Fixed-Parameter Enumerability of Cluster Editing and Related Problems[*]

Peter Damaschke

Department of Computer Science and Engineering

Chalmers University, 41296 Göteborg, Sweden

`ptr@cs.chalmers.se`

Abstract

Cluster Editing is transforming a graph by at most k edge insertions or deletions into a disjoint union of cliques. This problem is fixed-parameter tractable (FPT). Here we compute concise enumerations of all minimal solutions in $O(2.27^k + k^2 n + m)$ time. Such enumerations support efficient inference procedures, but also the optimization of further objectives such as minimizing the number of clusters. In an extended problem version, target graphs may have a limited number of overlaps of cliques, measured by the number t of edges that remain when the twin vertices are merged. This problem is still in FPT, with respect to the combined parameter k and t. The result is based on a property of twin-free graphs. We also give FPT results for problem versions avoiding certain artificial clusterings. Furthermore, we prove that all solutions with minimal edit sequences differ on a so-called full kernel with at most $k^2/4 + O(k)$ vertices, that can be found in polynomial time. The size bound is tight. We also get a bound for the number of edges in the full kernel, which is optimal up to a (large) constant factor. Numerous open problems are mentioned.

Keywords: cluster graphs, soft clustering, fixed-parameter tractability, enumeration problems, true twins

[*]This is a merger of two conference articles that appeared in the Proceedings of the *31st International Workshop on Graph-Theoretic Concept in Computer Science WG 2005* (Metz) and the *6th International Conference on Algorithms and Complexity CIAC 2006* (Rome), *Lecture Notes in Computer Science* (Springer) vol. 3787, pp. 283–294 and vol. 3998, pp. 344–355, respectively. Compared to the preliminary versions, some results have been added or improved, and some proofs simplified.

1 Introduction

1.1 Cluster editing in graphs

A *cluster graph* is a disjoint union of cliques, or equivalently, a graph free of induced P_3 (paths of three vertices). An *edit step* in graph $G = (V, E)$ inserts or deletes an edge, where V remains fixed. We define $n := |V|$ and $m := |E|$. The Cluster Editing problem asks to turn $G = (V, E)$ by at most k edit steps into a cluster graph. In Cluster Deletion, only edge deletions are allowed. These problems arise in computational biology, e.g., in phylogeny reconstruction and classification of gene expression data [1, 2, 24, 25, 23]. In the latter application, edges join coregulated genes (vertices) belonging to the same functional group. Generally speaking, G describes pairwise similarities of items, and we seek a hidden clustering close to these empirical data. G may deviate from a cluster graph due to experimental errors, noisy data, or a non-transitive similarity relation.

In some applications, vertices may belong to several clusters, e.g., in text document clustering by subjects [8] and in biology: Genes may be involved in several functional groups. In such cases, disjoint clusters give artificial and meaningless classifications. Recent research addresses the identification of functional groups of proteins from known pairwise interactions. These groups are relatively few cliques that may overlap [22]. Since experiments yield many false negative and positive edges, some graph editing as postprocessing may be able to infer the real clusters. We will define a generalization of Cluster Editing that allows some limited amount of overlaps.

1.2 Parameterized enumeration

For both Cluster Editing and Cluster Deletion, computing the smallest possible k given G is NP-hard [1, 2, 15, 23], even if the number of clusters is prescribed [23]. Since usually $k \ll n$, the framework of fixed-parameter tractability (FPT) can be applied. Some efficient FPT algorithms for Cluster Editing and Cluster Deletion run in $O(2.27^k + n^3)$ and $O(1.77^k + n^3)$ time, respectively [9]. The Cluster Editing result has been further reduced to $O(1.92^k + n^3)$, using a computer program for search tree construction [10]. A problem kernel for Cluster Editing with $4k$ vertices is computable in $O(n^3)$ time [11].

In the present paper we seek an *enumeration* of all possible solutions for given G and k. A particular optimal solution is by no means guaranteed to explain the data properly [6, 3]. It is safer to consider *all* solutions for some parameter value k, and to judge them afterwards by other criteria specific to the application: Pairs of vertices may be in a cluster with different prior probabilities, which allows discrimination of solutions by some Bayesian inference rule or by informed decisions of an expert, e.g., a biologist who examines gene expression data using his knowledge about gene functions and plausible clusterings. An enumeration is a *version space* [17], i.e., the set of hypotheses consistent with the data and assumptions. In our case, hypotheses are clusterings, and the observed graph is assumed to be at most k edit steps away from the true clustering. The version space is used as a basis for inference. If all consistent solutions are equally likely *a priori*, probabilities are calculated by *counting*. For example, the probablity that two vertices u, v are in the same cluster is the number of clusterings with this property, divided by the size of the version space.

For counting and inference purposes, *concise enumerations* save space and time. A concise enumeration may state certain "simple" parts of the solution space by set-theoretic expressions, thus making the descriptions much smaller than linear in the number of solutions. Concise enumeration is not a formal technical term, but a general idea. *Ad hoc* definitions saying which expressions are allowed can be introduced for any problem. Earlier we proposed one for the Vertex Cover problem [3]. In the present paper we give FPT algorithms that generate a concise enumeration of all solutions to Cluster Editing in a given graph.

Concise enumerations are also useful algorithmically, for solving multicriteria optimization problems that are in FPT in some parameter: In our case, we study Cluster Editing with the extra demand to minimize the number of clusters. For optimizing a further objective function like this, we only need one optimal solution from every "simple" part of the solution space. Provided that we can get them in polynomial time, the upper bound for the concise enumeration translates into an upper bound for the FPT optimization problem. This scheme has been discovered independently and used for improved FPT algorithms for vertex covers with additional demands [18, 19]. (Actually, we had an earlier result about concise enumerations of vertex covers [3], but with a larger base because we considered repetition-free enumerations.) Other enumeration-based FPT algorithms are known for edge dominating sets [7] and feedback vertex sets [12].

We assume basic knowledge about the analysis of FPT algorithms working with reduction rules, branching rules, and search trees, and familiarity with the notions of branching vector, characteristic equation, and branching number [5].

In general, we cannot simply translate an optimization algorithm into an enumeration or counting algorithm with the same complexity, for several reasons: FPT optimization algorithms may discard branches that cannot lead to optimal solutions anymore. Such reductions are no longer possible in search trees for enumeration problems. Every branching has to be exhaustive. Moreover, branching can stop when the residual problem instance belongs to a polynomial-time solvable special case, but counting problems can still be hard albeit the corresponding optimization problem is easy. Yet another issue is that branchings should be disjoint, so that the enumerations are repetition-free, i.e., no solution occurs twice. Then, we can sum up the numbers of solution represented by all leaves. In conclusion, even though many known branching rules can be recycled, we have to develop enumeration and counting algorithms from scratch.

In graph modification problems like Cluster Editing, a simple way to make branching rules disjoint is blocking. Selected items (here: vertex pairs) not edited in the current branching are *blocked*, which prohibits later editing. All edited items are blocked as well, since forth-and-back changes are useless. With carefully chosen blockings, no solution gets lost, and different branches lead to mutually distinct solutions. Later applications of branching rules that involve already blocked items are simply disabled.

We introduce a notion of problem kernels for FPT enumeration problems, called *full kernels* [3], that "include" all possible minimal solutions, rather than only some optimal solution. Disagreements between any two solutions are restricted to a small kernel. For example, for Cluster Editing we will show that any two clusterings reachable by k edit steps differ only on edges in the full kernel, and in small clusters whose sizes depend on k only. (We will also prove an optimal bound for the full kernel size.) Then, for any two vertices outside these small sets we can safely conclude whether they are in one cluster or not, provided that our k is at least the true edit distance. Full kernels are of twofold use: They contain all possible ambiguities between solutions, and enumeration and counting problems can be reduced to them.

