Even Faster Parameterized Cluster Deletion and Cluster Editing

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

.

Peter Damaschke*
Department of Computer Science and Engineering
Chalmers University, 41296 Göteborg, Sweden
ptr@chalmers.se

Abstract

Cluster Deletion and Cluster Editing ask to transform a graph by at most k edge deletions or edge edits, respectively, into a cluster graph, i.e., disjoint union of cliques. Equivalently, a cluster graph has no conflict triples, i.e., two incident edges without a transitive edge. We solve the two problems in time $O^*(1.415^k)$ and $O^*(1.76^k)$, respectively. These results round off our earlier work by considerably improved time bounds. For Cluster Deletion we use a technique that cuts away small connected components that do no longer contribute to the exponential part of the time complexity. As this idea is simple and versatile, it may lead to improvements for several other parameterized graph problems. The improvement for Cluster Editing is achieved by using the full power of an earlier structure theorem for graphs where no edge is in three conflict triples.

Keywords: graph algorithms, cluster editing, parameterized complexity

1 Introduction

A problem with input size n and another input parameter k is *fixed-parameter tractable (FPT)* if it can be solved in $O(p(n) \cdot f(k))$ time where p is a polynomial and f any function. Since we focus on the $f(k)$ factor, we adopt the $O^*(f(k))$ notation that suppresses polynomial factors. (While it cannot be neglected in practice, it is usually moderate, so that $f(k)$ "dominates" the complexity.) We deal with graph problems and denote by n the number of vertices. For general introductions we refer to [1, 2], in particular, we assume familiarity with bounded search trees, branching vector, and branching number. When mentioning concrete branching numbers we round them to some appropriate decimal. A b *rule* is a branching rule with branching number b or smaller.

*Corresponding author. Tel. 0046-31-772-5405. Fax 0046-31-772-3663.

A *cluster graph* is a disjoint union of cliques, or equivalently, a graph without induced P_3, which denotes a path of three vertices and two edges. Therefore P_3 is also called a *conflict triple* in this context. (Some more special graph-theoretic notions will be defined later.) The Cluster Deletion problem asks to delete at most k edges in a graph so as to obtain a cluster graph. Cluster Editing is similarly defined, but edge deletions and insertions are permitted. Both problems are special cases of Integer-Weighted Cluster Editing where the vertex pairs have individual edit costs. These problems have motivations from computational biology [3, 4, 5, 6], mainly clustering of genes based on similar expression profiles, and they are NP-hard [5]. Cluster Deletion is important if only adjacent vertices, representing similar genes or other items, are acceptable in a cluster. A variant with vertex (rather than edge) deletions is considered in [7].

The parameterized complexity of these problems is well studied. In [8] we give problem kernels and algorithms for enumerating all solutions to several clustering problems. For Cluster Editing, a first algorithm with running time $O^*(2.27^k)$ [4] was improved to $O^*(1.92^k)$ in [12]. The currently fastest algorithm runs in $O^*(1.82^k)$ time [9]. A problem kernel with $4k$ vertices [10] was recently improved to size $2k$ [11]. Faster algorithms are known for Cluster Deletion which has a similar history: A first $O^*(1.77^k)$ time algorithm [4] was improved to $O^*(1.53^k)$ in [12], and later to $O^*(1.47^k)$ in [13]. In [13] we also explored a technique (for a different problem) that splits off, by efficient branchings, components of an input graph that are small or simple, which allows immediate deductions from the parameter k during branching, since these components do not affect the exponential factor in the complexity. The NP-complete clustering problems may become easier in special graphs, in particular, geometric vicinity graphs are relevant. As a first step, Cluster Editing in unit interval graphs (1-dimensional vicinity graphs) can be solved in polynomial time, since clusters in an optimal solution are consecutive sets, which is expected but surprisingly hard to prove [14].

Our new contribution, however, is further progress on the parameterized bounds, by bringing together ingredients that were independently developed before. Using the aforementioned split-off technique we can now solve Cluster Deletion in $O^*(1.415^k)$ time. Then, we improve the time for Integer-Weighted Cluster Editing to $O^*(1.76^k)$ by combining the approach from [9] with a structure theorem from [13].

