Partitioning Biological Networks into Highly Connected Clusters with Maximum Edge Coverage

Falk Hüffner, Christian Komusiewicz, Adrian Liebtrau, and Rolf Niedermeier

Abstract—A popular clustering algorithm for biological networks which was proposed by Hartuv and Shamir [IPL 2000] identifies nonoverlapping highly connected components. We extend the approach taken by this algorithm by introducing 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 75 % of the edges that are deleted for an optimal solution; the column generation method can then optimally solve protein interaction networks with up to 6 000 vertices and 13 500 edges within five hours. Additionally, we present a new heuristic that finds more clusters than the method by Hartuv and Shamir.

Index Terms—Cluster analysis, PPI networks, fixed-parameter tractability, data reduction, integer linear programming, heuristics

1 INTRODUCTION

Network clustering is a computational tool to analyze biological systems by identifying functional subgroups within large biological networks generated for example from protein interaction data. Herein, a key idea is to identify densely connected subgraphs (clusters) that have many interactions within themselves and few with the rest of the graph [1–4]. Hartuv and Shamir [5] proposed a clustering algorithm producing so-called *highly connected* clusters. Their method has been successfully used to cluster cDNA fingerprints [6], to find complexes in protein–protein interaction (PPI) data [7, 8], to group protein sequences hierarchically into superfamily and family clusters [9], and to find families of regulatory RNA structures [10]. Hartuv and Shamir [5] 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 has degree greater than $n/2$ [11]. Thus, highly connected graphs are very similar to *0.5-quasi-complete graphs* [12, 13], that is, graphs 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 [5].

The algorithm by Hartuv and Shamir [5] partitions the vertex set of the given graph such that each partition set is highly connected, thus guaranteeing good intra-cluster density (including maximum cluster diameter two and the presence of more than half of all possible edges). Moreover, the algorithm needs no prespecified parameters (such as the number of clusters) and it naturally extends to hierarchical clustering. Essentially, Hartuv and Shamir's algorithm iteratively deletes the edges of a minimum cut in a connected component that is not yet highly connected.[1]

While Hartuv and Shamir's algorithm guarantees to output a partitioning into *highly connected* subgraphs, it iteratively uses a greedy step to find small edge sets to delete. Thus it does not guarantee to maximize the overall number of edges within clusters or, equivalently, to minimize 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 we propose a formally defined *combinatorial*

A preliminary version appeared in the proceedings the 9th International Bioinformatics Research and Applications Symposium (ISBRA 2013) held at Charlotte, NC, USA (volume 7875 in Lecture Notes in Computer Science, pages 99–111, Springer, 2013). The full version contains all missing proofs as well as experiments based on an extended data set. We furthermore propose and evaluate a new heuristic for HIGHLY CONNECTED DELETION.

- *F. Hüffner, C. Komusiewicz, and R. Niedermeier are with the Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany. E-mail: {falk.hueffner,christian.komusiewicz,rolf.niedermeier}@tu-berlin.de*
- *A. Liebtrau was with the Institut für Informatik, Friedrich-Schiller-Universität Jena, Germany.*

Supported by DFG projects PABI (NI 369/7-2) and ALEPH (HU 2139/1-1) and a post-doc fellowship of the region Pays de la Loire.
Manuscript received XX Sept. 201X; revised XX Sept. 201X; accepted X Sept. 201X; published online XX Sept.. 201X.
For information on obtaining reprints of this article, please send e-mail to: tcbb@computer.org, and reference IEEECS Log Number TCBB-201X-XX-XXXX. Digital Object Identifier no. 10.1109/TCBB.XX.X.

1. The CLICK algorithm [14] and the SIDES algorithm [15] follow the same scheme, but use edge weights and different stopping criteria, based on probabilistic models.

optimization problem that specifies the goal to minimize the number of edge deletions which is addressed implicitly by Hartuv and Shamir's algorithm.

> 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 [16], 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 [17], 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 [5], 2-CLUB DELETION poses a weaker demand on intra-cluster connectivity and density. Further related problems are those which allow a different set of modifications. If we allow insertion of edges instead of deletion, the problem reduces to increasing the connectivity of a graph to more than $n/2$ by edge insertions. This problem can be solved in polynomial time [18]. The vertex deletion version of HIGHLY-CONNECTED DELETION is NP-hard by a simple adaption of the NP-hardness proof of 2-CLUB CLUSTER VERTEX DELETION due to Liu et al. [17]: given a VERTEX COVER instance, create an equivalent 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.

It could be expected that the algorithm by Hartuv and Shamir [5] yields a good approximation for the optimization goal of HIGHLY CONNECTED DELETION. However, we can observe that in the worst case, its result has 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 one-vertex clusters, which Hartuv and Shamir counteract with postprocessing [5]. This tendency might introduce systematic bias [15]. Hence, exact algorithms for solving HIGHLY CONNECTED DELETION are desirable.

