Bounded-Degree Techniques Accelerate Some Parameterized Graph Algorithms

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

Abstract

Many parameterized algorithms for NP-hard graph problems are search tree algorithms with sophisticated local branching rules. But it has also been noticed that the global structure of input graphs can help improve the time bounds. Here we present some new results based on the global structure of bounded-degree graphs after branching away the high-degree vertices. Some techniques and structural results are generic and should find more applications. First, we decompose a graph by branchings along a separator into cheap or small components where we can further branch separately. We apply this technique to accelerate the $O^*(1.3803^k)$ time algorithm for counting the vertex covers of size k (Mölle, Richter, and Rossmanith, 2006) to $O^*(1.3740^k)$. Next we give a complete characterization of graphs where every edge is in at most two conflict triples, i.e., triples of vertices with exactly two edges. This enables us to improve to $O^*(1.47^k)$ the previous $O^*(1.53^k)$ time algorithm (Gramm, Guo, Hüffner, Niedermeier, 2004) for Cluster Deletion. i.e., for deleting k edges so as to destroy all conflict triples and obtain a disjoint union of cliques. For graphs where every edge is in $O(1)$ conflict triples we show a nice dichotomy: The graph or its complement has degree $O(1)$. This opens the possibility to apply the above decomposition technique and develop further improvements.

1 The Problems and Contributions

A problem is fixed-parameter tractable (FPT) if it is solvable in time $O(p(n)f(k))$ where p is a polynomial and f any function. We assume familiarity with the basic notions of FPT algorithms and their analysis, otherwise we refer to [5, 13]. Since we focus here on the exponential part of the complexity, we sometimes adopt the $O^*(f(k))$ notation that suppresses polynomial factors. In graph problems we usually denote by n the number of vertices in a graph. For brevity we say "component" instead of "connected component" of a graph.

Counting Vertex Covers: A vertex cover is a set of vertices with at least one vertex from every edge. The problem of counting all vertex covers of size k in a graph (more generally, counting all hitting sets of size k in a hypergraph of fixed rank) is very natural as such, but has also interesting applications in combinatorial inference, e.g., in computational biology as proposed in [3]. Briefly, the real problem is to infer a set of substances that explains experimental observations, modeled as vertices and edges of a hypergraph, but since many different solutions of a given expected size exist, we count the solutions containing every fixed vertex, in order to evaluate how likely the presence of every substance is. Many vertex cover variants have been studied in the literature, see, e.g., [11] for pointers.

The vertex cover counting algorithm in [12] first branches on vertices of degree 4 or larger. The branching vector $(1, 4)$ has branching number 1.3803, and vertex covers in the residual graphs of degree 3 can be counted in $O^*(1.26^k)$ time by a non-trivial technique. This gives an overall time bound of $O^*(1.3803^k)$, where branching on vertices of degree 4 is the bottleneck. In order to take this hurdle we must avoid the situation that only $(1, 4)$-branching takes place and cheaper rules are never invoked. Our idea is to reuse the algorithm but do the initial $(1, 4)$-branching in a special way that guarantees a “large and easy” residual graph. We either retain one large degree-3 component that can be solved in time $O^*(1.26^k)$ (with the residual k), or we can split the residual graph into several components. In the latter case we can do the branching separately on the components and finally combine the partial results in polynomial time. Loosely speaking, this quasi-parallelization of the remaining branchings has the effect that only one component with the largest residual k counts for the exponential term of the complexity. In all cases this drives the overall branching number below 1.3803. The observation that components of a graph can be treated independently is usually not even worth mentioning as it cannot be exploited in a worst-case analysis, but in our approach it becomes an essential point. The time analysis is not straightworward either. Upon the $(1, 4)$-branchings we deduct not only 1 and 4 from the parameter k, but also the number of vertices that must be added later to the vertex covers. On the other hand, we must charge the $(1, 4)$-branchings for these deferred decisions and increase the size bounds for the search trees after the branching accordingly, as every leaf in the search tree now represents the deferred decisions. But since these later branchings are cheaper, this pays off in the end.

This global technique of branching away the hardest parts of an instance and separating the rest into independent pieces seems to be new in this form. The idea is not deep, but this may be a strength. The conceptual simplicity should make it versatile and also applicable to other FPT problems, provided that they decompose in the desired way.

The bound for vertex cover counting we can prove so far is $O^*(1.3740^k)$. This may seem a marginal progress, however, this is apparently more a weakness of the current analysis rather than the algorithm itself. We will point out where our analysis gives something away, hence the true *worst-case* bound is likely to be way better. On the other hand, it is hard to take advantage of those observations in a worst-case analysis, since different bad cases can appear arbitrarily mixed throughout the search tree. (Cf. also [6, 7].)