1.3 Graph-theoretic definitions and preliminaries

A *clique* is a complete subgraph of $G = (V, E)$, not necessarily maximal. $G - X$ is graph G with $X \subseteq V$ and all incident edges removed. If $X = \{x\}$, we write $G - x$. For an induced subgraph H of G, and a vertex $w \notin V(H)$, we denote by $H + w$ the subgraph induced by $V(H) \cup \{w\}$. The open neighborhood $N(X)$ of $X \subset V$ is the set of vertices being not in X but incident to some vertex of X. If $X = \{x\}$, we write $N(x)$. Vertex x is *isolated* if $N(x) = \emptyset$. Two sets of vertices that share at most one vertex are *pair-disjoint*. $M \subseteq V$ is a *module* if every vertex outside M is adjacent to no or all vertices in M. Symbols P_n, C_n, K_n denote a chordless path, chordless cycle, and complete graph, respectively, of n vertices. Star graph $K_{1,s}$ has one central vertex adjacent to s other vertices, and no further edges. A (connected) *component* of G is an inclusion-maximal connected subgraph.

Let G' and G'' be any two solutions to an instance G, k of Cluster Editing. G'' *contains* G' if the edit steps leading from G to G' form a subset of those leading from G to G''. (The notion should not be confused with containment of the graphs.) A solution not containing any other solution is *minimal*. The Cluster Editing enumeration problem can be split in two parts: The nontrivial part is to enumerate *all minimal* solutions G'. Once we know them, it is pretty easy to characterize *all* solutions G'' reachable from G by at most k edit steps: The only way to obtain further solutions G'' from a minimal solution G' is to divide or merge clusters of G', by at most $k-d$ edit steps, where d is the edit distance of G' and G. Note that these changes can affect only clusters of size at most $k-d$, in any G'. Hence, if k is close to the optimal number of edit steps, these changes are very limited. From now on we consider the nontrivial part only:

Cluster Editing (enumeration version)
Given G and k, enumerate all minimal sets of at most k edit steps that transform G into a cluster graph. Let k_0 denote the minimum number of edit steps in any solution.

Note that an enumeration of all minimal solutions in a search tree may be "polluted" by some non-minimal solutions, even if the enumeration is repetition-free. Some edit steps may turn out to be redundant only later. But, these non-mimimal solutions can be recognized directly.

We extend the Cluster Editing problem to tolerate some overlapping cliques. An obvious idea is to control the number of intersections of maximal cliques in the target graph by a second parameter. Below we propose a parameterization based on the notion of twins. Vertices u, v in G are *(true) twins* if uv is an edge and $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. The true-twin relation is symmetric and transitive, thus we get equivalence classes of true twins, which are both cliques and modules. The *twin graph* $T(G)$ has one vertex for each equivalence class of twins in G, and joins two vertices by an edge iff edges exist between the corresponding equivalence classes in G. Note that $T(G)$ is isomorphic to any induced subgraph of G with of one representative from each equivalence class. Twin graphs are also known as critical clique graphs [16]. *False twins* u, v are defined similarly, but uv is not an edge. The false-twin relation gives rise to equivalence classes, too.

Twin Graph Editing
Given a graph G and parameters k and t, can we obtain by at most k edit steps a graph whose twin graph has at most t edges?

Cluster Editing is the special case when $t = 0$. Alternative parameters would be the number of non-isolated vertices in $T(G)$, which is polynomially equivalent to t, or the number c of maximal cliques involved in overlaps. But since c overlapping cliques can generate at most

2^c equivalence classes of true twins being non-isolated in $T(G)$, these parameters are already captured by our smoother parameter t.

The problem also arises naturally in computational biology: In an evolution model for gene interaction networks [26], gene duplications produce twins. Thus, Twin Graph Editing may be used to estimate the rate of creation or loss of single interactions, compared to the duplication rate. Overlapping clusters are also of interest in the analysis of gene expression experiments [4].

A graph G without true twins ($T(G) = G$) is *twin-free*. Vertex x is a *discriminator* of edge yz if exactly one of the edges xy, xz exists. Two adjacent vertices are no true twins iff they have a discriminator. Predicate $D(x, y, z)$ means that x is the *only* discriminator of edge yz, and xy is an edge. Note that $D(x, y, z)$ excludes the existence of some v with $D(v, z, y)$.

Vertices u, v form an *ambiguous* pair if uv is an edge in some but not all minimal solutions to Cluster Editing (given G, k). Ambiguous vertices are vertices in ambiguous pairs. A *full kernel* is a set of vertices containing all ambiguous vertices.

1.4 Overview of results

In Section 2 we give an algorithm with branching number 2.27 that computes some concise enumeration of all minimal solutions to Cluster Editing. In Section 3 we prove that Twin Graph Editing is in FPT, with combined parameters k, t. Membership in FPT is no longer trivial. We need a lemma that provides some elimination order for twin-free graphs and a branching rule. We achieve a branching number $O(t)$.

In Section 4 we study problem variants where also the total number c of clusters shall be minimized. Clearly, c is monotone decreasing in k, and c can properly decrease: For instance, a P_4 can be split in two clusters by one deletion, but also be completed to one clique by three insertions. Cluster Editing with at most k edit steps and the smallest possible number of clusters remains in FPT. By exploiting our concise enumerations we can solve the problems within the same time bounds.

However, just minimizing the number of clusters can force unmotivated merging of small clusters. This undesirable effect is avoided by our proposed "natural clusterings" that satisfy two modest requirements. They are equivalent to the minimal solutions of Cluster Editing. It follows that natural clusterings with a minimum number of clusters can be found within the same time bounds as before.

In Section 5 we prove for Cluster Editing that a full kernel with roughly $k^2/4$ vertices is computable in $O(k^2 n + m)$ time, and $1/4$ is the optimal constant factor. For the number of ambiguous pairs of vertices we get the bound $\Theta(k^4)$, however with a coarse estimate of the constant factor. Section 6 points out some further research directions.

2 Cluster Editing: Enumerating the Minimal Solutions

A trivial branching algorithm for Cluster Editing (enumeration version) would choose some induced P_3, and delete one of the two edges or insert the missing edge. The branching number is obviously 3, and $\Omega(3^k)$ time is needed in the worst case to enumerate all minimal solutions. For instance, the disjoint union of k copies of P_3 has 3^k solutions. However, this solution space can be simply described as the Cartesian product of k sets of three vertex pairs. In this section we give a faster algorithm that generates a concise enumeration of minimal solutions for any G.

Lemma 2.1 *G contains induced K_3 only in components which are cliques, or a branching rule is available with branching number 2.27.*

Proof. Consider a K_3 in G with vertices u, v, w, and another vertex z. If z is adjacent to exactly one vertex of the K_3, say zu is an edge, we branch as follows. Either we remove edge zu (1 edit step), or we block it. In the latter case, either none of v, w, or only v, or only w, or v and w are put in the same cluster as z, u. This results in the branching vector (1,2,3,3,2). The characteristic equation $x^3 = x^2 + 2x + 2$ yields branching number 2.27. If z is adjacent to exactly two vertices of the K_3, say zu and zv are edges, we branch as follows. Either we insert edge zw (1 edit step), or we block this pair. In the latter case, z and w belong to different clusters. If we block edge uv, then both edges connecting u, v to either z or w must be deleted. These are two branches with 2 edit steps. If we delete edge uv, we obtain a C_4 and are forced to delete another 2 non-incident edges. This can be done in two ways, giving two branches with 3 edit steps. Thus we obtain the same branching vector. □

Lemma 2.2 *G contains no induced $K_{1,4}$, or a branching rule is available with branching number 2.17.*

Proof. Consider an induced subgraph $K_{1,4}$ with central vertex u and leaves v, r, s, t. We branch as follows. Either we delete edge uv (1 edit step) or we block it. In the latter case we decide which of r, s, t shall belong to the same cluster as u, v. In each of the eight cases we have to insert or delete exactly 3 edges between r, s, t on one side and u, v on the other side. If exactly two of r, s, t join the cluster (three different branches), we must insert a 4th edge, namely one of rs, rt, st. If all of r, s, t join the cluster, we must insert all 3 edges rs, rt, st, altogether these are 6 edit steps. This guarantees the branching vector (1,3,3,3,3,4,4,4,6). The characteristic equation $x^6 = x^5 + 4x^3 + 3x^2 + 1$ yields $x < 2.17$. □

By Lemma 2.1 and 2.2, we can assume in the following that our graphs have no induced K_3 and $K_{1,4}$. (Note that edit steps affecting vertices from isolated cliques are redundant.) In particular, no vertex of degree 4 or more can appear.

