# Partitioning into Colorful Components by Minimum Edge Deletions

Sharon Bruckner[1⋆], Falk Hüffner[2⋆⋆], Christian Komusiewicz[2], Rolf Niedermeier[2], Sven Thiel[3], and Johannes Uhlmann[2⋆⋆⋆]

[1] Institut für Mathematik, Freie Universität Berlin, sharonb@mi.fu-berlin.de
[2] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, {falk.hueffner,christian.komusiewicz,rolf.niedermeier}@tu-berlin.de johannes.uhlmann@campus.tu-berlin.de
[3] Institut für Informatik, Friedrich-Schiller-Universität Jena, sven.thiel@uni-jena.de

**Abstract.** The NP-hard Colorful Components problem is, given a vertex-colored graph, to delete a minimum number of edges such that no connected component contains two vertices of the same color. It has applications in multiple sequence alignment and in multiple network alignment where the colors correspond to species. We initiate a systematic complexity-theoretic study of Colorful Components by presenting NP-hardness as well as fixed-parameter tractability results for different variants of Colorful Components. We also perform experiments with our algorithms and additionally develop an efficient and very accurate heuristic algorithm clearly outperforming a previous min-cut-based heuristic on multiple sequence alignment data.

## 1 Introduction

We study a maximum parsimony approach to the discovery of heterogeneous components in vertex-colored graphs:

> Colorful Components
> **Instance:** An undirected graph $G = (V, E)$ and a coloring of the vertices $\chi : V \to \{1, \ldots, c\}$.
> **Task:** Find a minimum subset of edges $E' \subseteq E$ such that in $G' = (V, E \setminus E')$, all connected components are *colorful*, that is, they do not contain two vertices of the same color.

Such an edge set $E'$ is called a *solution*, and we denote its size by $k$. Colorful Components is an edge modification problem originating from biological applications in sequence and network alignment as described next.

The first application of Colorful Components stems from *Multiple Sequence Alignment*. This is the process of aligning at least three protein, DNA, or

⋆ Supported by project NANOPOLY (PITN-GA-2009–238700).
⋆⋆ Supported by DFG project PABI (NI 369/7-2).
⋆⋆⋆ Supported by DFG project PABI (NI 369/7-2).

RNA sequences such that positions believed to be homologous, that is, resulting from inheritance from a common ancestor, are written in a common column. This serves to illustrate the similarity or dissimilarity between the sequences and makes it possible to investigate their evolutionary relationship. Corel et al. [5] present an algorithm for this problem where a central step is to find connected subgraphs in graphs whose vertices are positions of the sequences, edges indicate that a pair of positions should be aligned, and the colors one-to-one correspond to sequences. These subgraphs correspond to partial alignment columns and thus may contain at most one vertex from each input sequence. This yields the Colorful Components problem. The solution of Colorful Components is then used by the DIALIGN software to compute a multiple alignment. Corel et al. [5] solve Colorful Components using a greedy algorithm, subsequently called "min-cut heuristic": Find two vertices of the same color in some connected component, find a minimum edge cut between them, and remove it; repeat this until all connected components are colorful.

A second biological motivation for Colorful Components arises in *Network Alignment* for multiple protein–protein interaction (PPI) networks. We propose a method for network alignment that is based on solving Colorful Components. Given networks $G_i = (V_i, E_i)$ and a similarity relation $S$ between the proteins of different species, first create a network whose vertex set is $\bigcup V_i$ and in which vertex $v \in V_i$ receives color $i$. Then, add an edge $\{u, v\}$ if $uSv$. The detected colorful components are then sets of matched proteins. Every protein appears in exactly one component, and every component has at most one protein from each species, which is a very strict model. The results can then be viewed as functional orthologs [13], or they can form the basis for further analysis. Deniélou et al. [6] suggest a three-step framework for network alignment where the first step is to aggregate the proteins from the different species into subsets, and Colorful Components offers a way of performing this task that results in consistent, disjoint aggregated groups.

*Related combinatorial problems.* Colorful Components can be seen as the problem of destroying by edge deletions all *bad paths*, that is, simple paths between two vertices of the same color. Thus, it is a special case of the well-known NP-hard Multicut problem, which has as input an undirected graph and a set of vertex pairs and asks for a minimum number of edges to delete to disconnect each given vertex pair. Multicut is fixed-parameter tractable with respect to the number $k$ of edge deletions, with a running time of $2^{O(k^3)} \cdot |V|^{O(1)}$ [3, 12].

Colorful Components is also a special case of Multi-Multiway Cut [1]. This problem asks to disconnect by edge deletions all paths between vertices from the same vertex set of some given vertex sets. Thus, Colorful Components is the special case where the vertex sets form a partition. Finally, there is related work on "clustering with diversity" [10] which extends a traditional clustering problem by asking that in each resulting cluster all points of the underlying colored metric space must have different colors.

