Finding Highly Connected Subgraphs*

Falk Hüffner, Christian Komusiewicz, and Manuel Sorge

Institut für Softwaretechnik und Theoretische Informatik, TU Berlin,
{falk.hueffner,christian.komusiewicz,manuel.sorge}@tu-berlin.de

Abstract. A popular way of formalizing clusters in networks are *highly connected* subgraphs, that is, subgraphs of k vertices that have edge connectivity larger than $k/2$ (equivalently, minimum degree larger than $k/2$). We examine the computational complexity of finding highly connected subgraphs. We first observe that this problem is NP-hard. Thus, we explore possible parameterizations, such as the solution size, number of vertices in the input, the size of a vertex cover in the input, and the number of edges outgoing from the solution (edge isolation), and expose their influence on the complexity of this problem. For some parameters, we find strong intractability results; among the parameters yielding tractability, the edge isolation seems to provide the best trade-off between running time bounds and a small parameter value in relevant instances.

1 Introduction

A popular method of analyzing complex networks is to identify *clusters* or *communities*, that is, subgraphs that have many interactions within themselves and fewer with the rest of the graph (e. g. [18, 19]). Hartuv and Shamir [9] proposed a prominent clustering algorithm producing *highly connected* clusters, formalized 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 has degree at least $\lfloor n/2 \rfloor + 1$ [3]. Moreover, highly connected graphs have diameter at most two [9].

We study the following problem:

Highly Connected Subgraph
Input: An undirected graph $G = (V, E)$ and a nonnegative integer k.
Question: Is there a vertex set S such that $|S| = k$ and $G[S]$ is highly connected?

In addition to the natural application in analyzing complex networks [19], Highly Connected Subgraph also occurs (with vertex weights) as a subproblem in a column generation algorithm for partitioning graphs into highly connected components [11].

* FH and MS gratefully acknowledge support by Deutsche Forschungsgemeinschaft, projects ALEPH (HU 2139/1) and DAPA (NI 369/12) respectively. Due to space constraints, proofs for results marked by ⋆ are deferred to a full version of this extended abstract.

Since Highly Connected Subgraph is NP-hard (Theorem 1), we explore the "parameter ecology" [7] of this problem. We are looking for fixed-parameter algorithms, that is, we try to find problem parameters p that allow for a running time bounded by $f(p) \cdot |G|^{O(1)}$. The hope is that the function f grows not too fast (although it has to be superpolynomial unless P = NP), and that the parameter value p can be expected to be relatively small in interesting instances. Clearly, there is a trade-off between these goals. Similarly to NP-hardness, fixed-parameter tractability can be refuted by giving suitable reductions from hard problems of the classes W[1] or W[2]. For details, refer to the literature [5].

Results. We list the results going from the hardest parameters to the easiest, corresponding roughly to going from small expected parameter values to large ones. Let n be the number of vertices in G. For the parameter $\ell := n - k$ (the number of vertices to delete to obtain a highly connected subgraph), we obtain a strong hardness result: there is a trivial $n^{O(\ell)}$ time algorithm, but it is unlikely that $n^{o(\ell)}$ time can be achieved (Theorem 1). For the size of the solution k, a fixed-parameter algorithm is unlikely, even if we additionally consider the degeneracy of G as a parameter (Theorem 2). If we take the minimum size τ of a vertex cover for G as parameter, we obtain a fixed-parameter algorithm: the problem can be solved in $(2\tau)^\tau \cdot n^{O(1)}$ time (Theorem 3). Considering the number of vertices n, we can clearly solve the problem in $2^n \cdot n^{O(1)}$ time. We show that it is unlikely that this can be improved to $2^{o(n)} \cdot n^{O(1)}$ time (Theorem 4). If the parameter is the number γ of edges between $G[S]$ and the remaining vertices, then the problem can be solved in time $O(4^\gamma n^2)$ (Theorem 5). Finally, if we consider the number α of edges to delete to obtain a highly connected subgraph (plus singleton vertices), we obtain a $O(2^{4 \cdot \alpha^{0.75}} + \alpha^2 nm)$-time algorithm (Theorem 8). This running time is subexponential in α.

Related work. The algorithm by Hartuv and Shamir [9] partitions a graph heuristically into highly connected components; another algorithm tries to explicitly minimize the number of edges that need to be deleted for this [11]. Highly connected graphs can be seen as *clique relaxation* [18], that is, a graph class that has many properties similar to cliques, without being as restrictive. Highly connected graphs are very similar to *0.5-quasi-complete graphs* [17], that is, graphs where every vertex has degree at least $(n-1)/2$. These graphs are also referred to as *(degree-based) 0.5-quasi-cliques* [15]. Recently, also the task of finding subgraphs with high *vertex* connectivity has been examined [20].