1.1 Our contributions

We analyze the (parameterized) computational complexity of HIGHLY CONNECTED DELETION and propose several exact solution methods.[2] In particular, we show that HIGHLY CONNECTED DELETION is NP-hard even on 4-regular graphs and, provided the Exponential Time Hypothesis (ETH) [20] is true, cannot be solved in subexponential time. While biological networks are unlikely to be 4-regular, this result directly implies hardness for graphs with bounded degeneracy.[3] Biological networks often have low degeneracy, but the NP-hardness means that this fact cannot be directly exploited to obtain efficient algorithms for HIGHLY CONNECTED DELETION. In addition, we provide 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 an additional polynomial-time executable data reduction rule for HIGHLY CONNECTED DELETION preserving the possibility to solve the problem exactly. Further, we develop an ILP formulation (using column generation) and show how, combined with the presented data reduction rules, real-world instances with, e. g., 6 000 vertices and 13 500 edges can be solved within five 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 [21]. We believe that such advances should also be made for clustering problems and present a first step for one concrete problem.[4] Finally, we present a simple heuristic that can be employed when the instances are too large for the exact approach.

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, to which we will refer as min-cut method, and Markov clustering (MCL). While our exact approach is clearly the slowest when compared with the other two, we demonstrate that it is a viable alternative in terms of number and quality of the reported clusters.

More specifically, first we observe that our data re-

2. 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 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. [19] for a deeper elaboration.

3. A graph has degeneracy d if every subgraph has at least one vertex of degree at most d.

4. A related NP-hard clustering problem for which significant progress has been made towards efficient exact algorithms is CLUSTER EDITING [22].

duction rules typically identify more than 75 % of the edges to be deleted for an optimal solution for HIGHLY CONNECTED DELETION. Combining our data reduction rules with the min-cut method significantly improves this method'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 than the min-cut 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. Our newly proposed heuristic can also beat the min-cut method for the number of clusters found, which we illustrate in particular for a dense network.

1.2 Preliminaries

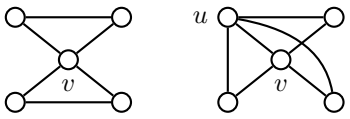
We consider only undirected and simple graphs $G = (V, E)$. We use n and m to denote the number of vertices and edges in the input graph, respectively, and k for the minimum size of an edge set whose deletion makes all components highly connected. The *order* of a graph G is the number of vertices in 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 as a parameterized problem [23–25], 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 [26]. 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) [20] 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. [27] 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) [20] is false. Our reduction is from the PARTITION INTO TRIANGLES problem.



Type I Type II

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

As shown by van Rooij et al. [28], 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 [28]; we refer to such graphs as *4-regular neighborhood-restricted graphs*. The variant of PARTITION INTO TRIANGLES that we use in the reduction thus is:

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

Our reduction is similar to a simple reduction that was used to show NP-hardness of CLUSTER DELETION on graphs of maximum degree four [29]. 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 $N[u] = N[v]$. 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 a 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 [28]. 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 $|V| = k = |E|/2$. □

3 PARAMETERIZED COMPLEXITY

In this section, we provide polynomial-time data reduction rules that 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. The resulting instance is called a *problem kernel* with respect to the parameter k and the data reduction algorithm is called *kernelization* [26]. The size of the resulting instance is used to measure the effectiveness of the data reduction rules. Further, we present a fixed-parameter algorithm for HIGHLY CONNECTED DELETION with running time $O(3^{4k} \cdot k^2 + n^2 mk \cdot \log n)$. 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 let u, v be two vertices in G. If u and v are adjacent, 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 $n_{uv} + n_u + c > n/2$ and $n_{uv} + n_v + c > n/2$, 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 shows 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 $D(S) := \{\{u, v\} \in E \mid u \in S \text{ and } v \in V \setminus S\}$ 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 there is no solution of size at most k. Otherwise, remove S from G and set $k := k - |D(S)|$.

Lemma 4: Rule 4 preserves optimal solvability 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 [30, 31]. This tree has n vertices and the set of weighted edges represents all pairwise min-cuts. A maximal inseparable vertex sets can be found by deleting all edges that have weight at most k. Once this set has been identified, the application of the reduction rule can be performed in $O(m)$ time. Since the reduction rule can be performed at most k times (it answers either "no" or reduces k), the overall running time follows. □

Note that a highly connected graph of order at least $2k$ forms 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 preserves optimal solvability 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 $\bigcup_{1 \leq i \leq q}(C_i \setminus S)$ 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

$$\begin{aligned} &|S|/2 - 0.3\sqrt{|S|} - \binom{\lfloor 0.6\sqrt{|S|}\rfloor}{2} \\ &> |S|/2 - 0.6|S|/2 - 0.36|S|/2 > 0. \end{aligned} \tag{1}$$

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 $O(m)$ 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

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

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 [32], and the best known fixed-parameter algorithm for 2-CLUB DELETION runs in $2.74^k \cdot n^{O(1)}$ time [17].

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 [5], 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 within 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 time becomes single-exponential 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 $O(3^{4k} \cdot k^2 + n^2 mk \cdot \log n)$ 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. Moreover, the algorithm 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 of purely theoretical nature. Hence, we now 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, formulate the problem as an *integer linear programming* (ILP) and apply it to the new, smaller instance. [5] The resulting ILP can be solved using sophisticated ILP solvers which are very efficient.

5. ILP formulations have proved useful for attacking other hard graph problems involving dense subgraphs such as computation of edge-weighted quasi-bicliques [33] or CLUSTER EDITING [22].