*Contributions.* On the theoretical side, we present a first systematic study on the computational complexity of Colorful Components, exhibiting both tractable and intractable cases. First, we observe that Colorful Components is NP-hard even in trees. Then, we present a complexity dichotomy concerning the number $c$ of colors showing that Colorful Components is polynomial-time solvable for two or less colors and NP-hard otherwise. For three or more colors, we also obtain super-polynomial running time lower bounds (based on the Exponential Time Hypothesis) even in the case that the input graph has bounded degree. On the positive side, we present fixed-parameter algorithms with running time $2^c \cdot |V|^{O(1)}$ for Colorful Components in trees and with running time $O((c-1)^k \cdot |E|)$ in general graphs. In experimental work we demonstrate that, somewhat surprisingly, we can get better results by solving the more general Weighted Multi-Multiway Cut problem, since this allows us to merge vertices. We take advantage of this in data reduction rules, a simplified branching, and a new heuristic. With the branching algorithms, we can solve to optimality more than half of the instances generated from the BAliBASE 3.0 benchmark [14] each time within five minutes on a standard PC, with up to 5 000 vertices and 13 000 edges. Our heuristic has an average error of 0.6 %, a large improvement over the 29.2 % of the previously suggested min-cut heuristic [5]. We also show the strength of the developed data reduction rules.

*Preliminaries.* We consider only undirected and simple graphs $G = (V, E)$ where $n := |V|$ and $m := |E|$. We assume that $n = O(m)$ since isolated vertices can be removed from the input in linear time. A *bad path* is a simple (that is, cycle-free) path between two vertices of the same color. The length of a path is the number of its edges. An *edge cut* is a set of edges whose deletion increases the number of connected components. For a nonnegative number $t$, a graph is *$t$-edge connected* if it does not have an edge cut of size less than $t$.

The Exponential Time Hypothesis (ETH) states that, for all $x \geq 3$, $x$-SAT, which asks whether a boolean input formula in conjunctive normal form with $n$ variables and $m$ clauses and at most $x$ variables per clause is satisfiable, cannot be solved within a running time of $2^{o(n)}$ or $2^{o(m)}$; see Lokshtanov et al. [11] for a recent survey. A problem is *fixed-parameter tractable* with respect to a parameter $k$ if it can be solved in $f(k) \cdot n^{O(1)}$ time for an arbitrary (typically exponential) function $f$ in $k$.

## 2 Computational Hardness

In this section, we present hardness results for two restricted variants of Colorful Components.

First, we consider the special case where the input graph is a tree. For obtaining our hardness result, we exploit the connection between Colorful Components and Multicut. Note that Multicut is NP-hard and MaxSNP-hard even if the input is a star, that is, a tree consisting of a central vertex with attached degree-1 vertices [7]. Multicut in stars can be reduced to Colorful

Components as follows: for every pair $\{s, t\}$ to be disconnected, create degree-1 vertices $s'$ and $t'$ attached to $s$ and $t$, respectively, and color $s'$ and $t'$ with the same unique color. Each original vertex gets a further unique color. Since this reduction produces trees whose diameter is four, we arrive at the following.

**Proposition 1.** Colorful Components *is NP-hard even in trees with diameter four.*

In stars, however, Colorful Components turns out to be polynomial-time solvable: If the central vertex $v$ has two neighbors with the same color, one can delete the edge between $v$ and one of the two identically colored degree-one vertices. If $v$ has no two neighbors of the same color, then every connected component is colorful.

Second, we study the computational complexity of Colorful Components if the number of colors is fixed. This is of interest since the number of colors may be small in practical cases.

**Theorem 1.** Colorful Components *with three colors in graphs with maximum degree six is NP-hard; it cannot be solved in* $2^{o(k)} \cdot n^{O(1)}$, $2^{o(n)} \cdot n^{O(1)}$, *or* $2^{o(m)} \cdot n^{O(1)}$ *time unless the ETH is false.*

*Proof.* We present a polynomial-time many-to-one reduction from the NP-hard 3-SAT problem which has as input a Boolean formula $\phi$ in 3-CNF.[4] For simplicity, we assume that every clause contains *exactly* three literals.

The basic idea of the reduction is as follows. For each variable $x_i$ of a given 3-CNF formula $\phi$, we construct a *variable cycle* of length $4m_i$, where $m_i$ denotes the number of clauses that contain $x_i$. These cycles are colored alternatingly with two colors $c_e$ and $c_o$ such that deleting every second edge yields a minimum-cardinality edge deletion set for obtaining colorful components for this cycle. The corresponding two possibilities are used to represent the two choices for the value of $x_i$. Then, for each clause $C_j$ of $\phi$ containing the variables $x_p$, $x_q$, and $x_r$, we connect the three corresponding variable cycles by a clause gadget. This gadget has the property that if the solutions for the variable gadgets correspond to an assignment that satisfies $C_j$, then one needs only four edge deletions for the clause gadget. Conversely, if four edge deletions are sufficient, then the assignment corresponding to the deletions in the variable cycle satisfies $C_j$. Let $m$ be the number of clauses in $\phi$ and observe that, since $\phi$ is a 3-CNF formula, the overall number of vertices in the variable cycles is $12m$. Our construction guarantees that there is a satisfying assignment for $\phi$ if and only if the constructed graph can be transformed into one with colorful components by exactly $6m + 4m = 10m$ edge deletions, where $6m$ edge deletions are used for the variable cycles and $4m$ modifications are used for the clause gadgets. The details follow.

