# A golden ratio parameterized algorithm for Cluster Editing

Sebastian Böcker

Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany, sebastian.boecker@uni-jena.de

**Abstract.** The Cluster Editing problem asks to transform a graph by at most $k$ edge modifications into a disjoint union of cliques. The problem is NP-complete, but several parameterized algorithms are known. We present a novel search tree algorithm for the problem, which improves running time from $O^*(1.76^k)$ to $O^*(1.62^k)$. In detail, we can show that we can always branch with branching vector $(2, 1)$ or better, resulting in the golden ratio as the base of the search tree size. Our algorithm uses a well-known transformation to the integer-weighted counterpart of the problem. To achieve our result, we combine three techniques: First, we show that zero-edges in the graph enforce structural features that allow us to branch more efficiently. Second, by repeatedly branching we can isolate vertices, releasing costs. Finally, we use a known characterization of graphs with few conflicts.

This is a preprint of: Sebastian Böcker. A golden ratio parameterized algorithm for Cluster Editing. In *Proc. of International Workshop on Combinatorial Algorithms (IWOCA 2011)*, volume 7056 of Lect. Notes Comput. Sci., pages 85-95. Springer, Berlin, 2011.

## 1 Introduction

Given an undirected graph $G$, the Cluster Editing problem asks for a minimal set of edge modifications such that the resulting graph is a vertex-disjoint union of cliques. In the corresponding Integer-Weighted Cluster Editing problem, we are given modification costs for each edge or non-edge, and we search for a set of edge modifications with minimum total weight. Here, one assumes that all edges have non-zero modification cost.

In application, the above task corresponds to clustering objects, that is, partitioning a set of objects into homogeneous and well-separated subsets. Similar objects are connected by an edge, and a cluster is a clique of the input graph. The input graph is corrupted and we have to clean (edit) the graph to reconstruct the clustering under the parsimony criterion. Clustering data still represents a key step of numerous life science problems. The weighted variant of the Cluster Editing problem has been frequently proposed for clustering biological entities such as proteins [18].

The Cluster Editing problem is NP-hard [13]. The parameterized complexity of Cluster Editing, using the number of edge modifications as parameter $k$, is well-studied, see also the FPT races column in [17]. A first algorithm with running time $O^*(2.27^k)$ [10] was improved to $O^*(1.92^k)$ by an extensive case analysis [9]. By transforming the problem to the integer-weighted variant, running time was advanced to $O^*(1.82^k)$ [1]. Using a characterization of graphs that do not contain many conflicts, results in the currently fastest algorithm with running time $O^*(1.76^k)$ [3]. There exist linear problem kernels for the unweighted [5] and the integer-weighted variant [4]. Recently, Cluster Editing with "don't care edges" (that is, edges whose modification cost is zero) has been shown to be fixed-parameter tractable [14]. To find exact solutions in practice, a combination of data reduction and Integer Linear Programming proved to be very efficient [2].

*Our contributions.* We present a new search tree algorithm for Cluster Editing with running time $O(1.62^k + k^2 + m + n)$ for $m$ edges and $n$ vertices, being the fastest known for the problem. The algorithm itself is rather simple, and is based on the merge branching introduced in [1]. We stress that our result only holds for the unweighted Cluster Editing problem, as general integer-weighted instances will not satisfy the "parity property" introduced below.

## 2 Preliminaries

A problem with input size $n$ and parameter $k$ is *fixed-parameter tractable* (FPT) if it can be solved in $O(f(k) \cdot p(n))$ time where $f$ is any computable function and $p$ is a polynomial. We naturally focus on the $f(k)$ factor, and sometimes adopt the $O^*(f(k))$ notation that suppresses polynomial factors. For a general introduction we refer to [7,15]; in particular, we assume familiarity with bounded search trees, branching vectors, and branching numbers. In the following, let $n$ be the number of vertices, and $k$ the number of edge modifications.