Cluster Modification Problems: A cluster graph is a disjoint union of cliques, also called clusters. A P_3 is a path of three vertices (and two edges). Cluster graphs are exactly the graphs without induced P_3 which are also called conflict triples.

In Cluster Deletion, a graph G and an integer k are given, and we are to delete at most k edges so as to obtain a cluster graph. Equivalently, deletions must destroy all induced P_3. Cluster Editing is similarly defined, but there we may delete and insert edges. Both problems are special cases of Weighted Cluster Editing, where pairs of vertices have individual edits costs. These problems have applications foremost in computational biology [4, 9, 14] and they are NP-hard [14].

Cluster Deletion is appropriate if non-adjacent vertices, representing dissimilar objects, are strictly required to be in distinct clusters.

The parameterized complexity of these problems is well studied. In [2] we give problem kernels and algorithms for the enumeration of *all* solutions to several clustering problems. In [1], Weighted Cluster Editing has been solved in $O(1.82^k + n^3)$ time. This is also the best known time bound for Cluster Editing. The other special case is still somewhat "easier": an $O(1.53^k + n^3)$ time algorithm for Cluster Deletion is known [8]. This result was an application example of automated generation of search trees and improved a "handmade" $O(1.77^k + n^3)$ result [9]. The algorithm works with branching rules on subgraphs with at most six vertices. Apparently no further progress on Cluster Deletion has been made since then.

Here we give an $O(1.47^k + n^3)$ time algorithm for Cluster Deletion. It starts from the most obvious branching rule with branching number 1.47: If some edge belongs to three or more P_3, then *branch on the edge*, i.e., delete the edge, or delete all edges building a P_3 with it. Key to our improvement is a new theorem showing that graphs where this rule is not applicable have simple structures, such that no other local branching rules are needed. (Special cases were already treated in [1, Lemma 4] and in a conference version of [2], but here we study this structure in full generality.) This result is yet another indication that local branching rules should be complemented by global structure analysis and techniques. Other global methods like dynamic programming on subsets, bounded treewidth and pathwidth [6, 7], and iterative compression [10], have recently shown their great potential.

Organization of the paper: In Section 2 we state our algorithm for vertex cover counting. It builds upon [12] and is only slightly more complicated. The only new features are that we branch on vertices of degree 4 in a certain breadth-first order and combine partial results from different components in moderate polynomial time. (The polynomial factor has not been explicitly specified in [12], but we can obviously state that it does not blow up by the routines we add.) In Section 3 we prove a time bound that beats $O^*(1.3803^k)$, and we discuss why the true bound should be even better. Then we turn to some graph theory useful for clustering. In Section 4 we give a complete characterization of graphs where every edge is in at most two P_3. The organization of the proof tries to avoid too many tiresome case distinctions. More generally, for graphs where every edge is in any constantly bounded number of P_3 we prove a dichotomy: the graph or its complement graph has constantly bounded degree. Section 5 deals with the algorithmic consequences of P_3 structure. First we give an $O^*(1.47^k)$ algorithm for Cluster Deletion. Improving some details in the polynomial-time parts solving the "simple" cases seems quite possible, however, the more intriguing question is about improved bases of the exponential term. We have to leave this for further research, but, using the above dichotomy we can at least show that the bottleneck case is graphs with bounded degree, and therefore it may be possible to apply the same separation technique as in Section 2-3. We discuss these possibilities also for Cluster Editing. Due to lack of space, some simpler proofs are omitted in this version.

2 Vertex Cover Counting Algorithm

In a graph G, let $c(G, k)$ be the number of vertex covers with exactly k vertices. The *degree* of a vertex v is the number of vertices adjacent to v, and the degree of a graph is the maximum vertex degree. We call a vertex of degree exactly d a *degree-d vertex*, and a graph of degree 3 *subcubic*.

We skip the definition of tree decomposition, because we use the following result only as a "black box". Any subcubic graph has a tree decomposition of width at most $(1/6+\epsilon)n$, and such a tree decomposition can be computed in polynomial time, for any fixed $\epsilon > 0$ [7]. Dynamic programming on this tree decomposition can be used to count the vertex covers of any given size k in $O^*(2^{(1/3+\epsilon)k}) = O^*(1.26^k)$ time [12], thus:

Lemma 1 *In subcubic graphs G one can compute $c(G,k)$ in $O^*(1.26^k)$ time.*

If a graph G is the disjoint union of graphs G_1 and G_2 then, obviously, $c(G,k) = \sum_{j=0}^{k} c(G_1,j)\cdot c(G_2,k-j)$. From this it is easy to conclude:

Lemma 2 *Once the $c(G',j)$ are known for all components G' of G, and for all $j \le k$, we can compute $c(G,k)$ in polynomial time.*

Branching on a vertex v means the following rule: Either put v in the vertex cover and remove v and all incident edges, or put all neighbors of v in the vertex cover and remove v, its neighbors, and all incident edges.

