Going Weighted: Parameterized Algorithms for Cluster Editing

Sebastian Böcker[1], Sebastian Briesemeister[2], Quang B. A. Bui[1], and Anke Truss[1]

[1] Lehrstuhl für Bioinformatik, Friedrich-Schiller-Universität Jena, Ernst-Abbe-Platz 2, 07743 Jena, Germany
{boecker,bui,truss}@minet.uni-jena.de
[2] Div. for Simulation of Biological Systems, ZBIT/WSI, Eberhard Karls Universität Tübingen, Germany
briese@informatik.uni-tuebingen.de

Abstract. The goal of the Cluster Editing problem is to make the fewest changes to the edge set of an input graph such that the resulting graph is a disjoint union of cliques. This problem is NP-complete but recently, several parameterized algorithms have been proposed. In this paper we present a surprisingly simple branching strategy for Cluster Editing. We generalize the problem assuming that edge insertion and deletion costs are positive integers. We show that the resulting search tree has size $O(1.82^k)$ for edit cost k, resulting in the currently fastest parameterized algorithm for this problem. We have implemented and evaluated our approach, and find that it outperforms other parametrized algorithms for the problem.

1 Introduction

The Weighted Cluster Editing problem is defined as follows: Let $G_w = (V, E)$ be an undirected graph. For every vertex pair $\{u, v\} \in \binom{V}{2} = \{\{u, v\} : u, v \in V, u \neq v\}$ we know the cost of deleting $\{u, v\}$ from G_w in case $\{u, v\} \in E$, or inserting $\{u, v\}$ into G_w in case $\{u, v\} \notin E$. Our task is to transform G_w into a transitive graph (a disjoint union of cliques) by applying edge modifications with minimum total cost. For our theoretical analysis we assume that all pairs have *non-zero integer weight*. In the unweighted Cluster Editing problem, insertion or deletion cost are one for each vertex pair.

In application, the above task corresponds to clustering objects, that is, partitioning a set of objects into homogeneous and well-separated subsets. Clustering data still represents a key step of numerous biological and medical problems, such as class discovery for tissue identification using gene expression data. Here, a clustering corresponds to a vertex disjoint union of cliques. The input graph is corrupted and we have to clean (edit) the graph to reconstruct the clustering [13] under the *parsimony criterion*.

Previous work. NP-hardness of the unweighted Cluster Editing problem [13] was proven by Křivánek and Morávek [10]. Several heuristics were developed for

B. Yang, D.-Z. Du, and C.A. Wang (Eds.): COCOA 2008, LNCS 5165, pp. 1–12, 2008.

the WEIGHTED VARIANT or rely on its graph-theoretic intuition, including CLICK [14] and FORCE [16]. The problem is APX-hard, and has a constant-factor approximation of 2.5 [15]. To find exact solutions, Grötschel and Wakabayashi [7] formulated the problem as an Integer Linear Program. The parameterized complexity of unweighted CLUSTER EDITING, using the minimum number of edge modifications as the parameter k, is well-studied: Until recently, the fastest implemented algorithm had running time $O(2.27^k+n^3)$ on an n-vertex graph [6,4], while in theory, the best known algorithm has running time $O(1.92^k + n^3)$ [5]. Guo [8] presented a linear problem kernel. In contrast, the fixed-parameter tractability of CLUSTER EDITING with "don't care edges", that is, edges whose modification cost is zero, is still an open problem [3].

For WEIGHTED CLUSTER EDITING, the authors presented a problem kernel in [1] and, in particular, introduced a new data reduction technique of merging vertices. Furthermore, we provided two branching strategies with search trees of size $O(3^k)$ and $O(2.42^k)$, respectively, where parameter k is the minimum total cost of edge modifications. We found that merging vertices significantly reduces running times and, in contrast to what theoretical bounds suggest, the $O(3^k)$ strategy consistently outperformed the $O(2.42^k)$ strategy. An experimental evaluation of exact methods for CLUSTER EDITING, including the branching strategy presented in this paper, can be found in [2].