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, as shown by Theorem 2, it is useful to reduce vertices of small degree, as found in protein interaction networks. However, Rules 4 and 5 that produce a kernel have the downside of requiring relatively large substructures unlikely to exist in our inputs. In contrast, we noted that Rule 3 leaves behind many degree-2 and -3 vertices. Thus, we additional devised the following rule.

We try to identify triangles uvw that form a highly connected cluster in at least one optimal solution. 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:

- There is no vertex connected to all three of u, v, and w;
- the set $N_{uv} \cup N_{uw} \cup N_{vw}$ does not contain an edge;
- for any $\{x,y,z\} = \{u,v,w\}$, the value of $\{x,y\}$ plus the value of z is at most three;
- the sum of the values of u, v, and w is at most three.

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

Proof (preservation of optimality): 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. □

4.2 Integer Linear Programming: Column Generation

We now consider integer linear programming (ILP) based approaches. With these, we can utilize the decades of engineering that went into commercial solvers like CPLEX or Gurobi to be able to tackle large instances. Our main approach is somewhat involved due to the use of column generation. We additionally tried a more straightforward approach based on a CLIQUE PARTITIONING formulation and row generation [22, 34]. However, it performed poorly compared to the column generation, and we omit a detailed comparison.

We describe an ILP formulation of HIGHLY CONNECTED DELETION, which in its basic scheme is similar to those of Mehrotra and Trick [35] and Ji and Mitchell [36] for constrained CLIQUE PARTITIONING and that of Aloise et al. [19] 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

$$\text{maximize} \sum_{T \in \mathcal{T}} c_T z_T, \tag{3}$$

$$\text{s.t.} \sum_{\{T \in \mathcal{T} | u \in T\}} z_T = 1 \quad \forall u \in V, \tag{4}$$

$$z_T \in \{0,1\} \quad \forall T \in \mathcal{T}, \tag{5}$$

where c_T is the number of edges in the subgraph induced by T. The objective (3) maximizes the number of edges within clusters, which is equivalent to minimizing the number of inter-cluster edges (deletions). The constraints of type (4) 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 instances, typically only a small subset of possible variables needs to be added.

More precisely, we try to add variables that improve the relaxed model where constraint (5) is replaced by $z_T \geq 0$. We thus first compute an optimal solution to the relaxed model. If we can find an improving column, then we add this column and compute a new solution for the relaxed model. If we cannot find an improving column, then there is no cluster that can improve the current fractional solution. If this solution is "by chance" integral, then we have found an optimal solution also for the model with integrality constraints. Otherwise, as suggested by Mehrotra and Trick [35], we use Ryan–Foster branching: we find two vertices such that there is both a fractional cluster containing both vertices and a fractional cluster containing exactly one of them, and branch into the two cases that the two vertices are together in a cluster or that they are in a different cluster.

The improvement of adding a column for cluster T in the relaxed model 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 (4) in the relaxation (see e. g. Aloise et al. [19] 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 subgraph 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:

$$\text{maximize} \sum_{\{u,v\} \in E} e_{uv} - \sum_{u \in t} \lambda_u v_u, \tag{6}$$

$$\text{s.t. } d = \sum_{u \in V} v_u, \tag{7}$$

$$e_{uv} \leq v_u, e_{vu} \leq v_v \quad \forall \{u,v\} \in E, \tag{8}$$

$$\text{if } v_u \text{ then } \sum_{v \in N(u)} e_{uv} > d/2 \quad \forall u \in V, \tag{9}$$

where the constraint (9) 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.

To get a speedup, 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 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. If this fails, we try local search: removing and adding up to two vertices, respectively, to a known cluster. Further, we abort solving the column generation ILP as soon as an improving solution is found. Also, to find a good set of disjoint clusters more quickly, we initially scale the dual variables by a factor of 5 until no improving clusters can be found with this factor.

4.3 Neighborhood Heuristic

One drawback of the method by Hartuv and Shamir [5] is that it uses a minimum cut routine, which has a fairly high worst-case time complexity and is difficult to implement. We suggest an alternative simpler heuristic. Recall that Rule 3 from Section 3.1 says that an edge whose two endpoints have no common neighbor can be deleted. The idea is then to greedily delete in a connected component that is not yet highly connected the edge for which the endpoints have the fewest common neighbors. We additionally weigh this by the vertex degree, that is, we delete the edge $\{u,v\}$ with $x := |N(u) \cap N(v)|$ that minimizes

$$\min\{x/\deg(u), x/\deg(v)\}. \tag{10}$$

5 Experimental Evaluation

We implemented the data reduction in OCaml and the ILPs in C++ using the CPLEX 12.5.1 ILP solver. For the minimum cut subroutine of the min-cut method, a highly optimized implementation in C was used [37]. Our source code and sample instances are available at http://fpt.akt.tu-berlin.de/hcd/. The test machine is a 4-core 3.6 GHz Intel Xeon E5-1620 (Sandy Bridge-E) with 10 MB L3 cache and 64 GB main memory, running under Debian GNU/Linux 7.0. CPLEX is allowed to use up to 8 threads; we report wall-clock times.

