Partitioning Biological Networks into Highly Connected Clusters with Maximum Edge Coverage

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

[1] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
{falk.hueffner,christian.komusiewicz,rolf.niedermeier}@tu-berlin.de
[2] Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany

Abstract. We introduce the combinatorial optimization problem Highly Connected Deletion, which asks for removing as few edges as possible from a graph such that the resulting graph consists of highly connected components. We show that Highly Connected Deletion is NP-hard and provide a fixed-parameter algorithm and a kernelization. We propose exact and heuristic solution strategies, based on polynomial-time data reduction rules and integer linear programming with column generation. The data reduction typically identifies 85 % of the edges that need to be deleted for an optimal solution; the column generation method can then optimally solve protein interaction networks with up to 5 000 vertices and 12 000 edges.

1 Introduction

A key idea of graph-based data clustering is to identify densely connected subgraphs (clusters) that have many interactions within themselves and few with the rest of the graph [1, 35, 43, 44]. Hartuv and Shamir [21] proposed a clustering algorithm producing so-called *highly connected* clusters. Their method has been successfully used to cluster cDNA fingerprints [22], to find complexes in protein–protein interaction (PPI) data [23, 39], to group protein sequences hierarchically into superfamily and family clusters [30], and to find families of regulatory RNA structures [38]. Hartuv and Shamir [21] 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 [12]. The concept of a highly connected graph is very similar to that of a *0.5-quasi-complete graph* [26, 33], that is, a graph where every vertex has degree at least $(n-1)/2$. Further, being highly connected also ensures that the diameter of a cluster is at most two [21].

The algorithm by Hartuv and Shamir [21] 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. [3] 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.

Note that, by definition, isolated edges are *not* highly connected. Hence, the smallest clusters are triangles; we consider all singletons as *unclustered*. The problem formulation resembles the Cluster Deletion problem [42], 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 [31], 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 also have diameter at most two [21], 2-Club Deletion poses a looser demand on intra-cluster connectivity.

It could be expected that the algorithm by Hartuv and Shamir [21] 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 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 counteract with postprocessing [21]. This tendency might introduce systematic bias [29]. Hence, exact algorithms for solving Highly Connected Deletion are desirable.

Our contributions. We analyze the (parameterized) computational complexity of Highly Connected Deletion and propose several exact solution methods.[4]

[3] The CLICK algorithm [17] and the SIDES algorithm [29] follow the same scheme, but use edge weights and different stopping criteria, based on probabilistic models.

[4] Exact (in contrast to heuristic and approximate) algorithms may be beneficial for several reasons. The availability of optimal solutions can be used to evaluate the

In particular, we show that Highly Connected Deletion is NP-hard even on 4-regular graphs and, provided the Exponential Time Hypothesis (ETH) [24] 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".

Geared more towards practical methods, we design several polynomial-time executable data reduction rules for Highly Connected Deletion preserving the possibility to solve the problem exactly. Further, we develop two ILP formulations (one using row generation and one using column generation) and show how, combined with the presented data reduction rules, real-world instances with, e. g., 5 000 vertices and 12 000 edges can be solved within several hours. From an algorithmic standpoint we thus make progress towards exact algorithms for NP-hard clustering problems on biological networks. The related application area of comparative network analysis has seen systematic algorithmic advances in recent years that have made it possible to find exact solutions for hard problems [3]. We believe that such advances should also be made for clustering problems and present a first step for one concrete problem.

On the biological and experimental side, we focus on the analysis of the three species *A. thaliana*, *C. elegans*, and *M. musculus* with moderate size protein interaction networks. We compare our approach with Hartuv and Shamir's and Markov clustering. While our exact approach is clearly the slowest when compared with the other two, we demonstrate that it mostly outperforms them in terms of quality of the reported clusters.

More specifically, first we observe that our data reduction rules typically identify more than 80 % of the edges to be deleted for an optimal solution for Highly Connected Deletion. Combining our data reduction rules with Hartuv and Shamir's algorithm significantly improves this algorithm's running time and the quality of its reported clusters. Still, with our data reduction and ILP approach we typically find more biologically relevant clusters[5] than the Hartuv and Shamir method extended with our data reduction algorithm does. Further, we compared our results with a state-of-the-art algorithm based on Markov clustering. Again, while this algorithm shows much faster running time and better coverage (higher number of vertices (that is, proteins) assigned to clusters), our algorithm is superior in terms of cluster quality.

To address the drawback that our approach has much higher running time, we find that our ILP formulation based on column generation can also be easily transformed into a fast heuristic providing within at most one hour high-

quality and performance of heuristics, help to separate possible model inadequacies from errors introduced by heuristic solutions, and along with the clear combinatorial model, they are easier to interpret; refer to Aloise et al. [2] for a deeper elaboration.

[5] Measured by determining for each cluster whether its protein set has a statistically significant enrichment of annotations describing processes in which the respective protein takes part.

quality results substantially superior to those provided by the min-cut algorithm. Moreover, we observe that our clear combinatorial model of the clustering process (formalization as Highly Connected Deletion problem) significantly helps to do a further simple and efficient postprocessing for our method that helps assigning low-degree vertices to clusters. In this way, the vertex coverage thus reaches two thirds of the Markov clustering algorithm (which has excellent coverage), still maintaining the superior quality of our clustering.

