On the Complexity of the Highly Connected Deletion Problem

Falk Hüffner [1], Christian Komusiewicz [1], Adrian Liebtrau [2], and Rolf Niedermeier [1]

[1]Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany

[2]Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany

January 7, 2013

Abstract

Removing as few edges as possible from a graph such that the resulting graph consists of highly connected components is a natural problem in graph-based data clustering. Building on work by Hartuv and Shamir [Inf. Proc. Lett. 2000], we introduce the corresponding combinatorial optimization problem Highly Connected Deletion. We show that Highly Connected Deletion is NP-hard even on 4-regular graphs and provide fixed-parameter tractability results for the problem.

1 Introduction

A key idea of graph-based data clustering is to identify highly connected subgraphs (clusters) that have many interactions within themselves and few with the rest of the graph [1, 22, 28, 29]. Hartuv and Shamir [9] proposed a clustering algorithm producing highly connected clusters. Their method has been successfully used to cluster cDNA fingerprints [10], to find complexes in protein–protein interaction (PPI) data [25], to group protein sequences hierarchically into superfamily and family clusters [18], and to find families of regulatory RNA structures [24]. Hartuv and Shamir [9] formalized the connectivity demand for a cluster as follows: the *edge connectivity* $\lambda(G)$ of a graph G is the minimum number of edges whose deletion results in a disconnected graph, and a graph G with n vertices is called *highly connected* if $\lambda(G) > n/2$. An equivalent characterization is that a graph is highly connected if each vertex is connected to a majority of the other vertices [3]. This demonstrates that the concept of a highly connected graph is very similar to that of a *0.5-quasi-complete graph* [14, 21], that is, a graph where every vertex has degree at least $(n-1)/2$. Further, it

can be shown that being highly connected also ensures that the diameter of a cluster is at most two [9].

The algorithm by Hartuv and Shamir [9] partitions the vertex set of the given graph such that each partition set is highly connected, thus guaranteeing good intra-cluster density (including maximum cluster diameter two and the presence of more than half of all possible edges). Moreover, the algorithm needs no prespecified parameters (such as the number of clusters) and it naturally extends to hierarchical clustering. Essentially, Hartuv and Shamir's algorithm iteratively deletes the edges of a minimum cut in a connected component that is not yet highly connected.[1] While Hartuv and Shamir's algorithm guarantees to output a partitioning into *highly connected* subgraphs, it does not guarantee to achieve this by minimizing inter-cluster connectivity. In other words, it is not ensured that the partitioning comes along with a *minimum number of edge deletions* making the resulting graphs consist of highly connected components. This is why here, "on top" of Hartuv and Shamir's work, we propose a formally defined *combinatorial optimization problem* that additionally specifies the goal to minimize the number of edge deletions.

Highly Connected Deletion
Instance: An undirected graph $G = (V, E)$.
Task: Find a minimum subset of edges $E' \subseteq E$ such that in $G' = (V, E \setminus E')$ all connected components are highly connected.

The formulation above resembles the Cluster Deletion problem [27], which asks for a minimum number of edge deletions to make each connected component a clique; thus, Cluster Deletion has a much stronger demand on intra-cluster connectivity. Also related is the 2-Club Deletion problem [19], which asks for a minimum number of edge deletions to make each connected component have a diameter of at most two. Since highly connected clusters have diameter at most two [9], 2-Club Deletion poses a looser demand on intra-cluster connectivity. Note that by definition, isolated edges are *not* highly connected. Hence, the smallest clusters are triangles; we consider all singletons as *unclustered*.

Since the algorithm by Hartuv and Shamir [9] repeatedly deletes the edges of a minimum cut from a component that is not yet highly connected, it could be expected that it yields a good approximation for the optimization goal of Highly Connected Deletion. However, we can observe that in the worst case, its result can have size $\Omega(k^2)$, where $k := |E'|$ is the size of an optimal solution. For this, consider two cliques with vertex sets $u_1, \ldots, u_n$ and $v_1, \ldots, v_n$, respectively, and the additional edges $\{u_i, v_i\}$ for $2 \leq i \leq n$. Then these additional edges form a solution set of size $n - 1$; however, Hartuv and Shamir's algorithm will (with unlucky choice of minimum cuts) transform one of the two cliques into an independent set by repeatedly cutting off one vertex, thereby

[1] The CLICK algorithm [6] and the SIDES algorithm [17] follow the same scheme, but use edge weights and different stopping criteria, based on probabilistic models. Hu et al. [11] present a variant that can also produce overlapping clusters.

deleting $n(n+1)/2-1$ edges. This also illustrates the tendency of the algorithm to cut off size-1 clusters, which Hartuv and Shamir tried to counteract with postprocessing [9]. This tendency might introduce systematic bias [11, 17]. Hence, exact methods for solving Highly Connected Deletion make sense.