Lemma 2.3 *A connected graph without induced K_3, where every edge belongs to at most two P_3, is P_n, C_n, or $K_{1,3}$.*

Proof. The only connected graphs with maximum vertex degree 2 are P_n and C_n. Now let u be a vertex of degree 3. With its neighbors it forms an induced $K_{1,3}$. Since any edge uv of this $K_{1,3}$ is already in two P_3, any further neighbor of v must be adjacent to u, but this gives a K_3, contradicting the assumption that G has no K_3. It follows that no vertex of our $K_{1,3}$ can possess further neighbors. $\Box$

By the preceding lemmas, we can assume in the following that our graphs have no induced K_3 and $K_{1,4}$, and some edge uv belongs to three different P_3. Let r, s, t be three vertices, each building a P_3 together with u, v. Not all of r, s, t are adjacent to u (this would give an induced $K_{1,4}$ or K_3). Hence u is adjacent to exactly r and s, while v is adjacent to exactly t. (Other cases are identical up to renaming.) Note that rs is not an edge, whereas rt, st may be edges or not.

Now we branch as follows. Either we delete edge uv (1 edit step), or we block it. In the latter case, we must insert or delete one edge in each of the three P_3 that include edge uv. This yields eight branches with at least 3 edit steps each. On top of that we study how many edges need to be edited between r, s, t. The rows of the table indicate which edges exist (the case that only st exists is symmetric and therefore omitted), the columns indicate which of r, s, t shall be in the same cluster C as u, v. We count the missing edges in C that must be inserted, and the edges between C and the rest that must be deleted.

	-	r	s	t	r,s	r,t	s,t	r,s,t
-	0	0	0	0	1	1	1	3
rt	0	1	0	1	2	0	2	2
rt,st	0	1	1	2	3	1	1	1

Together with the 3 edit steps between u, v and r, s, t and the single edit step from the branch where uv is deleted, we obtain branching vectors (1,3,3,3,3,4,4,4,6) and (1,3,3,3,4,4,5,5,5) in the first two cases, and the third case is superior to the first. The characteristic equations $x^6 = x^5 + 4x^3 + 3x^2 + 1$ and $x^5 = x^4 + 3x^2 + 2x + 3$ give branching numbers 2.17 and 2.08, respectively.

Our worst branching number was 2.27 from Lemma 2.1. Now we get a concise enumeration of all minimal solutions to CLUSTER EDITING by applying the branching rules repeatedly in any order. Every leaf of the resulting search tree represents a graph whose components are cliques (which need not be edited further), chordless paths and cycles or $K_{1,3}$.

Given the "residual" k for the leaf, these paths and cycles must be of length $O(k)$, otherwise this leaf is a dead end, without solution. Characterizing all minimal solutions in chordless paths and cycles is trivial, due to their highly regular structure. For the other components we can explicitly list their constantly many minimal solutions. Finally we take the disjoint unions of edit steps in the components, observing the allowed total number of these remaining edit steps. Details are straightforward.

Prior to branching we can compute a full kernel with $O(k^2)$ vertices in $O(k^2n + m)$ time (as we will show in Section 5). This gives:

Theorem 2.4 *A concise enumeration of all minimal solutions of* Cluster Editing *is computable in* $O(2.27^k + k^2n + m)$ *time.* $\square$

Since each of our branching rules is disjoint, and the number of solutions in chordless paths and cycles may be expressed by closed formulae, we also get a counting algorithm of the same complexity. A result similar to Theorem 2.4, but with base 1.47, exists for Cluster Deletion [20].

3 Twin Graph Editing is in FPT

3.1 The basic scheme of the algorithm

Let G, k, t be an instance of Twin Graph Editing. We seek an induced subgraph H, with s size limited by some function of the parameters, and where at least one edit step is forced. Then we can branch on the possible edit steps in H. Thus we refer to H as the *branching graph.*

Lemma 3.1 *If H is a twin-free induced subgraph of G with more than t edges, then any solution to* Twin Graph Editing *must insert or delete an edge in H.*

Proof. Since H is twin-free, every edge in H has a discriminator in H. If we do not edit H, it retains this property in the edited graph G', thus we get no true twins in H. Hence, $T(G')$ still contains $T(H)$ as induced subgraph and has therefore more than t edges, contradicting the specification of the Twin Graph Editing problem. $\square$

While this choice of a branching graph is fairly obvious, the difficulty is to find a small enough branching graph efficiently, without exhaustive search in G. We accomplish this by an elimination process based on:

Lemma 3.2 *A twin-free graph H always has a vertex r such that $D(r, s, t)$ does not hold for any edge st in H. We call r a* non-critical *vertex.*

We defer the proof to the next subsection. Due to Lemma 3.2 we can remove r and all incident edges from H, and no remaining edge will lose all its discriminators, hence $H - r$ is twin-free again. Now we can prove:

Theorem 3.3 TWIN GRAPH EDITING *is fixed-parameter tractable in combined parameters k and t.*

Proof. Given G, we first compute for every edge of G the list of discriminators, trivially in $O(mn)$ time. Now we easily obtain a (twin-free!) induced subgraph H of G isomorphic to $T(G)$ in $O(m)$ time: Edges with empty discriminator lists build disjoint cliques, and from every such clique we keep one vertex and remove the others and all their incident edges. (Actually, linear time would be sufficient for this part [13], but this does not help the overall time bound.)

If H has at most t edges, we are done. Otherwise we remove a non-critical r from H (see Lemma 3.2) and also all isolated vertices in $H-r$. Since $H-r$ is still twin-free, so is the smaller induced subgraph. Thus we can continue the process and never get stuck, and in every step we can select an *arbitrary* vertex r which does not occur alone in any discriminator list. Non-critical vertices are held in a separate set NC. In every step we update the data structure by removing the edges incident with r, their discriminator lists, all occurrences of r in the lists of other edges, and the isolated vertices. From NC we remove r and the vertices that grew lonely in a list. All this is done in $O(mn)$ time in total.

We stop as soon as an induced subgraph H' has at most t edges, whereas the previous H had more than t edges. This H is our branching graph. Clearly, H' has at most $2t$ vertices. The vertices of H that became isolated in this step are all adjacent to r, since they were not isolated in H. Together with r they induce a star graph which is twin-free. If more than t such vertices exist, we obviously get a twin-free induced subgraph with only $t+2$ vertices. Otherwise H has at most $3t+1$ vertices. We have to edit one of the at most $3t(3t+1)/2$ vertex pairs in H, which gives an $O(t^2)$ bound on the branching number. $\square$

3.2 Twin-free graphs have non-critical vertices

Lemma 3.4 *Suppose that $D(x,v,u)$ holds. Then:*
(i) $D(v,t,y)$ implies $t=x$, and (y,x,v,u) is an induced P_4.
(ii) $D(u,t,y)$ implies $y=x$, and t,v are identical or adjacent.

Proof. First assume that $\{x,v,u\} \cap \{t,y\} = \emptyset$. Since v and u are adjacent to exactly the same vertices except x, either of $D(v,t,y)$ and $D(u,t,y)$ implies the other relation, contradicting the definition of D. Thus we have $t=x$ or $t=u$ in (i). But if $D(v,u,y)$ then y is another discriminator of vu. There remains $t=x$, hence $D(v,x,y)$. If yu is an edge then y is again another discriminator of vu. In (ii) it must be $t=v$ or $y=x$. But if $t=v$ then y is another discriminator of vu, unless $y=x$. Finally, if t,v are distinct but not adjacent then t is another discriminator of vu. $\square$

Using this building block we will now prove Lemma 3.2. First we give the idea. Consider any vertex x. If x is critical then $D(x, v, u)$ for some edge vu. If both v and u are critical, there must exist edges they are unique discriminators of, and so on. The difficulty is that such edges can in fact be established, and they can involve new vertices. (Hence the construction shows by itself that the argument cannot be simplified.) On the other hand, Lemma 3.4 imposes strong enough restrictions that enforce some repeated pattern. Since the graph is finite, we must abort the construction at some point and are then left with some non-critical vertex. Now the detailed exposition follows.