We refer to a series of such branching decisions on different vertices as a *branch*. In other words, a branch corresponds to a node of the resulting search tree. The *residual graph* in a branch is the graph that remains after removal of the vertices and edges specified above. Note that branching on a vertex divides the family of vertex covers of the current residual graph exactly in two subfamilies, of the vertex covers of residual graphs in the two new branches. Hence, when the search tree is completed, we can sum up the numbers of vertex covers (of the proper size) found in the residual graphs at the leaves. Now we describe our algorithm.

1. Preparation phase: First branch, as long as possible, on degree-d vertices with $d \ge 5$. In every branch consider the residual graph of degree 4 and continue as described below. We will choose certain vertices and distinguish them as *roots*.

2. Separation phase: If a component without a root exists, then fix some vertex in this component as the *active root* and proceed as follows. By layer j we denote the set of vertices at distance j from the active root. As long as, in the component of the active root, there exist degree-4 vertices in some layer $j > 2$, choose one such degree-4 vertex closest to the root and branch on it. (Note that these branchings may change the layers, and even disconnect some vertices from the active root.) This process stops as soon as no degree-4 vertices remain in the layers $j > 2$ in the component of the active root. At this moment, set the active root passive. Iterate the process, until each component of the residual graph has a passive root.

3. Completion phase: Branch on the remaining degree-4 vertices near the roots. This results in a subcubic graph. Let $m \le k$ be the number of vertices that remain to be added to the vertex cover. (Number m can differ between the residual graphs in several branches, but m is exactly known in every branch.) Compute the $c(G',i)$ in all components G' of the residual graph, and for all $i \le m$, using the algorithm in Lemma 1. Finally combine these results in every residual graph, to compute the number of vertex covers of size k, using the algorithm in Lemma 2. In the last step, the counts from all leaves of the search tree are added.

Some refinement: For the separation phase we refine the rule for chosing the next degree-4 vertex to branch on. This is only a technicality that we insert just because we can prove a better worst-case bound when using it.

Whenever a new layer j for branching is entered (that is, $j > 2$ is the smallest index of a layer where still degree-4 vertices exist), pair up some degree-4 vertices in layer j to siblings: Every degree-4 vertex in layer j is assigned an adjacent *parent vertex* in layer $j-1$. (If several possible parent vertices exist, then select any one.) *Siblings* are degree-4 vertices with the same parent. It is not hard to see that, for $j > 3$, every degree-4 vertex gets at most one sibling. Now, whenever we have branched on a degree-4 vertex v and added v to the vertex cover (this happens in one branch), and v has a sibling which is still a degree-4 vertex, then branch on this sibling immediately.

3 Analysis

Theorem 3 *The vertex covers of size k can be counted in $O^*(1.3740^k)$ time.*

Proof. The branching number for branching on degree-d vertices with $d \geq 5$ in the preparation phase is the positive root of $x^5 = x^4 + 1$, that is, $x < 1.3248$.

For the separation procedure in the "rooted" components, the following observation is crucial: Since we always pick a degree-4 vertex in a layer $j > 2$ closest to the root, and removal of vertices and edges can make the distances between the remaining vertices only larger, the current distance j from root, of the degree-4 vertices we branch on, can only increase during the process. Hence, if j denotes the index of the current layer, any vertex being in a layer $i \leq j-2$ at this moment is never removed later, and it also stays in the same layer i forever. Accordingly we call these vertices *persistent*. Also note that any vertex being in layer $j-1$ at this moment can either disappear due to branchings in layer j, or stay in layer $j-1$ forever, but it cannot slide anymore into a layer with higher index.

Let j be the current layer where we branch on degree-4 vertices. As already stated in the algorithm, to every degree-4 vertex v in layer j we assign a *parent vertex* $p(v)$ which is a neighbor of v in layer $j-1$, and a *grandparent vertex* $g(v)$ which is a neighbor of $p(v)$ in layer $j-2$. Note that $g(v)$ is persistent. Since every vertex in a layer $i > 0$ has at least one edge to a neighbor in layer $i-1$, and vertex degrees are at most 3 in the layers i, $2 < i < j$, every persistent vertex in a layer $i > 2$ can have at most 2 children and at most 4 grandchildren vertices. Recall that any two degree-4 vertices v, v' with the same parent are *siblings*. If v is a degree-4 vertex without a sibling v' (a degree-4 vertex with the same parent), we still use the notation v' and simply say that "v' does not exist".

In the following we assign to certain vertices so-called *certificates*. Consider any vertex v and its sibling v' in layer j. We distinguish two cases.

Case (1): Either v' does not exist, or v, v' are adjacent. Then, when we branch on v we send a certificate to $g(v)$. The branching vector is $(1, 4)$, and after the branching, no child of $p(v)$ is a degree-4 vertex anymore, since either v' did not exist, or v' has been removed now, or vertex v has been removed, which reduces the degree of v'.