For brevity, we write $uv$ as shorthand for an unordered pair $\{u, v\} \in \binom{V}{2}$. Let $s : \binom{V}{2} \to \mathbb{Z}$ be a *weight function* that encodes the input graph: For $s(uv) > 0$ a pair $uv$ is an edge of the graph and has deletion cost $s(uv)$, while for $s(uv) < 0$, the pair $uv$ is not an edge (a *non-edge*) of the graph and has insertion cost $-s(uv)$. Let $N(u)$ be the set of all vertices $v \in V$ such that $s(uv) > 0$. If $s(uv) = 0$, we call $uv$ a *zero-edge*. We require that there are no zero-edges in the input graph. Nonetheless, zero-edges can appear in the course of computation and require additional attention when analyzing the algorithm.

When analyzing connected components we only consider edges of the graph. We say that $C \subseteq V$ is a *clique* in an integer-weighted graph if all pairs $uv \in \binom{C}{2}$ are edges. If all vertex pairs of a connected component are either edges or zero-edges, we call it a *weak clique*. Vertices $uvw$ form a *conflict triple* in an integer-weighted graph if $uv$ and $vw$ are edges but $uw$ is either a non-edge or a zero-edge. We distinguish two types of conflict triples $uvw$: if $uw$ has weight zero then the conflict triple is called *weak*, whereas if $uw$ is a non-edge then the conflict triple is called *strong*. If the integer-weighted graph contains no conflict triples then it is *transitive*, i.e. a disjoint union of weak cliques. But the converse is obviously not true, as the example of a single weak conflict triple shows: This graph is a weak clique but contains a (weak) conflict triple. To solve Weighted Cluster Editing we first identify all connected components of the input graph and calculate the best solutions for each component separately, because an optimal solution never connects disconnected components. Furthermore, if the graph is decomposed during the course of the algorithm, then we recurse and treat each connected component individually.

An unweighted Cluster Editing instance can be encoded by assigning weights $s(uv) \in \{+1, -1\}$. In the resulting graph, all conflict triples are strong. During data reduction and branching, we may set pairs $uv$ to "forbidden" or "permanent". Permanent edges can be merged immediately: *Merging* $uv$ means replacing the vertices $u$ and $v$ with a single vertex $u'$, and, for all vertices $w \in V \setminus \{u, v\}$, replacing pairs $uw$, $vw$ with a single pair $u'w$. In this context, we say that we *join* vertex pairs $uw$ and $vw$. The weight of the joined pair is $s(u'w) = s(uw) + s(vw)$. In case one of the pairs is an edge while the other is a non-edge, then we can decrease parameter $k$ by $\min\{|s(uw)|, |s(vw)|\}$. Note that we may join any combination of two edges, non-edges, or zero-edges when merging two vertices. We stress that joined pairs can be zero-edges.

We encode a *forbidden pair* $uv$ by setting $s(uv) = -\infty$. By definition, every forbidden pair $uv$ is a non-edge, since $s(uv) < 0$. A forbidden pair $uw$ can be part of a conflict

triple $uvw$, which then is a strong conflict triple. Assume that we join pairs $uv$ and $uw$ where $uw$ is forbidden and, hence, a non-edge. From the above definition, the resulting pair $u'w$ is forbidden, too, as $s(u'w) = s(uw) + s(vw) = -\infty + s(vw) = -\infty$ holds for all $s(vw) \in \mathbb{R} \cup \{-\infty\}$. Finally, if $uw$ is forbidden and $vw$ is an edge then $k$ is decreased by $\min\{\infty, |s(vw)|\} = s(vw)$.

The following branching was proposed in [1]: We *branch on an edge* $uv$ by recursively calling the algorithm two times, either removing $uv$ and setting it to forbidden, or merging $uv$. If $uv$ is part of at least one strong conflict triple, then merging $uv$ will generate cost: As there is both an edge $uw$ and a non-edge $vw$, we can reduce $k$ by $\min\{s(uw), -s(vw)\}$. In case $s(uw) = -s(vw)$, joining $uw$ and $vw$ into $u'w$ results in $u'w$ being a zero-edge. At a later stage of the algorithm, this would prevent us from decreasing our parameter when joining the zero-edge $u'w$. To circumvent this problem, the following bookkeeping trick was introduced in [1]: We assume that joining $uw$ and $vw$ with $s(uw) = -s(vw)$ only reduces the parameter by $\min\{s(uw), -s(vw)\} - \frac{1}{2} = |s(uw)| - \frac{1}{2} \geq \frac{1}{2}$. If at a later stage we join this zero-edge with another pair, we decrease our parameter by the remaining $\frac{1}{2}$. So, both generating and destroying a zero-edge generates cost of at least $\frac{1}{2}$. Note that joining with a forbidden pair cannot create a zero-edge.