Preliminaries. We consider only undirected and simple graphs $G = (V, E)$. We use n and m to denote the number of vertices and edges in the input graph, respectively, 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 [15, 16, 36], 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) [24] 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. [32] for a recent survey.

2 Computational Complexity

We present a proof that Highly Connected Deletion is NP-hard even on 4-regular graphs and that it cannot be solved in subexponential time unless the exponential-time hypothesis (ETH) [24] is false.

As recently shown by van Rooij et al. [46], 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 [46]; 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 $G =$

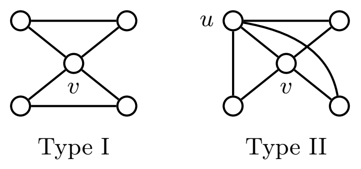


Fig. 1. The two different neighborhoods in a 4-regular neighborhood restricted graph. None of these graphs contains a clique of order four.

> (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 [28]. 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 [46]. 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 $|V| = k = |E|/2$.

Variants. If we allow insertion of edges instead of deletion, the problem reduces to increasing the connectivity of a graph to $\lambda = \lceil n/2 \rceil$ by edge insertions. For arbitrary λ, this problem can be solved in polynomial time [10].

The vertex deletion version is NP-hard by a simple adaption of the proof for the NP-hardness of 2-Club Cluster Vertex Deletion by Liu et al. [31]: given a Vertex Cover instance, create a Highly Connected Vertex Deletion instance by attaching to each vertex a large clique. It is then easy to see that the graph has a vertex cover of size at most k if and only if at most k vertices can be deleted to leave a graph where every component is highly connected.

3 Parameterized Complexity

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* [20] 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.

3.1 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, then 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, Rules 2 and 3 yield a linear-time algorithm for Highly Connected Deletion on graphs of maximum degree three, which together with Theorem 1 implies a complexity dichotomy with respect to the maximum degree.

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.

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 [18, 27]. 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. $\square$

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 *can be reduced in* $O(n^2 \cdot mk \log n)$ *time to an equivalent instance, called problem kernel, with at most* $10 \cdot k^{1.5}$ *vertices.*

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. Hence, the overall instance size is at most

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

A simple calculation shows that there is an assignment to the c_i's maximizing the sum such that at most one c_i 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 are at most $2k/0.3\sqrt{2k} = 10\sqrt{2k}/3$ such clusters. The overall instance size follows. □

3.2 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 [6], and the best known fixed-parameter algorithm for 2-Club Deletion runs in $2.74^k \cdot n^{O(1)}$ time [31].

The main idea of our algorithm is to branch until each connected component has diameter at most 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 [21], 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 are 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.

4 Practical Algorithms

The fixed-parameter tractability results for Highly Connected Deletion (Section 3.2) are currently mostly of theoretical nature. Hence, we follow an algorithmic approach that consists of two main steps: First, apply a set of *data reduction rules* that exploit the structure of biological networks and yield a new instance that is significantly smaller than the original one. Second, solve the new, smaller instance by devising an *integer linear programming* (ILP) formulation.[6] The resulting ILP can be solved using sophisticated ILP solvers which are very efficient. We devise two different ILP formulations. The more efficient one uses a nonstandard column generation approach.

[6] ILP formulations have proved useful for attacking other hard graph problems involving dense subgraphs such as computation of edge-weighted quasi-bicliques [11].

4.1 Further Data Reduction

As we demonstrate in the computational experiments presented in Section 5, Rule 3 tremendously simplifies many real-world input instances. In particular, it is useful to reduce vertices of small degree, as found in protein interaction networks; for instance, it can be proved that Highly Connected Deletion can be solved in linear time with Rules 2 and 3 when the input graph has degree at most three. However, Rules 4 and 5 that produce a kernel have the downside of requiring relatively large substructures. To improve performance in practice, we use the following two rules.

We try to identify triangles uvw that must form highly connected clusters. For a triangle edge $\{x, y\}$, let $N_{xy} := (N(x) \cup N(y)) \setminus \{u, v, w\}$ be the common neighbors of the edge outside the triangle. Let the value of an edge e be 3 if $N_e \neq \emptyset$ and 0 otherwise. Let the value of a vertex x be the size of the largest connected component in $G[N(x) \setminus \{u, v, w\}]$, or 0 if this size is 1.

Rule 6. Assume that for a triangle uvw the following conditions hold:

- for no two triangle edges $\{x, y\}, \{x, z\}$ ($\{x, y, z\} = \{u, v, w\}$) there is an edge in G between some vertex in N_{xy} and some vertex in N_{xz};
- for no triangle edge e is there an edge in $G[N_e]$;
- for any $\{x, y, z\} = \{u, v, w\}$, the value of $\{x, y\}$ plus the value of z is at most 3;
- the sum of the values of u, v, and w is at most three.

Then isolate the triangle by deleting all edges incident on u, v, and w except the triangle edges.

Proof (of correctness). By case distinction: if the triangle is not a solution cluster, then it must be part of a larger cluster, or the vertices are divided into two or three clusters. The conditions ensure that none of these situations yield a better solution than isolating the triangle. □

The following rule reduces some low-degree vertices.

Rule 7. Let u be a vertex and $N_2(u)$ be the neighbors of u that have degree 2. If $G[N_2(u)]$ contains an edge, then isolate all vertices of degree 0 in $G[N_2(u)]$. Otherwise, if there is a vertex v that is in G a neighbor of a vertex w in $N_2(u)$ and has degree 3 in G, then delete the edge from v to the neighbor that is not u or w.