Our contributions. We analyze the (parameterized) computational complexity of Highly Connected Deletion.

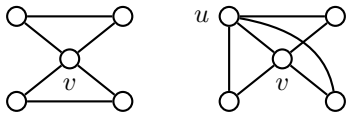
In particular, we show that Highly Connected Deletion is NP-hard even on 4-regular graphs and, provided the Exponential Time Hypothesis (ETH) [13] does not fail, cannot be solved in subexponential time. In addition, we provide strong polynomial-time executable data reduction rules (on the theoretical side also yielding a so-called problem kernel of polynomial size) and a fixed-parameter algorithm based on dynamic programming; both these results exploit the parameter "number of edge deletions".

Preliminaries. We consider only undirected and simple graphs $G = (V, E)$. We use n to denote the number of vertices in the input graph and m to denote the number of edges in the input graph, and k for the minimum size of an edge set whose deletion makes all components highly connected. The *order* of a graph G is the number of vertices in G. We use $G[S]$ to denote the *subgraph induced* by $S \subseteq V$. Let $N(v) := \{u \mid \{u, v\} \in E\}$ denote the *(open) neighborhood* of v and $N[v] := N(v) \cup \{v\}$. A *minimum cut* of a graph G is a smallest edge set E' such that deleting E' increases the number of connected components of G.

We view Highly Connected Deletion also as a parameterized problem [4, 5, 23], where the parameter is the number k of edge deletions. A parameterized problem with input size s is called *fixed-parameter tractable* (FPT) with respect to a parameter k if it can be solved in $f(k) \cdot s^{O(1)}$ time, where f is a computable function only depending on k. A *problem kernel* for a parameterized problem is a many-one polynomial-time self-reduction such that the produced instances have size upper-bounded by some function of the parameter. Usually, a problem kernel is achieved by applying polynomial-time executable data reduction rules. Referring to decision problems, we call a data reduction rule $\mathcal{R}$ *correct* if the new problem instance I' that results from applying $\mathcal{R}$ to the original instance I is a yes-instance if and only if I is a yes-instance. The Exponential Time Hypothesis (ETH) [13] states that 3-Sat cannot be solved in subexponential time. Several results on (relative) lower bounds for exponential-time algorithms have been shown by exploiting the ETH, see Lokshtanov et al. [20] for a recent survey.

2 Computational Complexity

In this subsection, we present a proof that Highly Connected Deletion is NP-hard even on 4-regular graphs and that it cannot be solved in $2^{o(k)} \cdot n^{O(1)}$, $2^{o(n)} \cdot n^{O(1)}$, or $2^{o(m)} \cdot n^{O(1)}$ time unless the exponential-time hypothesis (ETH) [13] is false.



Type I Type II

Figure 1: The two different neighborhoods in a 4-regular neighborhood restricted graph. None of these graphs contains a clique of order four.

The Partition Into Triangles problem is to partition the vertex set of a graph into $n/3$ sets such that each set induces a triangle. Partition Into Triangles is NP-hard and cannot be solved in $2^{o(n)} \cdot n^{O(1)}$ time unless the ETH is false, even for so-called 4-*regular neighborhood-restricted graphs* [26]. A reduction from Partition Into Triangles to Highly Connected Deletion would be easy if we could make sure the only highly connected subgraphs are triangles. We can show that 4-regular neighborhood-restricted graphs already cannot contain a highly connected subgraph of size four, five, or at least eight. The possibly remaining highly connected subgraphs of size six or seven can be eliminated with a data reduction rule, which completes the reduction.