In Section 2 we will need the following general lemma. We skip the obvious and short proof which is just application of the definitions.

Lemma 1 *Consider any two branching rules that have characteristic equations $x^a = p(x)$ and $x^b = q(x)$, where p and q are polynomials of degree smaller than a and b, respectively. Suppose that we apply the first rule and, in every branch, we apply the second rule. Then the compound rule has the characteristic equation $x^{a+b} = p(x)q(x)$.* $\diamond$

2 Faster Cluster Deletion by Edge Separation

An important routine in our Cluster Deletion algorithm is to split long paths into paths of some constant length:

Lemma 2 *If an instance of* Cluster Deletion *contains a path of* 8 *edges whose* 7 *inner vertices have degree* 2*, we can isolate a subpath using a* 1.389 *rule.*

Proof. Consider a path of edges $e_0, e_1, \ldots, e_{m-1}, e_m$ (we shall fix m below), whose inner vertices have degree 2. We branch on e_1 and e_{m-1}, that is, we decide which of these edges we delete. Of course, if we keep e_1, we can delete e_0 and e_2 immediately. Similarly, if we keep e_{m-1}, we can delete e_{m-2} and e_m immediately. In each of the four branches, the subpath starting in e_2 or e_3 and ending in e_{m-3} or e_{m-2} now forms a connected component of constant size (as m is fixed). No further branching on this component is needed, instead we delete the minimum number of edges that destroy all P_3, that is, we simply delete every other edge. Choosing $m := 7$, this yields the branching vector $(4, 4, 4, 5)$. The cases are illustrated below; symbols 0,1,x stand for "kept", "deleted", "decided later", and numbers in parentheses indicate how many further edges are deleted from the split-off path.

1-0-1-x-x-1-0-1 (+1)
1-0-1-x-x-x-1-x (+1)
x-1-x-x-x-1-0-1 (+1)
x-1-x-x-x-x-1-x (+2)

The characteristic equation $x^5 = 3x + 1$ has solution $x < 1.389$, thus the lemma is proven. $\diamond$

We define the *score* of an edge e to be the number of induced P_3 that contain e. The score of a graph is the maximum score among its edges. *Branching on an edge* e means to delete e, or to delete all edges that form induced P_3 with e. As usual, C_4 denotes the simple cycle of 4 vertices and 4 edges.

Lemma 3 *If some edge e of score larger than 3 exists in the graph, then a 1.381 rule for* CLUSTER DELETION *is available.*

Proof. We must destroy all induced P_3 containing e, thus we branch on e, with branching vector $(1, 4)$ or better. $\diamond$

Lemma 4 *If an induced C_4 exists, we can apply a 1.415 rule for* CLUSTER DELETION.

Proof. If we delete at most one edge from each pair of opposite edges in an induced C_4 then, obviously, some induced P_3 remains. Thus we must delete some pair of opposite edges, which yields the branching vector $(2, 2)$. $\diamond$

Let $A = \{u, v, w\}$ be the vertex set of some fixed induced P_3, say, with edges uv and vw. Let B be the set of all vertices adjacent to one or two vertices in A, and C be the set of all vertices adjacent to all vertices in A.

Lemma 5 *B has at most four vertices, otherwise some edge has score larger than* 3.

Proof. Consider one of the edges in A, say uv. Since uv forms already a P_3 with vw, and the score of uv is limited to at most 3, at most two vertices of B are adjacent to exactly one of u and v. The same argument applies to edge vw, thus at most two further vertices of B are adjacent to exactly one of v and w.

Now consider any vertex $z \in B$. By definition of B, vertex z has either one or two neighbors in A. It is easy to check that at least one of these two statements is true: z has exactly one neighbor among u, v; z has exactly one neighbor among v, w. Thus, at most four vertices $z \in B$ can exist. $\diamond$

The bound in Lemma 5 is optimal, as B can actually have four vertices: two adjacent to only u, and two adjacent to only w.