Preliminaries. Using standard graph notation, we consider only simple undirected graphs $G = (V, E)$ with $n := |V|$ and $m := |E|$; we call n the *order* of G. We use $N(v)$ to denote the set of *neighbors* of a vertex v. For a vertex set $S \subseteq V$, we denote $G[S] := (S, \{\{u, v\} \in E \mid u, v \in S\})$ the *subgraph of G induced by S*. We use $G - S$ as shorthand for $G[V \setminus S]$. A *cut* (A, B) in a graph $G = (V, E)$ is a vertex bipartition, that is, $A \cap B = \emptyset$ and $A \cup B = V$. The *cut edges* are the edges between vertices in A and B; the *size* of a cut is the number of its cut edges.

2 Vertex Deletion

For finding large cliques in a graph, one successful approach is to use fixed-parameter algorithms for the parameter "number of vertices in the graph that are not in the clique" [13, 14]. We show by a reduction from Hitting Set that such fixed-parameter algorithms are unlikely for Highly Connected Subgraph.

Theorem 1 ($\star$). Highly Connected Subgraph *is NP-hard and W[2]-hard parameterized by $\ell := n - k$. Moreover, an $n^{o(\ell)}$-time algorithm implies FPT=W[1].*

3 Solution Size and Degeneracy

A graph has degeneracy d if every subgraph contains at least one vertex that has degree at most d. In many graphs from real-world applications, the degeneracy of a graph is very small compared to the network size [6]. For yes-instances, the degeneracy of the input graph has to be at least $\lfloor k/2 \rfloor + 1$. Therefore, Highly Connected Subgraph is polynomial-time solvable if the input graph has constant degeneracy: trying all subgraphs with $k \leq 2d + 2$ vertices decides the problen in $n^{2d} \cdot n^{O(1)}$ time. This can be improved to the following running time.

Proposition 1 ($\star$). Highly Connected Subgraph *can be solved in $2^d \cdot n^{d+O(1)}$ time where d is the degeneracy of G.*

Unfortunately, if we regard the degeneracy as a parameter instead of a constant, we obtain hardness using a reduction from Clique, even if additionally the solution size k is a parameter.

Theorem 2 ($\star$). Highly Connected Subgraph *parameterized by the combined parameter (d, k), where d is the degeneracy of G, is W[1]-hard.*

4 Vertex Cover Size

We next consider the parameter τ, the minimum size of a vertex cover of G. This parameter is interesting because it can be smaller than the number of vertices of G. We show that Highly Connected Subgraph is solvable in $(2\tau)^\tau \cdot n^{O(1)}$ time. The algorithm first computes a vertex cover C, then determines via branching the intersection of C and the desired solution S, and then adds suitable vertices of the independent set $V \setminus C$. We identify suitable vertices in the independent set by solving an instance of Set Multicover.

> Set Multicover
> Input: A universe U with covering demands $d : U \to \mathbb{N}$, a family $\mathcal{F}$ of subsets of the universe with multiplicity values $m : \mathcal{F} \to \mathbb{N}$, and $p \in \mathbb{N}$.
> Question: Is there a multiset of at most p subsets from $\mathcal{F}$ that contains each $F \in \mathcal{F}$ at most $m(F)$ times, and covers each $u \in U$ with at least $d(u)$ subsets?

Lemma 1. *A given instance of* Highly Connected Subgraph *with a vertex cover of size τ can be solved using the answers to at most 2^τ instances of* Set Multicover*, each with $|U| \leq \tau$ and $2\max_{u\in U} d(u) \leq \tau$. Furthermore, all these instances can be computed in $O(2^\tau(n+\tau n))$ time.*

Proof. Fix some highly connected subgraph $G[S]$ of order k if it exists. First compute a minimum vertex cover C for G in $O(1.274^\tau + \tau n)$ time [4]. Enumerate all 2^τ possibilities for $C' \subseteq C$. Clearly, in one branch $C' = C \cap S$. In each branch, delete the vertices from $C \setminus C'$. Then remove vertices from the independent set $V \setminus C'$ that have $k/2$ or less neighbors in C', since they cannot be part of S. Let V' be the thus reduced vertex set. It remains to find $k' := k - |C'|$ vertices in $V' \setminus C'$ such that each vertex v in C' has more than $d(v) := k/2 - |N(v) \cap C'|$ neighbors among these k' vertices. This is an instance of Set Multicover: In our case, the universe is C', the covering demands are d as defined above, the family is $\mathcal{F} = \{N(v) \cap C' \mid v \in V' \setminus C'\}$, the multiplicity of $X \in \mathcal{F}$ is the number of vertices in $V' \setminus C'$ having neighborhood X in C', and $p = k'$. If the solution to Set Multicover has less than k' sets, then we can add arbitrary further vertices from $V' \setminus C'$ to make the vertex subset large enough (if $|V' \setminus C'| < k'$, then we can safely reject this branch for $C' \subseteq C$). □

Set Multicover with multiplicity constraints can be solved in $O((b+1)^{|U|}|\mathcal{F}|)$ time [10], where $b := \max_{u\in U} d(u)$. Note that $|C'| > k/2$, since the vertices outside of C' form an independent set and we cannot choose $k/2$ or more of them. Thus, $b < |C'| \leq \tau$. The size of $\mathcal{F}$ is at most n. Together with the enumeration of the instances, we obtain the following.