As recently shown by van Rooij et al. [26], the Partition Into Triangles problem is NP-hard even when the input graph $G = (V, E)$ is 4-regular. Moreover, NP-hardness persists even when for each vertex $v \in V$ the graph $G[N[v]]$ is isomorphic to one of the two graphs shown in Fig. 1 [26]; we refer to such graphs as 4-*regular neighborhood-restricted graphs*. The variant of Partition Into Triangles which we will use in the reduction thus is:

> Restricted Partition Into Triangles (RPIT)
> **Instance:** An undirected 4-regular neighborhood restricted graph $G = (V, E)$.
> **Question:** Can V be partitioned into $|V|/3$ sets such that each set of the partition induces a triangle, that is, a complete graph on three vertices, in G?

Our reduction is similar to a simple reduction that was used to show NP-hardness of Cluster Deletion on graphs of maximum degree four [16]. In this reduction, the graph remains basically the same, and one just has to find the appropriate k. The main difference is that for Highly Connected Deletion we have to show that there can be no clusters larger than triangles in G (in the case of Cluster Deletion this is easier since the cluster size is at most five in 4-regular graphs). In the following, we present two observations and a data reduction rule for RPIT and then use them to obtain a reduction from RPIT to Highly Connected Deletion. The overall aim is to show that we can assume for our reduction that all clusters are triangles.

Lemma 1. *Let $G = (V, E)$ be a 4-regular neighborhood-restricted graph. Then G*

does not contain any highly connected subgraph of order four, five, or at least eight.

Proof. Obviously, G does not contain highly connected subgraphs of order at least eight, since G is 4-regular and any highly connected graph of order at least eight has minimum degree five. Furthermore, the only highly connected graphs of order four are cliques on four vertices. Since G has only the neighborhoods shown in Fig. 1, it does not contain cliques of order four.

It remains to show that G does not contain highly connected subgraphs of order five. The main observation we use is that, in such a graph, every vertex has degree at least three. Let v be a vertex in G. Since $G[N[v]]$ contains at least two vertices of degree two (see Fig. 1), $G[N[v]]$ is not highly connected. Hence, if v is contained in a highly connected subgraph G' of order five, then G' contains at least one vertex from $V \setminus N[v]$ and exactly one vertex from $N[v]$ is not in G'. If $G[N[v]]$ is of Type I (see Fig. 1, then v is not contained in a highly connected subgraph of order five: deleting one vertex from $G[N[v]]$ produces one vertex with degree one and this vertex cannot obtain degree at least three by adding one vertex. Hence, assume that $G[N[v]]$ is of Type II. Clearly, every highly connected subgraph of order five containing v also has to contain u, since otherwise one creates again a degree-one vertex. Note that u and v have the same neighborhood. Since G is 4-regular, there is no vertex in $V \setminus N[v]$ that is adjacent to u or v. Consequently, every vertex of $V \setminus N[v]$ has degree at most two in a subgraph of G that has order five and contains u and v. Hence, there is no highly connected subgraph of order five that contains v. □

We now present one data reduction rule that removes small connected components from the RPIT instance.

Rule 1. *Let $G = (V, E)$ be an instance of* RPIT*. If G contains a connected component C of order at most seven, then check whether C can be partitioned into triangles. If this is the case; then remove C from G, otherwise, answer "no".*

The correctness of the reduction rule is obvious. Furthermore, it results in an instance with the following property.

Lemma 2. *Let $G = (V, E)$ be an instance of* RPIT *that is reduced with respect to Rule 1. Then, G does not contain any highly connected subgraphs of order six or seven.*

Proof. Assume that G contains a highly connected subgraph $G' = (V', E')$ on six vertices. Then, each vertex in G' has at least four neighbors in G'. Consequently, no vertex of G' has in G any neighbors in $V \setminus V'$. Hence, G' is a connected component of G. This contradicts the fact that G is reduced with respect to Rule 1. A similar argument applies for highly connected subgraphs of order seven. □

Theorem 1. Highly Connected Deletion *on 4-regular graphs is NP-hard and cannot be solved in* $2^{o(k)} \cdot n^{O(1)}$, $2^{o(n)} \cdot n^{O(1)}$, *or* $2^{o(m)} \cdot n^{O(1)}$ *time unless the exponential-time hypothesis (ETH) is false.*

Proof. We reduce from RPIT which is NP-hard and cannot be solved in $2^{o(n)} \cdot n^{O(1)}$ time unless the ETH is false [26]. Given an instance of RPIT, first apply Rule 1. Let $G = (V, E)$ be the resulting instance. We obtain an instance of Highly Connected Deletion by setting $k := |V|$.