Our contributions. We concentrate on the case that edge insertion and deletion costs are positive integers, and sketch how to adopt our results for real-valued graphs where necessary. We present a new branching strategy that is surprisingly simple, and show that the resulting search tree is of size $O(2^k)$. We then refine our analysis and show that by accurately choosing edges to branch on, we obtain running time $O(1.82^k + n^3)$. Our algorithm is the fastest known for unweighted CLUSTER EDITING and improves on the $O(1.92^k + n^3)$ algorithm in [5].

In Sec. 5 we compare running times of our algorithm to the previously best known results for a parameterized algorithm for WEIGHTED CLUSTER EDITING [1], and we observe improvements of several orders of magnitude. In our comparison, we use both simulated graphs and graphs that stem from protein similarity data and aim at clustering homologous proteins.

2 Preliminaries

Let V be the set of objects to be clustered, corresponding to the vertices of the graph. In this work, we consider only undirected graphs without self-loops and multiple edges. 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 of the graph (we call it a *non-edge*) and has insertion cost $-s(uv)$. If $s(uv) = 0$, we call uv a *zero-edge*. Note that there are no zero-edges in the input graph, so that each pair of vertices is either an edge or a non-edge whose edit cost (deletion or insertion cost) is a positive integer. This is necessary solely to achieve provable running times. Nonetheless, zero-edges can

appear in the course of computation and require additional attention when analyzing the algorithm.

When analyzing connected components we regard zero-edges as non-existing. Throughout this paper we assume that circles and paths do not contain zero-edges. A circle of length three is also called a *triangle*. 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*. If all connected components of a graph are weak cliques, it is called *transitive*. Weak cliques in a transitive graph are also called *clusters*. An unweighted graph $G = (V, E)$ is transitive if and only if there exists no *conflict triple* in G, that is, three vertices vuw such that $vu, uw \in E$ but $vw \notin E$. Unfortunately, there exists no direct analogue of this statement for integer-weighted graphs. Vertices vuw form a *conflict triple* in an integer-weighted graph G_w if uv and uw are edges of G_w but vw is either a non-edge or a zero-edge. We distinguish two types of conflict triples vuw: if vw has weight zero then the conflict triple is called *weak*, whereas if vw is a non-edge then the conflict triple is called *strong*. In case the integer-weighted graph G_w contains no conflict triples then G_w is transitive. But the converse is obviously not true, as the example of a single weak conflict triple shows. A graph that does not contain any strong conflict triple is not necessarily transitive: For $V = \{u, v, w, x\}$ let uv, vw, wx be edges, let uw, vx be zero-edges, and let ux be a non-edge. The resulting graph is connected and contains no strong conflict triple, but is *not* a weak clique.

To solve Weighted Cluster Editing we first identify all connected components of the input graph and calculate the best solutions for all components 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. Our fixed-parameter algorithms often 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.

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", meaning that the status of uv cannot be changed in the future. In fact, permanent edges can be merged immediately: As introduced in [1], *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 not, the parameter k is reduced 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.

Throughout this paper, let $n := |V|$. To decrease input size, we introduced kernelization rules for Weighted Cluster Editing in [1]. For unweighted Cluster Editing, Guo [8] uses the concept of *critical cliques* to construct

a kernel of size $4k_{\text{opt}}$ for unweighted Cluster Editing. Critical cliques are cliques in the input graph which share the same neighborhood. In unweighted graphs, all vertices of a critical clique must end up in the same cluster, so we can always merge critical cliques. This idea does not apply directly to Weighted Cluster Editing but it is possible to adapt the concept by considering cliques with similar neighborhood [2]. When given an unweighted instance of cluster editing, we merge all critical cliques to transform the graph into an integer-weighted instance. The resulting graph has at most $4k_{\text{opt}}$ vertices. The weighted graph can be constructed from the critical clique graph that, in turn, can be easily constructed in $O(m+n)$ time [9] for an n-vertex and m-edge graph. The weight of any tuple uv is simply the product of the corresponding critical clique sizes $|C_u| \cdot |C_v|$.

3 Edge Branching

We now describe a recursive algorithm for integer-weighted Cluster Editing, following the bounded search tree paradigm. In this algorithm, we identify a conflict triple and then branch into two sub-cases to repair this conflict. By this, we invoke recursive calls on "simplified" instances of the problem where parameter k is decreased by some constants a, b. For branching vector (a, b) we can compute a branching number using the characteristic polynomial, and this branching number in turn governs the asymptotic size of the search tree, see e.g. [11] for details.