Proof (of correctness). The vertex u can be contained in at most one triangle. Each of the deleted edges could only be part of a triangle with u, and for each such triangle there is another triangle which destroys fewer opportunities of using vertices for other clusters. □

4.2 Integer Linear Programming: Row Generation

The idea is to formulate Highly Connected Deletion as a Clique Partitioning problem. In this problem, vertex pairs are annotated as being similar or as being dissimilar, and the goal is to find a partition of the vertices that maximizes consistency with these annotations. We model the partition of the vertices as a *cluster graph*, that is, a graph where every connected component is a clique. The formal problem definition is then as follows:

Clique Partitioning
Instance: A vertex set V with a weight function $\delta : \binom{V}{2} \to \mathbb{Q}$.
Task: Find a cluster graph (V, E) that minimizes $\sum_{\{u,v\}\in E} \delta(u, v)$.

Herein, $\delta(u, v)$ denotes the *dissimilarity* between u and v. Clique Partitioning with the slightly modified objective

$$\sum_{\substack{\delta(u,v)>0 \\ \{u,v\}\notin E}} \delta(u, v) - \sum_{\substack{\delta(u,v)<0 \\ \{u,v\}\in E}} \delta(u, v), \tag{3}$$

which differs from the objective of Clique Partitioning only by the constant $\sum_{\delta(u,v)<0} \delta(u, v)$, is known as Correlation Clustering (e. g. [4]) or Weighted Cluster Editing (e. g. [9]). In this formulation, the objective can be thought of as minimizing the cost of edge deletions (first sum) and edge insertions (second sum) to transform a graph into a cluster graph.

To obtain a Clique Partitioning instance from a Highly Connected Deletion instance, we set $\delta(u, v) = -1$ if $\{u, v\} \in E$ and 0 otherwise. We will need additional constraints to enforce that each cluster is highly connected, which will be described later.

There is a well-known ILP formulation of Clique Partitioning [19, 41], which we can adapt for Highly Connected Deletion. It has been successfully implemented and augmented with cutting planes [9, 19, 37] to speed up the solving process. It uses binary variables e_{uv} for $u, v \in V, u < v$, where $e_{uv} = 1$ iff the edge $\{u, v\}$ is part of the solution cluster graph. Cluster graphs are exactly those graphs that do not contain a P_3 as induced subgraph, that is, three distinct vertices u, v, w with $\{u, v\} \in E$ and $\{v, w\} \in E$ but $\{u, w\} \notin E$. Thus, we can ensure that the graph is a cluster graph by avoiding a P_3 for each possible triple of vertices $u < v < w \in V$:

$$e_{uv} + e_{vw} - e_{uw} \leq 1 \tag{4}$$
$$e_{uv} - e_{vw} + e_{uw} \leq 1 \tag{5}$$
$$-e_{uv} + e_{vw} + e_{uw} \leq 1 \tag{6}$$

The objective is to minimize $\sum_{\{u,v\}\in E} \delta(u, v)e_{uv}$.

This ILP has a number of variables quadratic in n; however, we can shrink it substantially in practice by substituting $e_{uv} = 0$ for all pairs uv that according to Lemma 3 cannot be in the same cluster. Moreover, the number of constraints is $3\binom{n}{3} = O(n^3)$, and thus can get easily too large for memory. Therefore, we implemented a row generation scheme ("lazy constraints") as suggested by Grötschel and Wakabayashi [19]. That is, we initially add none of the constraints (4)–(6). When the ILP solver finds a putative solution, we check for violated constraints by a simple brute-force search, and add up to 300 of those constraints that are violated the most.

It remains to describe how to constrain clusters to be highly connected. As mentioned in the introduction, the following lemma is a corollary to a result of [12], which states that if the minimum degree in a graph with $n \geq 2$ vertices is at least $\lfloor n/2 \rfloor$, then the minimum degree equals its connectivity.

Lemma 8. *A graph with $n \geq 2$ vertices is highly connected if and only if each vertex has degree larger than $n/2$.*

The size of the cluster that vertex u is contained in is $\sum_{v \in V} e_{uv}$, and the degree of u in the cluster is $\sum_{v \in N(u)} e_{uv}$. Thus, by Lemma 8, if the cluster has size at least 2, we can express that it is highly connected by the linear constraint

$$\sum_{v \in N(u)} e_{uv} > \sum_{v \in V} e_{uv} \Big/ 2 \tag{7}$$

$$\Leftrightarrow \sum_{v \in N(u)} e_{uv} - \sum_{v \in V \setminus N(u)} e_{uv} \geq 1 \tag{8}$$

Unfortunately, this inequation does not hold for singleton clusters. Thus, we introduce additional binary variables $v_u, u \in V$, which indicate that a variable is in a nonsingleton cluster. Replacing the right-hand side of (8) with v_u will ensure that the condition is also fulfilled for singleton clusters. Further, we need the constraints

$$\forall \{u, v\} \in E : e_{uv} \leq v_u, e_{uv} \leq v_v \tag{9}$$

to ensure consistency between the vertex and edge variables.

4.3 Integer Linear Programming: Column Generation

In Section 4.2, we presented a more straightforward approach based on a Clique Partitioning formulation and row generation. The next approach is more involved, due to the use of column generation. Our experiments show that the extra complexity pays off and the column generation approach can solve larger instances exactly.