Consider a graph H where each vertex is the unique discriminator of some edge. We show by induction that H must contain pairwise distinct vertices u_j, v_j, x_j, y_j for all $j \geq 1$ that have certain properties listed below. It follows that such a finite H cannot exist. Let W_j be the set of all u_i, v_i, x_i, y_i with $i \leq j$, in particular $W_0 = \emptyset$. The properties are the following.
(1) (u_j, v_j, x_j, y_j) in this order form an induced P_4.
(2) v_j, x_j are adjacent to all vertices in W_{j-1}.
(3) u_j is adjacent to all u_i, v_i in W_{j-1}, but not to the x_i, y_i. Similarly, y_j is adjacent to all x_i, y_i in W_{j-1}, but not to the u_i, v_i.
(4) $D(v_j, s, y_j)$ holds for some $s \in W_j$. Similarly, $D(x_j, s, u_j)$ holds for another $s \in W_j$.
(5) All u_i, v_i with $i \leq j$ have the same neighbors outside W_j. Similarly, all x_i, y_i with $i \leq j$ have the same neighbors outside W_j.

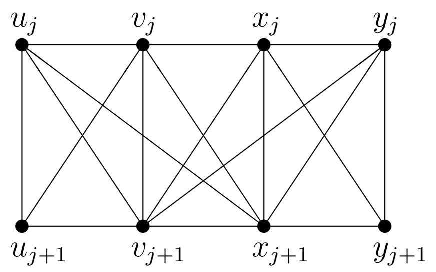

Figure 1. Two consecutive layers of the graph used in the proof.

Induction base ($j = 1$): Let x_1 be any vertex. By assumption on H there exist u_1, v_1 so that $D(x_1, v_1, u_1)$, and v_1 is also the only discriminator of some edge. Lemma 3.4 (i) allows only $D(v_1, x_1, y_1)$ for some new vertex y_1 and also implies (1) and (4) for $j = 1$. Conditions (2),(3) are vacuously true, and (5) follows from $D(x_1, v_1, u_1)$ and $D(v_1, x_1, y_1)$.

Induction step: Suppose that (1)–(5) are true for some j. By assumption on H, u_j is the only discriminator of some edge, and (4) for j says that $D(x_j, s, u_j)$ for some $s \in W_j$. Lemma 3.4 (ii) yields $D(u_j, t, x_j)$ for some t identical or adjacent to s. Case $t = v_j$ is impossible, since by (1), $v_j x_j$ has already another discriminator y_j, contradicting $D(u_j, v_j, x_j)$. If $t \in W_{j-1}$ then, by (3), t is one of the u_i or v_i. But then, due to (1),(2), y_i

with the same i is another discriminator of tx_j, contradicting $D(u_j, t, x_j)$. This shows $t \notin W_j$. Define $x_{j+1} := t$. Since x_{j+1} has neighbors u_j and x_j, property (5) yields that x_{j+1} is adjacent to all of W_j. Thus we have shown (2) for x_{j+1}. By symmetry there also exists v_{j+1} satisfying (2). Recall that $D(u_j, x_{j+1}, x_j)$ and, symmetrically, $D(y_j, v_{j+1}, v_j)$. Assume $v_{j+1} = x_{j+1}$. Then we also have $D(y_j, x_{j+1}, v_j)$. By Lemma 3.4 (i), any edge with unique discriminator x_{j+1} contains both u_j and y_j, which is obviously impossible. Thus $v_{j+1} \neq x_{j+1}$. Furthermore, these two vertices are adjacent, since otherwise v_{j+1} would be another discriminator of $x_j x_{j+1}$, besides u_j. (Note that $v_{j+1}x_j$ is an edge by (2).)

Applying Lemma 3.4 (i) again to $D(y_j, v_{j+1}, v_j)$, we see $D(v_{j+1}, y_j, z)$ for some z being not adjacent to v_{j+1}, hence $z \notin W_j$ due to (2). Since (5) holds for j, and v_j, v_{j+1} have the same neighbors except y_j, vertex z is adjacent to all the x_i, y_i in W_j, but to none of the u_i, v_i in W_j. Defining $y_{j+1} := z$ we get $D(v_{j+1}, y_j, y_{j+1})$. Altogether, y_{j+1} satisfies (3), and also (4) holds, with $j + 1$ in the role of j, and y_j is the role of s. By symmetry we also get a vertex u_{j+1} which satisfies the other half of (3) and (4), respectively. In particular we have $D(x_{j+1}, u_j, u_{j+1})$. Since the new vertices u_{j+1} and y_{j+1} have distinct neighborhoods in W_j, they are distinct. Moreover they are not adjacent, otherwise u_{j+1} would be another discriminator of $y_j y_{j+1}$. Edges $u_{j+1}v_{j+1}$ and $x_{j+1}y_{j+1}$ do exist, since without them, v_{j+1} and x_{j+1} would be another discriminator of $u_j u_{j+1}$ and $y_j y_{j+1}$, respectively. This shows (1) for $j + 1$.

Finally we recover (5) for $j + 1$. By the induction hypothesis and $D(x_{j+1}, u_j, u_{j+1})$, $D(y_j, v_{j+1}, v_j)$, both u_{j+1} and v_{j+1} have outside W_{j+1} the same neighbors as all other u_i, v_i have. Once more, the argument holds symmetrically for x_{j+1} and y_{j+1}. This completes the induction step and the proof.

3.3 Making the base linear in parameter t

Theorem 3.3 established that Twin Graph Editing is in FPT. Finally we improve the base to $O(t)$. Note that $t \geq 2$ in the following. Case $t = 0$ is Cluster Editing, and $t = 1$ is meaningless.

Lemma 3.5 *If, in a twin-free graph other than P_3, several vertices of degree 1 have the same neighbor, then either of them is non-critical.*

Proof. Let x, x' be any two vertices of degree 1 with common neighbor v. We remove x. If x was a discriminator of some edge other than vx', then x' is still discriminator of that edge. Since our graph is not P_3, v has yet another neighbor, which is a discriminator of vx'. Hence no edge loses its unique discriminator. □

Lemma 3.6 *Suppose that $t \geq 3$, and H a minimal twin-free graph with more than t edges (minimal in the sense that removing any non-critical* *vertex would leave at most t edges). Then H has at most $\frac{3}{2}t+2$ vertices and $\frac{5}{2}t$ edges. For $t=2$, H has at most 5 vertices and 6 edges.*

Proof. Lemma 3.2 says that a non-critical r exists in H. Since $H-r$ is twin-free, no edge in $H-r$ is isolated. Hence the t edges in $H-r$ can span at most $\frac{3}{2}t$ vertices. (The worst case is a union of disjoint copies of P_3.) Note that $H-r$ may contain further, isolated vertices, i.e., vertices with the only neighbor r in H. Since H has at least three edges, H is not P_3, and Lemma 3.5 applies. If several vertices of degree 1 are adjacent to r, any of them is non-critical. Let x be one of them. In this case H has exactly $t+1$ edges. Moreover, since the observation above holds for any non-critical vertex and $H-x$ has no isolated vertices, H has at most $\frac{3}{2}t+1$ vertices. The other case is that at most one vertex x exists with r as the only neighbor. Then H has at most $\frac{3}{2}t+2$ vertices.

Suppose there is at least one neighbor x of r with degree 1. If x is non-critical, we are back to the previous case. Otherwise there exist u,v such that $D(x,v,u)$. Clearly, $v=r$. Recall that u,v have the same neighbors apart from x. Furthermore, since $H-v$ has at most t edges, the degree of u is bounded by $t+1$, which limits the degree of v to $t+2$. Since uv is an edge, it follows that H has at most $2t+2$ edges.

Finally, if r has no neighbors of degree 1 in H, then the edge number of H is maximized if $H-r$ has $\frac{3}{2}t$ vertices, all adjacent to r. This gives $t+\frac{3}{2}t=\frac{5}{2}t$ edges. $\Box$

Theorem 3.7 TWIN GRAPH EDITING *can be solved in $O(mn\cdot b^k)$ time, where $b=6.12t+O(\sqrt{t})$.*

Proof. The elimination process in our FPT algorithm (Theorem 3.3) provides a branching graph with size bounds as in Lemma 3.6, in $O(mn)$ time. In H, either we remove one of the at most $\frac{5}{2}t$ edges, or we block all edges of H and insert new edges. In the following we discuss the latter case. Our edge insertions must destroy twin-free induced subgraphs with more than t edges. Since the edge number in H can only grow, our insertions must produce at least one new pair u,v of twins. Some u,v have to become twins in the entire graph G, not only in H. (Otherwise all vertices in H would, in the final $T(G)$, still represent pairwise distinct equivalence classes of twins, so that a twin-free induced subgraph with more than t edges remains.) In the following we branch on H by choosing the pair u,v and doing the enforced edit steps.