Given a 3-CNF formula $\phi$ consisting of the clauses $C_0, \ldots, C_{m-1}$ over the variables $\{x_0, \ldots, x_{n-1}\}$, construct a Colorful Components-instance ($G =$

[4] A similar reduction type was previously used to show analogous results for Transitivity Editing [15] and Cluster Editing [9], which, in contrast, are defined on uncolored graphs.

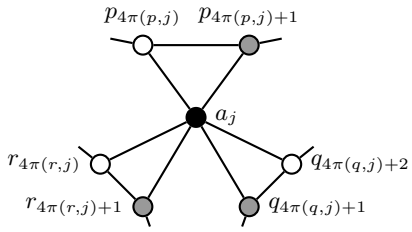


**Fig. 1.** The clause gadget for clause $C_j = (x_p \vee \bar{x}_q \vee x_r)$. White vertices have color $c_e$, gray vertices have color $c_o$, and black vertices have color $c_g$. The vertex $a_j$ is the reserved vertex for $C_j$, the other vertices lie on the variable cycles for $x_p$, $x_q$, and $x_r$, respectively.

$(V, E), k)$ as follows. For each variable $x_i$, $0 \leq i < n$, $G$ contains a *variable cycle* consisting of the vertices $V_i^v := \{i_0, \ldots, i_{4m_i-1}\}$ and the edges $E_i^v := \{\{i_k, i_{k+1}\} \mid 0 \leq k < 4m_i\}$ (for ease of presentation let $i_{4m_i} = i_0$). An edge $\{i_x, i_{x+1}\}$ is *even* if $x$ is even, and *odd* otherwise. A vertex $i_x$ receives color $c_e$ if $x$ is even; otherwise, it receives color $c_o$. So far, the constructed graph consists of a disjoint union of cycles and has $12m$ vertices and edges.

Next, add a *clause gadget* to $G$ for each clause of $\phi$. In the construction of the clause gadgets, we need for each clause $C_j$ in the variable cycles of $C_j$'s variables a fixed set of vertices that are "reserved" for $C_j$. To this end, suppose that for each variable $x_i$ an arbitrary but fixed ordering of the clauses containing $x_i$ is given, and let $\pi(i, j) \in \{0, \ldots, 4m_i - 1\}$ denote the position of a clause $C_j$ that contains $x_i$ in this ordering. We now give the details of the construction of the clause gadgets. Let $C_j$ be a clause containing the variables $x_p$, $x_q$, and $x_r$ (either negated or nonnegated). We construct a clause gadget connecting the variable cycles of $x_p$, $x_q$, and $x_r$. First, let $a_j$ be a new vertex that appears only in the clause gadget for clause $C_j$ and color $a_j$ with a third color $c_g$. Let $E_j^g$ denote the edge set of the clause gadget and let $E_j^g$ contain for each $i \in \{p, q, r\}$ the edges $\{a_j, i_{4\pi(i,j)}\}$ and $\{a_j, i_{4\pi(i,j)+1}\}$ if $x_i$ occurs nonnegated in $C_j$ or the edges $\{a_j, i_{4\pi(i,j)+1}\}$ and $\{a_j, i_{4\pi(i,j)+2}\}$, otherwise. See Figure 1 for an illustration. The construction of $G = (V, E)$ is completed by setting $V := \bigcup_{i=0}^{n-1} V_i^v \cup \bigcup_{j=0}^{m-1} \{a_j\}$ and $E := \bigcup_{i=0}^{n-1} E_i^v \cup \bigcup_{j=0}^{m-1} E_j^g$.

We show the correctness of the reduction by showing the following claim.

> $\phi$ is satisfiable $\Leftrightarrow$ $G$ can be transformed into a graph with colorful components by deleting at most $k := 10m$ edges.

"$\Rightarrow$": Given a satisfying assignment $\beta$ for $\phi$, we can transform $G$ into a graph with colorful connected components as follows. For each variable $x_i$ delete the odd edges of the variable cycle of $x_i$ if $\beta(x_i) = \text{true}$ and the even edges otherwise. After these deletions, there are no bad paths that contain only vertices from the variable cycles. Then, proceed as follows for each clause $C_j$. Assume without loss of generality that $C_j$ contains the variables $x_p$, $x_q$, and $x_r$, and that the literal corresponding to $x_p$ is true. Then, delete the four edges that are incident with $a_j$

and with one vertex of the variable cycles of $x_q$ and $x_r$. After the deletion of these four edges, there are no bad paths that contain $a_j$, which can be seen as follows. Clearly, $a_j$ is only adjacent to two vertices on the variable cycle of $x_p$. Since the literal corresponding to $x_p$ in $C_j$ is true, the edge between these two vertices corresponds to the truth assignment that makes this literal true. Consequently, this edge is *not* deleted and the edges of the variable cycle that are before and after this edge are deleted. Hence, $a_j$ and its two neighbors in the variable cycle of $x_p$ form an isolated triangle with three different colors.

Summarizing, this means that after deleting the four edges as described above for each clause gadget, all bad paths containing some $a_j$ have been destroyed. The overall number of edge deletions is $10m$: For the variable cycles, we perform altogether $\sum_{0 \leq i < n} 4m_i/2 = 6m$ edge deletions, and for each clause gadget four edges are deleted.