We used protein interaction networks available at the BioGRID repository [38] (release 3.2.101 from May 25th, 2013). The species for which we illustrate our results are *S. pombe*, *C. elegans*, *M. musculus*, and *A. thaliana*. For

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, Δk [%]:= $\Delta k/k$ is the relative 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''
SP phys.	1963	4772	33	1875	4709	2398	62.9	1249	611	2173
SP all	3735	51620	9	3716	51608	6446	≥13.0	959	2765	45156
CE phys.	3176	5465	101	2926	5314	4593	88.6	2844	268	747
CE all	3866	7707	73	3686	7599	5665	77.7	3292	542	1991
MM phys.	7354	14509	142	6983	14274	10693	79.6	6107	1115	3604
MM all	7414	14687	146	7037	14449	10757	79.1	6133	1159	3733
AT phys.	5999	13571	118	5727	13407	9388	79.0	4730	1009	3658
AT all	6038	13680	124	5739	13490	9396	78.5	4744	1042	3771

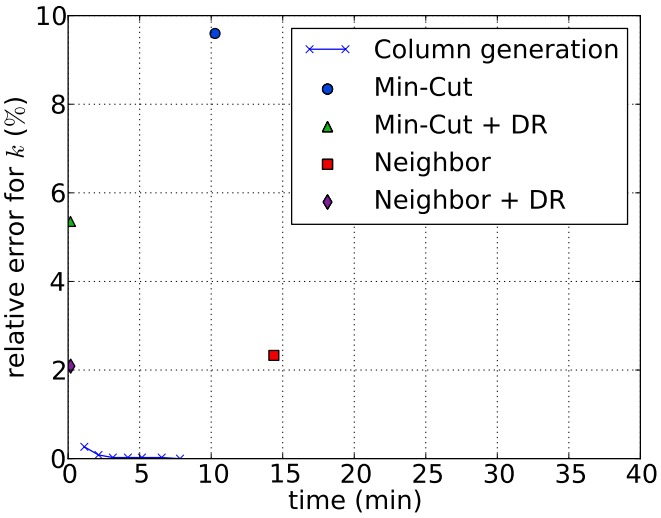


Fig. 2. Running time trade-off for the *A. thaliana* network

each species, we extracted one network with physical interactions only, and one with all interactions. Table 1 shows some basic properties of these networks.

5.1 Data Reduction and Running Time

Table 1 shows the effect of data reduction. Knowing the optimal k (see Table 2) allows us to state that typically 75 % of the edges that need to be deleted are identified. Since connected components can be treated separately, the most important time factor is the size of the largest connected component (m''). Here, the number of edges is reduced to typically 30 %. This demonstrates the effectivity of the data reduction, which preserves exact solvability, and suggests it should be applied regardless of the actual solution method that follows.

Table 2 summarizes 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 neighborhood method finds many more (mostly smaller) clusters than the min-cut method, and overall deletes less edges. However, the largest clusters it produces tend to be smaller. The column generation method is able to solve all but one test instance, although the hardest one takes almost 5 hours. It is not able to solve the network of all interactions of *S. pombe* within 32 hours; this is probably because this is a denser network, making data reduction less effective. Ryan–Foster branching is necessary for all instances, but the search tree is typically small, the largest having 37 nodes.

In Figure 2, we examine the trade-off between the solution size k and the running time for the min-cut method, the neighborhood method, and the column generation with a per-connected-component time limit. We see that data reduction speeds up the first two methods a lot, and also improves the solution quality, in particular for the min-cut heuristic. The column generation gets an almost optimal result very quickly, and finds the optimal solution after about 8 minutes (however, proving optimality takes several hours). Data reduction for column generation consistently improves running time, for example for CE-p by a factor of 24.

5.2 Biological evaluation

For the biological evaluation, we studied the *A. thaliana* network with all interactions in more detail. For the computation of the enrichment of annotation terms, we used the GOstats package [39] of Bioconductor with *A. thaliana* annotation data from the TAIR database [40]. The computed p-values are corrected for multiple hypothesis testing. We used a significance threshold of $p \leq 0.01$. Our findings are summarized in Figure 3. 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 large 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. 3, for all 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 neighborhood heuristic and the exact algorithm the results are less clear: they produce 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. A possible explanation could be as follows. By minimizing the number of deletions, some of the small reported clusters, for instance triangles,

TABLE 2
Results for the instances of Table 1. Here, k is the number of edges deleted, u is the number of unclustered vertices, K is the number of clusters, 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						neighborhood with DR						column generation with DR					
	k	u	K	n	m	t	k	u	K	n	m	t	k	u	K	n	m	t	k	u	K	n	m	t
SP-p	4324	1839	17	17	96	16	4165	1699	62	17	96	2	3961	1548	101	15	71	2	3811	1495	108	17	96	102
SP-a	50343	3663	4	63	1268	526	50331	3651	8	63	1268	214	49514	3263	95	60	1175	3491	—	—	—	—	—	—
CE-p	5437	3159	4	7	16	56	5268	3040	37	9	30	1	5215	2984	56	9	30	1	5184	2960	62	9	30	34
CE-a	7613	3849	1	17	94	93	7491	3758	27	17	94	3	7382	3664	55	15	78	4	7295	3625	63	19	113	149
MM-p	14413	7316	9	13	69	1198	14078	7072	80	13	50	15	13636	6718	185	13	69	15	13428	6600	209	13	67	2190
MM-a	14591	7376	9	13	69	1253	14265	7141	77	13	50	15	13791	6762	189	13	69	16	13591	6655	209	13	67	2458
AT-p	13009	5843	25	23	190	602	12497	5478	131	23	190	10	12119	5249	191	22	178	10	11885	5142	209	23	190	16721
AT-a	13121	5885	24	23	190	616	12613	5523	129	23	190	10	12222	5291	189	22	178	10	11972	5170	211	23	190	10536

