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 algorithms for FPT graph problems are search tree algorithms with sophisticated local branching rules. But it has also been noticed that using the global structure of input graphs complements the the search tree paradigm. Here we prove 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 "separating" branchings into cheaper or smaller components wich are then processed separately. Using this idea we 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 characterize the graphs where no edge is in three conflict triples, i.e., triples of vertices with exactly two edges. This theorem may find interest in graph theory, and it yields an $O^*(1.47^k)$ time algorithm for Cluster Deletion, improving upon the previous $O^*(1.53^k)$ (Gramm, Guo, Hüffner, Niedermeier, 2004). Cluster Deletion is the problem of deleting k edges to destroy all conflict triples and get 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 for future improvements via the above decomposition technique.

1 The Problems and Contributions

A problem is fixed-parameter tractable (FPT) if it can be solved in $O(p(n)f(k))$ time where p is a polynomial and f any function. We assume familiarity with the basic notions of FPT algorithms and their analysis [5,13]. Since we focus on the $f(k)$ factor, we sometimes adopt the $O^*(f(k))$ notation that suppresses polynomial factors. In graphs we usually denote by n the number of vertices. 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 graphs (or more generally, hitting sets of size k in hypergraphs 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 produced a given set of indicators, where

J. Chen and F.V. Fomin (Eds.): IWPEC 2009, LNCS 5917, pp. 98–109, 2009.

an indicator may come from several candidate substances. Substances and indicators are modeled as vertices and edges of a hypergraph. Since many different solutions of a certain 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 as well, 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 nontrivial technique. This gives an overall time bound of $O^*(1.3803^k)$, where the $(1, 4)$-branching 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 above algorithm but choose the initial $(1, 4)$-branchings 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 $O^*(1.26^k)$ time (with the residual k), or we can split the residual graph into several components, process them independently, and finally combine the partial results in polynomial time. Loosely speaking, due to this quasi-parallelization of the remaining branchings, only one component with the largest residual k counts for the exponential term in the complexity. In all cases this drives the overall branching number below 1.3803. The fact 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 essential. The time analysis is not straightworward either. Upon the $(1, 4)$-branchings we deduct not only 1 and 4 from parameter k, but also the number of vertices that will be added later to the vertex covers in a cheaper way. On the other hand, we must "charge" the $(1, 4)$-branchings for these deferred decisions and increase the size bounds of the search trees after the branching accordingly, as every leaf in the search tree now represents the deferred decisions. But since they are cheaper, this pays off in the end.

This global technique of branching away the hardest parts and splitting 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 that decompose neatly.

The bound for Vertex Cover Counting we can prove so far is $O^*(1.3740^k)$. Seemingly this is marginal progress, however, the weakness is apparently in the current analysis rather than in the algorithm itself. Our analysis gives something away, hence the true *worst-case* bound is likely to be way better. But it is hard to take advantage of those observations, since different bad cases can appear 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, or conflict triple, is a path of three vertices (and two edges). Cluster graphs are exactly the graphs without induced P_3. In Cluster Deletion we want to delete at most k edges in a graph G so as to obtain a cluster graph. That is, deletions must destroy all induced P_3. Cluster Editing is similarly defined with edge deletions *and* insertions. Both problems are special

cases of Weighted Cluster Editing where pairs of vertices have individual edits costs. These NP-hard problems [14] have also applications foremost in computational biology [4,9,14]. Cluster Deletion becomes important if only adjacent vertices (representing similar objects) are tolerated in any cluster.

The parameterized complexity of these problems is well studied. In [2] we give problem kernels and algorithms for enumerating *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 from [8]. It was an example of an automatically generated search tree algorithm and improved a "handmade" $O(1.77^k + n^3)$ algorithm [9]. It works with branching rules on subgraphs with at most six vertices. No further progress on Cluster Deletion has been made since then.

Here we come back to handicraft and give an $O(1.47^k + n^3)$ time algorithm for Cluster Deletion. It starts from the most obvious rule with branching number 1.47: If some edge is in 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, and no other local branching rules are needed. (Only special cases were already treated in [1, Lemma 4] and in a conference version of [2].)