The equivalence of the instances can be seen as follows. If G has a partition into triangles, then each of these triangles is a highly connected subgraph. The number of triangles is $|V|/3$ and the overall number of edges contained in these triangles is $|V|$. Since G is 4-regular, $|E| = 2|V|$. Hence, G can be transformed by at most k edge deletions into a highly connected cluster graph.

Conversely, if G can be transformed into a highly connected cluster graph G' by at most $|V|$ edge deletions, then G' has at least $|V|$ edges. By Lemmas 1 and 2, no cluster in G' has order at least four. Hence, all clusters are triangles or singletons. Since G' has $|V|$ edges, all clusters are triangles. Therefore, G can be partitioned into triangles.

Clearly, the reduction implies NP-hardness of Highly Connected Deletion on 4-regular graphs. The ETH-based lower bounds follow from the fact that $|V| = k = |E|/2$. □

2.1 Algorithms

In this section, we provide polynomial-time data reduction rules which reduce an instance of Highly Connected Deletion to an equivalent one with at most $10k^{1.5}$ vertices. Thus, after reduction, the instance size depends solely on k, implying a *problem kernel* [8] with respect to the parameter k. Further, we present a fixed-parameter algorithm for Highly Connected Deletion with running time $O(3^{4k} \cdot k^2 + n^2mk \cdot \log n)$. These results imply the fixed-parameter tractability of Highly Connected Deletion with respect to k and give hope for finding optimal solutions for instances where k is not too large.

2.2 Data Reduction and Problem Kernel.

The first data reduction rule is obvious.

Rule 2. *Remove all connected components from G that are highly connected.*

The following lemma can be proved by a simple counting argument.

Lemma 3. *Let G be a highly connected graph and u, v two vertices in G. If u and v are connected by an edge, they have at least one common neighbor; otherwise, they have at least three common neighbors.*

Proof. Let n_{uv} be the number of common neighbors of u and v and n_u and n_v the number of neighbors specific to u and v, respectively (excluding u and v). Let c be 1 if $\{u, v\} \in E$ and 0 otherwise. We have $n_{uv} + n_u + c \geq (n + 1)/2$

and $n_{uv} + n_v + c \geq (n+1)/2$, thus $2n_{uv} + n_u + n_v + 2c \geq n + 1$. Since $n \geq n_{uv} + n_u + n_v + 2$, we get $n_{uv} + 2c - 2 \geq 1$, thus $n_{uv} \geq 3 - 2c$. □

A simple data reduction rule follows directly from Lemma 3.

Rule 3. *If there are two vertices u and v with $\{u, v\} \in E$ that have no common neighbors, then delete $\{u, v\}$ and decrease k by one.*

Interestingly, these two rules can be used to obtain a linear-time algorithm for Highly Connected Deletion on graphs of maximum degree three:

Theorem 2. Highly Connected Deletion *can be solved in linear time when the input graph has degree at most three.*

Proof. We first apply Rule 3. This reduction rule can be applied in one pass since an edge that is in a triangle is never deleted by this rule. Consequently, the application of this rule does not produce new vertices u and v to which this rule applies. Hence, Rule 3 can be exhaustively applied in $O(n + m)$ time: for each edge in G we examine the neighborhoods of its endpoints; since G has maximum degree three this neighborhood has constant size. Next, we apply Rule 2, which can also be performed in linear time. After the application of this rule, G is reduced with respect to both rules.