Assume that $s(vw) = -s(uw)$ with $|s(vw)| = |s(uw)| \geq 2$. Then, merging an edge $uv$ in a conflict triple $uvw$ will also generate a zero-edge, and generates cost of at least $\frac{3}{2}$. In our analysis, we sometimes concentrate on the case that $s(vw) = -s(uw) = \pm 1$, where merging $uv$ has cost $\frac{1}{2}$. We do so only if it is absolutely obvious that $|s(vw)| = |s(uw)| \geq 2$ will result in the desired branching vector.

Our fixed-parameter algorithms require a cost limit $k$: In case a solution with cost $\leq k$ exists, the algorithm finds this solution; otherwise, "no solution" is returned. To find an optimal solution we call the algorithm repeatedly, increasing $k$.

## 3 Vertex parities

We need a simple observation about the input graphs to reach an improved running time: An integer-weighted graph $G$ with weight function $s : \binom{V}{2} \to \mathbb{Z}$ has the *parity property* if there is a *parity mapping* $p : V \to \{\text{EVEN}, \text{ODD}\}$ such that, for each pair $uv$, $s(uv)$ is odd if and only if both $p(u) = \text{ODD}$ and $p(v) = \text{ODD}$ holds. We ignore forbidden pairs in this definition, since $s(uv) = -\infty$ has no parity. Note that $p$ is not necessarily unique, as demonstrated by a graph with two vertices and even edge weight. We infer a few simple observations from this definition: If $s(uv)$ is even, then either $u$ or $v$ or both must have EVEN parity. If $u$ is EVEN then $s(uv)$ is even or $uv$ is forbidden, for all $v \neq u$.

Clearly, an unweighted instance of Cluster Editing has the parity property, as we can set $p(u) = \text{ODD}$ for all vertices $u \in V$. The interesting observation is that a graph does not loose the parity property if we merge two vertices. Quite possibly, this results has been stated before in a different graph-theoretical context. We defer the simple, constructive proof to the full paper.

**Lemma 1.** *Assume that an integer-weighted graph $G$ has the parity property. If we merge two vertices in $G$, then the resulting graph also has the parity property.*

If the input graph has the parity property then, after any sequence of merging operations, the resulting graph still has the parity property. This is particularly so for the edge branching from [1], as both operations (setting an edge to forbidden, or merging two vertices) preserve

the parity property. For our branching, it is important to notice that a zero-edge has even parity, so the parity of at least one of its incident vertices must be EVEN.

## 4 Isolation and vertices of even parity

Let $\varphi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots$ be the *golden ratio*, satisfying $\varphi = 1 + \frac{1}{\varphi}$. One can easily see that a search tree algorithm with branching vector $(2,1)$ results in a search tree of size $O(\varphi^k)$: This branching number is the positive root of $x^{-2} + x^{-1} - 1$, so $1 + x - x^2 = 0$, and dividing by $x$ results in the definition of the golden ratio.

Our branching strategy is based on a series of lemmata, ensuring that either there is an edge to branch on, or that the remaining graph is "easy". Clearly, branching on an edge that is part of four or more conflict triples results in the desired branching vector. To this end, we concentrate on the critical case of three conflict triples. First, we consider the case of three strong conflict triples:

**Lemma 2.** *Let $G$ be an integer-weighted graph that has the parity property. Assume that an edge $uv$ is part of exactly three conflict triples, all of which are strong. Then, we can branch with branching number $\varphi = 1.61803\ldots$.*

We use this lemma to show that we can find an edge to branch on, if we can find an edge that is part of at least three conflict triples.

**Lemma 3.** *Let $G$ be an integer-weighted graph that has the parity property. Assume that an edge $uv$ is part of three or more conflict triples. Then, we can either find an edge with branching number $\varphi$, or we can reduce $k$ without branching.*

The remainder of this section is devoted to proving these two central lemmata.