The result give new indications 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. In fact, the $O(1.3803^k)$ algorithm for Weighted Vertex Cover in [6] is close to the Vertex Cover Counting results, but here we add our separation idea.

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 some 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 completely characterize the graphs where no edge is in three P_3. In the proof we tried to avoid too many tiresome case distinctions. More generally, for graphs where every edge is in an $O(1)$ number of P_3 we prove a dichotomy: the graph or its complement graph has degree $O(1)$. Section 5 deals with the algorithmic consequences of P_3 structure. We solve Cluster Deletion in $O^*(1.47^k)$ time. Improving some details in the polynomial-time parts 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 separation technique again. We discuss these possibilities also for Cluster Editing. Due

to the page limit, some simpler proofs are omitted, graph-theoretic notions are only briefly reviewed, and there is no space for figures.

2 Vertex Cover Counting Algorithm

Let $c(G, k)$ be the number of vertex covers with exactly k vertices in graph G. The *degree* of a vertex v is the number of vertices adjacent to v. A *degree-d vertex* has degree exactly d. The degree of a graph is the maximum vertex degree. A *subcubic* graph has degree 3. 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 is computable in polynomial time for any fixed $\epsilon > 0$ [7]. By dynamic programming on this tree decomposition one can count the vertex covers of size k in $O^*(2^{(1/3+\epsilon)k}) = O^*(1.26^k)$ time [12], thus we have:

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

If 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 one easily concludes:

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

Branching on a vertex v means: Either put v or all neighbors of v in the vertex cover, remove the chosen vertices, all incident edges, and isolated vertices.

We refer to a series of such branching decisions on different vertices as a *branch*. Thus, a branch corresponds to a node of the resulting search tree. The *residual graph* in a branch is the graph that remains after the branchings. 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.

Phase 1: preparation. First branch, as long as possible, on degree-d vertices with $d \geq 5$. In every branch consider the residual graph of degree 4 and continue as described below. We will declare certain vertices *roots*.

Phase 2: separation. If a component without a root exists, then fix any vertex in this component as the *active root* and proceed as follows. Layer j contains the vertices at distance j from the active root. As long as degree-4 vertices exist in some layer $j > 2$ of this component, choose one such degree-4 vertex closest to the active 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 $j > 2$. At this moment, set the active root passive. – Iterate.

Phase 3: completion. Branch on the remaining degree-4 vertices near the roots. This results in a subcubic graph. Let $m \leq k$ be the number of vertices

that remain to be added to the vertex cover. (Note that m depends on the branch but is exactly known in every branch.) Compute the $c(G', i)$ in all components G' of the residual graph, and for all $i \leq m$, 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 phase 2 we refine the rule for selecting 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

A simple analysis would show that the residual graph after phase 2 (separation) always needs a vertex cover of at least yk vertices for some fraction y, giving a time bound $O^*(1.38028^{k-y}1.26^y)$. However y is very small in branches where often 4 vertices are taken. We got a better result by relating these "cheap" vertices in phase 3 directly to the number of branchings in phase 2.

Theorem 1. *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 phase 1 is the positive root of $x^5 = x^4 + 1$, that is, $x < 1.3248$.

For phase 2 the following observation is crucial: Since we always pick a degree-4 vertex in a layer $j > 2$ closest to the root, and removals can make the distances in the remaining graph 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 is 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 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 into a layer with higher index later.

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* $p(v)$ which is a neighbor of v in layer $j - 1$, and a *grandparent* $g(v)$ 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. 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".

We assign to certain vertices *certificates*: Consider any vertex v and its sibling v' in layer j. *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 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 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 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 to layer $i - 1$, at least $r/2$ edges remain after phase 2.

Let us count the branchings on the remaining degree-4 vertices near the roots already in phase 2. Since in each residual graph only one search tree for the components is decisive for the complexity (only the largest one, due to the polynomial-time combination procedure from Lemma 2) and every component has only $O(1)$ degree-4 vertices at that moment, this adds only a constant factor to the complexity. On the other hand, the graphs in phase 3 are now subcubic.