The **edge branching strategy** is as follows: Let uv be an edge of a (weak or strong) conflict triple vuw. Then, (a) set uv to forbidden, or (b) merge uv.

Let us first analyze this very simple strategy. One can easily check that this recursive procedure will at some point generate an optimal solution, because in every step we resolve a conflict triple. In the following we will analyze the size of the search tree. When deleting an edge uv we decrease the parameter by $s(uv)$. When merging vertices u, v, for each vertex $w \in V \setminus \{u, v\}$ we join the pairs uw and vw into a single pair with weight $s(uw) + s(vw)$. If $s(uw) \neq -s(vw)$ then parameter k can be lowered by $\min\{s(uw), -s(vw)\}$. In case $s(uw) = -s(vw)$ the new pair is a zero-edge, and this would prevent us from decreasing our parameter when joining the zero-edge in a later stage of the algorithm. To circumvent this problem, 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 latter stage we join this zero-edge with another pair, we decrease our parameter by the remaining $\frac{1}{2}$. Using this bookkeeping trick, our edge branching strategy has a branching vector of $(1, \frac{1}{2})$ that leads to a search tree of size $O(2.62^k)$.

We can easily improve this branching strategy by choosing a "good" edge uv, as follows: Choose the particular edge $uv \in E$ that *minimizes* the branching number of the corresponding branching step. The branching number is computed from branching vector (a, b) where a is the cost of deleting edge uv, while b is the cost of merging this edge. If one of these costs is zero, we say that the edge has infinite branching number. Using the bookkeeping trick introduced

above, an edge uv with finite branching number is not necessarily part of *any* conflict triple: joining a zero-edge uw with a vertex pair vw generates cost $\frac{1}{2}$ irrespective of whether vw is an edge, non-edge, or zero-edge. So, even the edge with minimum branching number might not be part of any conflict triple.

The following is a simple observation regarding unweighted graphs:

Lemma 1. *Given a connected, unweighted graph G. If every edge of G is part of at most one conflict triple, then G is either a clique or a clique minus a single edge.*

Proof. If $G = (V, E)$ contains no conflict triple then G is a clique. Assume that there is at least one conflict triple vuw in G with $uv, uw \in E$ and $vw \notin E$. We constructively show that G is a clique minus the edge vw. If another vertex $x \in V \setminus \{u, v, w\}$ is adjacent to v then $ux \in E$ must hold, too: otherwise, uv is part of two conflict triples vuw and uvx contrary to our assumptions. Similarly, $ux \in E$ implies $vx \in E$. In conclusion, $ux \in E$ if and only if $vx \in E$. The same holds replacing v by w, and we infer that if some vertex x is adjacent to one of u, v, or w then it is adjacent to all of u, v, and w.

Next, consider two vertices x, y adjacent to all u, v, w. If $xy \notin E$ then vxw and xvy are two conflict triples containing the edge xv which conflicts with our assumptions, so $xy \in E$ must hold. Finally, consider vertices x, z where x is adjacent to u, v, w while z is not adjacent to u, v, w, and assume $xz \in E$. Now the edge vx is part of the two conflict triples vxw and vxz, again a contradiction to our assumptions. So, any vertex $x \in V \setminus \{v, w\}$ must be adjacent to all other vertices in G. □

Lemma 2. *For an integer-weighted graph, the edge branching strategy that chooses an edge with minimum branching number has branching vector at least* $(1, 1)$.

Proof. Recall that if we create a zero-edge, this reduces k by at least $\frac{1}{2}$; and if we join a zero-edge, this reduces k by $\frac{1}{2}$. Let uv be the edge with minimum branching number. Note that removing uv induces cost $s(uv) \geq 1$, and let δ be the cost of merging uv. If $\delta \geq 1$ then we are done, so assume $\delta < 1$. This implies that at most one zero-edge was created or joined. In particular, uv is part of at most one conflict triple vuw, and there cannot be an edge that is part of two conflict triples. We transform the input graph into an unweighted graph G, where zero-edges and non-edges in the input graph are not present in G. By Lemma 1 above, the connected component containing vuw must be a clique minus vw in G. Regarding the weighted graph, all vertex pairs are edges except vw that may be a non-edge or a zero-edge. If vw is a zero-edge then our branching will stop when merging uv, so assume that vw is a non-edge. We now show that for this case, we can omit our bookkeeping trick of delayed parameter decrease.