We have at most $\frac{5}{2}t$ choices of vertices u,v that were already adjacent in H. Since u,v are not twins in H, some x discriminates uv, thus we must insert the unique missing edge between x and u or v.

It remains the case that u,v were not adjacent. Since H has at most $\frac{3}{2}t+2$ vertices, no more than $\frac{9}{8}t^2+\frac{9}{4}t+1$ such pairs u,v exist. To achieve branching number $O(t)$ we have to identify *two* edges that *must* be inserted to make u, v twins. One is obviously uv. If u, v have different neighborhoods in H, we are also forced to insert one more edge in H. Suppose that u, v have the same neighbors in H. But if some x outside H discriminates the new edge uv, we must either insert or delete an edge between x and u or v, in order to make u, v twins in G. The case that u, v are false twins is treated differently. For branching, it suffices to choose from each equivalence class of false twins only one pair u, v and to insert edge uv, because all other choices are isomorphic. In the worst case, H consists of $\frac{3}{4}t + 1$ classes of two false twins.

Reviewing all these cases we find that one edit step is enforced in at most $\frac{5}{2}t + \frac{5}{2}t + \frac{3}{4}t + 1 = \frac{23}{4}t + 1$ different ways, and two edit steps in at most $\frac{9}{4}t^2 + \frac{9}{2}t + 2$ different ways. Solving the characteristic equation gives the claimed branching number. □

The current bound is practical only for very small t, but it seems possible to improve the factor 6.12, and to derive smaller bases for $t = 2, 3, 4, \ldots$

4 Minimizing the Number of Clusters

4.1 Unrestricted clusterings

In Twin Graph Editing we measured the complexity of the target graph by the number of *edges* in its twin graph. Alternatively we may aim at a minimum number of *vertices* in the twin graph. We define the Twin Graph Editing (V) problem accordingly: Given G and k, t, can we obtain by at most k edit steps some G' so that $T(G')$ has at most t vertices? We can see that Twin Graph Editing (V) is in FPT, with combined parameters k, t, in very much the same way as in Theorem 3.3. Our branching graph H has exactly $t + 1$ vertices; we do not need a complicated argument to bound the size of H. This implies the branching number $t(t + 1)/2$. However, we could not achieve an $O(t)$ base, since the number of edges in $T(G)$ can be quadratic in t.

We define the problem Cluster Editing (Min): Given G and k, turn G by most k edit steps into a cluster graph with minimum number of clusters. Cluster Editing (Min) is in FPT in parameter k alone, as there is a trivial enumeration algorithm with branching number 3. In order to improve the branching number, we use our concise enumerations. The following lemma is just Theorem 2.4 rephrased, but we have to stress some technical details needed in the following.

Lemma 4.1 *A concise enumeration of all minimal solutions to an instance of* Cluster Editing *can be computed by a search tree algorithm that uses only branching rules with branching numbers no larger than*

2.27. All leaves of the resulting search tree represent graphs which may contain, besides cliques, also $K_{1,3}$ and chordless paths and cycles as components. All minimal solutions are reached by further, independent edit steps inside these non-clique components.

For solving Cluster Editing (Min) we compute a concise enumeration as in Lemma 4.1 and then further expand the paths of the search tree, starting from the current leaves. For every leaf it suffices to find some optimal solution below this leaf. Finally we take the best. In the following we consider any leaf. Let L denote the graph represented by this leaf, and k the number of edit steps still allowed for L (not to confuse with the initial k).

If L is already a cluster graph, we may further reduce the number of clusters by at most k further edit steps. The following lemma restricts the edit steps we have to take into account. The lemma is not surprising, we omit its simple proof.

Lemma 4.2 *Let L be a cluster graph that can be transformed by at most k edit steps into a cluster graph H with at most c clusters. Then we can do the transformation in such a way that clusters of L are only merged but never split.* $\square$

Due to Lemma 4.2, if L leads to an optimal solution at all, we can obtain some by successive merging of clusters. Moreover, we have:

Lemma 4.3 *In every merging step we can take a cluster C of currently minimum size and merge it with one of the other clusters.*

Proof. If C participates in further merge operations at all, we can merge C *right now* with another cluster. If not, then C becomes a separate cluster in the target graph H. But then we can switch the roles of C and some larger cluster, this only reduces the number of edge insertions. Hence it is safe to merge a smallest cluster C with some other cluster. $\square$

However, the greedy strategy that successively merges two smallest clusters fails. Counterexamples are easy to find. Instead we use a branching rule based on Lemma 4.3: Merge a smallest cluster C with another cluster. Every possible cardinality of the merging partner gives one branch. If $|C| = x$, and the different sizes of the other clusters are $x_1 < x_2 \cdots < x_d$, we get the branching vector $(xx_1, xx_2, \ldots, xx_d)$. Thus, the inverse y of the branching number fulfills the characteristic equation $\sum_{i=1}^{d} y^{xx_i} = 1$. For any d, the worst case (y minimal) appears if $x = 1$ and $x_i = i$ for all i, that is, $\sum_{i=1}^{d} y^i = 1$. Now we see $y > 1/2$, hence the branching number is always smaller than 2. (As a side remark, we do not know whether CLUSTER EDITING (MIN) is polynomial or NP-hard on cluster graphs.)

It remains to consider the components in L which are not cliques, see Lemma 4.1. Any transformation of L into a cluster graph can be rearranged so that we first reach a cluster graph through a minimal edit sequence (and then perhaps reduce the number of clusters). But we get all minimal solutions by turning the components independently into cluster graphs. It suffices to have one branch for every multiset of cardinalities. A $K_{1,3}$ can be divided into clusters of size 4, 3+1, 2+2, 2+1+1, or 1+1+1+1. The branching vector indicating the necessary number of edit steps is $(3, 2, 3, 2, 3)$, which gives a branching number below 2. The remaining components in L are chordless paths or cycles. In a P_3 component we need to consider only two branches, leading to clusters of size 3 or 2+1. The branching vector is $(1, 1)$, hence the branching number is 2. (Case 1+1+1 is not a minimal solution.) On a component P_s with $s \geq 4$ we branch as follows. Either we split P_s into s singleton clusters by $s-1$ deletions, or we decide on the size i, $2 \leq i < s$, of some subpath that we cut off and complete to a cluster, or we make the entire component a cluster. The middle case needs one deletion and $\binom{i}{2} - (i-1) = \binom{i-1}{2}$ insertions, and the last case needs $\binom{s-1}{2}$ insertions. Hence the branching vector is $(s-1, \binom{1}{2}+1, \binom{2}{2}+1, \binom{3}{2}+1, \ldots, \binom{s-2}{2}+1, \binom{s-1}{2})$. This vector is almost sorted, and if we put $s-1$ at the proper place in the increasing sequence, we easily see that our branching vector dominates $(1, 2, 3, \ldots, s)$ which yields a branching number below 2. For C_s we apply the same branching rule, but we need an extra deletion to get s singleton clusters or to split off the first subpath, and one insertion less if the whole cycle becomes a cluster. Hence, the branching vector becomes $(s, \binom{1}{2}+2, \binom{2}{2}+2, \binom{3}{2}+2, \ldots, \binom{s-2}{2}+2, \binom{s-1}{2}-1)$. Using the same argument, the branching number is smaller than 2, for every s. It follows:

Theorem 4.4 CLUSTER EDITING (MIN) *can be solved in* $O(2.27^k + k^2 n + m)$ *time.*

Proof. Prior to branching we can compute a full kernel with $O(k^2)$ vertices in $O(k^2 n + m)$ time (as we will show in Section 5). Furthermore, the worst branching number was 2.27. □

4.2 Natural clusterings

Clusterings with a forced small number of clusters tend to merge small clusters just because this is "cheap" in terms of edge insertions, but they need not be "really" related. Thus one may argue that Cluster Editing (Min) is somewhat ill-posed. To overcome this effect, we propose to prohibit unmotivated mergings and splittings as follows:

Definition 4.5 *Given a graph $G = (V, E)$, a cluster graph H on vertex set V is a* natural clustering *of G if:*
(1) The vertices of each cluster in H induce a connected subgraph of G.
(2) For any two clusters C and C' in H, not all possible edges between C and C' exist in E.