"$\Leftarrow$": Let $S$ denote an optimal solution for $G$ with $|S| \leq k := 10m$. To show that $\phi$ is satisfiable, we make some observations about the structure of $G$ and $S$.

First, we show that $10m$ is a lower bound on any solution for $G$, that is, $|S| \geq 10m$ and thus $|S| = 10m$. First, note that for each variable $x_i$ the variable cycle contains $4m_i/2$ edge-disjoint bad paths. Hence, $G$ contains overall $6m$ edge-disjoint induced bad paths such that all vertices of the bad paths are in the variable cycles. Clearly, at least $6m$ edge deletions are needed for these bad paths. For each clause $C_j$, $0 \leq j < m$, at least four edges incident with $a_j$ have to be deleted since $a_j$ has degree six and can have degree at most two in a colorful component. Hence, every solution has size at least $6m + \sum_{0 \leq j < m} 4 = 10m$ and thus $|S| = 10m$.

Now, since at least $6m$ edges are deleted in the variable cycles, this means that for each clause $C_j$ *exactly* four edges incident with $a_j$ are deleted by $S$. Consequently, for each variable cycle either all even or all odd edges are deleted.

Consider the assignment $\beta$ for $\phi$ that, for each $x_i$, $0 \leq i < n$, sets $\beta(x_i) :=$ true if all odd edges of $V_i^v$ are deleted and sets $\beta(x_i) :=$ false if all even edges of $V_i^v$ are deleted. We show that $\beta$ is a satisfying assignment. Consider an arbitrary clause $C_j$ containing the variables $x_p$, $x_q$, and $x_r$. Since $a_j$ is in a colorful component after the edge deletions, it can have at most two neighbors. Furthermore, these neighbors must be on the same variable cycle: otherwise, $a_j$ would be in a connected component of size five, because after the edge deletions, every vertex is adjacent to exactly one further vertex on its variable cycle. Hence, after the edge deletions, $a_j$ is adjacent to at most two vertices of the variable cycle of one of $x_p$, $x_q$, and $x_r$ of $C_j$. Let $x_p$ be this variable. Furthermore, since exactly four edge deletions are incident with $a_j$, *both* edges that are incident with the vertices of the variable cycle of $x_p$ are not deleted by $S$. Without loss of generality, assume that $x_p$ appears nonnegated in $C_j$. Then the two vertices of $V_p^v$ that are adjacent to $a_j$ are $p_{4\pi(p,j)}$ and $p_{4\pi(p,j)+1}$. Since $S$ is a solution, the edge $\{p_{4\pi(p,j)}, p_{4\pi(p,j)-1}\}$ is not deleted by $S$: otherwise, $a_j$ is in a connected component of size five. Hence, all odd edges of $V_p^v$ are deleted, and therefore the assignment $\beta$ fulfills clause $C_j$.

Altogether, this shows the correctness of the reduction. Since the reduction can be performed in polynomial time and produces a graph with maximum degree six, it implies NP-hardness in graphs with maximum degree six. Furthermore, for formulas with $m$ clauses, the reduction produces graphs with $|V| < 13m$, $k = 10m$ and $|E| = 18m$. Hence, any algorithm with running time $2^{o(k)} \cdot n^{O(1)}$, $2^{o(|V|)}$, or $2^{o(|E|)}$ implies an algorithm with running time $2^{o(m)}$ for 3-SAT, contradicting the ETH. □

## 3 Algorithms

*Two colors.* While Theorem 1 shows that Colorful Components is NP-hard for three colors, for two colors it can be solved in polynomial time via computing a maximum matching in bipartite graphs.

**Proposition 2.** Colorful Components *in two-colored graphs can be solved in* $O(\sqrt{n}m)$ *time.*

*Proof.* We begin by removing all edges $\{u, v\}$ where $u$ and $v$ have the same color. The remaining graph is bipartite, since it has a proper 2-coloring. This instance can be solved by computing a maximum matching: First, observe that the edges that are *not* deleted by a solution to Colorful Components must be a matching in the bipartite graph. This is because the degree of every vertex in the solution is either one (it is a part of a component of size two, which is the largest possible component size) or zero. Since maximizing the number of undeleted edges is equivalent to minimizing the number of deleted edges, we can obtain a minimum-cardinality solution by computing a maximum matching $M$, and then deleting all edges not contained in $M$.

Removing all edges between vertices with identical colors can be done in $O(m)$ time. A maximum matching in a bipartite graph can be found in $O(\sqrt{n}m)$ time using the Hopcroft–Karp algorithm. □

*An efficient algorithm in trees with few colors.* Let $T = (V, E)$ denote the input tree and assume that $T$ is rooted at an arbitrary vertex. The idea is to do dynamic programming bottom-up from the leaves, storing for each $v \in V$ and $C \subseteq \{1, \ldots, c\}$ the minimal cost $T(v, C)$ of a solution for the subtree rooted at $v$ where the connected component containing $v$ contains exactly the colors of $C$.