We call the vertices inserted in the vertex cover in phase 2 and 3 expensive and cheap vertices, respectively. Let $T(k, l)$ be the number of leaves of a search tree, when it remains to add $k + l$ vertices at most k of which are expensive and at least l are cheap. Initially, k is the given k, and $l = 0$. In the following, fractional numbers of objects make sense because their sum is finally rounded to the next integer. (They could be avoided by doing an equivalent analysis for blocks of 6 consecutive branchings.) As seen above, for every branching in phase 2, either on a single vertex or on siblings v, v' of degree 4, at least $1/2$ edges need to be covered later in phase 3. Since the maximum degree is 3, at least $1/6$ cheap vertices will be added later. Thus it is safe to transmit $1/6$ from k to l. This yields $T(k, l) \leq T(k - 1 - 1/6, l + 1/6) + T(k - 4 - 1/6, l + 1/6)$ and $T(k, l) \leq T(k-2-1/6, l+1/6)+T(k-4-1/6, l+1/6)+T(k-5-1/6, l+1/6)$ in the two cases. Together with $T(0, l) \leq 1.26^l$ from Lemma 1 we obtain recurrences for $T(k) := T(k, 0)$, the number of leaves in the overall search tree: Since every branching in phase 2 finally incurs another $1.26^{1/6}$ factor in all summands, we have $T(k) \leq (T(k - 1 - 1/6) + T(k - 4 - 1/6))1.26^{1/6}$ and similarly $T(k) \leq (T(k - 2 - 1/6) + T(k - 4 - 1/6) + T(k - 5 - 1/6))1.26^{1/6}$, with numerical solutions 1.3699^k and 1.3740^k, respectively. □

Our analysis guarantees the existence of at least one persistent vertex, with an edge attached, for any two $(1, 4)$-branchings made. We "certify" only grandparents and ignore further persistent edges, especially in the large component relevant for the complexity. If the analysis could use all persistent vertices, this would almost double the deduction from the parameter. Refinements of the algorithm itself may give further improvements: The worst case in our analysis appears if every vertex in the largest component has two children. But then the component is merely a binary tree, and vertex covers could be counted trivially there. Finally, some clever measure-and-conquer might yield a simpler analysis.

4 Conflict Triple Structure in Graphs

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, and pG is the disjoint union of p copies of G. The join $G * H$ 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 belongs to. A graph is score-s if every edge has score at most s. Note that a score-s graph is also score-$(s + 1)$, and an induced subgraph of a score-s graph is score-s. A main goal of this section is to characterize the connected score-2 graphs G. We say that G is *spanning score-2* if G has a spanning tree of score-2 edges. To *add a vertex* to a connected graph G means to introduce a new vertex 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 3. *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 non-adjacent to at least three other vertices of G, then G is maximal score-2.

Doubling a vertex x of a graph means to insert a new vertex adjacent exactly to x and its neighbors. The result of doubling several vertices does not depend on the order. Now we define several special 6-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 2. *The following graphs (with arbitrarily large positive n, q, p) and their connected induced subgraphs comprise the complete list of 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 of them 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 conditions of Lemma 3 (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 keeping 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 3 (ii), every added vertex is adjacent to all its $2p$ 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, as this would create new edges of score above 2.

Thus we have shown that our list contains only score-2 graphs and is closed under vertex addition. Next we prove that no further cases exist, that is, any connected score-2 graph G is mentioned in the Theorem.

Let $N(u)$ denote the set of neighbors of a vertex u in G. In G^c this means, $N(u)$ is the set of all non-neighbors of 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.