*Proof (Lemma 2).* We will show that we can find an edge to branch on, with branching vector $(1,2)$ or better. In our reasoning, we will show that either, we have already reached the desired branching vector; or, we can infer certain structural properties about the instance.

Let $a, b, c$ be the three vertices that are part of the three conflict triples with $u, v$. If $s(uv) \geq 2$ then branching on $uv$ results in deletion cost $s(uv) \geq 2$ and merging cost $3 \cdot \frac{1}{2}$, so we reach branching vector $(2, \frac{3}{2})$ and we are done. If $uvx$ with $x \in \{a, b, c\}$ is a conflict triple such that $s(vx) \geq 2$ or $s(ux) \leq -2$, then merging $uv$ into $u'$ will not create a new zero-edge incident to $u'$. So, branching on $uv$ has branching vector $(1, 2 \cdot \frac{1}{2} + 1) = (1, 2)$, and we are done. The same argumentation holds for a conflict triple $vux$. In the following, we may assume that $a, b, c$ are ODD, and that $s(uv) = 1$ and $|s(wx)| = 1$ holds for all $w \in \{u, v\}$ and $x \in \{a, b, c\}$; for all other cases, we have just shown that the desired branching vector can be reached.

Assume that $u, v$ do not have a common neighbor, $N(u) \cup N(v) = \{u, v, a, b, c\}$. Then, merging $u, v$ into $u'$ generates three zero-edges $u'a, u'b, u'c$, and $u'$ is isolated, $N(u') = \emptyset$. But then, we do not have to use bookkeeping for these edges, as $\{u'\}$ will also be a separated cluster of size one in the solution. So, branching on $uv$ results in branching vector $(1, 3)$.

We will now use the same trick that the merged vertex $u'$ can be isolated, but this is slightly more involved in case $u, v$ have at least one common neighbor. Let $D := N(u) \cap N(v)$, then $N(u) \cup N(v) = D \cup \{u, v, a, b, c\}$ and $|D| \geq 1$. Our first step is to branch on $uv$: We delete $uv$ with cost 1, and set it to forbidden.

Next, we merge $u, v$ into a new vertex $u'$. This generates three zero edges $u'a, u'b, u'c$ with costs $\frac{3}{2}$. Here, $s(u'd) \geq 2$ holds for all $d \in D = \{d_1, \ldots, d_l\}$. We will now branch on all edges $u'd_j$ where the case that $u'd_j$ is deleted, is further analyzed. In detail, we either merge $u'd_i$ with costs $\frac{3}{2}$; or, we delete $u'd_i$ with cost 2 and branch on $u'd_{i+1}$, if $i < l$. Note that we either delete all $d_1, \ldots, d_l$, or we finally merge some $u'd_i$ with cost $\frac{3}{2}$. In the latter case, the total costs of this branch are $2(i-1) + \frac{3}{2}$. But in the very last case where all $d_1, \ldots, d_l$ are deleted, we separate $u'$. Hence, by the reasoning introduced above, we can "cash" cost $\frac{3}{2}$ we have put aside when generating the three zero-edges $u'a, u'b, u'c$. So, the costs of this final branch are $2l + \frac{3}{2}$. Recall that in all cases, we have additional cost $\frac{3}{2}$ for generating the three zero-edges. In total, we reach the partial branching vector $(0+3, 2+3, \ldots, 2l+3) = (3, 5, 7, \ldots, 2l+3)$.

We combine these two partial branching vectors into one branching vector $(1, 3, 5, 7, 9, \ldots, 2l+3)$. We claim that any such branching vector corresponds to a branching number $x < \varphi$, and that the numbers converge towards $\varphi$. To this end, first note that $1/\varphi$ is the unique positive root of the polynomial $x^2 + x - 1$, that is the characteristic polynomial of branching vector $(2, 1)$. We analyze the infinite series $f(x) := x^0 + x^2 + x^4 + \ldots$ that converges for all $|x| < 1$. Now, $x^2 \cdot f(x) = f(x) - 1$ and

$$(x^2 + x - 1) \cdot f(x) = f(x) - 1 + xf(x) - f(x) = xf(x) - 1.$$

So, for the series $g(x) := xf(x) - 1$ we have

$$g(x) = xf(x) - 1 = (x^2 + x - 1) \cdot f(x)$$

and, hence, $g(1/\varphi) = 0$. For the partial sums $S_l(x) := x^{2l+3} + x^{2l+1} + \cdots + x^3 + x^1 - 1$ we infer $S_l(x) < S_{l+1}(x)$ and $S_l(x) < g(x)$ for $x \in (0, \infty)$. Also, $S_l$ is strictly increasing in $[0, \infty)$.