We describe the algorithm for binary trees. Define $T_v$ to be the subtree of the tree $T$ that is rooted in node $v$. We then find, for every vertex $v$ and every subset of colors $C \subseteq \{1, \ldots, c\}$, the minimal cost $T(v, C)$. We compute this by dynamic programming using a table $T[\cdot, \cdot]$. Performing a bottom-up traversal starting at the leaves, for each $v$ we compute $T(v, C)$ for every $C$. When computing the cost $T(v, C)$, we choose the minimal cost between four options: In the first case, we do not delete the two edges from $v$ to its children. The cost of the solution then is the minimum value of the sum of the costs of the two subtrees for every combination of colors that will give $C$. In the second and third case, we delete the edge to the left or the right subtree, respectively. The cost is obtained by

summing the cost of the solution for the subtree taken and the minimal possible cost for the rest of the tree. In the last case both edges are deleted. More formally, this reads as follows.
*Initialization:* For each leaf $v$, set $T[v, \{\chi(v)\}] := 0$ and $T[v, C] := \infty$ for $C \subseteq \{1, \ldots, c\}$ and $C \neq \{\chi(v)\}$.
*Recursion:* Let $l$ and $r$ denote the two children of an inner node $v$.

$$T[v, C] := \min \begin{cases} \min_{C_1 \uplus C_2 \uplus \{\chi(v)\} = C} T[l, C_1] + T[r, C_2], \\ T[r, C \setminus \{\chi(v)\}] + 1 + \min_{C' \subseteq X} T[l, C'], \\ T[l, C \setminus \{\chi(v)\}] + 1 + \min_{C' \subseteq X} T[r, C'], \\ 2 + \min_{C_1 \subseteq X} T[l, C_1] + \min_{C_2 \subseteq X} T[r, C_2] \end{cases}$$

The running time of this algorithm is $O(3^c \cdot n)$. The size of our dynamic programming table is $O(2^c \cdot n)$ to include all possible color subsets. Overall, the computation can be executed in $O(3^c \cdot n)$ time: for each vertex we need to consider at most $O(3^c)$ combinations of color subsets (every subset and the possibilities to split it into two). For each such combination the computation of the recursion can be performed in constant time if we maintain for each $v$ the minimum cost of $T_v$. To extract the actual colorful components found, one can use a traceback procedure within the same running time bound. The exponential factor can be further improved (increasing the polynomial factor) to $2^c$ by using the convolution-based techniques of Björklund et al. [2].

To extend this algorithm to work in general trees, we use a standard trick for dynamic programming in trees: Order the children of every node and add an additional dimension $i = 1, \ldots, d$ to the dynamic programming table, where $d$ is the maximum degree in the tree. We then compute $T[v, i, C]$ by an adaption of the above approach. We omit the straightforward details.

**Theorem 2.** Colorful Components *on trees can be solved in* $2^c \cdot n^{O(1)}$ *time.*

*An efficient fixed-parameter algorithm for graphs with few colors.* Whereas on general graphs due to Theorem 1 there is no hope for fixed-parameter tractability with respect to the parameter "number $c$ of colors", additionally using the parameter "number $k$ of edge deletions" leads to fixed-parameter tractability.

First, we describe a simple $O(c^k \cdot m)$-time search tree algorithm. Using breadth-first search, it finds a bad path between two vertices of the same color. This strategy will be referred to as *bad-path branching* in the experimental part. This path has length at most $c$, since after visiting $c + 1$ vertices in the breadth-first search there must be a bad path. Now, branch into the $c$ cases to destroy this bad path by edge deletion, and for each case recursively solve the resulting instance. Since at most $k$ edge deletions are needed, the search tree has depth at most $k$ and therefore size $O(c^k)$; a bad path can be found in $O(m)$ time.

We can get a speed-up by using the observation that we can either find a bad path of length at most $c - 1$ or solve the problem in polynomial time. As a motivation for this improvement, observe that in practical applications the parameter $c$ denoting the number of colors can be quite small with values

in the one-digit range. Moreover, according to Theorem 1 we cannot expect a $2^{o(k)} \cdot n^{O(1)}$-time algorithm for Colorful Components on three-colored graphs.

**Theorem 3.** *For $c \geq 3$,* Colorful Components *can be solved in $O((c-1)^k \cdot m)$ time.*

*Proof.* We first describe the algorithm and then bound its running time. In the following analysis, assume that $k$ is fixed in advance.

As long as the input graph contains a vertex $v$ with degree at least three, perform a breadth-first search starting at $v$ until either two vertices with the same color have been found or all vertices of $v$'s connected component have been visited. In the second case, $v$'s connected component is colorful and can therefore be removed from the graph. In the first case, we have visited at most $c+1$ vertices until a vertex pair with the same color has been found. The bad path between these two vertices has length at most $c-1$: since $v$ has degree at least three, at least one neighbor of $v$ is not on this path.

After a bad path has been found, branch into the at most $c-1$ edges to destroy it. Clearly, one of the edges has to be deleted. Hence, a solution can be found by recursively solving Colorful Components for each of these cases; now with $k-1$ edge deletions.

In case the graph has only vertices of degree at most two, proceed as follows. If the connected component is a cycle, then one of the following two cases can occur. If there is a bad path of length at most $c-1$ between any pair of two vertices, then branch as described above. Otherwise, the coloring on the cycle is ordered, that is, we can assume without loss of generality that each vertex with color 1 is adjacent to one vertex with color 2 and one vertex with color $c$; each vertex with color 2 is adjacent to one vertex with color 1 and one vertex with color 3, and so on. In this case, a solution for the connected component is simply to delete all edges between vertices of color $c$ and 1. In the last remaining case, the connected component is a simple path. A solution for a path can be found by visiting the edges along the path starting from one of the two degree-one vertices until there is a color that has already been visited. Then, the last visited edge can be deleted; this is repeated until the path is colorful.