We define the problem Cluster Editing (Min,Nat): Given G and k, turn G into a natural clustering of G with a minimum number of clusters, by at most k edit steps (if possible). Neatly, the following equivalence holds:

Theorem 4.6 *The natural clusterings of a graph are exactly the minimal solutions to* Cluster Editing.

Proof. Let H be a natural clustering of G, and τ the set of edit steps turning G into H. We claim that any $\sigma \subset \tau$ cannot lead to a cluster graph, hence H is a minimal solution. If σ omits some of the edge deletions in τ, then at least two clusters of H, say C and C', fall into the same component of G. Since σ cannot delete more edges than τ, it follows that C and C' remain connected after σ. But then σ must insert all missing edges between C and C' (due to (2) there are some), whereas τ does not insert such edges. This contradicts $\sigma \subset \tau$. We conclude that all edges deleted by τ are deleted by σ, too. After the deletions, σ must complete the components to cliques. Hence σ must insert all missing edges in the clusters of H (unless some cluster of H is not a connected subgraph of G, which is excluded by (1)). This implies $\sigma = \tau$, a contradiction. The claim is proved.

Conversely, let H be a minimal solution to Cluster Editing, and τ defined as above. By minimality, any $\sigma \subset \tau$ cannot produce a cluster graph. Assume that (1) is violated, i.e., some cluster C of H is a disconnected subgraph of G. If we fail to insert the edges between the components of C, we get a proper subset of τ generating a cluster graph (with C divided in several clusters), a contradiction. Assume that (2) is violated, so that E contains all possible edges between two of the clusters, say C and C'. Since C and C' are clusters in H, τ has deleted all these edges. If we do not delete them, we get a proper subset of τ producing a cluster graph (with cluster $C \cup C'$ rather than C and C'), a contradiction. $\Box$

Theorem 4.7 Cluster Editing (Min,Nat) *can be solved in $O(2.27^k + k^2 n + m)$ time.*

Proof. Prior to branching we can compute a full kernel with $O(k^2)$ vertices in $O(k^2n+m)$ time (as we will show in Section 5). The exponential part follows directly from Theorem 4.6 and the result for CLUSTER EDITING (MIN). Note that only the non-clique components must be turned into cluster graphs, while the entire cluster merging phase is rendered superfluous in this problem version. □

5 Cluster Editing: The Full Kernel

5.1 The cluster decomposition

In this section we leave the area of branching rules and deal with the size and computation of full kernels. The following decomposition is the basis for our estimate of full kernel size.

Definition 5.1 *A* cluster decomposition *of a connected graph $G = (V, E)$ is a partition of V in disjoint sets, called a* head Q *and* pre-clusters, *with the following properties:*
(i) Every pre-cluster is a clique.
(ii) There are no edges between pre-clusters.
(iii) Every pre-cluster is a module.
(iv) For any two pre-clusters C and D, sets $N(C), N(D)$, called the tags *of C and D in Q, are disjoint.*

Lemma 5.2 *If graph G is at most k edit steps away from a cluster graph, then a cluster decomposition of G with $|Q| \leq 3k$ can be computed in $O(k^2n+m)$ time.*

Proof. Consider a maximal set $\mathcal{P}$ of mutually pair-disjoint induced P_3 in G. Let the head Q be the vertex set spanned by $\mathcal{P}$. Since $\mathcal{P}$ is maximal, $G-Q$ does not contain another P_3, hence $G-Q$ induces a cluster graph. We define the components of $G-Q$ to be the pre-clusters. Then (i), (ii) are obvious. Condition (iii) is true, since otherwise some $u, v \in C$ and $w \in Q$ induce a P_3 $u-v-w$, contradicting the maximality of $\mathcal{P}$. Finally, if some C, D violate (iv), then some $u \in C$, $v \in Q$, $w \in D$ form a P_3, again contradicting the maximality of $\mathcal{P}$. In conclusion, $\mathcal{P}$ as specified above yields a cluster decomposition. Since $|Q| \leq 3k$, at most $3k$ pre-clusters are connected to Q.

In order to construct $\mathcal{P}$ efficiently, we start from the subgraph H of G with empty vertex set and from an empty $\mathcal{P}$. Then we insert vertex by vertex in H, along some spanning tree of G, so that H always remains connected. Moreover, we maintain a cluster decomposition of the current H, until $H = G$. For any new vertex w added to H we form several new P_3, each consisting of w and a pair of vertices u, v in H. We call u, v an eligible pair if u, v, w form a P_3, and u, v do not already belong to the same P_3 in $\mathcal{P}$. Then, we take disjoint eligible pairs and add the resulting new P_3 to $\mathcal{P}$ in a greedy fashion, until $\mathcal{P}$ cannot be further extended in the current H. Correctness is obvious. We analyze the time.

We have to find a greedy set of eligible pairs u, v efficiently in every step. First of all, u, v must form an induced P_3 with w. Since $|Q| = O(k)$, only $O(k)$ of the pre-clusters are connected with Q, and the pre-clusters are modules in H, we can easily partition subgraph $H + w$ in $O(k)$ modules, using the cluster decomposition of H. In detail: Every pre-cluster in H is split into at most two modules (consisting of the vertices being adjacent to w or not), and every vertex in $Q \cup \{w\}$ may form a module on its own, in the worst case.

Now, it suffices to choose one representative vertex from every module of $H + w$, and to check $O(k^2)$ pairs u, v whether they form a P_3 with w, since the results carry over to all vertices u', v' in the same modules. Once we know the pairs of candidate modules, we can restrict our greedy extension procedure to vertices from these pairs of modules. Vertices that became members of any new P_3 in $\mathcal{P}$ will move, of course, from the pre-clusters to Q.

As we have seen above, for every w we need $O(k^2)$ preprocessing time to identify the new members of $\mathcal{P}$, from a partitioning of $H + w$ into modules. This gives the $O(k^2 n)$ term. We limit the time for all other operations in the whole algorithm as follows. Since the final $\mathcal{P}$ has still at most k P_3, inserting them in $\mathcal{P}$ costs $O(k)$ time. Edges incident to each new w are inserted in the growing induced subgraph H in $O(m)$ time. Thus we need $O(m)$ time in total, in order to compute modules of $H + w$ from the cluster decomposition of H, and for all updates of the cluster decomposition. $\Box$

Next we give a cleaning procedure which transforms an instance G, k of Cluster Editing, preserving the set of minimal solutions. It does some forced edit steps immediately and removes parts of G being irrelevant for the problem. Recall that every $v \in Q$ is in the tag of at most one pre-cluster C. To avoid case distinctions, we introduce a dummy clique of size 0 and define its tag as the set of all vertices in Q which are not in the tag of any (real) pre-cluster.

Cleaning Procedure:
(1) For every pre-cluster C with more than k vertices, insert an edge uv between any two $u, v \in N(C)$ that are not yet adjacent.
(2) For any two pre-clusters C and D with c and d vertices, respectively, where $c + d > k$, delete every edge uv with $u \in N(C)$ and $v \in N(D)$. (In particular, D may be the dummy pre-cluster and $d = 0$.)
(3) Remove every clique which is disconnected from the rest of the graph.
Apply these rules as long as possible.

Lemma 5.3 *The cleaning procedure does not alter the set of minimal solutions and can be implemented to run in $O(k^2 + m)$ time if a cluster decomposition as in Lemma 4.7 is already given.*

Proof. In any solution, every clique with $k + 2$ vertices in G must entirely be in a cluster, otherwise we had to disconnect the clique, which is impossible with k deletions. To see that (1) is correct, note that both $C \cup \{u\}$ and $C \cup \{v\}$ are cliques of size at least $k+2$, hence u, v must be in the same cluster. As for (2), observe that if we keep edge uv, every vertex in $C \cup D$ must be incident to an inserted or deleted edge with one of u, v, requiring more than k edit steps, a contradiction. Correctness of (3) is trivial: Extra edges that connect an isolated clique to other vertices cannot be part of a minimal solution.