We now either delete uv with cost $s(uv) \geq 1$, or merge uv. We distinguish the cases $s(uw) \geq -s(vw)$ and $s(uw) < -s(vw)$. If $s(uw) \geq -s(vw)$ holds then the joined pair has weight $s(uw) + s(vw) \geq 0$, the resulting connected component is a clique that can be removed from the graph, and we reduce the parameter

k by $\min\{s(uw), -s(vw)\} \geq 1$. For $s(uw) < -s(vw)$ the joined pair has weight $s(uw) + s(vw) < 0$, so we have not generated a zero-edge. We can assume in our analysis that parameter k is reduced by the full $\min\{s(uw), -s(vw)\} \geq 1$. So, the branching vector is at least $(1, 1)$ as claimed. □

Hence, edge branching results in a search tree of size $O(2^k)$ for integer-weighted graphs.

4 Refined Edge Branching

We now refine our edge branching and present a sketch of proof showing that the search tree has size $O(1.82^k)$. This results in the fastest known algorithm for unweighted Cluster Editing: the previously best-known branching strategy by Gramm et al. [5] results in a search tree of size $O(1.92^k)$. This algorithm uses complicated branching rules (more than 1300 cases) and has never been implemented. To the best of our knowledge, the fastest implementation for unweighted Cluster Editing has running time $O(2.27^k + n^3)$ using 11 branching cases [6,4]. In contrast, our branching strategy is both fast and simple using only two branching cases.

Theorem 1. *For an integer weighted graph that contains no zero-edges, the* Weighted Cluster Editing *problem can be solved in* $O(1.82^k + n^3)$ *time.*

We modify the order in which edges are processed by the edge branching strategy, what allows for a simpler analysis of the running time behavior. We conjecture that Theorem 1 is also true for edge branching where edges are sorted with respect to branching number, but this requires many more case distinctions.

Let G_w be an integer-weighted and connected graph. We say that we *branch on* an edge uv by setting uv to forbidden and recursing, and merging uv and recursing. To deal with zero-edges, we use the above bookkeeping trick: Creating a zero-edge induces cost $\geq \frac{1}{2}$, and resolving a zero-edge induces the remaining cost $\frac{1}{2}$. We choose an edge to branch on according to the following order:

(A) If there is an edge with branching vector $(1, \frac{3}{2})$ or better then we branch on this edge.
(B) If there is an edge xy and a vertex z in G_w such that x,y,z form a triangle, and if there exist two additional vertices v_1,v_2 such that for both v_1,v_2 one of the following conditions holds (where x and y may be exchanged):
  (B1) xv_i is an edge and yv_i is a non-edge
  (B2) xv_i is a zero-edge and yv_i is a zero-edge
  (B3) xv_i is a zero-edge and yv_i is a non-edge, and zv_i is an edge or a zero-edge
  (B4) xv_i is an edge and yv_i is a zero-edge, and zv_i is a non-edge or a zero-edge
  Then branch on xy.

If no such edge exists, we stop the recursion. We will show below that the remaining graph must be a clique, a clique minus one edge (where the last edge is either a zero-edge or a non-edge), a path, a circle, or contains only 4 vertices. We

will also show how to solve this remaining instance in polynomial time. See Fig. 1 for the four initial cases of condition (B). To be more precise, there are ten different subcases of condition (B) which are combinations of (B1),...,(B4), taking into account that we can exchange x and y. We denote them by (B11) to (B44).

If there exists an edge satisfying condition (A) then branching on this edge has branching number 1.76. The following lemma corresponds to condition (B) of edge sorting, and shows how we analyze two branching steps together: The first branching step can in fact result in a branching vector of $(1, 1)$ but the next branching steps result in better branching vectors, leading to an overall branching number as desired.