Lemma 6 *C is a clique, otherwise the graph has an induced C_4.*

Proof. This is obvious since any vertex in C is also adjacent to u and w. $\diamond$

Next, let D be the set of all vertices that are not already in $A \cup B \cup C$ and have neighbors in C. Note that vertices in D have no neighbors in A.

Lemma 7 *Suppose that a graph has score 3 and no induced C_4. Then, D has at most three vertices, every vertex of D is adjacent to all vertices of C, and to no vertex outside $B \cup C \cup D$.*

Proof. Consider any $x \in D$, and some neighbor $y \in C$ of x. The edge xy is already involved in three P_3 with the vertices of A. Since C is a clique by Lemma 6, we can keep the score of xy at 3 only if x is adjacent to all of C. Now assume that D has four or more vertices. Since any $y \in C$ is adjacent to all of them, we find an edge with score 4 (for instance vy), a contradiction. Finally, assume that some $x \in D$ has a neighbor $z \notin B \cup C \cup D$. Again, let y be a neighbor of x in C. Since z has no neighbor in C, edge xy forms another P_3 with z, on top of the three P_3 with the vertices of A, which contradicts score 3. $\diamond$

In the following we suppose that our graph has score 3 and no induced C_4.

Finally we define $B_0 := B$, and for $i > 0$ we define B_i to be the set of all vertices that are not already in $A \cup B \cup C \cup D$ and have minimum distance i to the vertices in B. By definitions and by Lemma 7, vertices in B_i, $i > 0$, cannot be adjacent to any vertices in $A \cup C \cup D$.

For any two disjoint subsets X and Y of vertices we use $X - Y$ to denote the set of edges with one vertex in X and one in Y. At this point, remember that at most the following edge sets between the vertex sets considered so far can be nonempty: $A - B$, $A - C$, $B - C$, $B - D$, $C - D$, and $B_i - B_{i+1}$ for $i \geq 0$. We refer to the edge set $B_i - B_{i+1}$ as *layer i*.

Let j be minimal such that some edge of score 3 exists in layer j. In other words, all edges in the layers $i < j$ have score 1 or 2; the score is at least 1, since every edge in layer i forms a P_3 with some edge in layer $i - 1$, or with $A - B$ if $i = 0$. It follows immediately that every edge in a layer $i < j$ is incident to at most one edge of the next layer $i + 1$.

Hence the set of edges in the layers $i < j$ can be partitioned into paths of edges of score 2, each starting in B_0 and having at most one edge from every layer. (If two edges from different paths share a vertex in some B_i, then at most one edge can continue both paths in layer i, and then these paths terminate. In this case we arbitrarily assign this last edge to one of the involved paths.) We refer to these paths from B_0 to B_j as *chains*. By Lemma 5 we have at most four vertices in B_0, thus at most four chains exist if $j > 0$ (and there can actually exist four chains). No edges exist inside the B_i, $0 < i < j$, as they would render some edges in the chains to have score larger than 2, contradicting the choice of j.

Theorem 8 Cluster Deletion *can be solved in* $O^*(1.415^k)$ *time.*

Proof. In a first phase we branch on edges of score 4 or larger, and on induced C_4, as long as possible, according to Lemma 3 and 4. Then we fix some P_3 (if none exists, we are done), and define A, B, C, D and the B_i as above.

Now let us branch on some edge e of score 3 in layer j, with minimum j. In particular, we delete e, or we delete the edge f that precedes e in the chain e belongs to. In both branches, the subgraph induced by $A \cup C \cup D$ is not affected, and the chain of e is interrupted. Hence, after at most four such branchings (where j may increase each time), no chain from B to an edge of score 3 is left. At this stage, in the remaining graph, the connected component of A consists of a clique (C, see Lemma 8), a constant number of other vertices that are fully connected to this clique ($A \cup D$, see Lemmas 5 and 7), and at most four paths of edges of score 2 whose internal vertices are, thus, not linked to each other. Due to the very restricted structure of this graph, we can finish as follows. If some chain is larger than some constant length, we can use one branching to disconnect a simple path of that length from the rest, according to Lemma 2. This is repeated until the chain has length bounded by a constant. After this step, there remains a clique with a constant number of other vertices attached. As shown in [13] (proof of Theorem 4), this case can be solved in polynomial time.