Case (2): v' exists and v, v' are not adjacent. Then, when we branch on the first sibling, say v, we send a certificate to $g(v)$. In one branch we have put the 4 neighbors of v in the vertex cover. Otherwise we have put only v in the vertex cover, and then, by the refined rule, we branch on v' which is still a degree-4 vertex. In total we achieve the branching vector $(2, 4, 5)$. After that, no child of $p(v)$ is a degree-4 vertex anymore, since $p(v)$ has been removed, which reduces the degree of its children, or both v and v' have been removed.

This way, every persistent vertex in layers $i > 2$ receives at most 2 certificates through its (at most 2) children. For each residual graph let r denote the total number of certificates issued in all components. Hence at least $r/2$ persistent vertices exist in layers $i > 2$. Since every such vertex is incident to an edge leading to the previous layer, at least $r/2$ edges remain after the separation phase.

For easier understanding we first establish a weaker bound of $O^*(1.3755^k)$: Basically, analysis is done in the usual way, by establishing a recurrence for the number $T(k)$ of leaves of a search tree, if the initial parameter is k. The recurrences for branching on siblings v, v' of degree 4, as specified above, would be $T(k) \leq T(k-1) + T(k-4)$ and $T(k) \leq T(k-2) + T(k-4) + T(k-5)$. Here we establish modified recurrences where we exploit the fact (see above) that for every branching at least $1/2$ edges need to be covered later in the completion phase. (Here and in the following, fractional numbers make sense because their sum is finally rounded to the next integer.) Since the maximum degree is 4, per branching at least $1/8$ vertices of the residual graph will be added to the vertex cover in the completion phase. Thus it is safe to deduct an extra value $1/8$ from the parameter already in the separation phase. On the other hand, the leaves of the search tree for the separation phase now represent residual graphs prior to the completion phase. Thus we have to multiply the sizes of search trees on the right-rand side of the recurrences by the search tree size for the completion phase. This seems to be tricky, as these tree sizes can be very different in the various branches, but we will next figure out the worst case for the recurrence.

Consider the residual graph in any leaf of the search tree, after the separation phase. Note that every component has only constantly many remaining degree-4 vertices near the root, hence branching on them adds only a constant factor to the total time bound. Let m be defined as in the algorithm (completion phase). From Lemma 1 and 2 it follows that the vertex covers of size m in the residual graph can be counted in $O^*(1.26^m)$ time. In the worst case, this cheap completion phase can be applied only with the smallest possible m in all leaves. On the other hand, we know from our analysis above that $m \geq r/8$ in all leaves, where r was the number of branchings, i.e., the path length in the search tree. Thus it suffices to charge every branching in the separation phase for one $(1.26)^{1/8}$ factor of the complexity of the completion phase, i.e., to multiply the tree size bounds after branching by this factor. For subtrees where the separation phase stops earlier and m is larger, the size can only be smaller.

Thus we obtain the recurrences $T(k) \leq (T(k-1-1/8) + T(k-4-1/8))1.26^{1/8}$ and $T(k) \leq (T(k-2-1/8) + T(k-4-1/8) + T(k-5-1/8))1.26^{1/8}$ with solutions 1.3723^k and 1.3755^k, respectively.

Finally we strengthen one of our bookkeeping arguments as follows. We move the branchings on the remaining degree-4 vertices near the roots to the separation phase. Since in each residual graph only one search tree for the components counts for the complexity (only the largest one, due to the polynomial-time combination procedure from Lemma 2) and every component has only constantly many degree-4 vertices at that moment, this adds a constant factor to the total complexity. On the other hand, the maximum degree in the completion phase is 3 rather than 4. Hence we can replace 1/8 with 1/6 in the recurrences and obtain the solutions 1.3699^k and 1.3740^k, respectively. $\Box$

Our analysis guarantees the existence of at least one persistent vertex, with an edge attached, for any two local branchings made. We "certify" only grandparent vertices, but not their ancestors. That is, our analysis may not notice the existence of further persistent edges, especially in the large component that is relevant for the complexity. If we could incorporate all persistent vertices in the analysis, this would almost double the deduction from the parameter. Further improvements may come from refinements of the algorithm itself. The worst case in our analysis appears if every vertex in the largest component has two children. But just in this case the component is merely a binary tree, and vertex covers could be counted there in linear time. One could make use of such easy cases by kernelization on the components.

