leads to a branching vector of $(1, 1, 1)$. Since branching is performed only as long as $k > 0$, 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. Altogether, we obtain the following.

Proposition 3. 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 $\mathcal{O}(|V|^2)$.

Proposition 4. Weighted Module Map *can be solved in* $\mathcal{O}(3^k \cdot |V|^2)$ *time.*

3 An Improved Search Tree Algorithm

To improve the running time, we adapt a branching strategy for Cluster Editing [5]. To apply this strategy, we first introduce a generalization of Weighted Module Map. Then, we explain our branching strategy. Finally, we solve certain instances in polynomial time to obtain an $\mathcal{O}(2^k \cdot |V|^3)$-time search tree algorithm.

A more flexible scoring 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 transformation costs are directly related. From now on, we allow independent transformation costs for the different possibilities. To this end, we introduce an *edge-cost function* $s : \binom{V}{2} \to \mathbb{R}^3$ for all pairs of vertices $\{u, v\}$ of a given graph G where $s(u, v) := (b_{u,v}, n_{u,v}, r_{u,v})$. This 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 also write $(b, n, r)_{u,v}$. If there is no danger of confusion we omit the index of the associated vertices

u and v. For example, let $\{u, v\}$ be a blue edge in an instance of Weighted Module Map with weight ω. Then we get cost vector $(0, \omega, 2\omega)$.

We call a vertex pair $\{u, v\}$ with its cost vector (b, n, r) a *blue pair* if $b = 0$ and $n, r > 0$, a *non-pair* if $n = 0$ and $b, r > 0$, and a *red pair* if $r = 0$ and $b, n > 0$. As for unweighted graphs, three vertices u, v, and w form a *blue* P_3 if $\{u, v\}$ and $\{u, w\}$ are blue and $\{v, w\}$ is a non-pair, they form a *two-colored* K_3 if $\{u, v\}$ and $\{u, w\}$ are blue and $\{v, w\}$ is red, they form a *two-colored* P_3 if $\{u, v\}$ is blue, $\{u, w\}$ is red and $\{v, w\}$ is a non-pair. Finally, a graph G is called a *pair module graph* if each pair $\{u, v\}$ of vertices in G is a blue pair, a non-pair or a red pair and G contains no blue P_3, two-colored K_3 and two-colored P_3.

We do not allow arbitrary scoring functions but demand the following three properties. The first property restricts the relation between the three costs.

Property 1. For each cost vector $(b, n, r)_{u,v}$, we have $b + r \geq 2n$.

Property 1 is essentially a more relaxed version of the property that transforming a blue edge into a red edge is at least as expensive as transforming this edge first into a non-edge and subsequently into a red edge.

Property 2. In each cost vector $s(u, v)$ either all components are non-negative integers or all three are non-negative and half-integral. In the latter case, at least two components are equal to $1/2$.

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

The final property demands that each vertex pair whose cost vector is not half-integral has exactly one component equal to zero. This guarantees the unambiguous construction of a pair module graph from each vertex pair.

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

Properties 1–3 are fulfilled by the scoring function obtained from Weighted Module Map instances. Moreover, we can observe the following.

Proposition 5. *Let $(b, n, r)_{u,v}$ be a cost vector fulfilling Property 1–3.*

- *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 and that only one component is zero. The third claim can be seen as follows. By Property 2, all three components are half-integral and at least two of them are equal to $1/2$. By Property 1, $b + r \geq 2n$. If $n > 1/2$, then $b = 1/2 = r$ and Property 1 is violated. Thus, $n = 1/2$. □

We may now define Module Map with Scoring Function (MMS).

MMS
Input: A graph G with an edge-cost function $s : \binom{V}{2} \to \mathbb{R}^3$ which fulfills Properties 1–3 and a non-negative integer k.
Question: Can we transform G into a pair module graph with transformation costs at most k?

Our aim is to show the following.

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

Merge-based branching. We branch on blue pairs since each forbidden subgraph contains at least one blue edge. In one case we will delete this blue edge and in the other case we will keep this blue edge.

Definition 2. *A blue pair* $\{u, v\}$ *forms a* conflict triple *with a vertex* w *if* $\{u, w\}$ *and* $\{v, w\}$ *are not both blue, not both non-pairs, or not both red.*

To resolve all conflict triples, we branch on blue pairs that are contained in at least two conflict triples. In the corresponding branching, similar to the approach of Böcker et al. [5], we *merge* the vertex pair $\{u, v\}$ in one of the cases.