Altogether, the maximum branching number in the second phase is determined by the at most four branchings on edges of score 3, in each component of a fixed set A inducing a P_3. The key argument why we can improve the branching number 1.47 of branching vector $(1, 3)$ is that at least one P_3 still exists in the component after these branchings, namely the unchanged subgraph induced by A. We can deduct 1 from the parameter, for the edge deleted later in A, because this deletion is part of a polynomial-time step. To sum it up, four branchings have the characteristic equation $x^3 = x^2 + 1$, and the deduction 1 we get for free has the trivial characteristic equation $x = 1$. Using Lemma 1, this yields the characteristic equation $x^{4 \cdot 3+1} = x^{13} = (x^2 + 1)^4$ with solution $x = 1.394 < 1.415$. $\diamond$

As a side remark, the algorithm may be modified to output a concise enumeration of all clusterings reachable by at most k edge deletions. (The interested reader may consult [8] where concise enumerations are addressed.)

3 Improved Search Tree Algorithm for Cluster Editing

For brevity, we write uv as shorthand for an unordered pair $\{u, v\}$ of elements (vertices) of a set V. Let s be a *weight function* that assigns to every pair of vertices an integer and encodes the input graph: For $s(uv) > 0$, the pair uv is an edge of the graph and has deletion cost $s(uv)$, while for $s(uv) < 0$, the pair uv is a *non-edge* of the graph and has insertion cost $-s(uv)$. If $s(uv) = 0$, we call uv a *zero-edge.* No zero-edges are in the input graph, but they can appear in the course of computation and require additional attention when analyzing the algorithm. In the following, G refers to a graph and G_s is the integer-weighted graph encoded by the weight function s.

When analyzing connected components we regard zero-edges as non-existing. We say that $C \subseteq V$ is a *clique* in an integer-weighted graph if all pairs uv in C are edges. If all vertex pairs of a connected component are either edges or zero-edges, we call it a *weak clique*. Vertices v, u, w form a *conflict triple* vuw in an integer-weighted graph if uv and uw are edges of G_s but uw is a non-edge or zero-edge. A conflict triple vuw where $s(vw) = 0$ is called *weak*, whereas if $s(vw) < 0$ 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. The Integer-Weighted Cluster Editing problem asks for a set of edge modifications of minimum total weight that turns the input graph into a cluster graph. We explicitly forbid zero-edges in the input. To solve the problem we first identify all connected components of the input graph and calculate the best solutions for each component separately, because an optimal solution never inserts edges between 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 not, 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.

Graphs that do not allow for an efficient branching (see below) are characterized in [13]. As this characterization is central for our novel running time analysis, we repeat it here. 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 connected to every vertex from H. Finally, the graph G^c has the same vertex set as G, and uv is an edge of G^c if and only if it is no edge of G. Now, Theorem 5 from [13] states:

Theorem 9 *Let G be a connected graph where no edge is part of three conflict triples. Then, G has at most six vertices, or it is a connected induced subgraph of one of the following graphs, for positive integers n, p, q: C_n, $K_q * C_5$, $K_q * K_3^c$, $K_q * (K_2 + K_2)$, or $(q \cdot K_1 + p \cdot K_2)^c$ for $p \geq 2$.* $\diamond$

We use that to prove:

Theorem 10 Integer-Weighted Cluster Editing *without zero-edges can be solved in $O^*(1.76^k)$ time.*

Proof. Our branching proceeds as follows: Let uvw be a (weak or strong) conflict triple. We *branch on an edge uv* by branching into two cases: In the first case, we remove uv and set it to forbidden; in the second case, we set uv to permanent, and immediately merge it. One can easily check that this branching will ultimately come up with all possibilities of transforming the input graph into a disjoint union of cliques [9].

If uv is part of at least one strong conflict triple, then merging uv will generate cost: As there are both an edge uw and a non-edge vw, we can reduce k by $\min\{s(uw), -s(vw)\}$. In case $s(uw) = -s(vw)$, 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 $\frac{1}{2}$. So, both generating and destroying a zero-edge generates cost of at least $\frac{1}{2}$. This bookkeeping trick was introduced in [9].