Rules (1) and (2) only add edges inside (or delete edges between) tags in a set of $O(k)$ vertices, hence they apply at most once to each of $O(k^2)$ vertex pairs. Rule (3) simply removes isolated cliques. Hence the time is linear in the size of G. $\Box$

Corollary 5.4 *After the cleaning procedure, every pre-cluster C has at most k vertices.*

Proof. Assume that a larger C exists. Since (1) does not apply, $C \cup N(C)$ is a clique. Since (2) does not apply, no edge connects $N(C)$ and vertices outside $C \cup N(C)$. This gives an isolated clique, and (3) applies, a contradiction. $\Box$

Corollary 5.5 *A full kernel with at most $3k^2 + 3k$ vertices is computable in $O(k^2 n + m)$ time.*

Proof. The remaining graph is a full kernel, $|Q| \leq 3k$, all tags are disjoint, and the pre-clusters are bounded due to Corollary 5.4. $\Box$

5.2 Ambiguous vertices

Corollary 5.5 establishes an $O(k^2)$ bound for the full kernel. Next we also achieve the optimal constant factor. In the following we *fix the cluster decomposition and a certain minimal solution.* With respect to this minimal solution, we distinguish several cases of pre-clusters C and "charge" them for edit steps that touch vertices in C. All pre-clusters will be charged. Since at most k edit steps are allowed in total, this will eventually limit the full-kernel size.

Lemma 5.6 *If the graph (full kernel) G' after the cleaning procedure is disconnected, then a minimal solution never adds edges between vertices from different components of G'.*

Proof. The cleaning procedure performs only enforced edit steps (which must be done in any minimal solution). Any cluster with vertices from different components of G' can be split in smaller clusters, each containing vertices from one component. This finer clustering was produced by a proper subset of further edit steps (starting from G'), hence the given clustering was not a minimal solution. $\square$

Theorem 5.7 *At most $k^2/4 + 7k/2 + 1/4$ vertices are ambiguous, and a full kernel of that size can be computed in $O(k^2n + m)$ time.*

Proof. Consider any component H of the full kernel after the cleaning procedure. Let $c_1 \geq \cdots \geq c_r$ be the vertex numbers of all r pre-clusters in H. If the tags of all pre-clusters in H are cliques, we set the weight of H to $c_1 + c_r$. Otherwise, the weight of H is just c_1. Finally, w is the maximum weight of a component in our full kernel. Every tag is non-empty, since the cleaning procedure has removed isolated cliques.

We call a pre-cluster *inert*, with respect to a fixed minimal solution, if the vertex set of this pre-cluster and its tag forms exactly one cluster there. Let C, with c vertices, be a largest pre-cluster in a component H of weight w. *In the following we suppose that C is not inert in some minimal solution, and we fix such a minimal solution.*

Phase 1:

First assume that all other pre-clusters in H are inert. Since, by Lemma 5.6, edit steps after the cleaning procedure occur only inside the components, it follows that $C \cup N(C)$ is split in at least two clusters. This splitting requires at least c deletions of edges incident to vertices of C, which is easily seen from $N(C) \neq \emptyset$ and the fact that C is a clique and a module. Moreover, since we consider a minimal solution, $N(C)$ was not a clique. By the definition of w, we get $c = w$.

The other case is that some other pre-cluster D in H, say with $|D| = d$, is not inert either. We claim that at least $c + d$ edit steps that involve vertices of $C \cup D$ must be done. It is easy to check the few different cases that $C \cup N(C)$ (or $D \cup N(D)$) is cut in different clusters or stays in one cluster that gets at least one more vertex. Trivially, we also have $c + d \geq w$.

Thus we can already charge one or two non-inert pre-clusters for at least w edit steps, in both cases.

Phase 2:

Next we also charge *all* the remaining pre-clusters in *all* components. In the following, consider any component with, say, r pre-clusters. By connectivity, at least $r - 1$ edges tie their tags together. Two pre-clusters are called neighbors if there exists an edge between their tags.

If the considered component has at least one non-inert pre-cluster C, or an inert pre-cluster C whose tag $N(C)$ is not a clique, we can successively charge all r pre-clusters, each for one edit step, as follows. First charge all pre-clusters C of the two mentioned types: If C is not inert, there is an edit step involving a vertex of C. If C is inert but $N(C)$ not a clique, an edge insertion must be done in $N(C)$. Next, pick a yet uncharged pre-cluster D (inert, with a clique as tag) which has already a charged neighbor C. Since D is inert, an edge between $N(C)$ and $N(D)$ is deleted in the solution, and we charge D for this deletion. Since C was already charged, we did not count this edge twice. By connectivity, this procedure never gets stuck, until all pre-clusters are charged. If one of the pre-clusters in the component is dummy, we also consider it as "charged" in the beginning and proceed as above. In either case, every non-empty pre-cluster in the component is now charged for a different edit step. Also note that every pre-cluster contains at most w vertices.

It remains to discuss components where all pre-clusters are inert and have cliques as tags. Obviously, at least $r-1$ edges must be deleted. We charge the largest and the smallest pre-cluster together for one deletion, and the other $r-2$ pre-clusters together for $r-2$ deletions. By definition of w, at most w vertices from pre-clusters are now charged for each of the $r-1$ edge deletions.

Putting things together:

Recall that at most k edit steps in total are allowed. In Phase 1 we charged, for $y \geq w$ of them, no more than y vertices from one or two pre-clusters. In Phase 2, at most $w \leq y$ vertices from nonempty pre-clusters have been charged for each of the, at most, $k-y$ other edit steps. Since all pre-clusters are charged, all pre-clusters together contain at most $(1+k-y)y$ vertices. This term is maximized if $y=(k+1)/2$, hence the pre-clusters contain at most $k^2/4+k/2+1/4$ vertices. Adding the at most $3k$ vertices from Q yields the result.

Finally we give the time complexity. Again, let C be a largest pre-cluster in a component of maximum weight. (C is easy to find in the cluster decomposition.) Our construction and analysis shows: If C is not inert in some minimal solution, the union of pre-clusters has already a size of at most $k^2/4+k/2+1/4$, and we can stop. By contraposition, if this union is larger, we know that C is inert in all minimal solutions. Hence, removing $C \cup N(C)$ leaves us with a smaller full kernel. Moreover, we still have a cluster decomposition of this smaller graph where $|Q| \leq 3k$, since the part outside Q is still a cluster graph. Thus we can simply iterate the procedure. A cluster decomposition with $|Q| \leq 3k$ must be computed only once in the beginning. The only thing to recompute after every removal is the components of Q. Now the time bound follows from the previous results. □

The asymptotic $k^2/4$ bound is tight even for Cluster Deletion, as the following example shows. For simplicity let k be even.

Proposition 5.8 *There exist graphs with $k^2/4 + 3k/2 + 2$ ambiguous vertices.*

Proof. Take $k/2 + 1$ disjoint cliques, each with $k/2 + 1$ vertices, and attach to every clique another vertex being adjacent to one vertex in the clique. Put any one of the extra vertices x and its only neighbor y in one cluster, the other $k/2$ edges incident to y are deleted. The other $k/2$ extra edges are also deleted. Every such solution is minimal, hence all vertices are ambiguous. □

There remains a gap in the linear term only. For the graphs in the previous proof, the minimum number of edit steps to reach a cluster graph is only $k_0 = k/2 + 1$. It arises the question whether the full kernel size is even smaller than $k^2/4$ for $k = k_0$, or already for some $k < 2k_0$.

5.3 Ambiguous pairs

Theorem 5.9 *The number of ambiguous pairs is bounded by $k^4/32 + o(k^4)$, and there exist graphs with $k^4/800$ ambiguous pairs.*

Proof. The first statement follows from Theorem 5.7. The worst case would be that all pairs in a full kernel of maximum size are ambiguous.