Theorem 3. Highly Connected Subgraph *can be solved in $O((2\tau)^\tau \cdot \tau n)$ time.*

5 Number of Vertices

A trivial algorithm for Highly Connected Subgraph is to enumerate all vertex subsets S of size k and to check for each subset whether it is highly connected. This algorithm has running time $O(2^n \cdot m)$. We now show by a reduction from Clique that a running time improvement to $2^{o(n)} \cdot n^{O(1)}$ is unlikely. The idea of the reduction is to add to a Clique instance some new graph that is so large, that, in the resulting instance of Highly Connected Subgraph, every highly connected graph of size k must contain this new graph. The remaining vertices must form a clique in order to have sufficiently high degree. The following lemma shows that we can efficiently construct the graph which we need to add to the Clique instance.

Lemma 2 ($\star$). *For any two integers $a, b \in \mathbb{N}$ such that a is even, $b-3 \geq 8$ is a power of two, and $a - 2 \geq 2b$, there is a graph $G = (X \cup W, E)$ on the disjoint vertex sets X and W, such that*

i) $G[X]$ is connected,
ii) $|X| = a - 2$, $|W| = a - b + 1$,

iii) $N_G(X) \setminus X = W$,
iv) each vertex in X *has degree* a*, and each vertex in* W *has degree* $a - b$.
Moreover, G *can be constructed in time polynomial in* a.

We call the graph G described in the above lemma an (a, b)-*equalizer* and the vertices in W are its *ports*.

Lemma 3. *There is a polynomial-time many-one reduction from* Clique *to* Highly Connected Subgraph *that is parameter-linear with respect to the number of vertices.*

Proof. Let (G, p) represent an instance of Clique. Without loss of generality, assume that $p - 3 \geq 8$ and $p - 3$ is a power of two. Otherwise, repeatedly add a universal vertex and increase p by one, until $p - 3 \geq 8$, and $p - 3$ is a power of two. Note that this at most doubles p. Furthermore, assume that $n - 1 \geq p$; otherwise, solve the instance in polynomial time.

Denote $|V(G)| = n$. We construct the instance (G', k) of Highly Connected Subgraph where $k = 4n-1$. Note that the minimum degree in a highly connected graph with k vertices is $2n$. Graph G' is constructed as follows. First, copy G into G'. Then add a vertex-disjoint $(2n, p)$-*equalizer*. By Lemma 2, a $(2n, p)$-*equalizer* exists and is computable in polynomial time, because, by choice of (G, p), $2n$ is even, $p-3 \geq 8$ is a power of two, and $2n - 2 \geq 2p$. Denote the ports of the equalizer by W and its remaining vertices by X. Add an edge between each port and each vertex in $V(G)$; this finishes the construction. The graph G' has less than $5n$ vertices, since the $(2n, p)$-*equalizer* has less than $4n$ vertices. It remains to show equivalence of the instances, that is,

$$(G, p) \text{ is a yes-instance} \Leftrightarrow (G', k = 4n - 1) \text{ is a yes-instance.}$$

"$\Rightarrow$": Let $G[S]$ be a clique of order p in G. Then, $G'[S \cup X \cup W]$ is highly connected: Each vertex in S is adjacent to $p - 1$ vertices in S and to $2n - p + 1$ vertices in W. Hence, each vertex in S has $2n$ neighbors in $S \cup X \cup W$, as required. Each port has $2n - p$ neighbors in $X \cup W$ and p neighbors in S. Finally, each vertex in X has $2n$ neighbors in $X \cup W$.

"$\Leftarrow$:" Let $G'[S]$ be a highly connected graph of order k in G'. There are at most n vertices in $V(G) \cap S$, thus there is at least one vertex in $S \cap X$. Since $G'[X]$ is connected and each vertex in X has degree exactly $2n$ (the minimum degree in $G'[S]$), we have $X \subseteq S$. Furthermore, since $\{v \mid X \cap N(v) \neq \emptyset\} \setminus X = W$, also $W \subseteq S$, leaving $4n - 1 - |X| - |W| = p$ vertices in $S \cap V(G)$. Since $N_{G'}(V(G)) \setminus V(G) = W$ and $|W| = 2n - p + 1$, each vertex in $S \cap V(G)$ has at least $p - 1$ neighbors in $S \cap V(G)$. Thus $G[S \cap V(G)]$ is a clique. □

Using Lemma 3, we can connect the running time with respect to parameter n with the Exponential Time Hypothesis (ETH) [16].