Consider a connected component in G. We show that G contains only four vertices. Let $\{u, v\}$ be some edge in this connected component. Since G is reduced with respect to Rule 3, there is a vertex w that is a common neighbor of u and v. Since G is reduced with respect to Rule 2, one of these three vertices, say v has a further neighbor x. Now, x has a common neighbor with v, say u. The connected component does not contain any further vertices: First, u and v can have no further neighbors since G has maximum degree three. Second, w and x cannot be adjacent since then the connected component is a clique of order four which contradicts that G is reduced with respect to Rule 2. Finally, neither x nor w have a further neighbor since this neighbor has to be adjacent to either u or v, which already have degree three. Hence, each remaining connected component can be solved in constant time. □

As computational experiments demonstrate [12], Rule 3 tremendously simplifies many real-world input instances. The next two data reduction rules are concerned with finding vertex sets that have a small edge cut. For $S \subseteq V$, we use $D(S) := \{\{u, v\} \in E \mid u \in S \wedge v \in V \setminus S\}$ to denote the set of edges outgoing from S, that is, the edge cut of S.

The idea behind the next reduction rule is to find vertex sets that cannot be separated by at most k edge deletions. We call two vertices u and v *inseparable* if the minimum edge cut between u and v is larger than k. Analogously, a vertex set S is inseparable if all vertices in S are pairwise inseparable.

Rule 4. *If G contains a maximal inseparable vertex set S of size at least $2k$, then do the following. If $G[S]$ is not highly connected, then return "no". Otherwise, remove S from G and set $k := k - |D(S)|$.*

Lemma 4. *Rule 4 is correct and can be exhaustively applied in $O(n^2 \cdot mk \log n)$ time.*

Proof. We show that the rule produces equivalent instances. First, assume that (G, k) is a yes-instance. Clearly, an inseparable vertex set S has to be subset of a cluster C in the solution graph. Now, since $|S| \geq 2k$ there can be no vertex in $C \setminus S$: Assume that C contains such a vertex. Then by the maximality of S, the graph $G[C]$ has an edge cut of size at most k. Since $|C| > 2k$, this means that $G[C]$ is not highly connected. Hence, S is a cluster in the solution and thus $G[S]$ is highly connected. The rule performs precisely the edge deletions needed to cut S from $V \setminus S$ and reduces the parameter accordingly. Hence, it produces a yes-instance.

Now, if the instance is a no-instance, then either the rule returns "no" or performs some edge deletions and reduces the parameter accordingly. This cannot transform a no-instance into a yes-instance.

We now describe how to achieve an exhaustive application of the rule in the described running time. First, build a so-called Gomory-Hu tree in $O(n^2 \cdot m \log n)$ time [7, 15]. This tree has n vertices and the set of weighted edges represents all pairwise min-cuts. A maximal inseparable vertex sets can be found by deleting all edges that have weight at most k. Once this set has been identified, the application of the reduction rule can be performed in $O(m)$ time. Since the reduction rule can be performed at most k times (it answers either "no" or reduces k), the overall running time follows. □

Note that a highly connected graph of size at least $2k$ is an inseparable vertex set. Hence, after exhaustive application of Rule 4, every cluster has bounded size. While Rule 4 identifies clusters that are large with respect to k, Rule 5 identifies clusters that are large compared to their neighborhood.

Rule 5. *If G contains a vertex set S such that*

- $|S| \geq 4$,
- *$G[S]$ is highly connected, and*
- $|D(S)| \leq 0.3 \cdot \sqrt{|S|}$,

then remove S from G and set $k := k - |D(S)|$.

Lemma 5. *Rule 5 is correct and can be exhaustively applied in $O(n^2 \cdot mk \log n)$ time.*

Proof. We show that there is an optimal solution in which S is a cluster. To this end, suppose that there is an optimal solution which produces some clusters $C_1, \ldots, C_q$ that contain vertices from S and vertices from $V \setminus S$. We show how to transform this solution into one that has S as a cluster and needs at most as many edge deletions.

First, we bound the overall size of the C_i's. Note that deleting all edges between S and $\{C_i \setminus S \mid 1 \leq i \leq q\}$ cuts each C_i. By the condition of the rule,

such a cut has at most $0.3\sqrt{|S|}$ edges. Since each $G[C_i]$ is highly connected, this implies that $\sum_{1 \leq i \leq q} |C_i| < 0.6\sqrt{|S|}$.

Now, transform the solution at hand into another solution as follows. Make S a cluster, that is, undo all edge deletions within S and delete all edges in $D(S)$, and for each C_i, delete all edges in $G[C_i \setminus S]$. This is indeed a valid solution since $G[S]$ is highly connected, and all other vertices that are in "new" clusters are now in singleton clusters.