Lemma 3. *Let G_w be an integer-weighted and connected graph, and assume that there is an edge xy that satisfies condition (B). Then, branching on xy and performing another branching step where edges to branch on are chosen according to the edge sorting, results in a branching vector of $(2, \frac{5}{2}, 2, 3)$ with branching number ≤ 1.82.*

Proof. Branching on edge xy leads to a branching vector of $(1, 1)$: Deleting xy induces cost at least 1, and merging xy results in cost at least $2 \cdot \frac{1}{2} = 1$ since for each v_i a conflict triple or a zero-edge will be resolved. We will now show that after setting xy to "forbidden" there exists an edge with branching vector $(1, \frac{3}{2})$ and after merging xy there exists an edge with branching vector $(1, 2)$. These are the worst-case branching vectors for the edge which is chosen in the next branching step.

First we analyze the case where xy is set to "forbidden", see Fig. 2: We show that now one of the edges xz or yz has branching vector $(1, \frac{3}{2})$. Setting xz or yz to forbidden results in cost 1. Merging xz or yz resolves the conflict triple xzy, resulting in cost 1 since xy is forbidden. If condition (B2), (B3), or (B4) holds then in addition, a zero-edge is resolved when merging xz or yz. If condition (B1) holds we distinguish two cases: If v_1z is a non-edge or a zero-edge, then we branch on xz which either resolves an additional zero-edge, or resolves the conflict triple v_1xz. If v_1z is an edge then we branch on yz which resolves the conflict triple v_1zy. Hence, either xz or yz have merging costs $\frac{3}{2}$.

Second we consider the case where x, y have been merged, see Fig. 3. Let w_{xy} be the vertex resulting from merging xy: We show that now, the edge $w_{xy}z$ has branching vector $(1, 2)$. Deleting $w_{xy}z$ induces cost of 2 as $s(w_{xy}z) \geq 2$. Merging

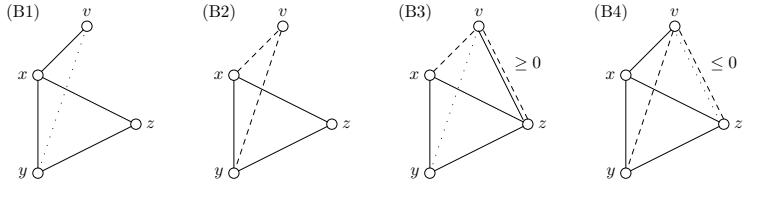

Fig. 1. Conditions (B1) to (B4) of edge sorting. Solid lines are edges, dashed lines are zero-edges, dotted lines are non-edges.

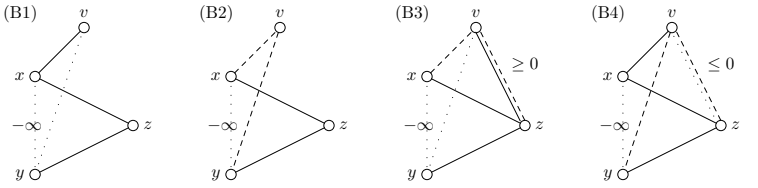

Fig. 2. Branching conditions (B1) to (B4) after xy is set to "forbidden"

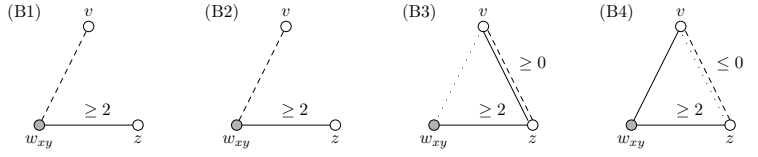

Fig. 3. Conditions (B1) to (B4) after merging xy

$w_{xy}z$ induces cost of $\frac{1}{2}$ for each v_i: If condition (B1) holds for v_i then $w_{xy}v_i$ is a zero-edge. Otherwise, we infer $s(xv_i) > 1$ or $s(yv_i) < -1$, so the initial branching on xy would have resulted in a branching vector of $(1, \frac{3}{2})$. Merging $w_{xy}z$ resolves this zero-edge. If condition (B2) holds then $w_{xy}v_i$ clearly is a zero-edge. For conditions (B3) and (B4) merging $w_{xy}z$ either resolves a zero-edge v_iz, or resolves a conflict triple $w_{xy}zv$ or $vw_{xy}z$. These observations hold both for v_1 and v_2, so merging $w_{xy}z$ results in total cost $\frac{2}{2}$.