Theorem 4. *If the Exponential Time Hypothesis (ETH) is true, then* Highly Connected Subgraph *does not admit a* $2^{o(n)} \cdot n^{O(1)}$*-time algorithm.*

6 Edge Isolation

We now present a single-exponential FPT algorithm for the number γ of edges between the desired highly connected subgraph $G[S]$ and the remaining graph. In this case, S is called "γ-isolated". More formally, if $G = (V, E)$ is a graph, we call a set $S \subseteq V$ *γ-isolated* if $(S, V \setminus S)$ is a cut of size at most γ. To our knowledge, Ito et al. [13] were the first to consider a formal notion of isolation in the context of dense subgraph identification. There is the following difference between the isolation definitions: we count the total size of the cut $(S, V \setminus S)$, whereas previous definitions count the size of $(S, V \setminus S)$ divided by the size of S [12, 13] or the minimum of the number of outgoing edges per vertex [14]. Our isolation definition leads to the following problem.

> Isolated Highly Connected Subgraph
> Input: An undirected graph $G = (V, E)$, nonnegative integers k and γ.
> Question: Is there a k-vertex γ-isolated highly connected subgraph contained in G?

The notion of isolation is not only motivated from an algorithmic point of view but also from the application. Ideally, communities in a network have fewer connections to the rest of the network [18]. Thus, putting an additional constraint on the number of outgoing edges may yield better communities than merely demanding high edge connectivity.

In the following, it will be useful to consider an augmented version of Isolated Highly Connected Subgraph: we place integer labels on the vertices which imply that these vertices are harder to isolate. We thus additionally equip each instance of Isolated Highly Connected Subgraph with a labeling $f\colon V \to \mathbb{N}$ and we call $V' \subseteq V$ *γ-isolated under f* if there are at most $\gamma - \sum_{v \in V'} f(v)$ edges between V' and $V \setminus V'$ in G. Without loss of generality, assume $k \geq 2$ in the following.

The algorithm first performs three reduction rules. The first simple rule removes connected components that are too small.

Rule 1. *Remove all connected components with less than k vertices from G.*

The next rule finds connected components that are either trivial solutions or cannot contain any solution since proper subgraphs violate the isolation condition.

Rule 2. *If there is a connected component $C = (V', E')$ of G that has minimum cut size at least $\gamma + 1$, then accept if C is highly connected, $|V'| = k$, and V' is γ-isolated under f. Otherwise remove C from G.*

Proof (Correctness of Rule 2). The rule is clearly correct if it accepts. If the rule removes C, then C has a minimum cut of size at least $\gamma + 1$. Thus, for every induced subgraph $C[S]$ of C that does not contain all of its vertices, set S is not γ-isolated. Hence, no subgraph of C is a solution and we can safely remove C. □

Rule 3. *If G has a connected component C with a minimum cut (A, B) of size at most $k/2$, then do the following. For each $v \in A$ redefine $f(v) := f(v)+|N(v)\cap B|$ and for each $v \in B$ redefine $f(v) := f(v) + |N(v) \cap A|$. Then, delete all edges between A and B.*

Proof (Correctness of Rule 3). Any k-vertex subgraph of C with nonempty intersection with both sides of (A, B) is not highly connected as it has a minimum cut of size at most $k/2$. Hence, any highly connected induced subgraph $C[S]$ of C is either contained in $C[A]$ or in $C[B]$. If S is γ-isolated under f in G, then it is also γ-isolated under the modified f in the modified graph (and vice-versa) by the way we have redefined f. □

Exhaustive application of these rules yields a relation between γ and $k/2$.

Lemma 4. *If Rules 1 to 3 are not applicable, then $\gamma > k/2$.*

Proof. Assume the contrary. Each connected component has a minimum cut cutting at least one edge because Rule 1 is not applicable and $k \geq 2$. Further, each connected component has a cut of size at most γ because Rule 2 is not applicable. By assumption $\gamma \leq k/2$ and, hence, each connected component has a cut of size at most $k/2$ which contradicts the inapplicability of Rule 3. □

As shown by the following lemma, the reduction rules can be applied efficiently.

Lemma 5 (⋆). *Rules 1 to 3 can be exhaustively applied in $O((kn+\gamma)nm)$ time.*

Using the above, we can now present the branching algorithm.