Definition 3. *Let* (G, s, k) *be an instance of* MMS. Merging *two vertices* u *and* v *is the following operation: Remove* u *and* v *from* G *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 $s(v, w)$. Note that $s(u', w)$ may not fulfill Properties 1–3. Because of this, we have to *reduce* joint cost vectors as far as possible. Herein, reducing (b, n, r) by a value t is to decrease each of its components by t. Simultaneously we can reduce the parameter k by t.

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

Let $x = min(b, n, r)$ be a minimal value of (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 give a parameter decrease for vectors such as $(1, 1, 3)$. According to the bookkeeping trick introduced in [5], in such a case, we reduce this vector by $x-1/2$ to circumvent the above problem. For example, we reduce the vector $(1, 1, 3)$ by $1/2$ and get vector $(1/2, 1/2, 5/2)$.

Branching Rule 1. *Let* (G, s, k) *be an instance of* MMS. *If* (G, 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 Reduction Rule 1.*
Case 2 : Merge the vertex pair $\{u, v\}$*. Afterwards apply Reduction Rule 1.*

Lemma 4. *Branching Rule 1 is safe.*

Now we prove that Properties 1–3 remain true if we set $b_{u,v} := k + 1$ for a blue pair $\{u, v\}$ and apply Reduction Rule 1, and if we merge a blue pair $\{u, v\}$ and apply Reduction Rule 1.

Lemma 5. *Let $\{u, v\}$ be a blue pair in an instance (G, s, k) of* MMS*, and let (G', s', k') be obtained by setting $b_{u,v} := k + 1$ and applying Reduction Rule 1 or by merging u and v and applying Reduction Rule 1. Then, (G', s', k') is an instance of* MMS*. In particular, s' fulfills Properties 1–3.*

We now show that if we merge a blue pair $\{u, v\}$ where $\{u, w\}$ and $\{v, w\}$ are not both blue, non-, or red, then Reduction Rule 1 reduces the resulting cost vector $(b, n, r)_{u',w}$ by at least $1/2$.

Proposition 6. *Let $s(u', w)$ be a joint cost vector that is the join of two cost vectors $s(u, w)$ and $s(v, w)$, where $\{u, w\}$ and $\{v, w\}$ are not both blue, non- or red. Then Reduction Rule 1 applied to $s(u', w)$ decreases k by at least $1/2$.*

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

Lemma 6. *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 Reduction Rule 1 to (b^*, n, r) decreases k by at least 1 or this instance has no solution.*

Proof. Form Proposition 5 we conclude: $b = 0$, $n \geq 1$ and $r \geq 2n$. If $n \geq k$, then each component of (b^*, n, r) will be larger than k. Hence there exists no solution for this instance. Thus, the reduced cost vector is integral with a unique minimum component. Consequently, it is reduced by at least 1. □

Now consider the instances obtained by an application of Branching Rule 1. In Case 1, the new parameter is at most $k - 1$ due to Lemma 6. In Case 2, the new parameter is also at most $k - 1$ because $\{u, v\}$ is in two conflict triples. By Proposition 6 this means that we create two cost vectors which are both reduced by at least $1/2$.

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

Applying Branching Rule 1 states that every blue pair which is contained in at least two conflict triples has branching vector $(1, 1)$ or better. Branching Rule 2 deals with blue pairs $\{u, v\}$ which are contained in exactly one conflict triple with vertex w where $\{u, w\}$ and $\{v, w\}$ give a join that can be reduced by at least 1.

Branching Rule 2. *If (G, s, k) contains a blue pair $\{u, v\}$ and a vertex w such that u, v, and w form a conflict triple and the joined vertex pair can be reduced by at least 1, then branch into the following two cases:*

Case 1 : Set $b_{u,v} := k + 1$. Afterwards apply Reduction Rule 1.
Case 2 : Merge *the vertex pair $\{u, v\}$. Afterwards apply Reduction Rule 1.*

Lemma 7. *Branching Rule 2 is correct and has branching vector $(1, 1)$ or better.*

Solving the remaining instances in polynomial time. We now show that instances to which Branching Rules 1 and 2 do not apply can be solved efficiently.

Lemma 8. *Let (G, s, k) be an instance of* MMS*. If Branching Rules 1 and 2 do not apply, then (G, s, k) can be solved in $\mathcal{O}(|V|^2)$ time.*

We now can prove Theorem 2.