We cannot guarantee that the edge branching strategy will actually branch on edges xz or yz (after xy has been set to forbidden) and $w_{xy}z$ (after merging xy) in the next step of the branching. But we have shown that edges with branching numbers 1.755 and 1.6191 exist after the first step of the branching. With regards to the first case, one can easily check that all possible branching vectors with branching number ≤ 1.755 are of the form $(a, b/2)$ for integers $a \geq 1$ and $b \geq 3$. Similarly, all branching vectors with branching number ≤ 1.6191 are of the form $(a, b/2)$ for integers $a \geq 1$ and $b \geq 4$, or $a \geq 2$ and $b \geq 2$. This shows that even if we pick other edges in the second step of our branching, we still can guarantee branching vector $(2, \frac{5}{2}, 2, 3)$ with branching number 1.82. □

The following is again an observation regarding unweighted graphs:

Lemma 4. *Let G be a connected, unweighted graph. Assume that there is no edge in G that is part of three conflict triples, and there exists no triangle uvw in G such that uv is part of two conflict triples. Then G is a clique, a graph with at most one non-edge, a $K_{1,3}$, a path, or a circle.*

Proof. Assume that $G = (V, E)$ contains at least one edge xy that is part of two conflict triples: Otherwise, Lemma 1 guarantees that G is a clique or a clique minus a single edge. Let u and w be two vertices involved in conflict triples for

xy. This implies that either xu or yu is an edge, and that either xv or yv is an edge. Assume there exists another vertex $z \notin \{x, y, u, v\}$ with $xz \in E$: If $yz \in E$ then xyz is a triangle as excluded by our assumptions, and if $yz \notin E$ then xy is part of three conflict triples. So, no such z can exist and neither x nor y can be connected to any other vertex.

We distinguish two cases: the two conflict triples are either of the form uxy and uxv (asymmetric case), or uxy and xyv (symmetric case). For the asymmetric case, we can exchange u and v. Assume there exists another vertex $w \notin \{x, y, u, v\}$ with $uw \in E$. Then the edge xu is part of two conflict triples yxu and xuw and an additional edge xv exists. If $uv \notin E$ then xu is part of three conflict triples. If $uv \in E$ then the edge xu is part of a triangle xuv that is excluded by our assumptions. This implies that no such vertex w can exist, and the connected graph G is a $K_{1,3}$.

For the symmetric case, assume that there exists another vertex $w \notin \{x, y, u, v\}$ with $uw \in E$. Now, the edge xu is part of two conflict triples yxu and xuw, again in symmetric arrangement. If some $z \notin \{x, y, u, v, w\}$ exists with uz, we can show again that xu is part of three conflict triples or part of a triangle excluded by our assumptions. The same holds true for a vertex w with $vw \in E$. Repeating this argument we show that all vertices in the connected graph G have degree one or two, so G is a path or a circle. □

Let us now assume that there is no edge that satisfies branching conditions (A) or (B). Again, we transform the integer-weighted graph into an unweighted graph G where zero-edges of the integer-weighted graph are transformed into non-edges in G. Clearly, G does not contain an edge that is part of three conflict triples. Using Lemma 4 we infer that G is either one of the graph structures described there, or there exists an edge xy that is part of a triangle xyz and that is part of two conflict triples. In the first case, we have reduced the weighted graph as claimed: The weighted graph is a clique, a clique minus one edge, a path, a circle, or contains only four vertices. In the second case there is an edge xy which is contained in a triangle and two conflict triples for which branching condition (B) does not apply. It can be shown by rather technical analysis that in all cases the weighted graph is a weak clique or a graph with exactly one non-edge. We defer the details to the full paper.

If the remaining graph is a (weak) clique, we are finished. If it is a graph with one non-edge uv, we can solve it in polynomial time by calculating a minimum u-v-cut. In case the cost of the cut is higher than $-s(uv)$, we insert uv and are finished, otherwise we cut the graph according to the minimum u-v-cut and obtain two (weak) cliques. If the remaining graph is a path or a circle, it can be solved in polynomial time with dynamic programming. Again, we defer the details to the full paper. If the graph has at most four vertices, we can easily try all possibilities of solving it.