Theorem 5. *There is an $O(4^\gamma n^2 + (kn+\gamma)nm)$-time algorithm for* Isolated Highly Connected Subgraph.

Proof. We first reduce the instance with respect to Rules 1 to 3. By Lemma 5 this can be done in $O((kn+\gamma)nm)$ time. Next, we guess one vertex v that is in the solution S (by branching into n cases according to the n vertices). We start with $S' := \{v\}$ and try to extend S' to a solution. More precisely, we choose a vertex v' from the neighborhood of S' (that is, from $\bigcup_{u\in S'} N(u) \setminus S'$), and branch into two cases: add v' to S', or exclude v', that is, delete v' and increase $f(u)$ by one for all $u \in N(v')$. In the first case, we increase $|S'|$ by one. In the second case, we increase $\sum_{u\in S'} f(u)$ by at least one. Branching is performed until $|S'| = k$ or $\sum_{u\in S'} f(u)$ exceeds γ or the neighborhood of S' is empty. When $|S'|$ reaches k, we check whether S' is highly connected and γ-isolated under f, and if this is the case, we have found a solution. Otherwise, when $\sum_{u\in S'} f(u)$ exceeds γ or no branching is possible because the neighborhood of S' is empty, we abort the branch; in this case, clearly no superset of S' can be a solution. The height of the search tree is bounded by $k+\gamma$, and each branch can be executed in $O(n)$ time, yielding a running time bound of $O(n \cdot 2^{k+\gamma} \cdot n)$.

We now distinguish two cases: $k \leq \gamma$ and $k > \gamma$. In the first case $2^{k+\gamma} \leq 4^\gamma$, as required. If $k > \gamma$, there is at least one vertex in S that has no neighbors outside of S. Thus, instead of $S' = \{v\}$, we can start with $S' := \{v\} \cup N(v)$. Since v has more than $k/2$ neighbors in S, we have $|S'| > k/2+1$, and thus there are less than $k/2$ branches of adding a vertex. By Lemma 4, $2^{k/2+\gamma} \leq 4^\gamma$. □

We now present a further way of analyzing the presented data reduction rules by giving a *Turing kernelization* [1] for Isolated Highly Connected Subgraph parameterized by γ. Informally, a Turing kernelization is a reduction of the input instance of a parameterized problem to many instances of the same problem which are small measured in the parameter. Then, the solution to the original input instance can be computed by solving the small problem instances separately.

To motivate the Turing kernelization result we first observe that Isolated Highly Connected Subgraph does not admit a problem kernel, that is, a Turing kernelization which produces only one small problem instance. The disjoint union of a set of graphs has an isolated highly connected subgraph if and only if at least one of the graphs has one. Hence, Isolated Highly Connected Subgraph has a trivial OR-composition which implies the following [2].

Proposition 1. Isolated Highly Connected Subgraph *does not admit a polynomial-size problem kernel with respect to γ unless NP $\subseteq$ coNP/poly.*

Before describing the Turing kernelization, we give a formal definition.

Definition 1. *Let L be a parameterized problem and let $g : \mathbb{N} \to \mathbb{N}$ be a computable function. A* Turing kernelization *for L is an algorithm that, for each instance (x, k), decides whether $(x, k) \in L$ in polynomial time using an oracle for $\{(x', k') \mid |x'| + k' \leq g(k) \wedge (x', k') \in L\}$. The sequence of queries posed to the oracle is called* Turing kernel. *We call $g(k)$ the* size *of the Turing kernel.*

We now describe the algorithm in detail. The first step is to reduce to the augmented version of Isolated Highly Connected Subgraph in which we introduce the vertex labeling f. Then, apply Rules 2 and 3 exhaustively. Afterwards, apply the following reduction rule which removes high-degree vertices.

Rule 4 ($\star$). *Let (G, k, γ) be an instance of* Isolated Highly Connected Subgraph *that is reduced with respect to Rules 2 and 3. If G contains a vertex v of degree at least $3\gamma - f(v)$, then remove v from G, and for each $u \in N(v)$ increase $f(u)$ by one.*

Now we construct n instances of Isolated Highly Connected Subgraph that have $O(\gamma^3)$ vertices each. The original instance is a yes-instance if and only if one of these instances is a yes-instance. The idea is to exploit the fact that highly connected graphs have diameter two [9]. Thus, to find highly connected graphs, it is sufficient to explore the two-neighborhood of each vertex. More precisely, the instances are constructed as follows.