We describe an ILP formulation of Highly Connected Deletion, which in its basic scheme is similar to those of Mehrotra and Trick [34] and Ji and Mitchell [25] for constrained Clique Partitioning and that of Aloise et al. [2] for modularity maximization; however, we need a new approach for solving the column generation subproblem. Let $\mathcal{T}$ be the set of all vertex sets that induce a highly connected subgraph. We use binary variables z_T to indicate that the cluster $T \in \mathcal{T}$ is part of the solution. Then the model is

$$\text{maximize} \sum_{T \in \mathcal{T}} c_T z_T, \tag{10}$$

$$\text{s.t.} \sum_{\{T \in \mathcal{T} | u \in T\}} z_T = 1 \quad \forall u \in V, \tag{11}$$

$$z_T \in \{0, 1\} \quad \forall T \in \mathcal{T}, \tag{12}$$

where c_T is the number of edges in the subgraph induced by t. The objective (10) maximizes the number of edges within clusters, which equivalent to minimizing the number of inter-cluster edges (deletions). The constraints of type (11) ensure that each vertex is contained in exactly one cluster.

Due to the large number of variables, this model cannot be solved directly except for tiny instances. Thus, the idea is to only consider "relevant" variables. More precisely, we start with an initial set of z_T variables that yields a feasible solution (e. g., all singleton clusters). Then we successively add variables ("columns") that improve the objective, until this is no longer possible. Due to the structure of real-world instance, typically only a small subset of possible variables needs to be added.

Now the improvement of adding a column for cluster T is c_T minus the contribution of the vertices in T to the objective function. This contribution for some vertex u can be calculated as the value of the dual variable λ_u for the corresponding constraint of type (11) in the continuous relaxation of the problem (10)–(12) (see e. g. Aloise et al. [2] for details). The values of the dual variables can be easily calculated by a linear programming solver. Thus, we need to find a cluster T that maximizes $c_T - \sum_{u \in T} \lambda_u$. In other words, we need to find a highly connected cluster that maximizes the number of edges minus vertex weights. For this, we again use an ILP formulation, using binary edge variables e_{uv} and binary vertex variables v_u to describe the cluster selected, and a positive integral variable d to describe the cluster size:

$$\text{maximize} \sum_{\{u,v\} \in E} e_{uv} - \sum_{u \in t} \lambda_u v_u, \tag{13}$$

$$\text{s.t. } d = \sum_{u \in V} v_u, \tag{14}$$

$$e_{uv} \leq v_u, e_{vu} \leq v_v \quad \forall \{u, v\} \in E, \tag{15}$$

$$\text{if } v_u \text{ then} \sum_{v \in N(u)} e_{uv} > d/2 \quad \forall u \in V, \tag{16}$$

where the constraint (16) can be linearized using the big-M method (that is, by adding $M(1 - v_u)$ on the left-hand side with a sufficiently large constant M); in our implementation, we instead use indicator constraints as supported by CPLEX.

We can make use of the fact that it is not necessary to find a maximally improving column. Therefore, we can solve the column generation problem

Table 1. Instance properties and data reduction results. Here, K is the number of connected components, n' and m' are the number of vertices and edges in the largest connected component, respectively, Δk is the number of edges deleted during data reduction, K' is the number of connected components after data reduction, and n'' and m'' are the number of vertices and edges in the largest connected component after data reduction, respectively.

	n	m	K	n'	m'	Δk	Δk [%]	K'	n''	m''
CE phys.	157	153	39	23	24	100	92.6	137	11	38
CE all	3613	6828	73	3434	6721	5204	80.1	3202	373	1562
MM phys.	4146	7097	114	3844	6907	5659	85.3	3656	426	1339
MM all	5252	9640	135	4890	9407	7609	84.8	4566	595	1893
AT phys.	1872	2828	82	1625	2635	2057	83.1	1605	187	619
AT all	5704	12627	128	5393	12429	8797	79.5	4579	866	3323
SP all	2698	16089	17	2661	16065	2936	$\geq$ 18.2	1299	1372	13111

heuristically, and only solve it optimally using the ILP when no improving solution was found. As heuristic, we use a simple greedy method that starting from each vertex repeatedly adds the vertex that maximizes the value of the cluster, and records the best cluster that was highly connected. Further, we abort solving the column generation ILP as soon as an improving solution is found.

5 Experimental Evaluation

5.1 Experimental Setup & Running Time Evaluation

We implemented the data reduction in OCaml and the ILPs in C++ using the CPLEX 12.4 ILP solver. For the minimum cut subroutine of the algorithm of Hartuv and Shamir [21] (called *min-cut method* below), a highly optimized implementation in C was used [13]. Our source code and sample instances are available at http://www.user.tu-berlin.de/hueffner/hcd/. The test machine is a 3.6 GHz Intel Xeon E5-1620 with 10 MB L3 cache and 64 GB main memory, running under Debian GNU/Linux 7.0. Only a single thread was used.

Test instances. We used protein interaction networks available at the BIOGRID repository [45]. The three species for which we illustrate our results are *A. thaliana*, *C. elegans*, and *M. musculus*. For each species, we extracted one network with physical interactions only, and one with all interactions. In Section 5.3, we also consider the network of all interactions of *S. pombe*. Table 1 shows some basic properties of these networks. For the computation of the enrichment of annotation terms, we used the GO:TermFinder tool [7] with *A. thaliana* annotation data from the TAIR database [5]. The computed p-values are corrected for multiple hypothesis testing. We used a significance threshold of $p \leq 0.01$.