4 Conflict Triple Structure in Graphs

As usual, symbols P_n, C_n, K_n denote a chordless path, cycle, and a clique, respectively, of n vertices, and $K_{m,n}$ a complete bipartite graph with m and n vertices in the partite sets. The disjoint union $G + H$ of graphs G and H consists of vertex-disjoint copies of G and H. Let pG be the union of p copies of G. The join $G * H$ of two graphs is obtained from $G + H$ by inserting all possible edges between the vertices of G and H. The complement G^c of G is obtained by switching all edges into non-edges and vice versa. In an obvious sense, a P_4 has two *inner vertices* forming the *central edge*, and two *outer vertices*, and a P_5 has a *central vertex*.

Let the *score* of an edge be the number of different P_3 the edge is involved in. A graph is score-s if every edge appears in at most s different P_3. Note that a score-s graph is also score-$(s + 1)$, and an induced subgraph of a score-s graph is score-s. The main goal of this section is to characterize the connected score-2 graphs G. We say that a graph is spanning score-2 if it contains a spanning tree of score-2 edges. To *add a vertex* to a connected graph G means to introduce a new vertex that is adjacent to at least one vertex of G. A graph is maximal score-2 if we cannot add another vertex while keeping the graph score-2 and connected.

Lemma 4 *Suppose that we add a vertex x to a connected score-2 graph G.*
(i) If the extended graph remains score-2, and x is adjacent to vertex u of G, then x is also adjacent to all vertices reachable from u via score-2 edges in G.
(ii) If G is spanning score-2 and the extended graph remains score-2, then x is adjacent to all vertices of G.
(iii) If G is spanning score-2 and has a vertex which is non-adjacent to at least three other vertices of G, then G is already maximal score-2.

Doubling a vertex x of a graph means to insert a new vertex that is adjacent exactly to x and its neighbors. We can double several vertices in a graph, and the result does not depend on the order. Now we define several special six-vertex graphs, with the understanding that only the explicitly mentioned edges exist.

- 3-asterisk: a K_3 where each vertex is adjacent to one further vertex.

- 3-sun: a K_3 where each two vertices are adjacent to one further vertex.
- fat P_4: obtained from a P_4 by doubling both inner vertices.
- fat P_5: obtained from a P_5 by doubling its central vertex.

Theorem 5 *The following graphs (with arbitrarily large positive n, q, p) and their connected induced subgraphs include the complete list of the connected score-2 graphs:*

- *3-asterisk, 3-sun, fat P_4, fat P_5,*
- *C_n $(n \geq 4)$,*
- *$K_q * C_5$, $K_q * K_3^c$, $K_q * (K_2 + K_2)$,*
- *$(qK_1 + pK_2)^c$ $(p \geq 2)$.*

Proof. It is easy to check that all listed graphs are score-2. All listed graphs except $(qK_1 + pK_2)^c$ are also spanning score-2. The 3-asterisk, 3-sun, fat P_4, fat P_5, and C_n $(n \geq 6)$ also fulfill the assumptions of Lemma 4 (iii), hence they are maximal score-2. Adding a vertex to any of $K_q * C_5$, $K_q * K_3^c$, $K_q * (K_2 + K_2)$ while preserving score 2 yields a graph of the same type, with q increased by 1.

The reasoning for $(qK_1 + pK_2)^c$ is slightly more complicated. Graph $(pK_2)^c$ $(p \geq 2)$ is spanning score-2, hence, by Lemma 4 (ii), every added vertex is adjacent to all these $2p$ vertices. Moreover, each of the added vertices must be adjacent to all other added vertices except at most one. Thus we obtain only graphs $(qK_1 + pK_2)^c$. It is also impossible to add further vertices adjacent only to vertices in the qK_1 part of $(qK_1 + pK_2)^c$, as this would create new edges of score larger than 2.

Thus we have shown that our list contains only score-2 graphs and is closed under vertex addition. Next we shall prove that no further cases exist, that is, any connected 2-score graph G is in our list, either directly or as an induced subgraph of a listed graph.

For any vertex u let $N(u)$ denote the set of neighbors of u in G. In G^c this means, $N(u)$ is the set of all vertices non-adjacent to u. Let $H(u)$ be the subgraph of G^c (!) induced by $N(u)$. If some vertex v has degree larger than 2 in $H(u)$, then the non-edge uv is in three P_3^c in G^c, a contradiction. If $H(u)$ has an induced $2K_2$ then G has an induced $(K_1 + 2K_2)^c$. The other cases are that $H(u)$ has no edges, or the edges in $H(u)$ form one of the induced subgraphs K_2, P_3, C_3, P_4, C_4, C_5. We examine these cases one-by-one. Note that we always obtain graphs from our list.