For each vertex $v \in V$, construct the graph $G_v := G[N_2[v]]$ where $N_2[v]$ is the set of all vertices that have distance at most two from v (including v). When solving the Isolated Highly Connected Subgraph instances we need to determine whether a subgraph is γ-isolated. Thus, the graph G_v has to contain information on the original vertex degrees. Note that for each $u \in V$, $f(u)$ denotes the number of edges deleted during the data reduction that are incident with u. Moreover, for each vertex u in G_v, let $g(u)$ denote the number of neighbors of u in G in $V \setminus N_2[v]$. To obtain instances of Isolated Highly Connected

Subgraph one may not use vertex labelings. Thus, for each u of G_v add $g(u)+f(u)$ new vertices and make them adjacent to u. This completes the construction of G_v. In this way we obtain n instances (G_v, k, γ) of Isolated Highly Connected Subgraph. The following lemma shows that it is sufficient to solve these instances in order to determine whether the original Isolated Highly Connected Subgraph instance $(G = (V, E), k, \gamma)$ is a yes-instance.

Lemma 6 ($\star$). *Let $(G = (V, E), k, \gamma)$ be an instance of* Isolated Highly Connected Subgraph *and, for each $v \in V$, let (G_v, k, γ) denote the instance as constructed above. Then, (G, k, γ) is a yes-instance if and only if there is a $v \in V$ such that (G_v, k, γ) is a yes-instance.*

We now show that the instances have bounded size.

Lemma 7 ($\star$). *Let (G_v, k, γ) be an instance of* Isolated Highly Connected Subgraph *constructed from G as described above. Then G_v has less than $(3\gamma)^3$ vertices and less than $3\gamma^4$ edges.*

Combining Lemmas 6 and 7 leads to the following.

Theorem 6. Isolated Highly Connected Subgraph *admits a Turing kernel of size $O(\gamma^4)$ which has less than $(3\gamma)^3$ vertices.*

7 Edge Deletion

We now show that there is a subexponential fixed-parameter algorithm for Highly Connected Subgraph with respect to α, the number of edges we are allowed to delete in order to obtain a highly connected graph of order k. The algorithm is a search tree algorithm which branches on whether or not a given vertex is part of the highly connected graph. Repeated application of two reduction rules (similar to Rules 2 and 3 above) ensures that the branches are effective in reducing the remaining search space. To give a precise presentation of the branching step and the reduction rules, we define the problem with an additional *seed* S, a set of vertices which have to be in the highly connected graph.

> Seeded Highly Connected Edge Deletion
> Input: An undirected graph $G = (V, E)$, a vertex set $S \subseteq V$, and nonnegative integers k and α.
> Question: Is there a set $E' \subseteq E$ of at most α edges such that $G - E'$ consists only of degree-zero vertices and a $(k + |S|)$-vertex highly connected subgraph containing S?

For $S = \emptyset$ we obtain the plain edge deletion problem. The reduction rules are as follows.

Rule 5 ($\star$). *If there is a connected component $C = (V', E')$ of G that has minimum cut size at least $\alpha + 1$, then accept if C is highly connected, $S \subseteq V'$, $|V' \setminus S| = k$, and the remaining connected components of G contain at most α edges. Otherwise reject.*

Rule 6 ($\star$). *If there is a connected component of G that has a minimum cut of size at most $(k + |S|)/2$, then delete all cut edges and reduce α by their number.*

Similarly to the edge isolation parameter, after using the reduction rules k, $|S|$, and α are related.

Lemma 8 ($\star$). *If Rules 5 and 6 are not applicable, then $\alpha > (k + |S|)/2$.*

As the ones presented in Section 6, both rules can be applied efficiently.

Lemma 9 ($\star$). *Rules 5 and 6 are exhaustively applicable in $O(\alpha^2 nm)$ time.*

Exhaustively applying the reduction rules lets us bound the number of the remaining vertices linearly in α. This will be useful in the branching algorithm below.

Theorem 7 ($\star$). Seeded Highly Connected Edge Deletion *admits a problem kernel with at most $2\alpha + 4\alpha/k$ vertices and $\binom{2\alpha}{2} + \alpha$ edges computable in $O(\alpha^2 nm)$ time.*

In the subexponential branching algorithm, we use the following simple branching rule. It simply takes a vertex and branches on whether or not it should be added to the seed S for the desired highly connected graph.

Branching Rule 1. *If $\alpha + k \geq 0$, then choose an arbitrary vertex $v \in V \setminus S$ and branch into the cases of adding v to S or removing v from G. That is, create the instances $I_1 = (G, S \cup \{v\}, k - 1, \alpha)$ and $I_2 = (G - v, S, k, \alpha - \deg_G(v))$. Accept if I_1 or I_2 is accepted.*

It is clear that Branching Rule 1 is correct. We now describe the complete algorithm and bound its running time.

Theorem 8. *There is an $O(2^{4 \cdot \alpha^{0.75}} + \alpha^2 nm)$-time algorithm for* Highly Connected Edge Deletion.