Running time evaluation. Table 1 shows the effect of data reduction. Knowing the optimal k (see Table 2) allows us to state that typically 85 % of the edges that need to be deleted are identified. Since connected components can be treated separately,

Table 2. Results for the instances of Table 1. Here, k is the number of edges deleted, s and K are the number of singleton and nonsingleton clusters, respectively, n and m are the number of vertices and edges in the largest cluster, respectively, and t is the running time in seconds.

	min-cut without DR						min-cut with DR						column generation with DR					
	k	s	K	n	m	t	k	s	K	n	m	t	k	s	K	n	m	t
CE-p	111	136	5	9	30	0.01	108	133	6	9	30	0.01	108	133	6	9	30	0.06
CE-a	6714	3589	2	17	94	86.46	6630	3521	22	17	94	6.36	6499	3436	45	19	113	2088.35
MM-p	7004	4116	5	12	57	126.30	6882	4003	41	12	57	7.42	6638	3845	80	11	41	898.13
MM-a	9563	5227	5	13	65	267.63	9336	5044	61	13	65	17.84	8978	4812	120	13	65	3858.62
AT-p	2671	1796	19	14	76	5.82	2567	1723	39	14	76	0.68	2476	1675	49	14	76	60.34
AT-a	12096	5559	23	23	190	434.52	11590	5213	122	23	190	32.09	11069	4944	180	23	190	34121.23

the most important time factor is the size of the largest connected component. Here, the number of edges is reduced to typically 23 %. This demonstrates the effectivity of the data reduction, which preserves exact solvability, and suggests it should be applied regardless of the actual solution method.

Table 2 shows the clustering results and running times. Doing data reduction before running the min-cut method actually improves the running time, since it reduces the number of costly min-cut calls. The column generation method is able to solve all six test instances, although the hardest one takes more than 9 hours. However, it is not able to solve e. g. the network of all interactions of *S. pombe* with 1541 vertices and 3036 edges; this is probably because this is a denser network, making data reduction less effective.

We omit the results for the row generation method (Section 4.2), because it was not able to solve three of the instances within 12 hours. Thus, for exactly solving Highly Connected Deletion, it is clearly preferable to use the column generation method despite the added complexity. Data reduction for column generation always improves running time, for example for AT-p by a factor of 20.

5.2 Biological evaluation

For the biological evaluation, we studied the *A. thaliana* network with all interactions in more detail since it was the largest instance for which the exact algorithm finished. Our findings are summarized in Figure 2. Solving Highly Connected Deletion exactly produces more clusters than using the min-cut algorithm with data reduction which in turn produces more clusters than the min-cut algorithm without data reduction. This behavior can be observed for small and for larger clusters.

To assess the biological relevance of these clusters, we determined for each cluster whether the corresponding protein set has a statistically significant enrichment of annotations describing processes in which the protein take part. As shown in Fig. 2, for all three methods a large portion of clusters shows such an enrichment. The min-cut algorithm with data reduction clearly outperforms the min-cut algorithm without data reduction: it produces more clusters without producing a larger fraction of nonenriched clusters. For the exact algorithm the

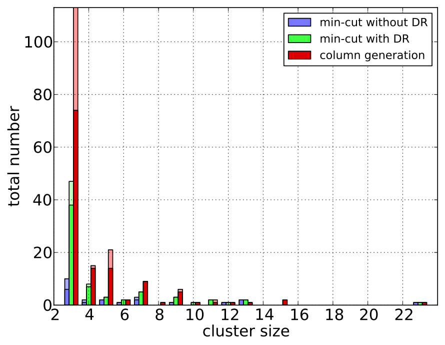


Fig. 2. Clusters in the *A. thaliana* network with all interactions. The brighter part of each bar shows the fraction of clusters without significant enrichment of biological process annotation terms.

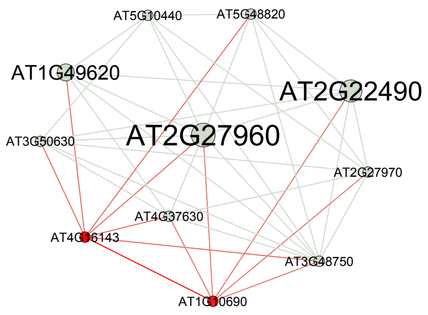
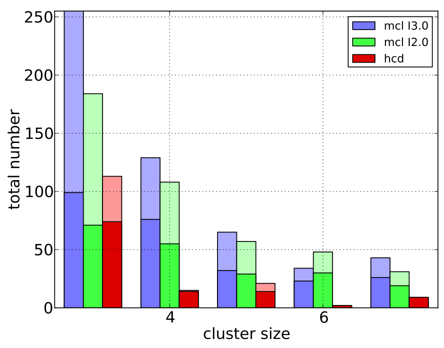


Fig. 3. An example cluster in the *A. thaliana* network.

results are less clear: it produces even more clusters, but a larger fraction is nonenriched. This behavior is particularly pronounced for small clusters of size at most three, but also for some larger cluster sizes.