Proof (of Theorem 2). First, check for each blue pair $\{u, v\}$ if Branching Rule 1 or 2 applies. This needs $\mathcal{O}(|V|^3)$ time. If this is the case, we will branch on $\{u, v\}$. According to Corollary 1 and Lemma 7, Branching Rules 1 and 2 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. Hence, we delete the vertices u and v from the graph and replace them by a new vertex u' and join all incident pairs of vertices. These are n pairs. So we can calculate the cost vector for each new, joined pair in $\mathcal{O}(|V|)$ time. In both cases we reduce the parameter accordingly. Hence, we need $\mathcal{O}(|V|^3)$ time per search tree node. By Lemma 8, MMS can be solved in $\mathcal{O}(|V|^2)$ time if Branching Rules 1 and 2 do not apply. Hence, we obtain a $\mathcal{O}(2^k \cdot |V|^3)$-time algorithm for MMS. □

4 A Polynomial Problem Kernel

We also obtain a problem kernelization for Weighted Module Map that yields problem kernel with $\mathcal{O}(k^2)$ vertices. The basic idea is the following: Let $\{u, v\}$ be a vertex pair of an instance $(G = (V, E_b, E_r), k, g)$ of Weighted Module Map. We investigate if it is possible that the vertex pair $\{u, v\}$ can be a blue, 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 [5, 12].

Theorem 3. Weighted Module Map *admits a problem kernel of $\mathcal{O}(k^2)$ vertices which can be found in $\mathcal{O}(|V|^3 + k \cdot |V|^2)$ time.*

5 Conclusion

There are many open questions: Does Module Map admit a problem kernel 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 above a lower bound as it was done for Cluster Editing [4]?

References

[1] A. Agrawal, D. Lokshtanov, A. E. Mouawad, and S. Saurabh. Simultaneous feedback vertex set: A parameterized perspective. In *Proc. 33rd STACS*, volume 47 of *LIPIcs*, pages 7:1–7:15. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[2] D. Amar and R. Shamir. Constructing module maps for integrated analysis of heterogeneous biological networks. *Nucleic Acids Research*, 42(7): 4208–4219, 2014.
[3] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1-3):89–113, 2004.
[4] R. van Bevern, V. Froese, and C. Komusiewicz. Parameterizing edge modification problems above lower bounds. *Theory of Computing Systems*, 62 (3):739–770, 2018.
[5] S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truß. Going weighted: Parameterized algorithms for cluster editing. *Theoretical Computer Science*, 410(52):5467–5480, 2009.
[6] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.
[7] R. Bredereck, C. Komusiewicz, S. Kratsch, H. Molter, R. Niedermeier, and M. Sorge. Assessing the computational complexity of multi-layer subgraph detection. In *Proc. 10th CIAC*, volume 10236 of *LNCS*, pages 128–139, 2017.
[8] J. Chen, H. Molter, M. Sorge, and O. Suchý. A parameterized view on multi-layer cluster editing. *CoRR*, abs/1709.09100, 2017.
[9] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.
[10] F. V. Fomin and D. Kratsch. *Exact Exponential Algorithms*. Texts in Theoretical Computer Science. An European Association for Theoretical Computer Science Series. Springer, 2010.
[11] F. V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and Y. Villanger. Tight bounds for parameterized complexity of cluster editing. In *Proc. 30th STACS*, volume 20 of *LIPIcs*, pages 32–43. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
[12] J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Exact algorithms for clique generation. *Theory of Computing Systems*, 38(4):373–392, 2005.
[13] S. Hartung and H. H. Hoos. Programming by optimisation meets parameterised algorithmics: A case study for cluster editing. In *Proc. 9th LION*, volume 8994 of *LNCS*, pages 43–58. Springer, 2015.
[14] A. Itai and M. Rodeh. Finding a minimum circuit in a graph. *SIAM Journal on Computing*, 7(4):413–423, 1978.
[15] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. Multilayer networks. *J. Complex Networks*, 2(3):203–271, 2014.
[16] C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012.
[17] M. Krivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Informatica*, 23(3):311–323, 1986.
[18] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.
[19] I. Ulitsky, T. Shlomi, M. Kupiec, and R. Shamir. From E-maps to module maps: dissecting quantitative genetic interactions using physical interactions. *Molecular Systems Biology*, 4(1):209, 2008.