Note that any polynomial of the form $p(x) := a_n x^n + \cdots + a_1 x^1 - 1$ with $a_i \geq 0$ for all $i$, has exactly one positive root for $p \not\equiv -1$. This follows as $p$ is continuous, $p'(x) > 0$ for all $x > 0$, so $p$ is strictly increasing in $(0, \infty)$, $p(0) = -1$, and $\lim_{x \to \infty} p(x) = \infty$. Let $x_l$ be the unique positive root of $S_l(x)$. With $S_l(x_{l+1}) < S_{l+1}(x_{l+1}) = 0$ we finally infer

$$x_1 > x_2 > x_3 > \cdots > 1/\varphi.$$

By definition, $1/x_l$ is the branching number for branching vector $(1, 3, 5, 7, 9, \ldots, 2l+3)$, and we reach

$$1/x_1 < 1/x_2 < 1/x_3 < \cdots < \varphi.$$

Since the series $S_l$ converges uniformly to $g$ in the interval $[0, \alpha]$ for every $\alpha < 1$, we infer that $\lim_l 1/x_l = \varphi$ must hold, which concludes the proof of the lemma. □

*Proof (Lemma 3).* Again, we will show that either, we have already reached the desired branching vector $(1, 2)$ or better; or, we can infer certain structural properties about the instance.

If $uv$ is part of four conflict triples then we reach branching vector $(1, 4 \cdot \frac{1}{2}) = (1, 2)$. If $uv$ is part of three strong conflict triples then Lemma 2 guarantees branching number $\varphi$. So, assume that $uv$ is part of exactly three conflict triples, and that $uvw$ is a weak conflict triple, so $uw$ is a zero-edge. As $uv$ is part of three conflict triples, we can choose $a, b$ such that $N(u) \triangle N(v) = \{w, a, b\}$. Clearly, for $s(uv) = 2$ we have branching vector $(2, \frac{3}{2})$, so we may assume $s(uv) = 1$. This implies that both $u$ and $v$ must have ODD parity. Since $uw$

is a zero-edge, we infer that $w$ has EVEN parity and, hence, that $s(vw) \geq 2$ holds. For our worst-case considerations, we may assume $s(vw) = 2$.

If $vw$ is part of any additional conflict triples besides $wvu$, then we reach branching vector $(2, 1)$ for branching on $vw$: Deleting $vw$ has cost 2, and merging $vw$ then has cost $2 \cdot \frac{1}{2}$. The same holds true if $v$ or $w$ are incident to additional zero-edges besides $uw$. So, assume there are no zero-edges incident to $v$ or $w$ besides $uw$, and $vx$ is an edge if and only if $wx$ is an edge for all $x \neq u, v, w$. Let $X \subseteq V \setminus \{u, v, w\}$ be the set of vertices incident to $v$ and, consequently, also to $w$. Let $X' := X \setminus \{a, b\}$, and note that this set can be empty. All $x \in X'$ are also incident with $u$; otherwise, there is a fourth conflict triple for the edge $uv$. We infer $N(\{u, v, w\}) \subseteq \{u, v, w, a, b\} \cup X'$.