Biological example. Fig. 3 gives an example of a cluster found in the *A. thaliana* network with physical interactions. The annotation terms with the lowest p-values for this cluster were 'negative regulation of cyclin-dependent protein kinase activity' and the related 'negative regulation of cell cycle'. Both terms are common annotations for 9 out of the 11 proteins in the cluster. The two proteins that are not annotated with this term are colored red in Fig. 3. The protein labeled 'AT4G16143' plays a role in protein import into the cell nucleus; the role of the protein labeled 'AT1G10690' is unknown. The cluster produced by our algorithm suggests to examine whether this protein plays a role in the cell cycle regulation.

Interestingly, this cluster is completely destroyed by both nonexact approaches, that is, *all* of the proteins end up in singleton clusters when using these approaches.



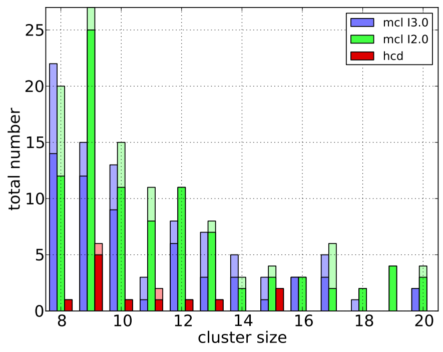
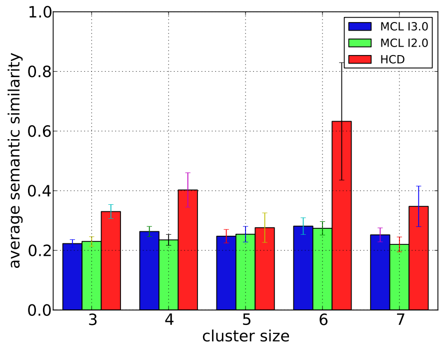


Fig. 4. Clusters in the *A. thaliana* networks produced by the MCL algorithm and our algorithm (HCD)The brighter part of each bar shows the fraction of clusters without significant enrichment of biological process annotation terms.

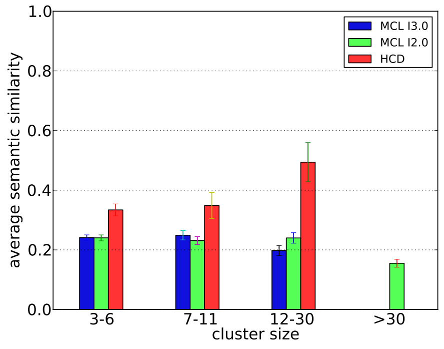
Comparison with Markov Clustering. Next, we compare our clustering algorithm with a popular clustering algorithm for protein interaction networks. As comparison, we choose the so-called Markov Clustering Algorithm (MCL), which was shown to outperform several other clustering algorithms on protein interaction networks [8]. For details concerning MCL refer to [14]; in the experiments, we used the MCL-implementation available at http://micans.org. One parameter that can be set when using MCL is the "inflation" I. We performed experiments with the default value of $I = 2.0$ and with $I = 3.0$ which produces a more fine-grained clustering (as does our algorithm). Unless stated otherwise, we use MCL to refer to the algorithm with default setting.

When comparing the two algorithms, our exact approach (in the following referred to as HCD) and the MCL algorithm, there are some clear advantages of the MCL algorithm: MCL finishes within less than a second, MCL assigns almost all proteins to nonsingleton clusters, and MCL produces more clusters than HCD. MCL also produces larger clusters than HCD. For instance, it finds 30 clusters of size more than 20, and the largest cluster has size 280. As shown in Figure 4, the number of produced clusters is higher across all cluster sizes. The fraction of clusters whose proteins share a significantly enriched GO annotation term, however, is for small and medium-size clusters much lower in the clustering produced by MCL than in the clustering produced by HCD. For large clusters (not shown), 85% of the clusters produced by MCL show a significant enrichment of some annotation term.

In order to assess how informative the clusters provided by the algorithms are, we examined the nature of the relevant enriched terms more closely. For instance, the largest cluster produced by MCL has 280 proteins and the annotation term with the lowest p-value was 'transport' shared by 30% of the proteins, followed by 'localization'. In contrast, the largest cluster produced by HCD only has size



(a) Small clusters for *A. thaliana*.



(b) Cluster categories for *A. thaliana*.

Fig. 5. The average pairwise semantic similarity for protein pairs in the same clusters grouped by cluster sizes; error bars show standard error.

23. The term with the lowest p-value is 'negative regulation of cyclin-dependent protein kinase activity' shared by 21 of 23 proteins.

To provide a more systematic analysis of the similarity of annotation terms for the clusters, we computed for each protein pair in the same cluster a semantic similarity score for the GO annotations using the score definition of Wang et al. [47]. The computed scores lie in $[0, 1]$; a higher score indicates higher similarity between the two considered proteins. The average semantic similarity score for a protein pair in the same cluster is 0.410 for HCD and 0.192 for MCL. This pure numeric score, however, could be skewed in favor of HCD: since MCL produces larger clusters, it would be acceptable that the pairs show less similarity because the obtained clustering could simply be coarser. We therefore further examined the effect of the cluster size on the average semantic score for protein pairs in the same cluster. Our results are shown in Figure 5. For small clusters, our clusters show better similarity values for all cluster sizes with the exception of size-five clusters, where the difference is marginal. Note that the similarity value increases for size-six clusters and then decreases again for size-seven clusters. We believe that this behavior is due to the following fact: Size-five clusters are the smallest clusters that can have a missing edge, they can contain vertices with $3 = 5 - 2$ neighbors in the cluster. For size-six clusters, every protein can again have at most one missing neighbor and thus has at least $4 = 6 - 2$ neighbors in the cluster. Now, size-seven clusters can contain proteins with *two* missing neighbors. This might indicate that this "jump" in semantic similarity is due to the rounding effect in the definition of highly connected graphs.