- If $H(u)$ has an induced C_5 then G has an induced $K_1 * C_5$.
- If $H(u)$ has an induced C_4 then G has an induced $K_1 * (K_2 + K_2)$.
- If $H(u)$ has an induced P_4 then G has an induced $K_1 * P_4$. Since the two edges from u to the outer vertices of the P_4 and the central edge of the P_4 have score 2, Lemma 4 (i) gives that any added vertex is adjacent to both inner vertices, or to u and both outer vertices, or to all five vertices. These cases yield an induced 3-sun, $K_1 * C_5$, and $K_2 * P_4$ (induced subgraph of $K_2 * C_5$), respectively. By essentially the same argument, adding further vertices to $K_q * P_4$ can only lead to $K_{q+1} * P_4$ or $K_q * C_5$.
- If $H(u)$ has an induced C_3 then G has an induced $K_{1,3} = K_1 * K_3^c$.

The P_3 case requires some more work. If $H(u)$ has an induced P_3 (but none of P_4, C_4, C_5) then G has an induced subgraph with vertices x, y, u, z and edges xy, xu, yu, uz. Since uz has score 2, any new vertices adjacent to u or z are adjacent to both u and z (Lemma 4 (i)), hence also to x and y and to each other (as no further edges in $H(u)^c = N(u)$ exist by assumption). This yields only graphs of the form $K_q * (K_2 + K_1)$. Now let q be maximum, that is, further vertices that we add are adjacent to x or y only. If some added vertex v is adjacent to x only, we must have $q = 1$. Since vx, xu, uz have score 2, by Lemma 4 (i) and the assumptions of our case, we can add at most one further vertex, and this vertex is adjacent to y only, which yields a 3-asterisk. It remains the case that v is adjacent to x and y. First observe $q \leq 2$. If $q = 2$, we have a fat P_4. For $q = 1$, edges xu, yu, uz have score 2, hence any further vertex added is adjacent to v only, and we get a fat P_5.

Now we have settled all cases where $H(u)$ contains more than one edge, for some u. It remains to study connected score-2 graphs G where, for every u, the neighborhood $N(u)$ has at most one non-edge. Clearly, such G cannot have induced $K_{1,3}$. If G also lacks K_3 then the maximum degree in G is 2, hence G is P_n or C_n. Otherwise consider some maximal clique K_q, $q \geq 3$. Any vertex x added to this K_q has some neighbor u in the K_q, hence $x \in H(u)$. In G^c, vertex x is therefore adjacent to exactly one vertex in the K_q^c. It also follows that we can add at most one vertex to the considered K_q. This yields an induced $K_{q-1} * (2K_1)$. Since the same reasoning applies to the other K_q (where x replaces u), u is the only vertex that could be added. Hence $K_{q-1} * (2K_1)$ is already the entire G in this case. □

Theorem 5 is best possible in the sense that already score-3 graphs are not limited to special structures but can be arbitrarily complicated: Subdividing the edges of an arbitrary graph of degree 3 by further vertices generates a score-3 graph. However, we can still prove an interesting dichotomy for graphs of any fixed score s. Let $N^i(u)$ denote the set of vertices at distance exactly i from u.

Theorem 6 *Let G be any connected score-s graph. Then G has degree at most $(s+1)(2s+1)$, or G^c has degree at most $3s+1$.*

Proof. Let u be a vertex with maximum degree d in G. Note that $N(u) = N^1(u)$ contains d vertices. Since every edge uv, $v \in N(u)$ has score at most s, every vertex $v \in N(u)$ must be adjacent to at least $d-s-1$ other vertices in $N(u)$. Let $x \in N^2(u)$ be adjacent to some $v \in N(u)$. Since vx forms a P_3 with uv and is in at most $s-1$ other P_3, it follows that x is also adjacent to at least $d-2s-2$ of v's $d-s-1$ neighbors in $N(u)$, provided that $d > 2s-2$. Next, let $y \in N^3(u)$ be adjacent to some $x \in N^2(u)$. Since x has at least $d-2s-1$ neighbors in $N(u)$ as shown above, each of them is involved in a P_3 with xy, and xy has score at most s, it follows $d-2s-1 \leq s$, that is $d \leq 3s+1$, or $N^3(u) = \emptyset$.

Finally we limit $|N^2(u)|$. We use again that every vertex in $N^2(u)$ has at least $d-2s-1$ neighbors in $N(u)$, or equivalently, at most $2s+1$ non-neighbors in $N(u)$. Assume for a moment that $d > (s+1)(2s+1)$ and $|N^2(u)| \geq s+1$. Then some $v \in N(u)$ is still adjacent to $s+1$ vertices of $N^2(u)$, a contradiction since uv has score larger at most s. Thus we have $d \leq (s+1)(2s+1)$ and the assertion is proved, or we get $|N^2(u)| \leq s$. In the latter case, u and the vertices in $N(u)$ and $N^2(u)$ have degree at most s, $2s$, and $3s+1$, respectively, in G^c. □

5 FPT Algorithms Using the Conflict Triple Structure

As usual, $O(1)$ means "bounded by a certain constant".