Proof. We first apply the kernelization from Theorem 7, which entails applying Rules 5 and 6 exhaustively. Then, if $k \leq 2\sqrt{\alpha}$ we check whether $S \cup V'$ induces a highly connected subgraph, for every vertex subset $V' \subseteq V \setminus S$ of size k. We accept or reject accordingly. If $k > 2\sqrt{\alpha}$, then we apply Branching Rule 1 and recurse on the two created instances.

From the correctness of the rules it is clear that this algorithm finds a solution if there is one. Let us analyze its running time. Note that in each recursive call, except the first one, the input instance has $O(\alpha)$ vertices according to Theorem 7. Thus applying Rules 5 and 6 in a recursive call amounts to $O(\alpha^5)$ time except in the first one where it is $O(\alpha^2 nm)$ time, by Lemma 9. Next, in each recursive call we may have to check whether $S \cup V'$ is highly connected for all k-vertex subsets V'. This is done only after Rules 5 and 6 have been exhaustively applied and only if $k \leq 2\sqrt{\alpha}$. Thus, the graph G is of order at most $2\alpha + 4\alpha/k \leq 4\alpha$ (note that $k \geq 2$ without loss of generality). Hence, testing the subgraphs

amounts to $O((4\alpha)^{2\sqrt{\alpha}+2})$ time. In total, the time spent per search tree node is $O((4\alpha)^{\max\{5,2\sqrt{\alpha}+2\}})$.

Now let us bound the number of leaves C of the search tree. Note that the total number of search tree nodes is within a constant factor of C. For an instance $I = (G, S, k, \alpha)$ of Highly Connected Subgraph, consider the value $\mu(I) = k + \alpha$ in the root of the search tree, after applying Rules 5 and 6. Then, $\mu(I) \leq 3\alpha$ by Lemma 8. Let $C(\mu(I))$ denote an upper bound on the number of leaves that a search tree with a root with value $\mu(I)$ can have. Whenever we apply Branching Rule 1, μ is reduced by a certain amount. More precisely, $C(\mu(I))$ fulfills $C(0) = 1$, and $C(\mu(I)) \leq C(\mu(I_1)) + C(\mu(I_2))$. Hence, $C(\mu(I))$ is monotone. Further, since Rule 6 is not applicable, $\deg_G(v) \geq (|S| + k)/2 \geq k/2 \geq \sqrt{\alpha}$ in the application of Branching Rule 1. This implies $C(\mu(I)) \leq C(\mu(I) - 1) + C(\mu(I) - \sqrt{\alpha})$. Hence $C(\mu(I))$ is at most the number of paths in $\mathbb{R}^2$ from the origin to some point (x, y) that take only steps $(1, 0)$ or $(0, \sqrt{\alpha})$, where $x + y = \mu(I)$. Scaling the y-axis by a factor of $1/\sqrt{\alpha}$, computing $C(\mu(I))$ reduces to the problem of counting such paths from the origin to some (x, y') taking only steps $(1, 0)$ or $(0, 1)$ such that $x + \sqrt{\alpha}y' = \mu(I)$. We now bound the number of these paths.

The number of $(0, 1)$ steps is at most $3\sqrt{\alpha}$. If the path contains i $(0, 1)$-steps, then the total number of steps in the path is $i + 3\alpha - \sqrt{\alpha}i$. Hence, there are $\binom{i+3\alpha-\sqrt{\alpha}i}{i}$ paths with exactly i steps $(0, 1)$. This implies $C(\mu(I)) \leq \sum_{i=0}^{3\sqrt{\alpha}} \binom{i+3\alpha-\sqrt{\alpha}i}{i}$. To bound this number we use the fact that $\binom{a+b}{a} \leq 2^{2\sqrt{ab}}$ [8, Lemma 9]. Hence $C(\mu(I)) \leq \sum_{i=0}^{3\sqrt{\alpha}} 2^{2\sqrt{i\cdot(3\alpha-\sqrt{\alpha}i)}}$. Consider the derivative $f(i)$ of $2\sqrt{i \cdot (3\alpha - \sqrt{\alpha}i)}$ with respect to i. We have

$$f(i) = \frac{\sqrt{\alpha}(3\sqrt{\alpha} - 2i)}{\sqrt{\sqrt{\alpha}i(3\sqrt{\alpha} - i)}}.$$