Next, we grouped the reported clusters into four categories: small (3–6 proteins), medium (7–11 proteins), large (12–30 proteins), and very large (> 30 proteins) and computed the average similarity scores for clusters in these categories. While HCD did not find any very large clusters, the average similarity score is, for the three other categories, higher in HCD than in MCL. Our results also confirm that the very large clusters show lower pairwise similarity than the small clusters. Summarizing, our results for the *A. thaliana* network indicate that HCD outperforms MCL in terms of quality of the reported clusters while MCL shows better coverage and a better running time.

5.3 Variants & Extensions

The comparison of HCD with the MCL clustering algorithm showed that two drawbacks of HCD are the running time explosion and the fact that a large fraction of proteins remains unclustered in the optimal HCD solution. We discuss here two strategies to lessen both drawbacks.

Heuristics. The exact column generation approach is not able to solve the hardest instances. Therefore, we consider a heuristic variant, where we stop the column generation process after a time limit is exceeded, and compare it to the min-cut algorithm with and without data reduction (**??**). In the network of all interactions of *S. pombe*, all methods find a dense core with 53 vertices and 899 edges. The min-cut algorithm finds in addition only three triangles; with data reduction, another 13 triangles are found. Clearly, these results are not very informative. Column generation with a time limit of 15 minutes finds 16 more clusters with sizes ranging from 9 to 21 vertices; with a time limit of one hour, 120 more mostly smaller clusters are found. Extending the running time further does not yield substantial changes. Thus, the column-generation based heuristic finds many potentially interesting clusters where the min-cut algorithm fails.

Post-Processing of the Clustering. Another intrinsic problem of demanding highly-connected clusters is the fact that biological networks contain many low-degree vertices. For instance, the *A. thaliana* network with all interactions contains 5704 proteins of which 2407 have degree one and 1033 have degree two. These proteins cannot be contained in any highly connected cluster. Hence, HCD essentially computes a clustering of the dense core of the network.

Similar to a post-processing suggested by Hartuv and Shamir [21], we used the following simple post-processing to "readd" the proteins not included in any cluster returned by HCD: For each unclustered protein p, check whether there is a cluster C in the solution such that $N(p) \subseteq C$, that is, a cluster that contains *all* proteins for which interactions with p are known. If there is such a cluster, then add p to the set C_x. Finally, for each cluster C output $C_x \cup C$ as a new cluster. An overview of our first results is shown below. Compared to HCD, the new clustering has a much higher number of clustered proteins; the number of clustered proteins is now roughly two thirds of the number of proteins clustered by MCL. Furthermore, a first examination of the enrichment statistics indicates that this version of HCD produces better clusters than MCL.

6 Outlook

Proposing the NP-hard problem Highly Connected Deletion, we introduced a more combinatorial view on partitioning into highly connected components with

Table 3. Number of clustered proteins and fraction of proteins in clusters with significantly enriched annotation terms for MCL, HCD and HCD with post-processing (HCD-ext).

network	algorithm	clustered	enriched	enriched/clustered
A. thaliana all	MCL I2.0	5114	3703	0.724
	HCD	751	575	0.765
	HCD ext.	3067	2641	0.861

a clearly defined (and naturally interpretable) optimization goal. We developed practically useful and efficient data reduction rules which can be combined with different approaches, including heuristic and ILP-based ones. Hence, we suggest these data reduction rules for wider use. Furthermore, we chartered the border of practical feasibility for finding optimal solutions to Highly Connected Deletion, showing that medium-size instances can be solved within few hours. Finally, we demonstrated the practical relevance of our approach for clustering protein interaction networks, favorably comparing with established clustering approaches without clear optimization goal (on the contrary, Highly Connected Deletion seeks to minimize the number of deleted network edges). We believe that our simple formal model of partitioning into highly connected clusters provides a viable alternative to existing approaches, first of all justified by the quality of the achieved clusterings.

We conclude with a few promising directions for future work. We plan to perform further evaluation of the quality of the clusters found by our approach. First, we plan to evaluate the column-generation-based heuristic on larger standard protein interaction networks such as *S. cerevisiae* and perform comparisons with further clustering algorithms, for example the RN algorithm [40]. Second, a main feature of Highly Connected Deletion is that the cluster definition is easy to interpret. This makes it easy to modify the produced clustering as shown in Section 5.3. There, the presented post-processing is just a first step, more sophisticated approaches are conceivable and should be explored to further increase clustering quality. Finally, it seems useful to consider edge-weighted Highly Connected Deletion, that is, to maximize the sum of edge weights in the clustering. This could be useful to model different degrees of reliability in the data [11]. Our ILP can be adapted to solve this problem as well.

Acknowledgments. We are indebted to Nadja Betzler and Johannes Uhlmann for their early contributions in the theoretical part of this research.