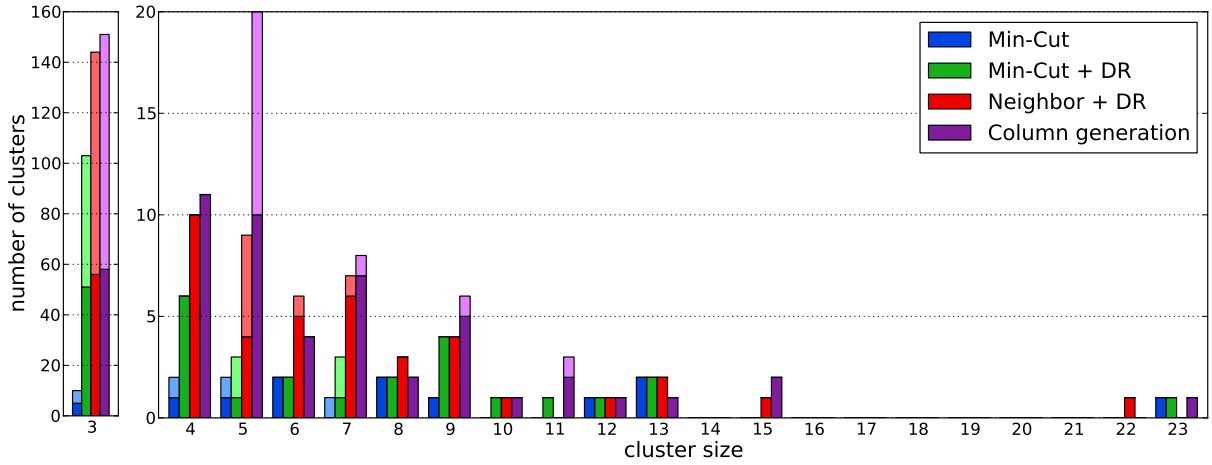


Fig. 3. Clusters in the *A. thaliana* network with all interactions. The darker part of each bar shows the fraction of clusters with significant enrichment of biological process annotation terms.

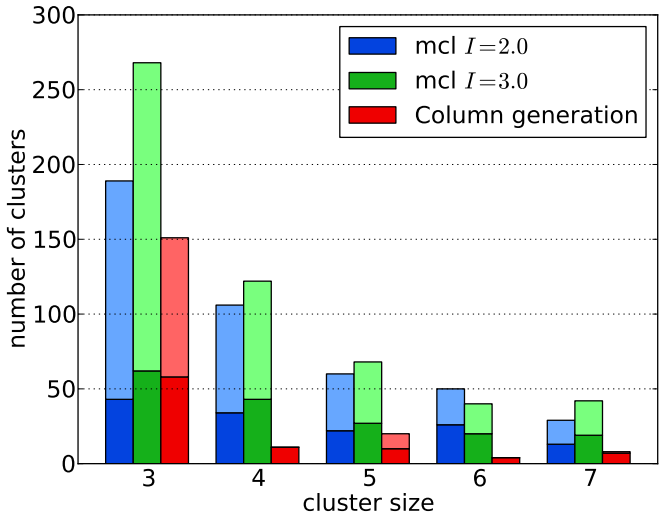
contain some nodes of very low degree and one high degree node that is "by chance" not included in any large cluster. The resulting cluster is unlikely to have similar GO annotations across all three proteins. We discuss a possibility to counter this behavior in the conclusion.

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 [41]. For details concerning MCL refer to [42]; in the experiments, we used the MCL implementation available at http://micans.org/mcl/ (version 12-135). 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 clusters, and MCL produces more clusters than HCD. MCL also produces larger clusters than HCD. For instance, it finds 38 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. Hence, MCL places many more vertices in enriched clusters than the other methods, but the average cluster quality is much lower (see Table 3).

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



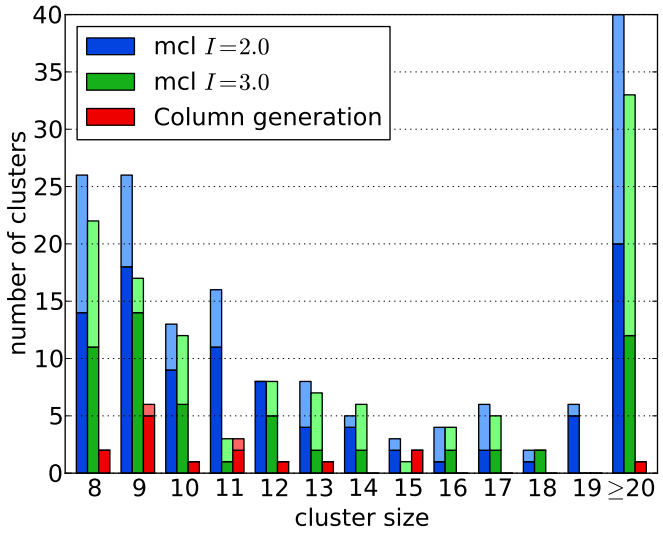
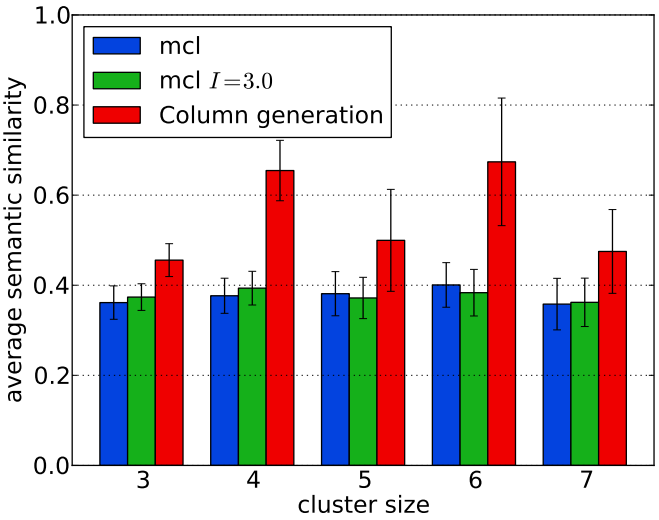