Inspecting $f(i)$ shows that $\sqrt{i \cdot (3\alpha - \sqrt{\alpha}i)}$ is maximized over $0 \leq i \leq 3\sqrt{\alpha}$ if $i = 3\sqrt{\alpha}/2$. This gives $C(\mu(I)) \leq 3\sqrt{\alpha} \cdot 2^{3\sqrt{\alpha\sqrt{\alpha}}}$. Finally,

$$3\sqrt{\alpha} \cdot 2^{3\sqrt{\alpha\sqrt{\alpha}}} \cdot (4\alpha)^{\max\{5,2\sqrt{\alpha}+2\}} \in O(2^{4\alpha^{0.75}}),$$

giving the overall running time bound of $O(2^{4\alpha^{0.75}} + \alpha^2 nm)$. □

Although the presented algorithm is a subexponential-time algorithm with relatively small constants in the exponential functions, it is unclear whether it can be useful in practice. This is because the parameter α is likely to be large in real-world instances. With further substantial running time improvements, however, one might obtain practical algorithms. For example, an algorithm with running time $O(2^{\alpha^{0.5}} \cdot nm)$ should perform well on many real-word instances.

References

[1] D. Binkele-Raible, H. Fernau, F. V. Fomin, D. Lokshtanov, S. Saurabh, and Y. Villanger. Kernel(s) for problems with no kernel: On out-trees with many leaves. *ACM Trans. Algorithms*, 8(4):38, 2012.

[2] H. L. Bodlaender, R. G. Downey, M. R. Fellows, and D. Hermelin. On problems without polynomial kernels. *J. Comput. Syst. Sci.*, 75(8):423–434, 2009.
[3] G. Chartrand. A graph-theoretic approach to a communications problem. *SIAM J. Appl. Math.*, 14(4):778–781, 1966.
[4] J. Chen, I. A. Kanj, and G. Xia. Improved upper bounds for vertex cover. *Theor. Comput. Sci.*, 411(40–42), 2010.
[5] R. G. Downey and M. R. Fellows. *Fundamentals of Parameterized Complexity.* Texts in Computer Science. Springer, 2013.
[6] D. Eppstein, M. Löffler, and D. Strash. Listing all maximal cliques in sparse graphs in near-optimal time. In *Proc. 22nd ISAAC*, volume 6506 of *LNCS*, pages 403–414. Springer, 2010.
[7] M. R. Fellows, B. M. P. Jansen, and F. A. Rosamond. Towards fully multivariate algorithmics: Parameter ecology and the deconstruction of computational complexity. *Eur. J. Combinatorics*, 34(3):541–566, 2013.
[8] F. V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and Y. Villanger. Tight bounds for parameterized complexity of cluster editing with a small number of clusters. *J. Comput. Syst. Sci.*, 80(7):1430–1447, 2014.
[9] E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Inf. Process. Lett.*, 76(4–6):175–181, 2000.
[10] Q.-S. Hua, Y. Wang, D. Yu, and F. C. M. Lau. Dynamic programming based algorithms for set multicover and multiset multicover problems. *Theor. Comput. Sci.*, 411(26-28):2467–2474, 2010.
[11] F. Hüffner, C. Komusiewicz, A. Liebtrau, and R. Niedermeier. Partitioning biological networks into highly connected clusters with maximum edge coverage. *IEEE/ACM Trans. Comput. Biol. Bioinf.*, 11(3):455–467, 2014.
[12] H. Ito and K. Iwama. Enumeration of isolated cliques and pseudo-cliques. *ACM Trans. Algorithms*, 5(4):Article 40, 2009.
[13] H. Ito, K. Iwama, and T. Osumi. Linear-time enumeration of isolated cliques. In *Proc. 13th ESA*, volume 3669 of *LNCS*, pages 119–130. Springer, 2005.
[14] C. Komusiewicz, F. Hüffner, H. Moser, and R. Niedermeier. Isolation concepts for efficiently enumerating dense subgraphs. *Theor. Comput. Sci.*, 410(38-40):3640–3654, 2009.
[15] G. Liu and L. Wong. Effective pruning techniques for mining quasi-cliques. In *Proc. ECML/PKDD*, volume 5212 of *LNCS*, pages 33–49. Springer, 2008.
[16] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 105:41–71, 2011.
[17] H. Matsuda, T. Ishihara, and A. Hashimoto. Classifying molecular sequences using a linkage graph with their pairwise similarities. *Theor. Comput. Sci.*, 210(2):305–325, 1999.
[18] J. Pattillo, N. Youssef, and S. Butenko. On clique relaxation models in network analysis. *Eur. J. Operational Research*, 226(1):9–18, 2013.
[19] R. Sharan, I. Ulitsky, and R. Shamir. Network-based prediction of protein function. *Mol. Syst. Biol.*, 3:88, 2007.
[20] A. Veremyev, O. A. Prokopyev, V. Boginski, and E. L. Pasiliao. Finding maximum subgraphs with relatively large vertex connectivity. *Eur. J. Operational Research*, 239(2):349–362, 2014.