Proof (Theorem 1). From the above we infer that our search tree has size $O(1.82^k)$. This results in a total running time of $O(1.82^k \cdot k^8 + n^3)$: Initially, we run the parameter-dependent data reduction from [1] in time $O(n^3)$. This data reduction results in a problem kernel with $O(k^2)$ vertices. For every edge

Table 1. Average running times for artificial data, edge branching and $O(3^k)$ branching strategy from [1]. Ten instances per bucket for sizes 10–50, five instances for sizes 60–100. For size 70 (80, 90, 100) one (four, all five, all five) instances did not stop after 20 days of computation using the $O(3^k)$ strategy. For size 90 (100) two (three) instances did not stop after 20 days of computation using the edge branching strategy. For average running times, we ignored these unfinished instances (*).

Size of instance	10	20	30	40	50	60	70	80	90	100
average # edit	8.3	28.1	66.7	115.5	183.2	263.0	351.6	459.0	594.0	728.6
3^k strategy [1]	10 ms	54 ms	1.0 s	29 s	7.6 min	27 h	58 h*	19 days*	n/a*	n/a*
edge branching	4 ms	16 ms	238 ms	2.5 s	18.2 s	5.5 h	17.7 h	13.8 h	34.8 h*	17.1 h*
with reduction [2]	3 ms	14 ms	163 ms	1.2 s	1.6 s	32 s	43 s	23 s	166 s	36 s

we compute the branching number that results from deleting and merging this edge in total time $O(k^6)$. Similarly, we can check for the substructures for branching condition (B) in time $O(k^8)$. In fact, we can get rid of the polynomial factor: We use interleaving [12] by performing data reduction repeatedly during the course of the search tree algorithm whenever possible. This reduces the total running time to $O(1.82^k + n^3)$. The remaining structures can be solved in polynomial time. □

Regarding Weighted Cluster Editing instances with real-valued weights, the edge branching strategy is also guaranteed to find the optimal solution. Let k be the cost parameter, we want to decide whether there is a solution of cost at most k. To estimate the worst-case running time we have to assume that all vertex pairs have weight at least one [1]. We redo our simple analysis from Sec. 3: Whenever joining two pairs of vertices results in a pair with absolute weight smaller than one, we put aside $\frac{1}{2}$ using our bookkeeping technique. This pair may later be part of a conflict triple, and when editing this pair we decrease k by $\frac{1}{2}$ we put aside earlier because the absolute weight of this pair can be arbitrarily small. A similar analysis to that given in this section, shows that the worst-case branching vector reduces to $(\frac{1}{2}, 2, 2)$ and the size of the search tree is $O(2.39^k)$. We defer the details to the full paper.

5 Computational Results

We have implemented the edge branching algorithm with search for the edge with maximum branching number in C++. We apply our data reduction from [1] to every instance in advance and when traversing the search tree. The program accepts nonnegative real values as edge modification costs. All running times were measured on an AMD Opteron-275 2.2 GHz with 6 GB of memory running Solaris 10.

We want to explore the performance of our algorithms and compare it to the previously fastest branching strategy for Weighted Cluster Editing from [1]. As reported there, branching strategies that do not merge vertices are clearly and consistently outperformed by those that do so, and unlike what theoretical running times suggest, the $O(2.42^k)$ was consistently outperformed

by the $O(3^k)$ strategy. For our evaluation, we use artificial data. We generate artificial instances by first constructing a transitive graph with n vertices by uniformly drawing clique sizes in $\{1, \ldots, n\}$ until all vertices have been used up. Next, we perturb this graph: for each pair uv we delete or insert an edge uv with probability 0.15. Running times are reported in Table 1. We also run experiments on the protein similarity data used in [1] and observed similar results. As one can see, edge branching is much faster than the previously fastest branching algorithm, and performance is increased by several orders of magnitude. For comparison, we also report running times of the FPT algorithm from [2] that uses the same edge branching strategy but, in addition, employs new parameter-independent reduction rules to cut down instance sizes before branching, and further heuristic improvements.