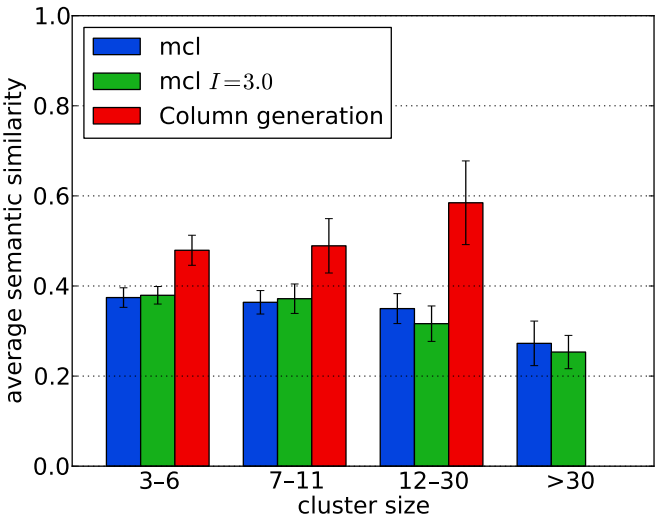
Fig. 4. Clusters produced by the MCL algorithm and column generation for *A. thaliana*. The darker part of each bar shows the fraction of clusters with significant enrichment of biological process annotation terms.

	#clust.	clust. enriched [%]	vert. enriched [%]
Min-Cut	24	66.7	2.0
Min-Cut + DR	129	56.6	5.5
Neighbor-Heur.	189	49.7	7.4
Column gen.	211	49.8	8.5
MCL $I = 2$	1045	29.9	48.1
MCL $I = 3$	1514	22.4	31.6

TABLE 3
Number and percentage of clusters with gene annotation enrichment in the *A. thaliana* network



(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 the 95 % confidence interval.

Semantic similarity

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. [43]. 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 0.538 for HCD, 0.258 for MCL with $I = 2.0$, and 0.261 for MCL with $I = 3.0$. This purely 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 except for size-five and size-seven clusters, where the difference is less pronounced. 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, significantly 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.

Biological interpretation of an example cluster

We further examined a cluster of size 15 that was found by the column generation algorithm and the neighborhood heuristic in the *A. thaliana* network with all interactions. The corresponding proteins are AT2G04660, AT2G18290, AT2G20000, AT2G39090, AT2G42260, AT3G48150, AT3G57860, AT4G11920, AT4G21530, AT4G22910, AT4G33270, AT5G05560, AT5G13840, AT1G06590, and AT1G78770. The annotation term with the lowest p-value for this cluster is 'regulation of DNA endoreduplication' shared by 12 out of the 15 proteins. The three proteins that are not annotated with this term are AT2G42260, AT4G21530, and AT4G33270. Protein AT2G42260 is annotated by 'DNA endoreduplication' and has unknown function, but mutants undergo further rounds of endoreduplications, so a role in *regulation* of DNA endoreduplication is not unlikely. Protein AT4G21530 is part of the anaphase promoting complex (APC) which plays a crucial role in cell cycle regulation. Protein AT4G33270 is a signal transducing protein interacting with subunits of the APC. So far, it is only annotated by 'signal transduction'; a more specific role in regulation of DNA endoreduplication is suggested by its many interactions with the other proteins of the cluster.

Interestingly, this cluster is completely destroyed by the min-cut method (with or without data reduction), that is, *all* proteins are unclustered when using these approaches. The MCL algorithm finds other clusters that overlap with this cluster. The cluster with the largest overlap has an enrichment of annotation terms; the term with the lowest p-value is again 'regulation of DNA endoreduplication', but here only 9 of 19 proteins have this annotation.