We now compare the number of edge modifications for both edge deletion sets and show that the new solution needs less edge modifications. To this end, we consider each vertex $u \in S$ that is contained in some C_i. On the one hand, since $G[S]$ is highly connected, and since there is at least some $v \in S$ that is not contained in any C_i we undo at at least $|S|/2$ edge deletions between vertices of S. On the other hand, an additional number of up to $0.3\sqrt{|S|} + \binom{\lfloor 0.6\sqrt{|S|} \rfloor}{2}$ edge deletions may be necessary to cut all the C_i's from S and to delete all edges in each $G[C_i \setminus S]$. By the preconditions of the rule we have $\sqrt{|S|} \leq |S|/2$ and thus the overall number of saved edge modifications for u is at least

$$|S|/2 - 0.3\sqrt{|S|} - \binom{\lfloor 0.6\sqrt{|S|} \rfloor}{2} > |S|/2 - 0.6|S|/2 - 0.36|S|/2 > 0. \quad (1)$$

Hence, the number of undone edge modifications is larger than the number of new edge modifications. Consequently, S is a cluster in every optimal solution.

The running time can be bounded analogously to the running time of Rule 4. The only difference is that after constructing the Gomory-Hu tree, one can find a vertex set that fulfills the conditions of the rule by trying all m possibilities for "guessing" $|S|/2$. Assuming the correct guess, deleting all edges with weight at most $|S|/2$ in the tree produces one connected component that is exactly S. □

Theorem 3. Highly Connected Deletion *admits a problem kernel with at most* $10 \cdot k^{1.5}$ *vertices which can be computed in* $O(n^2 \cdot mk \log n)$ *time.*

Proof. Let $I = (G, k)$ be an instance that is reduced with respect to Rules 2, 4 and 5. We show that every yes-instance has at most $10 \cdot k^{1.5}$ vertices. Hence, we can answer no for all larger instances.

Assume that I is a yes-instance and let $C_1, \ldots, C_q$ denote the clusters of a solution. Since I is reduced with respect to Rule 4, we have $|C_i| \leq 2k$ for each C_i. Furthermore, for every C_i we have $D(C_i) \geq 0.3\sqrt{|C_i|}$ since I is reduced with respect to Rules 2 and 5. In other words, every cluster C_i "needs" at least $0.3\sqrt{|C_i|}$ edge deletions. This implies that the overall size of the instance is upper-bounded by

$$\max_{(c_1,\ldots,c_q) \in \mathbb{N}^q} \sum_{i=1}^{q} c_i \text{ s.t. } \forall i \in \{1, \ldots, q\} : c_i \leq 2k, \sum_{1 \leq i \leq q} 0.3 \cdot \sqrt{c_i} \leq 2k.$$

A simple calculation shows that there is an assignment to the c_i's maximizing the sum such that there is at most one c_i that is smaller than $2k$. Hence, the

sum is maximized when a maximum number of c_i's have value $2k$. Each of the corresponding clusters is incident with at least $0.3\sqrt{2k}$ edge deletions. Hence, there can be at most $2k/0.3\sqrt{2k} = 10\sqrt{2k}/3$ of these clusters. The overall instance size follows. □

3 Fixed-Parameter Algorithm

We present a fixed-parameter algorithm solving Highly Connected Deletion in $3^{4k} \cdot n^{O(1)}$ time. Fixed-parameter algorithms have also been given for related clustering problems; the best known fixed-parameter algorithm for Cluster Deletion (after a long line of improvements) runs in $1.415^k \cdot n^{O(1)}$ time [2], and the best known fixed-parameter algorithm for 2-Club Deletion runs in $2.74^k \cdot n^{O(1)}$ time [19].

The main idea of our algorithm is to branch until each connected component has diameter two and solve these instances by a dynamic programming algorithm. The details are as follows.

Since each highly connected graph has diameter at most two [9], we can perform the following branching rule.