If there is an edge uv such that merging u, v and setting uv to forbidden leads to a branching number of 1.76 or better, we branch on this edge. Any edge that is part of three (weak or strong) conflict triples satisfies this condition, as merging u, v will result in costs $\frac{3}{2}$ for generating or destroying three zero edges. Suppose uv is part of a strong conflict triple vuw satisfying $s(uw) \geq 2$ or $s(vw) \leq -2$, and an additional (weak or strong) conflict triple. Then, branching on uv also leads to branching vector $(1, \frac{3}{2})$ with branching number 1.76, as joining vw and uw generates cost of at least 1. Finally, if uv is part of two (weak or strong) conflict triples vuw and $s(uv) \geq 2$, then branching on uv results in branching vector $(2, 1)$ and satisfies the above condition.

Now, assume that we cannot find an edge uv such that branching on uv has branching number 1.76 or better. In particular, this implies that there is no edge in the graph that is part of three or more conflict triples. We want to characterize the weighted graph G_s using Theorem 9; but as this theorem is on unweighted graphs, we first transform our weighted graph G_s into its unweighted counterpart G. It must be understood that the unweighted graph G is used solely to apply the theorem, and discarded thereafter. For our transformation, all edges of G_s become edges of G; all non-edges of G_s become non-edges of G; and, finally, all zero-edges of G_s become non-edges of G. In this way, any (weak or strong) conflict triple in G_s is also a conflict triple in G. Consequently, there is no edge uv in the unweighted graph G that is part of three or more conflict triples. Thus, we can apply Theorem 9 to characterize G. In the following, we again concentrate on the corresponding weighted graph G_s: Any non-edge in this characterization can be a non-edge or zero-edge in G_s, and edges and non-edges can be arbitrarily weighted. In contrast, G only represents the "type" of G_s.

We now show that we can efficiently solve all remaining instances. If the graph is a clique, we are done. Next, assume that the graph is a clique minus a single non-edge or zero-edge uv. Such an instance can be solved in polynomial time using an u-v-mincut algorithm [15]. In fact, the solution is even simpler: for $-s(uv) \leq k$ the answer is YES, as we can insert uv. Otherwise, we infer $s(uw) = s(vw) = 1$ for all $w \neq u, v$, as there is no edge to branch on. One can easily see that any u-v-mincut will either separate u or v in the graph. So, we test $\deg(u) \leq k$, in which case the answer is again YES, as we can separate u.

Finally, assume that the graph G_s contains at least two non-edges or zero-edges. If G_s has constant size, we can solve it in constant time. Otherwise, the corresponding unweighted graph must be an induced subgraph of one of the graphs $K_q * C_5$, $K_q * K_3^c$, $K_q * (K_2 + K_2)$, or $(q \cdot K_1 + p \cdot K_2)^c$. For any non-edge vw of the weighted graph, we can find an edge of G, say uv, such that vuw is a conflict triple, and uv is part of another conflict triple. This implies

$s(vw) = -1$ as otherwise, we could branch on uv with branching vector $(1, \frac{3}{2})$. So, the cost for inserting all edges into the graphs is at most 5, 3, 4, or p, respectively. We focus on the last case, the others follow trivially. For the unweighted graph $(q \cdot K_1 + p \cdot K_2)^c$, we show that transforming G_s into a clique is always optimal.

Note that for every vertex in G_s, there is at most one non-edge or zero-edge that contains the vertex. Let $p' \leq p$ be the actual costs for transforming G_s into a clique. Assume that the optimal solution instead splits the vertex set into sets $X, Y \subseteq V$ with $X \cap Y = \emptyset$ and $|X| \leq |Y|$. If $|X| = 1$ then we save cost at most 1 for not inserting a single edge, so costs are at least $1(p + q - 1) + (p' - 1) \geq p + q + p' - 2 > p'$ for $p + q \geq 3$. In general, if $|X| = x$ then we save cost at most x for not inserting edges, so the costs are at least $x(p + q - x) + (p' - x) \geq x(p - x - 1) + p' > p'$ for $p \geq 2$ and $1 \leq x \leq p/2$. Thus, an optimal solution cannot partition the vertex set, and we are done. $\diamond$