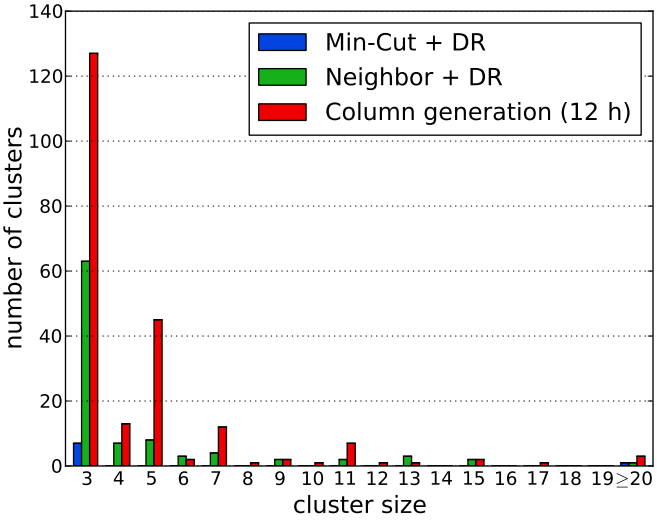


Fig. 6. Heuristic methods for the *S. pombe* network

5.3 Heuristics

We examine the heuristic methods for HIGHLY CONNECTED DELETION for the *S. pombe* network with all interactions, which our exact approach cannot solve (see Fig. 6). The min-cut method finds a dense core with 63 vertices and 1268 edges. The neighborhood heuristic finds almost the same cluster, omitting 5 vertices and adding 2 others. However, the min-cut method finds additionally only 7 triangles, whereas the neighborhood heuristic finds an additional 94 clusters, of which 31 are not triangles, the largest being two clusters of size 15 each. The column generation method with time limit finds even more clusters, but only after several hours (note that the heuristic column generation is not well-optimized for such large graphs, and this time could easily be improved). In summary, the neighborhood heuristic and column generation with time limit find many more potentially interesting clusters where the min-cut algorithm fails.

6 CONCLUSION

Proposing the NP-hard problem HIGHLY CONNECTED DELETION, we introduced a new view on partitioning into highly connected components with a clearly defined and natural optimization goal. We developed effective 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. We believe that our simple formal model of partitioning into highly connected clusters

provides a viable alternative to existing approaches. Compared to the min-cut method with data reduction, we find more clusters of slightly inferior quality; compared with Markov clustering, we find fewer clusters, but these have higher quality.

For future work, it seems interesting to perform comparisons with further clustering algorithms, for example the RN algorithm [44]. One drawback of the HIGHLY CONNECTED DELETION clustering definition is that many vertices remain unclustered. This could be counteracted with postprocessing as suggested by Hartuv and Shamir [5]. Another drawback is that the biological quality of the small clusters for the exact solutions is worse than for the min-cut method. Thus, two possible extensions for improving the overall cluster quality could be as follows: First, one could demand that small clusters contain only vertices of low degree. Second, one could demand for instance for size-five clusters that they have to be cliques, thus putting a stronger connectivity demand on these small clusters. 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 [33]. Our ILP can be adapted to solve this problem as well.

ACKNOWLEDGMENT

We are indebted to Nadja Betzler and Johannes Uhlmann for their early contributions in the theoretical part of this research. We also thank Andrea Kappes (Karlsruhe Institute of Technology) for pointing out a flaw in a previous version of the column generation algorithm.

REFERENCES

[1] R. Albert, "Scale-free networks in cell biology," *Journal of Cell Science*, vol. 118, no. 21, pp. 4947–4957, 2007.

[2] R. Sharan, I. Ulitsky, and R. Shamir, "Network-based prediction of protein function," *Molecular Systems Biology*, vol. 3, p. 88, 2007.

[3] V. Spirin and L. A. Mirny, "Protein complexes and functional modules in molecular networks," *PNAS*, vol. 100, no. 21, pp. 12 123–12 128, 2003.

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

[5] E. Hartuv and R. Shamir, "A clustering algorithm based on graph connectivity," *Information Processing Letters*, vol. 76, no. 4–6, pp. 175–181, 2000.

[6] E. Hartuv, A. O. Schmitt, J. Lange, S. Meier-Ewert, H. Lehrach, and R. Shamir, "An algorithm for clustering cDNA fingerprints," *Genomics*, vol. 66, no. 3, pp. 249–256, 2000.

[7] N. Pržulj, D. A. Wigle, and I. Jurisica, "Functional topology in a network of protein interactions," *Bioinformatics*, vol. 20, no. 3, pp. 340–348, 2004.

[8] W. Hayes, K. Sun, and N. Pržulj, "Graphlet-based measures are suitable for biological network comparison," *Bioinformatics*, vol. 29, no. 4, pp. 483–491, 2013.

[9] A. Krause, J. Stoye, and M. Vingron, "Large scale hierarchical clustering of protein sequences," *BMC Bioinformatics*, vol. 6, p. 15, 2005.

[10] 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*, vol. 21, no. 11, pp. 1929–1943, 2011.

[11] G. Chartrand, "A graph-theoretic approach to a communications problem," *SIAM Journal on Applied Mathematics*, vol. 14, no. 4, pp. 778–781, 1966.

[12] H. Matsuda, T. Ishihara, and A. Hashimoto, "Classifying molecular sequences using a linkage graph with their pairwise similarities," *Theoretical Computer Science*, vol. 210, no. 2, pp. 305–325, 1999.

[13] D. Jiang and J. Pei, "Mining frequent cross-graph quasi-cliques," *ACM Transactions on Knowledge Discovery from Data*, vol. 2, no. 4, pp. 16:1–16:42, 2009.