Branching Rule 1. *If a connected component in G has diameter three or more, then find two vertices u and v with distance three and pick an arbitrary shortest path $P = uxyv$ between u and v. Branch into the three possibilities to destroy P by deleting either $\{u, x\}$, $\{x, y\}$, or $\{y, v\}$. In each recursive branch, set $k := k - 1$.*

The rule is obviously correct in the sense that at least one of the three edges has to be destroyed. Now assume that the branching rule does not apply anymore, that is, each connected component has diameter two. If the graph is also highly connected, then we are done. Otherwise, we can apply Rule 4 to obtain an instance that is small compared to k, as shown by the following lemma.

Lemma 6. *Let $I = (G, k)$ be an instance of* Highly Connected Deletion *such that G has diameter two and I is reduced with respect to Rule 4. Then, G has at most $4k$ vertices.*

Proof. Consider a solution for I. Since G has diameter two, there is at most one cluster in this solution that has vertices that are not incident with an edge that is deleted by the solution (call these vertices *unaffected*): if there are two clusters C_i and C_j with unaffected vertices u and v, then these vertices are withing distance at least three (u is not adjacent to a neighbor of v). Let C_1 be the, possibly empty, cluster that has unaffected vertices. Then, since I is reduced with respect to Rule 4, C_1 has at most $2k$ vertices. Since all other vertices are affected, there can be at most $2k$ further vertices. The overall size of the instance follows. □

In the following lemma, we describe an algorithm that solves Highly Connected Deletion for arbitrary (not necessarily diameter-2) instances. The

main trick in the fixed-parameter algorithm is that with the above lemma this running becomes a single-exponential running time for the parameter k.

Lemma 7. Highly Connected Deletion *can be solved in* $O(3^n \cdot m)$ *time.*

Proof. We describe a dynamic programming algorithm. The idea of the algorithm is that if V can be two-partitioned into V_1 and V_2 such that all clusters are subsets of either V_1 or V_2, then we can obtain the overall solution by combining best solutions for the induced subgraphs $G[V_1]$ and $G[V_2]$. The details are as follows.

We build a dynamic programming table T with entries of the type $T[V']$, $V' \subseteq V$ which store an optimal solution for Highly Connected Deletion for $G[V']$. The table is initialized by setting $T[V'] = 0$ for each $V' \subseteq V$ such that $G[V']$ is highly connected. The remaining entries can be computed by the recurrence

$$T[V'] = \min_{V_1, V_2, V_1 \dot\cup V_2 = V'} T[V_1] + T[V_2] + |\{\{u, v\} \in E \mid u \in V_1 \wedge v \in V_2\}|. \quad (2)$$

The third summand is exactly the number of edges needed to cut V_1 from V_2 in G. After all entries have been computed, $T[V]$ stores the number of edge deletions needed for obtaining a highly connected cluster graph; an actual clustering can be obtained by a traceback. The correctness of the recurrence follows from the discussion above.

The running time can be bounded as follows. For each table entry, the initialization can be performed in $O(m)$ time, leading to an overall time of $O(2^n \cdot m)$ for this part of the algorithm. In the second part of the algorithm, an overall number of $O(3^n)$ recurrences have to be evaluated: each partition of V' into V_1 and V_2 uniquely defines a three-partition of V into $V \setminus V'$, V_1 and V_2. Since the number of edges needed for the third summand can be counted in $O(m)$ time, the overall running time follows. □

Combining the above two lemmas with the branching rule, we obtain our main result of this section.

Theorem 4. Highly Connected Deletion *can be solved in* $O(3^{4k} \cdot k^2 + n^2mk \cdot \log n)$ *time.*

Proof. The algorithm performs Rules 2 and 4 and the branching rule as long as possible. By Lemma 6, the remaining instances have at most $4k'$ vertices for some $k' \leq k$. Using Lemma 7, these instances can be solved in $O(3^{4k'} \cdot k^2)$ time. The overall running time follows from a simple worst-case analysis, we omit the details. □

The running time given by Theorem 4 is impractical. Even worse, the presented algorithm behind relies partially on dynamic programming which has the disadvantage that, compared to branching algorithms, its average-case running time often comes close to its worst-case running time. We conjecture that a running time improvement using only branching algorithms is possible.

Acknowledgment

We are indebted to Nadja Betzler and Johannes Uhlmann for their early contributions in this research.