Choose an arbitrary $x \in X'$. If $wx$ is part of an additional conflict triple besides $wxu$, or if $x$ is incident to a zero-edge, then we again reach branching vector $(2, 1)$ for branching on $wx$: Deleting $wx$ has cost 2 since $w$ is EVEN, and merging $wx$ has cost $2 \cdot \frac{1}{2}$. Hence, we infer three things: Firstly, each $y$ adjacent to some $x \in X'$ is also adjacent to $w$ and, hence, $y \in X$. So, $N(X') \subseteq \{u, v, w, a, b\} \cup X'$. Secondly, each pair $x, y \in X'$ must be connected by an edge. We distinguish three cases:

1. Assume $a, b \in X$, so $va$ and $vb$ are edges. In this case, $u, v, w, a, b, X'$ form a connected component. If $ab$ is a zero-edge or non-edge, then branching on $wa$ results in branching vector $(2, 2 \cdot \frac{1}{2})$: although $w, a, u$ do not form a conflict triple, merging $wa$ still destroys the zero-edge $uw$. So, we may assume that $ab$ is an edge. By the same reasoning, $ax$ and $bx$ must be edges, for all $x \in X'$. Next, $s(ux) = 1$ must hold for all $x \in X'$; otherwise, we can branch on $ux$ with branching vector $(2, 3 \cdot \frac{1}{2})$. The cost of separating $u$ from all other vertices is $|X'|+1$, and the resulting graph consists of two cliques $\{u\}$ and $\{v, w, a, b\} \cup X'$. The cost of any other cut in this connected component is at least $|X'|+3$ (for separating $a$ or $b$), since $w$ is adjacent to all vertices but $u$ with edges of weight at least 2. The cost of transforming the connected component into a clique is $|s(ua)| + |s(ub)|$. So, we can test in constant time if one of the two possible transformations has cost at most $k$.
2. Assume $a \in X$ and $b \notin X$, so $va$ and $ub$ are edges. Then, $N(\{u, v, w, a\} \cup X') \subseteq \{u, v, w, a, b\} \cup X'$. For $s(ua) < -1$ we reach branching vector $(1, 2 \cdot \frac{1}{2}+1)$ for branching on $uv$, as merging $u, v$ will not generate a zero-edge incident to $a$ and, hence, no bookkeeping is required. (Obviously, this includes the case that $ua$ is forbidden.) So, $s(ua) \in \{0, 1\}$ must hold. Since $bv$ is a non-edge, $bw$ and $bx$ for all $x \in X'$ are also non-edges. If $s(ub) \geq 2$ then branching on $ub$ results in branching vector $(2, 1)$, as $vub$ is a conflict triple. Now, one can easily see that no optimal solution can bisect $v, w, a, X'$: For $X' = \emptyset$ a bisection of vertices $v, w, a$ costs at least 3, and for $X' \neq \emptyset$ costs are at least 4. Given a solution that bisects $v, w, a, X'$, we modify the solution by putting $u, v, w, a, X'$ in a separate clique, with cost at most 1 for inserting $ua$, and cost 1 for removing $ub$. Clearly, this new solution has smaller total cost than the initial solution, so the initial solution cannot be optimal. Hence, we can merge $v, w, a, X'$ without branching, generating cost of at least $\frac{1}{2}$ for destroying the zero-edge $uw$.
3. Assume $a, b \notin X$, so $ua$ and $ub$ are edges. Then, $va$ and $vb$ are non-edges, since no zero-edges can be incident to $v$. Similar to above, this implies that $wa$ and $wb$, as well as $ax$ and $bx$ for all $x \in X'$, are non-edges, too: Otherwise, we can branch on $vw$ or $wx$. If $s(ua) \geq 2$ then branching on $uv$ results in branching vector $(1, 2)$. So, we infer $s(ua) = 1$ and, by symmetry, $s(ub) = 1$. Now, merging $uv$ into some vertex $u'$ results in a separated clique with vertex set $u', w, X$ that is not connected to the rest of the graph, and can be removed immediately. Hence, branching on $uv$ leads to branching

vector $(1, 2)$ as we do not have put away $2 \cdot \frac{1}{2}$ for potentially destroying zero-edges $u'a$ and $u'b$ later.