Theorem 7 Cluster Deletion *is solvable in* $O(1.47^k + n^3)$ *time.*

Proof. As long as possible, delete an edge e of score larger than 2, or delete all edges building a P_3 with e. The branching number is 1.47. If this rule is not applicable, then every component is one of the graphs from Theorem 5. We show how to solve Cluster Deletion in polynomial time in each of these cases. For graphs with $O(1)$ vertices and for the C_n (and P_n) this is evident.

In $(qK_1 + pK_2)^c$ we take one vertex from each K_2^c to form a clique with p vertices, and the rest is a clique with $p+q$ vertices. This solution needs $p(p+q-1)$ edge deletions, whichis optimal: Any two of the p pairs K_2^c build an induced C_4, hence two edges must be deleted. Since all these C_4 are edge-disjoint, no deletion is counted twice. Thus we must delete at least $p(p-1)$ edges between these pairs. Moreover, each combination of the p pairs and the q vertices in the $(qK_1)^c$ part builds a P_3, and all these P_3 are edge-disjoint. Thus we must delete pq of these edges. The sum is $p(p+q-1)$.

The other graphs in Theorem 5 consist of one clique K with q vertices, joined with at most five other vertices. If we first disconnect these extra vertices from K, we always get a solution with at most $5q+3$ deletions. (The constant term 3 is easy to verify.) Assume that some solution disconnects r vertices from the other $q-r$ vertices of K, where $r \leq q/2$ without loss of generality. This costs already $r(q-r)$ deletions, which cannot be optimal unless $r(q-r) \leq 5q+3$. Since $q-r \geq q/2$, this yields $rq/2 \leq 5q+3$, hence $r \leq 10 + 6/q$ and finally $r \leq 10$ regardless of q. Since the q vertices of K are undistinguishable, we may select any 10 of them as candidate vertices for split-off. It follows that there exists an optimal solution that deletes edges only within some fixed set of at most 15 vertices, and we are back to the $O(1)$ size case.

Now we have treated all cases. The polynomial term in the time bound is dominated by the time needed to enumerate the P_3. Note that during the process of edge deletions, every triple of vertices becomes a new P_3 at most once. □

A straightforward modification of the algorithm can output a concise enumeration of all solutions (cf. [2]) within the same time bound. Theorem 6 has some interesting algorithmic consequences, too:

Corollary 8 *Let* $b > 1.3803$ *be any fixed base. If we can solve* Cluster Deletion *in* $O^*(b^k)$ *time for graphs of degree* $O(1)$*, we can also do so for general graphs.*

Proof. Theorem 6 gives that any connected score-3 graph G has degree at most 28, or G^c has degree at most 10. We first branch on edges of score 4 or larger, as long as possible. The branching number is 1.3803 or better. It remains a score-3 graph G which is connected without loss of generality, otherwise we consider the components separately.

In G^c we observe that the sum of degrees of any two vertices u, v with distance larger than 2 (in G^c) is at most 3, since non-edge uv belongs to at most three P_3^c. Consequently, G^c has at most one non-trivial component H with more than one edge. If the second case of Theorem 6 applies, the degree of G^c is $O(1)$, hence the size of H is also $O(1)$ (or we get again a forbidden pair u, v as above). That means, G has the form $(qK_1 + pK_2 + H)^c$ with a graph H of $O(1)$ size, and we can solve this case in polynomial time, similarly as in Theorem 7. If the first case of Theorem 6 holds, the degree of G is $O(1)$.

Hence, either a rule with branching number at most 1.3803 is available, or the instance can be solved in polynomial time, or G has degree $O(1)$. $\Box$

Remarkably, Corollary 8 implies that, in order to improve the base 1.47, we only need to consider score-3 graphs of bounded degree. This also suggests that the separation technique from Section 2-3 may be applicable, however it remains to work out how much reduction we can get in this way. For CLUSTER EDITING we get a similar "reduction to bounded degree" below.

Lemma 9 *For any $b > 1.62$ there exists s such that, if a graph has an edge of score larger than s then a branching rule for* CLUSTER EDITING *with branching number at most b is available.*

Proof. Consider an edge uv with score $s+1$, and let S be the set of those $s+1$ vertices that form P_3 with uv. We branch as follows: Either delete uv or keep it. If we keep uv, then for each $w \in S$ we must edit (insert or delete) one of the edges uw, vw. Accordingly, we refer to w as an insertion or deletion vertex. Now decide to make all $w \in S$ deletion vertices, or decide on some insertion vertex $w \in S$. In each of the last $s + 1$ branches continue as follows. Make every $y \in S \setminus \{w\}$ independently an insertion or deletion vertex. In both branches we must edit one of the edges uy, vy, and in one branch we must also edit wy, because any insertion (deletion) vertex must be adjacent (non-adjacent) to w. It is easy to verify that the branching number of the whole branching rule on S, u, v satisfies $x^{2s+1} \leq x^{2s} + x^s + (s + 1)(x + 1)^s$. This can be rewritten as $x \leq 1 + 1/x^s + (s + 1)((x + 1)/x^2)^s$. For any $x > 1.62$ we have $(x + 1)/x^2 < 1$. It follows that the branching number tends to 1.62 as s grows. $\Box$