4 Further Research

The splitting technique can prove useful for more (hyper)graph problems. For instance, transversal computation in hypergraphs of fixed rank can be translated into a one-sided domination problem in the bipartite incidence graph of the given hypergraph. (Vertices are the hypergraph nodes and hyperedges.) Then, vertices from hyperedges have degrees bounded by the rank, and vertices from nodes with high degrees can be "branched away" cheaply. Thus we get a bounded-degree graph on which we can apply the method. This is likely to yield faster FPT algorithms for this problem. Maybe this technique can systematically give only relatively small improvements of bases (because splitting requires quite some previous branching), but as we argued before, such improvements are worthwhile, too. The technique also gives some structural insights into the problems, as it exploits the "locality" of constraints. We remark that, at the moment, we cannot use it to further reduce the time for CLUSTER EDITING: We lack a chain structure as in Theorem 8 since we cannot branch on C_4. However, there might exist a (probably more complicated) similar structure that helps.

Acknowledgment

The second author has been supported by the Swedish Research Council (Vetenskapsrådet), grant 2007-6437, "Combinatorial inference algorithms – parameterization and clustering". He also thanks Leonid Molokov for inspiring discussions.

References

[1] R.G. Downey, M.R. Fellows, Parameterized Complexity, Springer, 1999.

[2] R. Niedermeier, Invitation to Fixed-Parameter Algorithms, Oxford Lecture Series in Math. and its Appl., Oxford Univ. Press, 2006.

[3] F. Dehne, M.A. Langston, X. Luo, S. Pitre, P. Shaw, Y. Zhang, The cluster editing problem: implementations and experiments, in: H.L. Bodlaender, M.A., Langston (Eds.),

Int. Workshop on Parameterized and Exact Comp., IWPEC 2006, LNCS 4169, Springer, Heidelberg, 2006, pp. 13–24.

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

[5] R. Shamir, R. Sharan, D. Tsur, Cluster graph modification problems, Discr. Appl. Math. 144 (2004) 173–182.

[6] T. Wittkop, D. Emig, S. Lange, S. Rahmann, M. Albrecht, J.H. Morris, S. Böcker, J. Stoye, J. Baumbach, Partitioning biological data with transitivity clustering, Nat. Methods 7 (2010) 419–420.

[7] F. Hüffner, C. Komusiewicz, H. Moser, R. Niedermeier, Fixed-parameter algorithms for cluster vertex deletion, Theory Computing Syst. 47 (2010) 196–217.

[8] P. Damaschke, Fixed-parameter enumerability of cluster editing and related problems, Theory Computing Syst. 46 (2010) 261–283.

[9] S. Böcker, S. Briesemeister, Q.B.A. Bui, A. Truss, Going weighted: Parameterized algorithms for cluster editing, Theor. Comput. Sci. 410 (2009) 5467–5480.

[10] J. Guo, A more effective linear kernelization for cluster editing, Theor. Comput. Sci. 410 (2009) 718–726.

[11] J. Chen, J. Meng, A $2k$ kernel for the cluster editing problem, in: M.T. Thai, S. Sahni (Eds.), Computing and Combinatorics Conf., COCOON 2010, LNCS 6196, Springer, Heidelberg, 2010, pp. 459–468.

[12] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier, Automated generation of search tree algorithms for hard graph modification problems, Algorithmica 39 (2004) 321–347.

[13] P. Damaschke, Bounded-degree techniques accelerate some parameterized graph algorithms, in: J. Chen, F.V. Fomin (Eds.), Int. Workshop on Parameterized and Exact Comp., IWPEC 2009, LNCS 5917, Springer, Heidelberg, 2009, pp. 98–109.

[14] B. Mannaa, Cluster editing problem for points on the real line: A polynomial time algorithm, Info. Proc. Letters 110 (2010) 961–965.

[15] A.V. Goldberg, S. Rao, Beyond the flow decomposition barrier, J. ACM 45 (1998) 783–797.