The running time of the algorithm can be shown as follows. The search tree has size $O((c-1)^k)$ since at each search tree node, we branch into at most $c-1$ cases, and the depth of the tree is at most $k$. A path to branch on can be found in $O(m)$ time since the procedure only uses breadth-first search. Finally, all presented algorithms for the polynomial-time special cases can be performed in $O(m)$ time as well; the overall running time follows.

If $k$ is not given in advance, we can start the algorithm described above for increasing values of $k$ until a solution is found; the running time bound remains the same since $\sum_{1 \leq i \leq k} (c-1)^i = O(c-1)^k$. □

The observation that we can either find a bad path of length $c-1$ or solve the problem in polynomial time also implies the following factor-$(c-1)$ approximation algorithm: As long as the graph contains a bad path of length at most $c-1$, delete

all $c-1$ of these edges. If the graph has none of these bad paths, solve the problem in linear time. The approximation factor follows from the observation that at least one of the $c-1$ edges has to be deleted, and that deleting "unnecessary" edges does not create new bad paths.

**Corollary 1.** Colorful Components *can be approximated within a factor of* $c-1$ *in* $O(m^2)$ *time.*

Note that for $c \geq 11$ the factor-$(4\ln(c+1))$ approximation which is implied by the relation to Multi-Multiway Cut [1] gives better approximation ratios, for $c < 11$ our bound is better.

*Data reduction.* The following two polynomial-time executable data reduction rules for Colorful Components are relevant for the experimental work.

**Rule 1** *If a connected component is colorful, then remove it from* $G$*.*

Rule 1 can be executed in linear time. We note that Rule 1 provides a trivial *kernelization* [8][5] for Colorful Components with respect to the combined parameter $(k, c)$: obviously, after exhaustive data reduction, the instance has at most $2kc$ vertices, since an edge deletion can produce at most two colorful components, each of size at most $c$. This can be improved to a kernelization yielding only $(1+\epsilon)kc$ vertices for any $\epsilon > 0$: The idea of the corresponding data reduction is to choose any constant $\ell$ and to check (by say brute-force) for every connected component $C$ and for all $1 \leq i \leq \ell$ whether $(C, i)$ forms a yes-instance of Colorful Components and, if so, decrease the parameter $k$ accordingly by $i$. The larger we choose $\ell$, the smaller $\epsilon$ gets. We omit the details.

Rule 2 is less obvious.

**Rule 2** *Let* $B = \{b_1, \ldots, b_t\}$ *be a minimal edge cut, let* $G_B$ *be one side of the cut (that is, a connected component of* $G - B$ *such that each edge in* $B$ *has exactly one endpoint in* $G_B$*), and let* $N$ *denote the vertices that are incident with* $B$ *but not in* $G_B$*. If* $G_B$ *is colorful and* $t$*-edge connected and each color of* $N$ *also occurs in* $G_B$*, then delete* $B$ *and decrease* $k$ *by* $|B|$*.*

*Proof (of correctness).* The correctness of Rule 2 can be seen as follows. Let $S$ be a solution that does not contain some $\{u, v\} \in B$ with $u \in N$. Then, the bad path from $u$ to the vertex in $G_B$ with color $\chi(u)$ is destroyed by a set $X$ of at least $t$ edge deletions within $G_B$. Hence, the set $S' := (S \setminus X) \cup B$ is also a solution: First, $|S'| \leq |S|$. Second, the deletion of $X$ only destroys bad paths that visit at least one vertex of $V(G_B)$ and all of these bad paths are also destroyed by deleting $B$ since $G_B$ is colorful. □

Note that so far it is not clear whether Rule 2 is applicable in polynomial time if $t$ is not a constant.

[5] Informally, a kernelization transforms in polynomial time the original instance into a smaller equivalent instance whose size is upper-bounded by a function solely depending on the parameter; ideally, this function is a small polynomial.

## 4 Formulation as Weighted Multi-Multiway Cut

In the Colorful Components formulation, it is not possible to simplify a graph based on the knowledge that two vertices belong to the same connected component; we would like to be able to merge two such vertices. For this, we first need to allow not just a single color per vertex, but a set; moreover, we need to allow edge weights. Thus, we arrive at the edge-weighted version of Multi-Multiway Cut [1]: given an undirected graph $G = (V, E)$ with edge weights $w : E \to \{x \in \mathbb{Q} \mid x \geq 1\}$ and vertex sets $S_1, \ldots, S_c \subseteq V$, find a minimum-weight subset of edges $E' \subseteq E$ such that in $G' = (V, E \setminus E')$ no connected component contains two vertices from the same $S_i$.

To emphasize the connection to Colorful Components, for Weighted Multi-Multiway Cut we define the colors $\chi(u)$ of a vertex $u$ as $\{i \mid u \in S_i\}$. Note that we require weights to be at least 1.