References

[1] R. Albert. Scale-free networks in cell biology. *Journal of Cell Science*, 118 (21):4947–4957, 2007. 1

[2] S. Böcker and P. Damaschke. Even faster parameterized cluster deletion and cluster editing. *Information Processing Letters*, 111(14):717–721, 2011. 10

[3] G. Chartrand. A graph-theoretic approach to a communications problem. *SIAM Journal on Applied Mathematics*, 14(4):778–781, 1966. 1

[4] R. G. Downey and M. R. Fellows. *Parameterized Complexity*. 1999. 3

[5] J. Flum and M. Grohe. *Parameterized Complexity Theory*. 2006. 3

[6] I. Gat-Viks, R. Sharan, and R. Shamir. Scoring clustering solutions by their biological relevance. *Bioinformatics*, 19(18):2381–2389, 2003. 2

[7] R. E. Gomory and T. C. Hu. Multi-Terminal Network Flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961. 8

[8] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. 6

[9] E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4–6):175–181, 2000. 1, 2, 3, 10

[10] E. Hartuv, A. O. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir. An algorithm for clustering cDNA fingerprints. *Genomics*, 66 (3):249–256, 2000. 1

[11] H. Hu, X. Yan, Y. Huang, J. Han, and X. J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. In *Proc. 13th International Conference on Intelligent Systems for Molecular Biology (ISMB '05)*, volume 21 of *Bioinformatics*, pages 213–221, 2005. 2, 3

[12] F. Hüffner, C. Komusiewicz, and R. Niedermeier. Partitioning biological networks into highly connected clusters with maximum edge coverage, 2013. Manuscript. 7

[13] R. Impagliazzo, R. Paturi, and F. Zane. Which problems have strongly exponential complexity? *Journal of Computer and System Sciences*, 63(4): 512–530, 2001. 3

[14] D. Jiang and J. Pei. Mining frequent cross-graph quasi-cliques. *ACM Transactions on Knowledge Discovery from Data*, 2(4):16:1–16:42, 2009. 1

[15] V. King, S. Rao, and R. E. Tarjan. A faster deterministic maximum flow algorithm. *J. Algorithms*, 17(3):447–474, 1994. 8

[16] C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. 4

[17] M. Koyutürk, W. Szpankowski, and A. Grama. Assessing significance of connectivity and conservation in protein interaction networks. *Journal of Computational Biology*, 14(6):747–764, 2007. 2, 3

[18] A. Krause, J. Stoye, and M. Vingron. Large scale hierarchical clustering of protein sequences. *BMC Bioinformatics*, 6:15, 2005. 1

[19] H. Liu, P. Zhang, and D. Zhu. On editing graphs into 2-club clusters. In *Proc. Joint International Conference on Frontiers in Algorithmics and Algorithmic Aspects in Information and Management (FAW-AAIM '12)*, volume 7285 of *LNCS*. Springer, 2012. 2, 10

[20] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 84:41–71, 2011. 3

[21] H. Matsuda, T. Ishihara, and A. Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theoretical Computer Science*, 210(2):305–325, 1999. 1

[22] M. E. J. Newman. *Networks: An Introduction*. Oxford University Press, 2010. 1

[23] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006. 3

[24] B. J. Parker, I. Moltke, A. Roth, S. Washietl, J. Wen, M. Kellis, R. Breaker, and J. S. Pedersen. New families of human regulatory RNA structures identified by comparative analysis of vertebrate genomes. *Genome Research*, 21(11):1929–1943, 2011. 1

[25] N. Pržulj, D. A. Wigle, and I. Jurisica. Functional topology in a network of protein interactions. *Bioinformatics*, 20(3):340–348, 2004. 1

[26] J. M. M. van Rooij, M. E. van Kooten Niekerk, and H. L. Bodlaender. Partition into triangles on bounded degree graphs. In *Proceedings of the 37th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM '11)*, volume 6543 of *LNCS*, pages 558–569. Springer, 2011. 4, 6

[27] R. Shamir, R. Sharan, and D. Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1–2):173–182, 2004. 2

[28] R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Molecular Systems Biology*, 3:88, 2007. 1

[29] V. Spirin and L. A. Mirny. Protein complexes and functional modules in molecular networks. *PNAS*, 100(21):12123–12128, 2003. 1