As an example for the lower bound, take ak disjoint cliques, each with bk vertices, and another special vertex z. Constants a, b are fixed later. For simplicity assume that all numbers that denote cardinalities are in fact integer. Join one vertex from every clique by an edge to z. In the following, K_v denotes the clique with vertex v connected to z. For any K_u and K_v and $w \in K_v$ ($w \neq v$) we specify a minimal solution as follows. We keep edges uz, zv, vw, and all edges in K_u. This implies that K_u and z, v, w belong to the same cluster. In order to put z, v, w in K_u we have to insert $3bk$ edges. The other $ak - 2$ edges indicent to z are deleted, as well as the $2bk - 4$ edges between v and w, respectively, and the rest of K_v. Obviously, these $(a + 5b)k - 6$ edit steps yield a cluster graph. This solution is minimal, by the following argument. In any solution using a subset of edit steps we must keep uz, zv, vw and the edges in K_u, too, hence K_u, z, v, w are in the same cluster. Furthermore we must keep $K_v \setminus \{v, w\}$ and all other cliques. Since merging two $\Theta(k)$ cliques would require $\Theta(k^2)$ insertions, we must also delete the same edges as above. Note that all $a^2b^2k^4/2$ edges (lower-order terms neglected) are ambiguous. The constant factor is maximized under constraint $a+5b \leq 1$ if $a = 1/2$ and $b = 1/10$. □

The constant factor is left as an open problem. Similar remarks as above apply to the graphs used in this proof: Note that $k_0 = ak = k/2$. We conjecture that we get much less ambiguous pairs if k is close to k_0.

The number of ambiguous edges in Cluster Deletion is open, only an upper bound of $3k^3/4 + o(k^3)$ and a quadratic lower bound is known [20]. Another question arises from Theorem 5.7: Our construction yields a bound in terms of k, but not the smallest full kernel for any given graph. We conjecture that computing the exact set of ambiguous pairs requires the actual computation of all minimal solutions, and thus exponential time in k.

6 Other Problem Variants and Conclusions

If a graph $G = (V, E)$ is at most k edit steps away from a graph with a given hereditary property (preserved on induced subgraphs), we also reach that property by k vertex deletions (or much less, if the "graph of edit steps" has a small vertex cover). Thus, any edge modification problem gives rise to a related vertex deletion problem.

For example, Cluster Vertex Deletion, corresponding to Cluster Editing, asks to find some $X \subset V$, $|X| \leq k$, such that $G - X$ is a cluster graph. This allows for small overlaps of many clusters, whose number is no longer limited by the parameter. For the currently best results on this problem see [14]. A concise enumeration is easy to obtain by a search tree algorithm with branching number 2.42 (based on the observation that any graph with components other than cliques and P_3 has two induced P_3 with two common vertices), but this can certainly be improved. All complexity bounds in the paper are subject to further improvements as well, e.g., by more sophisticated branching rules and analysis.

Can we efficiently solve edge modification problems in a two-stage process, using their vertex deletion counterparts? The idea would be to compute concise enumerations for the vertex deletion problem, and to assign edge changes to the affected vertices only.

Cluster editing with arbitrary individual edge weights (at least 1) is solvable in $O(3^k + n^3)$ time, where k is the total cost of editing [21]. Can we do better if weights are distances in a metric space? Furthermore, vicinity graphs in a metric space cannot contain induced stars $K_{1,s}$ (s depending on the dimension). Can our bounds be improved for such graphs G? Can we efficiently compute posterior probabilities of clusterings when prior probabilities of edges are given? This problem seems to be quite different from weighted cluster editing [21].

Instead of weights we could have two parameters for the number of insertions and deletions. This is appropriate in applications where insertions and deletions have different evidence.

Acknowledgements

This work was partially supported by the Swedish Research Council (Vetenskapsrådet), grant no. 2007-6437, "Combinatorial inference algorithms – parameterization and clustering".

I am grateful to Tomas Nolander and the anonymous referees at WG 2005 and this journal, for pointing out some inaccuracies in earlier drafts and for many technical remarks, Mike Langston for encouraging discussions, and Henning Fernau for drawing my attention to some recent articles.

References

[1] N. Bansal, A. Blum, S. Chawla. Correlation clustering, *Machine Learning* 56 (2004), 89–113

[2] Z.Z. Chen, T. Jiang, G. Lin. Computing phylogenetic roots with bounded degrees and errors, *SIAM J. Computing* 32 (2003), 864–879

[3] P. Damaschke. Parameterized enumeration, transversals, and imperfect phylogeny reconstruction, *Theoretical Computer Science* 351 (2006), 337–350

[4] F. Dehne, M.A. Langston, X. Luo, S. Pitre, P. Shaw, Y. Zhang. The cluster editing problem: Implementations and experiments, *2nd International Workshop on Parameterized and Exact Computation IWPEC 2006*, *LNCS* 4169, 13–24

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

[6] H. Fernau. On parameterized enumeration, *8th Computing and Combinatorics Conference COCOON 2002*, *LNCS* 2387, 564–573

[7] H. Fernau. Edge dominating set: Efficient enumeration-based exact algorithms, *2nd International Workshop on Parameterized and Exact Computation IWPEC 2006*, *LNCS* 4169, 142–153

[8] H. Frigui, O. Nasraoui. Simultaneous clustering and dynamic keyword weighting for text documents, in: M. Berry (ed.), *Survey of Text Mining*, Springer 2004, 45–70

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

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

[11] J. Guo. A more effective linear kernelization for Cluster Editing, *1st Int. Symposium on Combinatorics, Algorithms, Probabilistic and Experimental Methodologies ESCAPE 2007*, *LNCS* 4614, 36–47

[12] J. Guo, J. Gramm, F. Hüffner, R. Niedermeier, S. Wernicke. Compression-based fixed-parameter algorithms for feedback vertex set and edge bipartization, *Journal of Computer and System Sciences* 72 (2006), 1386–1396

[13] W.L. Hsu, T.H. Ma. Substitution decomposition on chordal graphs and applications, *2nd International Symposium on Algorithms ISA 1991*, *LNCS* 557, 52–60

[14] F. Hüffner, C. Komusiewicz, H. Moser, R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion, *8th Latin American Theoretical Informatics Symposium LATIN 2008)*, *LNCS* 4957, pp. 711-722

[15] M. Krivanek, J. Moravek. NP-hard problems in hierarchical-tree clustering, *Acta Informatica* 23 (1986), 311–323

[16] G.H. Lin, T. Jiang, P.E. Kearney. Phylogenetic k-root and Steiner k-root. *11th International Symposium on Algorithms and Computation ISAAC 2000*, *LNCS* 1969, 539–551

[17] T.M. Mitchell. *Machine Learning*, McGraw-Hill 1997

[18] D. Mölle, S. Richter, P. Rossmanith. Enumerate and expand: Improved algorithms for connected vertex cover and tree cover, *1st International Computer Science Symposium in Russia CSR 2006*, *LNCS* 3967, 270–280

[19] D. Mölle, S. Richter, P. Rossmanith. Enumerate and expand: New runtime bounds for vertex cover variants, *12th Computing and Combinatorics Conference COCOON 2006*, *LNCS* 4112, 265–273

[20] T. Nolander. On the fixed-parameter enumerability of cluster deletion, Master's thesis, Computing Science, Chalmers (Göteborg) 2006

[21] S. Rahmann, T. Wittkop, J. Baumbach, M. Martin, A. Truß, S. Böcker. Exact and heuristic algorithms for weighted cluster editing, *6th Annual International Conference on Computational Systems Bioinformatics CSB 2007*, 391–401

[22] D. Scholtens, M. Vidal, R. Gentleman. Local modeling of global interactome networks, *Bioinformatics* 21 (2005), 3548–3557

[23] R. Shamir, R. Sharan, D. Tsur. Cluster graph modification problems, *Discrete Applied Mathematics* 144 (2004), 173–182

[24] R. Sharan, A. Maron-Katz, R. Shamir. CLICK and EXPANDER: A system for clustering and visualizing gene expression data, *Bioinformatics* 19 (2003), 1787–1799

[25] R. Sharan, R. Shamir. Algorithmic approaches to clustering gene expression data, in: *Current Topics in Computational Molecular Biology*, MIT Press, 2002, 269–300

[26] S. Wu, X. Gu. Gene network: Model, dynamics and simulation, *11th Computing and Combinatorics Conference COCOON 2005*, *LNCS* 3595, 12–21