Now, we can *merge* two vertices $u$ and $v$ with disjoint colors. This means to replace them by a new vertex $u'$ with colors $\chi(u) \cup \chi(v)$ and $N(u') := N(u) \cup N(v) \setminus \{u, v\}$, where $w(\{u', x\}) := w(\{u, x\}) + w(\{v, x\})$ (assuming $w(\{x, y\}) = 0$ for $\{x, y\} \notin E$).

*Edge branching.* Using the merge operation, we can do a simple branching on an edge: either delete the edge, or merge its endpoints; in the experimental part this will be referred to as *edge branching*. Note that merging does not necessarily decrease the parameter; but it is easy to see that if we branch on each edge of a forbidden path successively, then the last edge of the path cannot be merged since it connects vertices with an intersecting color set. This allows us to immediately delete the edge; thus, the $O(c^k \cdot m)$-time branching is still possible.

*Data reduction.* We can also adapt Rule 2 to Weighted Multi-Multiway Cut; the proof is similar to that of Rule 2.

**Rule 3** *Let $V' \subseteq V$ be a colorful subgraph. If the cut between $V'$ and $V \setminus V'$ is at least as large as the connectivity of $V'$, then merge $V'$ into a single vertex. Herein, connectivity is defined as the minimum total* weight *of edges to be deleted to obtain at least one more connected component.*

*Merge heuristic.* The idea of the heuristic is to repeatedly merge the two vertices "most likely" to be in the same component. During the process, we immediately delete edges connecting vertices with intersecting color sets. The *merge cost* of two vertices $u$ and $v$ is the weight of the edges that would need to be deleted when merging $u$ and $v$, while the *cut cost* is defined as

$$3w(\{u, v\}) + \sum_{w \in V \mid \{\{u,w\},\{v,w\}\} \subseteq E} \min\{w(\{u, w\}), w(\{v, w\})\}$$

as a rough approximation of the minimum cut between $u$ and $v$. The factor 3 has been tuned heuristically. We then always merge the endpoints of the edge that maximizes cut cost minus merge cost.

**Table 1.** Instances before and after data reduction. Herein, $n$, $m$, and $c$ are the number of vertices, edges, and colors, respectively, for the whole graph while $n'$,$m'$,$c'$ denote those values for the largest connected component of the instances.

| | original | | | | | | after Rule 2 | | | | | | after Rule 3 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $n$ | $m$ | $c$ | $n'$ | $m'$ | $c'$ | $n$ | $m$ | $c$ | $n'$ | $m'$ | $c'$ | $n$ | $m$ | $c$ | $n'$ | $m'$ | $c'$ |
| min. | 178 | 156 | 3 | 8 | 7 | 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| max. | 9311 | 26344 | 10 | 3048 | 6063 | 10 | 2769 | 5583 | 10 | 2769 | 5583 | 10 | 2602 | 5336 | 10 | 2602 | 5336 | 10 |
| avg. | 1702 | 3159 | 6.2 | 504 | 921 | 6.2 | 486 | 839 | 5.2 | 407 | 697 | 4.7 | 429 | 712 | 5.9 | 354 | 607 | 5.3 |
| med. | 1187 | 1401 | 6 | 149 | 232 | 6 | 172 | 262 | 6 | 46 | 90 | 5 | 119 | 128 | 6 | 42 | 58 | 5 |

## 5 Experiments

We performed experiments with instances from the multiple sequence alignment application. The search tree algorithm, data reduction, and merge heuristic were implemented in OCaml and compiled with the OCaml native-code compiler version 3.11.2. The test machine is a 2.66 GHz Intel Xeon X5550 with 8 MB cache and 16 GB main memory, running under openSUSE 11.3 Linux.

The source code and the test instances are available under the GNU GPL license at http://fpt.akt.tu-berlin.de/colcom/.

*Data.* We generated one Colorful Components instance for each multiple alignment instance from the BAliBASE 3.0 benchmark [14], using the diafragm 1.0 software [5]. We restricted the experiments to the 135 of the 386 instances that have at most 10 colors (that is, 10 sequences to be aligned). Instances with more colors can mostly not be solved with our exact methods.

*Implementation details.* To speed up the branching algorithms from Section 3 and Section 4, we use a transposition table in order to avoid recomputing the solutions of identical search tree nodes. We also track upper and lower bounds. These bounds are seeded with the result of the heuristic and a simple greedy packing, respectively. The branching is always on a shortest bad path. If during the branching process the instance decomposes into connected components, we solve them separately.

To efficiently find data reduction opportunities with Rule 2 and Rule 3, we try starting with each vertex and successively add more vertices with disjoint colors that minimize the cut to other edges, until we have either found a reduction opportunity or no more vertices can be added.

*Results.* We first examine the effect of data reduction (see Table 1), that is, we compare the size of the instances before and after exhaustively applying the data reduction rules. With Rule 2, we can solve 47 instances by data reduction alone, the largest among those having 3115 vertices and 4383 edges. The number of edges is reduced on average by 76.1 % (median 92.8 %). When considering the largest connected component, we get an average reduction of 64.5 % (median 65.8 %). Rule 2 reduces the largest component only by 55.4 % on average (median

54.9 %). Thus, clearly for many instances only data reduction makes the exact approaches feasible.