References

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

[2] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen, S. Perron, and L. Liberti. Column generation algorithms for exact modularity maximization in networks. *Physical Review E*, 82:046112(046112), 2010. 3, 15, 16

[3] N. Atias and R. Sharan. Comparative analysis of protein networks: hard problems, practical solutions. *Communications of the ACM*, 55(5):88–97, 2012. 3

[4] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56 (1–3):89–113, 2004. 14

[5] T. Z. Berardini, S. Mundodi, R. Reiser, E. Huala, M. Garcia-Hernandez, et al. Functional annotation of the Arabidopsis genome using controlled vocabularies. *Plant Physiology*, 135(2):1–11, 2004. 17

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

[7] E. I. Boyle, S. Weng, J. Gollub, H. Jin, D. Botstein, J. M. Cherry, and G. Sherlock. GO::TermFinder–open source software for accessing gene ontology information and finding significantly enriched gene ontology terms associated with a list of genes. *Bioinformatics*, 20(18):3710–3715, 2004. 17

[8] S. Brohée and J. van Helden. Evaluation of clustering algorithms for protein-protein interaction networks. *BMC Bioinformatics*, 7(1):488, 2006. 20

[9] S. Böcker, S. Briesemeister, and G. W. Klau. Exact algorithms for cluster editing: Evaluation and experiments. *Algorithmica*, 60(2):316–334, 2011. 14

[10] G.-R. Cai and Y.-G. Sun. The minimum augmentation of any graph to a k-edge-connected graph. *Networks*, 19(1):151–172, 1989. 7

[11] W.-C. Chang, S. Vakati, R. Krause, and O. Eulenstein. Exploring biological interaction networks with tailored weighted quasi-bicliques. *BMC Bioinformatics*, 13(S-10):S16, 2012. 12, 23

[12] G. Chartrand. A graph-theoretic approach to a communications problem. *SIAM Journal on Applied Mathematics*, 14(4):778–781, 1966. 1, 15

[13] C. Chekuri, A. V. Goldberg, D. R. Karger, M. S. Levine, and C. Stein. Experimental study of minimum cut algorithms. In *Proc. 8th SODA*, pages 324–333, 1997. 17

[14] S. van Dongen. *Graph Clustering by Flow Simulation*. PhD thesis, University of Utrecht, 2000. 20

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

[16] J. Flum and M. Grohe. *Parameterized Complexity Theory*. 2006. 4

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

[18] 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. 9

[19] M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Mathematical Programming*, 45(1–3):59–96, 1989. 14, 15

[20] J. Guo and R. Niedermeier. Invitation to data reduction and problem kernelization. *ACM SIGACT News*, 38(1):31–45, 2007. 7

[21] E. Hartuv and R. Shamir. A clustering algorithm based on graph connectivity. *Information Processing Letters*, 76(4–6):175–181, 2000. 1, 2, 11, 17, 22

[22] 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

[23] W. Hayes, K. Sun, and N. Pržulj. Graphlet-based measures are suitable for biological network comparison. *Bioinformatics*, 2013. To appear. 1

[24] 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, 4

[25] X. Ji and J. E. Mitchell. Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement. *Discrete Optimization*, 4(1): 87–102, 2007. 15

[26] 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

[27] V. King, S. Rao, and R. E. Tarjan. A faster deterministic maximum flow algorithm. *J. Algorithms*, 17(3):447–474, 1994. 9

[28] C. Komusiewicz and J. Uhlmann. Cluster editing with locally bounded modifications. *Discrete Applied Mathematics*, 160(15):2259–2270, 2012. 5

[29] 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

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

[31] H. Liu, P. Zhang, and D. Zhu. On editing graphs into 2-club clusters. In *Proc. FAW-AAIM '12*, volume 7285 of *LNCS*, pages 235–246. Springer, 2012. 2, 7, 10

[32] D. Lokshtanov, D. Marx, and S. Saurabh. Lower bounds based on the Exponential Time Hypothesis. *Bulletin of the EATCS*, 84:41–71, 2011. 4

[33] 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

[34] A. Mehrotra and M. A. Trick. Cliques and clustering: A combinatorial approach. *Operations Research Letters*, 22(1):1–12, 1998. 15

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

[36] R. Niedermeier. *Invitation to Fixed-Parameter Algorithms.* OUP, 2006. 4

[37] M. Oosten, J. H. G. C. Rutten, and F. C. R. Spieksma. The clique partitioning problem: Facets and patching facets. *Networks*, 38(4):209–226, 2001. 14

[38] 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

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

[40] P. Ronhovde and Z. Nussinov. Local resolution-limit-free Potts model for community detection. *Physical Review E*, 81(4):046114, 2010. 23

[41] S. Régnier. Sur quelques aspects mathématiques des problèmes de classification automatique. *I.C.C. Bulletin*, 4:175–191, 1965. 14

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

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

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

[45] C. Stark, B.-J. Breitkreutz, A. Chatr-aryamontri, L. Boucher, R. Oughtred, et al. The BioGRID interaction database: 2011 update. *Nucleic Acids Research*, 39 (Database-Issue):698–704, 2011. 17

[46] J. M. M. van Rooij, M. E. van Kooten Niekerk, and H. L. Bodlaender. Partition into triangles on bounded degree graphs. *Theory of Computing Systems*, 2013. To appear. 4, 6

[47] J. Z. Wang, Z. Du, R. Payattakool, P. S. Yu, and C.-F. Chen. A new method to measure the semantic similarity of GO terms. *Bioinformatics*, 23(10):1274–1281, 2007. 21