6 Conclusion

We have presented a surprisingly simple branching strategy that lead to the fastest known parameterized algorithm for (integer-weighted) Cluster Editing with respect to theoretical running time bounds. We believe that we can prove even better worst-case running times for this same strategy, using a refined, automated analysis similar to [5].

We implemented our algorithm and evaluated its performance. Together with further improvements reported in [2], our algorithm allows to solve weighted Cluster Editing instances with several hundred edge modifications in a matter of seconds. This clearly proves the practical usefulness of our approach and constitutes a huge improvement over [4] where unweighted instances with 50 edge modifications required several hours of computation. Wittkop et al. [16] recently demonstrated the power of Weighted Cluster Editing for clustering homologous proteins, so algorithm both fast in theory and efficient in practice are highly desirable.

Acknowledgments

We thank Svenja Simon and Thilo Muth for helping with the implementation. S. Briesemeister gratefully acknowledges financial support from LGFG Promotionsverbund “Pflanzliche Sensorhistidinkinasen” at the University of Tübingen.

References

1. Böcker, S., Briesemeister, S., Bui, Q.B.A., Truß, A.: A fixed-parameter approach for weighted cluster editing. In: Proc. of Asia-Pacific Bioinformatics Conference (APBC 2008). Series on Advances in Bioinformatics and Computational Biology, vol. 5, pp. 211–220. Imperial College Press (2008)
2. Böcker, S., Briesemeister, S., Klau, G.W.: Exact algorithms for cluster editing: Evaluation and experiments. In: McGeoch, C.C. (ed.) WEA 2008. LNCS, vol. 5038, pp. 289–302. Springer, Heidelberg (2008)

3. Bodlaender, H.L., Cai, L., Chen, J., Fellows, M.R., Telle, J.A., Marx, D.: Open problems in parameterized and exact computation — IWPEC 2006. Technical Report UU-CS-2006-052, Department of Information and Computing Sciences, Utrecht University (2006)
4. Dehne, F., Langston, M.A., Luo, X., Pitre, S., Shaw, P., Zhang, Y.: The cluster editing problem: Implementations and experiments. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 13–24. Springer, Heidelberg (2006)
5. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Automated generation of search tree algorithms for hard graph modification problems. Algorithmica 39(4), 321–347 (2004)
6. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-modeled data clustering: Fixed-parameter algorithms for clique generation. Theor. Comput. Syst. 38(4), 373–392 (2005)
7. Grötschel, M., Wakabayashi, Y.: A cutting plane algorithm for a clustering problem. Math. Program. 45, 52–96 (1989)
8. Guo, J.: A more effective linear kernelization for Cluster Editing. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 36–47. Springer, Heidelberg (2007)
9. Hsu, W.-L., Ma, T.-H.: Substitution decomposition on chordal graphs and applications. In: Hsu, W.-L., Lee, R.C.T. (eds.) ISA 1991. LNCS, vol. 557, pp. 52–60. Springer, Heidelberg (1991)
10. Křivánek, M., Morávek, J.: NP-hard problems in hierarchical-tree clustering. Acta Inform. 23(3), 311–323 (1986)
11. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford University Press, Oxford (2006)
12. Niedermeier, R., Rossmanith, P.: A general method to speed up fixed-parameter-tractable algorithms. Inform. Process. Lett. 73, 125–129 (2000)
13. Shamir, R., Sharan, R., Tsur, D.: Cluster graph modification problems. Discrete Appl. Math. 144(1–2), 173–182 (2004)
14. Sharan, R., Maron-Katz, A., Shamir, R.: CLICK and EXPANDER: a system for clustering and visualizing gene expression data. Bioinformatics 19(14), 1787–1799 (2003)
15. van Zuylen, A., Williamson, D.P.: Deterministic algorithms for rank aggregation and other ranking and clustering problems. In: Proc. of Workshop on Approximation and Online Algorithms (WAOA 2007). LNCS, vol. 4927, pp. 260–273. Springer, Heidelberg (2008)
16. Wittkop, T., Baumbach, J., Lobo, F., Rahmann, S.: Large scale clustering of protein sequences with FORCE – a layout based heuristic for weighted cluster editing. BMC Bioinformatics 8(1), 396 (2007)