- 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 3 (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 3 (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, then $q = 1$. Since vx, xu, uz have score 2, by Lemma 3 (i) and the assumptions of our case, we can add at most one further vertex, and

this one 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 2 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 any 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 3. *Let G be any connected graph of score s graph that has more than $(s+1)^2(2s+1)^2 + 1 \sim 4s^4$ vertices. Then G or G^c has degree at most $3s+1$.*

Proof. Let u be a vertex of maximum degree d in G, hence $N(u)$ has 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 $x \in N^2(u)$, and let $v \in N(u)$ be some neighbor of x. Since vx forms a P_3 with uv, it is in at most $s - 1$ other P_3. It follows that x is adjacent to at least $d - 2s - 2$ of v's $d - s - 1$ neighbors in $N(u)$, provided that $d > 2s - 2$. Next, let $y \in N^3(u)$, and let $x \in N^2(u)$ be some neighbor of y. As shown above, x has at least $d - 2s - 1$ neighbors in $N(u)$, each of which is involved in a P_3 with xy, and xy has score at most s, we get $d - 2s - 1 \leq s$. This shows $d \leq 3s + 1$ or $N^3(u) = \emptyset$.

In case $d \leq 3s + 1$ we are done, so let $N^3(u) = \emptyset$. If $d \leq (s + 1)(2s + 1)$ then, since d is the maximum degree, we have $|N^0(u)| + |N^1(u)| + |N^2(u)| \leq (s+1)^2(2s+1)^2 + 1$. Hence assume $d > (s + 1)(2s + 1)$ in the following. Recall again that every vertex in $N^2(u)$ has at least $d - 2s - 1$ neighbors in $N(u)$, that is, at most $2s + 1$ non-neighbors in $N(u)$. Thus, if $|N^2(u)| \geq s + 1$ then some $v \in N(u)$ is still adjacent to $s + 1$ vertices of $N^2(u)$. But uv has score at most s. This contradiction shows $|N^2(u)| \leq s$. Now we see that the vertices in $N^0(u)$, $N^1(u)$, and $N^2(u)$ have, in G^c, degree at most s, $2s + 1$, and $3s + 1$. □

5 FPT Algorithms Using the Conflict Triple Structure

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

Theorem 4. Cluster Deletion *is solvable in* $O(1.47^k + n^3)$ *time.*

Proof. As long as possible, take an edge e of score larger than 2, and delete e or all edges forming a P_3 with e. The branching number is 1.47. Once the graph is score-2, every component is one of the graphs from Theorem 2. We show how to solve Cluster Deletion in polynomial time in these cases. For graphs of size $O(1)$ and for 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 of size p, and the rest is a clique of size $p + q$. This solution needs $p(p + q - 1)$ edge deletions, which is 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 $(qK_1)^c$ 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 2 consist of one clique K of size q, 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. (Summand 3 is easy to verify.) Assume that some solution disconnects some $r \leq q/2$ vertices from the other $q - r$ vertices of K. 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 vertices of K are undistinguishable, we may select any 10 of them as candidate vertices for split-off. Thus there exists an optimal solution that deletes edges only between at most 15 predefined vertices, and we are back to the $O(1)$ size case.

In 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. □

With minor modifications, the algorithm can output a concise enumeration of all solutions (cf. [2]) within the same time bound. Theorem 3 has some interesting algorithmic consequences, too:

Corollary 1. *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 do so for general graphs.*

Proof. As long as possible, branch on edges of score 4 or larger. The branching number is 1.3803. It remains a score-3 graph G. Assume G is connected, otherwise we consider the components separately. Theorem 3 gives that G or G^c has degree at most 10 (or G has $O(1)$ size).

In G^c we also 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 3 applies, the degree of G^c is $O(1)$, thus H has size $O(1)$ (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 size $O(1)$, and we can solve this case in polynomial time, similarly as in Theorem 4. If the first case of Theorem 3 holds, the degree of G is $O(1)$.

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

Remarkably, Corollary 1 implies that efforts to improve the base 1.47 only need to consider score-3 graphs of $O(1)$ degree. This also suggests that the separation technique from Section 2-3 may be applicable. We have to leave the question how much we can gain in this way for further research. For Cluster Editing we get a similar "reduction to fixed degree" below.