We have shown that we can find an edge that allows for the desired branching vectors, simplify the instance and reduce $k$ without branching, or solve the remaining instance in constant time. □

## 5 Solving remainder instances

Assume that there is no edge in the graph that is part of three or more (weak or strong) conflict triples. We transform our weighted graph into an unweighted counterpart $G_u$, where zero-edges are counted as non-existing. This graph $G_u$ is called the *type graph* of the weighted graph. Then, there is no edge $uv$ in the unweighted graph $G_u$ that is part of three conflict triples. Damaschke [6] characterizes such graphs: Let $P_n, C_n, K_n$ be the chordless path, cycle, and clique on $n$ vertices, respectively. Let $G + H$ denote the disjoint union of two graphs, and let $p \cdot G$ denote $p$ disjoint copies of $G$. Let $G * H$ be the graph $G + H$ where, in addition, every vertex from $G$ is adjacent to every vertex from $H$. Finally, the graph $G^c$ has the same vertex set as $G$, and $\{u, v\}$ is an edge of $G^c$ if and only if it is no edge of $G$. Now, Theorem 2 from [6] states:

**Lemma 4.** *Let $G$ be a connected, unweighted graph such that no edge is part of three or more conflict triples. Then, $G$ has at most six vertices, is a clique, a path, a cycle, or a graph of type $K_q * H$ for $q \geq 0$ and $H \in \{K_1+K_1, C_5, P_4, K_1+K_1+K_1, K_2+K_2, K_2+K_1, (p \cdot K_2)^c\}$, $p \geq 2$.*

In fact, the characterization in [6] is slightly more complicated: To this end, note that $K_q * P_3 = K_{q+1} * (K_1 + K_1)$. Any non-edge in the type graph can be a non-edge or zero-edge in the weighted graph, and edges and non-edges can be arbitrarily weighted. We now show that we can efficiently solve all remaining, "simple" instances. This is similar to our argumentation in [3] but as we want to reach branching vector $(2, 1)$, our argumentation is slightly more involved. We defer the proof of Lemma 5 to the full version of this paper.

**Lemma 5.** *Let $G$ be a connected graph that has the parity property. Assume that there is no edge that is part of three conflict triples. Then, we can find an edge with branching number $\varphi$; reduce $k$ without branching; or, we can solve the instance in polynomial time.*

## 6 A golden ratio base for search tree size

Assume that $G$ has the parity property. We want to show that we can either find an edge to branch on with branching number $\varphi$; decrease $k$ without branching; or, solve the remaining instance in polynomial time. If there is an edge $uv$ that is part of at least three (weak or strong) conflict triples, we branch on this edge. By Lemma 3, doing so results in branching number $\varphi$, or we reduce $k$ without branching, as desired. We can find an edge to branch on, in time $O(n^3)$. Similarly, we can perform all other tasks required for one step of the branching, in this time. If there is no edge $uv$ that is part of at least three conflict triples, then Lemma 5 guarantees that we can branch with branching number $\varphi$; reduce $k$ without branching; or, solve the instance in polynomial time. To compute minimum $s$-$t$-cuts as part of Lemma 5, we use the Goldberg-Tarjan algorithm [8] to compute a maximum $s$-$t$-flow in time $O(n^3)$, independent of edge weights. We reach:

**Lemma 6.** *Given an integer-weighted instance of the* Cluster Editing *problem with no zero-edges that satisfies the parity property, this instance can be solved in* $O(\varphi^k \cdot n^3)$ *time.*

We can combine this with the weighted kernel from [4] of size $O(k)$ with running time $O(n^2)$, resulting in running time $O(\varphi^k \cdot k^3 + n^2)$. To get rid of the multiplicative polynomial factor, we use interleaving [16]: Here, a small trick is required to make this kernel work with instances that may contain zero-edges; we defer the details to the full paper.

**Theorem 1.** *Given an integer-weighted instance of the* Cluster Editing *problem with no zero-edges that satisfies the parity property, this instance can be solved in* $O(\varphi^k + n^2)$ *time.*

Given an unweighted Cluster Editing instance, we first identify all critical cliques in time $O(m+n)$ for a graph with $n$ vertices and $m$ edges [12], and merge the vertices of each critical clique [1,11]. The resulting integer-weighted instance has $O(k)$ vertices and no zero-edges, and satisfies the parity property. Using Theorem 1 we reach:

**Theorem 2.** Cluster Editing *can be solved in* $O(1.62^k + k^2 + m + n)$ *time.*

## 7 Conclusion

We have presented a parameterized algorithm for the Cluster Editing problem, that finally reaches the golden ratio as the base for the exponential growth of the running time. It is noticeable that search tree approaches plus additional structural observations still have a lot of potential to yield better FPT algorithms for well-known problems, even without extensive case handling. Note that the underlying edge branching is also very swift in practice, and can usually process instances with thousands of edge modifications in a matter of minutes [2].