If the score of G is at most our fixed s, then Theorem 6 applies. The case that G^c has degree $O(1)$ is easily settled, similarly as in Theorem 7:

Lemma 10 *In any class of graphs G where G^c has degree d,* CLUSTER EDITING *is solvable in polynomial time.*

Corollary 11 *Let $b > 1.62$ be any fixed base. If we can solve* CLUSTER EDITING *in $O^*(b^k)$ time for graphs of degree $O(1)$ (depending on b), we can also do so for general graphs.*

Proof. Combine Theorem 6 with Lemma 9 and 10. Either a rule with branching number at most b is available, or the instance is solved in polynomial time, or G has degree $O(1)$. $\Box$

We conclude with another observation supporting the conjecture that the separation technique is applicable to CLUSTER EDITING:

Proposition 12 *In graphs of degree d, all clusters in an optimal solution to* CLUSTER EDITING *have at most $2d + 1$ vertices.*

It is also interesting to notice that an optimal solution to CLUSTER EDITING in connected graphs of degree $O(1)$ needs $k = \Theta(n)$ edits, since the cluster size is limited (Proposition 12) and links between the clusters must be removed.

Acknowledgment

The work is supported by the Swedish Research Council (Vetenskapsrådet), grant 2007-6437, “Combinatorial inference algorithms – parameterization and clustering”.

References

[1] S. Böcker, S. Briesemeister, Q.B.A. Bui, A. Truß. Going weighted: Parameterized algorithms for cluster editing, *2nd COCOA 2008*, *LNCS* 5165, 1-12

[2] P. Damaschke. Fixed-parameter enumerability of cluster editing and related problems, *Theory of Computing Systems*, to appear

[3] P. Damaschke, L. Molokov. The union of minimal hitting sets: Parameterized combinatorial bounds and counting, accepted for *J. Discrete Algorithms*, preliminary version in: *24th STACS 2007*, *LNCS 4393*, 332-343

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

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

[6] F.V. Fomin, S. Gaspers, S. Saurabh, A.A. Stepanov. On two techniques of combining branching and treewidth, *Algorithmica*, to appear

[7] F.V. Fomin, K. Hoie. Pathwidth of cubic graphs and exact algorithms, *Information Processing Letters* 97 (2006), 191-196

[8] J. Gramm, J. Guo, F. Hüffner, R. Niedermeier. Automated generation of search tree algorithms for hard graph modification problems, *Algorithmica* 39 (2004), 321347

[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] F. Hüffner, C. Komusiewicz, H. Moser, R. Niedermeier. Fixed-parameter algorithms for cluster vertex deletion, *8th LATIN 2008*, *LNCS* 4957, 711-722

[11] J. Kneis, A. Langer, P. Rossmanith. Improved upper bounds for partial vertex cover, *34th WG 2008*, *LNCS* 5344, 240-251

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

[13] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*, Oxford Lecture Series in Mathematics and Its Applications, Oxford University Press 2006

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

Appendix

Proof of Lemma 2. Let $G_1, \dots, G_m$ be the components of G, and U_l the union of the first l components. By assumption we know the $c(U_1, j)$, $j \leq k$. Then we inductively calculate $c(U_{l+1}, j) = \sum_{i=0}^{j} c(U_l, i) \cdot c(G_{l+1}, j-i)$ for all $j \leq k$, and this for $l < m$. $\square$.

Proof of Lemma 4. If x is adjacent to u, and uv is a score-2 edge, then x must also be adjacent to v, since otherwise the edge uv gets score 3. Now (i) follows by an obvious inductive argument, and (ii) is an immediate consequence. For (iii) notice that one of the edges incident with x gets score 3, a contradiction, thus we cannot add a new vertex x. $\square$

Proof of Lemma 10. One solution is to insert the at most $dn/2$ missing edges to make G a clique. On the other hand, since every vertex has degree at least $n-1-d$ in G, separating more than $d/2$ vertices from the rest costs more than $dn/2$ deletions. Hence any optimal solution has a giant clique and some clusters of total size at most $d/2$. Even checking all potential solutions naively costs polynomial time, as $d = O(1)$. $\square$

Proof of Proposition 12. Consider a cluster C with c vertices in an optimal solution. Every vertex $v \in C$ must be adjacent to at least $(c-1)/2$ other vertices of C, since otherwise it would be cheaper to disconnect v from the rest of C to form a cluster of its own. Hence $(c-1)/2 \leq d$, and the assertion follows. $\square$