Lemma 4. *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 this whole branching rule on S, u, v satisfies $x^{2s+1} \leq x^{2s} + x^s + (s+1)(x+1)^s$, equivalently $x \leq 1 + 1/x^s + (s+1)((x+1)/x^2)^s$. For any $x > 1.62$ we have $(x+1)/x^2 < 1$, hence the branching number tends to 1.62 as s grows. □

If the score of G is at most our fixed s, then Theorem 3 applies. The case that G^c has degree $O(1)$ is easily settled, similarly as in Theorem 4:

Lemma 5. *In any class of graphs G where G^c has degree d,* Cluster Editing *is solvable in polynomial time.*

Corollary 2. *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 3 with Lemma 4 and 5. □

We conclude with another observation supporting the conjecture that the separation technique is applicable to Cluster Editing:

Proposition 1. *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 1) and links between the clusters must be removed.

Acknowledgment

This work has been supported by the Swedish Research Council (Vetenskapsrådet), grant 2007-6437, "Combinatorial inference algorithms – parameterization and clustering".

References

1. Böcker, S., Briesemeister, S., Bui, Q.B.A., Truß, A.: Going Weighted: Parameterized Algorithms for Cluster Editing. In: Yang, B., Du, D.-Z., Wang, C.A. (eds.) COCOA 2008. LNCS, vol. 5165, pp. 1–12. Springer, Heidelberg (2008)
2. Damaschke, P.: Fixed-Parameter Enumerability of Cluster Editing and Related Problems. Theory Comp. Systems (to appear)
3. Damaschke, P., Mololov, L.: The Union of Minimal Hitting Sets: Parameterized Combinatorial Bounds and Counting. J. Discr. Alg. (to appear)
4. Dehne, F., Langston, M.A., Luo, X., Pitre, S., Shaw, P., Zhang, Y.: The Cluster Editing Problem: Implementations and Experiments. In: Bodlaender, H.L., Langston, M.A. (eds.) IWPEC 2006. LNCS, vol. 4169, pp. 13–24. Springer, Heidelberg (2006)
5. Downey, R.G., Fellows, M.R.: Parameterized Complexity. Springer, Heidelberg (1999)
6. Fomin, F.V., Gaspers, S., Saurabh, S., Stepanov, A.A.: On Two Techniques of Combining Branching and Treewidth. Algorithmica (to appear)
7. Fomin, F.V., Hoie, K.: Pathwidth of Cubic Graphs and Exact Algorithms. Info. Proc. Letters 97, 191–196 (2006)
8. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Automated Generation of Search Tree Algorithms for Hard Graph Modification Problems. Algorithmica 39, 321–347 (2004)
9. Gramm, J., Guo, J., Hüffner, F., Niedermeier, R.: Graph-Modeled Data Clustering: Fixed-Parameter Algorithms for Clique Generation. Theory Comp. Systems 38, 373–392 (2005)
10. Hüffner, F., Komusiewicz, C., Moser, H., Niedermeier, R.: Fixed-Parameter Algorithms for Cluster Vertex Deletion. In: Laber, E.S., Bornstein, C., Nogueira, L.T., Faria, L. (eds.) LATIN 2008. LNCS, vol. 4957, pp. 711–722. Springer, Heidelberg (2008)
11. Kneis, J., Langer, A., Rossmanith, P.: Improved Upper Bounds for Partial Vertex Cover. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 240–251. Springer, Heidelberg (2008)
12. Mölle, D., Richter, S., Rossmanith, P.: Enumerate and Expand: New Runtime Bounds for Vertex Cover Variants. In: Chen, D.Z., Lee, D.T. (eds.) COCOON 2006. LNCS, vol. 4112, pp. 265–273. Springer, Heidelberg (2006)
13. Niedermeier, R.: Invitation to Fixed-Parameter Algorithms. Oxford Lecture Series in Math. and its Appl. Oxford Univ. Press, Oxford (2006)
14. Shamir, R., Sharan, R., Tsur, D.: Cluster Graph Modification Problems. Discr. Appl. Math. 144, 173–182 (2004)