Next, we consider the running times of the branching algorithms. For the bad-path branching, we have 61 instances that can be solved in less than 1 second, 6 instances that can be solved in 1 second to 10 minutes, and 68 cannot be solved in 10 minutes. With edge branching, we can solve 70 instances in less than 1 second, 9 in 1 second to 10 minutes, and 56 remain unsolved. We note that in ongoing research, we are able to solve several more instances to optimality with integer linear programming (ILP) based approaches [4].

For the heuristics, we compare the solution quality for the 112 instances for which we know the optimal solution. The min-cut heuristic [5] has an error between 0 % (once) and 70.0 %, with an average error of 29.2 % (median 27.8 %). In contrast, the merge heuristic has an error between 0 % (76 times) and 12.7 %, with the average error 0.6 % (median 0 %). Without data reduction, the results are slightly worse with 66 times an optimal result and an average error of 1.0 % (median 0 %). Thus, clearly the merge heuristic is much superior for these instances, and in fact solves the majority of the instances optimally. Both heuristics take at most two seconds to solve an instance.

Finally, for the instances for which an exact solution was found, we compared the solution quality of the alignments obtained by using DIALIGN with and without the partial alignment columns indicated by an exact solution for Colorful Components, by the merge heuristic, and by the min-cut heuristic. The found alignments were compared with the BAliBASE reference alignments concerning the reconstruction of total columns (TC score) and position pairs (SP score). The exact algorithm had a TC score of 56.6 %, the merge heuristic achieved 55.1 %, the min-cut heuristic 53.6 %, and the alignment without anchors achieved 54 %; for SP-score the results are similar. This indicates that minimizing edge deletions for obtaining Colorful Components is indeed helpful for obtaining better alignments. Note that, concerning TC score, DIALIGN with the min-cut heuristic is about 10 percentage points worse than current state-of-the-art multiple alignment methods [5]. Hence, an improvement of roughly 3 percentage points is a sizable step towards closing the gap between DIALIGN and these methods.

## 6 Outlook

It is open to obtain a smaller problem kernel, a problem kernel with size independent of $c$, and branching algorithms with branching number less than $c - 1$. So far, it is also undetermined whether Rule 2 and Rule 3 can be exhaustively applied in polynomial time. From the modeling perspective, it is interesting to consider a relaxation of the colorfulness constraint: In preliminary experiments with network alignment data, we found that allowing only one protein of each species to be matched was, while a natural model, too strict. Generalizing Color Components to allow a constant number of occurrences of each color for the connected components could result in improved network alignments.

## References


[1] A. Avidor and M. Langberg. The multi-multiway cut problem. *Theoretical Computer Science*, 377(1–3):35–42, 2007. 2, 10, 11

[2] A. Björklund, T. Husfeldt, P. Kaski, and M. Koivisto. Fourier meets Möbius: fast subset convolution. In *Proc. 39th STOC*, pages 67–74. ACM, 2007. 8

[3] N. Bousquet, J. Daligault, and S. Thomassé. Multicut is FPT. In *Proc. 43rd STOC*, pages 459–468. ACM, 2011. 2

[4] S. Bruckner, F. Hüffner, C. Komusiewicz, and R. Niedermeier. Entity disambiguation by partitioning under heterogeneity constraints. Manuscript, submitted, Feb. 2012. URL http://fpt.akt.tu-berlin.de/publications/disambiguation.pdf. 13

[5] E. Corel, F. Pitschi, and B. Morgenstern. A min-cut algorithm for the consistency problem in multiple sequence alignment. *Bioinformatics*, 26(8): 1015–1021, 2010. 2, 3, 12, 13

[6] Y.-P. Deniélou, F. Boyer, A. Viari, and M.-F. Sagot. Multiple alignment of biological networks: A flexible approach. In *Proc. 20th CPM*, volume 5577 of *LNCS*, pages 263–273. Springer, 2009. 2

[7] N. Garg, V. V. Vazirani, and M. Yannakakis. Primal–dual approximation algorithms for integral flow and multicut in trees. *Algorithmica*, 18(1):3–20, 1997. 3

[8] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007.

[9] C. Komusiewicz. *Parameterized Algorithmics for Network Analysis: Clustering & Querying*. PhD thesis, Technische Universität Berlin, Berlin, Germany, 2011. 4

[10] J. Li, K. Yi, and Q. Zhang. Clustering with diversity. In *Proc. 37th ICALP*, volume 6198 of *LNCS*, pages 188–200. Springer, 2010. 2

[11] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–71, 2011. 3

[12] D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proc. 43rd STOC*, pages 469–478. ACM, 2011. 2

[13] D. Park, R. Singh, M. Baym, C.-S. Liao, and B. Berger. IsoBase: a database of functionally related proteins across PPI networks. *Nucleic Acids Research*, 39(Database-Issue):295–300, 2011. 2

[14] J. D. Thompson, P. Koehl, R. Ripp, and O. Poch. BAliBASE 3.0: latest developments of the multiple sequence alignment benchmark. *Proteins: Structure, Function, and Bioinformatics*, 61(1):127–136, 2005. 3, 12

[15] M. Weller, C. Komusiewicz, R. Niedermeier, and J. Uhlmann. On making directed graphs transitive. *Journal of Computer and System Sciences*, 78 (2):559–574, 2012. 4