The base $\varphi = \frac{1+\sqrt{5}}{2} = 1.61803\ldots$, resulting from branching vector $(2,1)$, appears repeatedly in the analysis of advanced algorithms for the problem [1,3]. Hence, it is an interesting question for the future if we can get beyond the $O^*(\varphi^k)$ barrier. One possible extension lies in the split-off technique introduced in [3] for Cluster Deletion, even though it cannot be directly applied, as branching on a $C_4$ results in branching vector $(1,1)$ for Cluster Editing. Improving upon the running time should not be problematic for the rather technical Lemma 5, though. Here, the open question is, which of these special cases are tractable (such as $H = K_1 + K_1$) and which are intractable (such as $H = K_1 + K_1 + K_1$), and what FPT algorithms can be derived for the hard ones.

## References

1. S. Böcker, S. Briesemeister, Q. B. A. Bui, and A. Truss. Going weighted: Parameterized algorithms for cluster editing. *Theor. Comput. Sci.*, 410(52):5467–5480, 2009.
2. S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011.
3. S. Böcker and P. Damaschke. Even faster parameterized cluster deletion and cluster editing. *Inform. Process. Lett.*, 111(14):717–721, 2011.
4. Y. Cao and J. Chen. Cluster editing: Kernelization based on edge cuts. In *Proc. of International Symposium on Parameterized and Exact Computation (IPEC 2010)*, volume 6478 of *Lect. Notes Comput. Sc.*, pages 60–71. Springer, 2010.

5. J. Chen and J. Meng. A $2k$ kernel for the cluster editing problem. In *Proc. of Computing and Combinatorics Conference (COCOON 2010)*, volume 6196 of *Lect. Notes Comput. Sc.*, pages 459–468. Springer, 2010.
6. P. Damaschke. Bounded-degree techniques accelerate some parameterized graph algorithms. In *Proc. of International Workshop on Parameterized and Exact Computation (IWPEC 2009)*, volume 5917 of *Lect. Notes Comput. Sc.*, pages 98–109. Springer, 2009.
7. R. G. Downey and M. R. Fellows. *Parameterized Complexity*. Springer, 1999.
8. A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.
9. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems. *Algorithmica*, 39(4):321–347, 2004.
10. J. Gramm, J. Guo, F. Hüffner, and R. Niedermeier. Graph-modeled data clustering: Fixed-parameter algorithms for clique generation. *Theor. Comput. Syst.*, 38(4):373–392, 2005.
11. J. Guo. A more effective linear kernelization for cluster editing. *Theor. Comput. Sci.*, 410(8-10):718–726, 2009.
12. W.-L. Hsu and T.-H. Ma. Substitution decomposition on chordal graphs and applications. In *Proc. of International Symposium on Algorithms (ISA 1991)*, volume 557 of *Lect. Notes Comput. Sc.*, pages 52–60. Springer, 1991.
13. M. Křivánek and J. Morávek. NP-hard problems in hierarchical-tree clustering. *Acta Inform.*, 23(3):311–323, 1986.
14. D. Marx and I. Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *Proc. of ACM Symposium on Theory of Computing (STOC 2011)*, 2011. To be presented, see also `http://arxiv.org/abs/1010.3633`.
15. R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
16. R. Niedermeier and P. Rossmanith. A general method to speed up fixed-parameter-tractable algorithms. *Inform. Process. Lett.*, 73:125–129, 2000.
17. F. Rosamond, editor. *FPT News: The Parameterized Complexity Newsletter*. Since 2005. See `http://fpt.wikidot.com/`.
18. T. Wittkop, D. Emig, S. Lange, S. Rahmann, M. Albrecht, J. H. Morris, S. Böcker, J. Stoye, and J. Baumbach. Partitioning biological data with transitivity clustering. *Nat. Methods*, 7(6):419–420, 2010.