[14] I. Gat-Viks, R. Sharan, and R. Shamir, "Scoring clustering solutions by their biological relevance," *Bioinformatics*, vol. 19, no. 18, pp. 2381–2389, 2003.

[15] M. Koyutürk, W. Szpankowski, and A. Grama, "Assessing significance of connectivity and conservation in protein interaction networks," *Journal of Computational Biology*, vol. 14, no. 6, pp. 747–764, 2007.

[16] R. Shamir, R. Sharan, and D. Tsur, "Cluster graph modification problems," *Discrete Applied Mathematics*, vol. 144, no. 1–2, pp. 173–182, 2004.

[17] H. Liu, P. Zhang, and D. Zhu, "On editing graphs into 2-club clusters," in *Proc. FAW-AAIM '12*, ser. LNCS, vol. 7285. Springer, 2012, pp. 235–246.

[18] G.-R. Cai and Y.-G. Sun, "The minimum augmentation of any graph to a k-edge-connected graph," *Networks*, vol. 19, no. 1, pp. 151–172, 1989.

[19] 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*, vol. 82:046112, no. 046112, 2010.

[20] R. Impagliazzo, R. Paturi, and F. Zane, "Which problems have strongly exponential complexity?" *Journal of Computer and System Sciences*, vol. 63, no. 4, pp. 512–530, 2001.

[21] N. Atias and R. Sharan, "Comparative analysis of protein networks: hard problems, practical solutions," *Communications of the ACM*, vol. 55, no. 5, pp. 88–97, 2012.

[22] S. Böcker, S. Briesemeister, and G. W. Klau, "Exact algorithms for cluster editing: Evaluation and experiments," *Algorithmica*, vol. 60, no. 2, pp. 316–334, 2011.

[23] R. G. Downey and M. R. Fellows, *Parameterized Complexity*, 1999.

[24] J. Flum and M. Grohe, *Parameterized Complexity*

Theory, 2006.
[25] R. Niedermeier, *Invitation to Fixed-Parameter Algorithms*. Oxford University Press, 2006.
[26] J. Guo and R. Niedermeier, "Invitation to data reduction and problem kernelization," *ACM SIGACT News*, vol. 38, no. 1, pp. 31–45, 2007.
[27] D. Lokshtanov, D. Marx, and S. Saurabh, "Lower bounds based on the Exponential Time Hypothesis," *Bulletin of the EATCS*, vol. 105, pp. 41–71, 2011.
[28] 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*, vol. 52, no. 4, pp. 687–718, 2013.
[29] C. Komusiewicz and J. Uhlmann, "Cluster editing with locally bounded modifications," *Discrete Applied Mathematics*, vol. 160, no. 15, pp. 2259–2270, 2012.
[30] R. E. Gomory and T. C. Hu, "Multi-Terminal Network Flows," *Journal of the Society for Industrial and Applied Mathematics*, vol. 9, no. 4, pp. 551–570, 1961.
[31] V. King, S. Rao, and R. E. Tarjan, "A faster deterministic maximum flow algorithm," *Journal of Algorithms*, vol. 17, no. 3, pp. 447–474, 1994.
[32] S. Böcker and P. Damaschke, "Even faster parameterized cluster deletion and cluster editing," *Information Processing Letters*, vol. 111, no. 14, pp. 717–721, 2011.
[33] W.-C. Chang, S. Vakati, R. Krause, and O. Eulenstein, "Exploring biological interaction networks with tailored weighted quasi-bicliques," *BMC Bioinformatics*, vol. 13, no. S-10, p. S16, 2012.
[34] M. Grötschel and Y. Wakabayashi, "A cutting plane algorithm for a clustering problem," *Mathematical Programming*, vol. 45, no. 1–3, pp. 59–96, 1989.
[35] A. Mehrotra and M. A. Trick, "Cliques and clustering: A combinatorial approach," *Operations Research Letters*, vol. 22, no. 1, pp. 1–12, 1998.
[36] X. Ji and J. E. Mitchell, "Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement," *Discrete Optimization*, vol. 4, no. 1, pp. 87–102, 2007.
[37] C. Chekuri, A. V. Goldberg, D. R. Karger, M. S. Levine, and C. Stein, "Experimental study of minimum cut algorithms," in *Proc. 8th SODA*. ACM/SIAM, 1997, pp. 324–333.
[38] C. Stark, B.-J. Breitkreutz, T. Reguly, L. Boucher, A. Breitkreutz, and M. Tyers, "BioGRID: a general repository for interaction datasets," *Nucleic Acids Research*, vol. 34, no. suppl. 1, pp. D535–D539, 2006.
[39] S. Falcon and R. Gentleman, "Using GOstats to test gene lists for go term association," *Bioinformatics*, vol. 23, no. 2, pp. 257–258, 2007.
[40] 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*, vol. 135, no. 2, pp. 1–11, 2004.
[41] S. Brohée and J. van Helden, "Evaluation of clustering algorithms for protein-protein interaction networks," *BMC Bioinformatics*, vol. 7, no. 1, p. 488, 2006.
[42] S. van Dongen, "Graph clustering by flow simulation," Ph.D. dissertation, University of Utrecht, 2000.
[43] 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*, vol. 23, no. 10, pp. 1274–1281, 2007.
[44] P. Ronhovde and Z. Nussinov, "Local resolution-limit-free Potts model for community detection," *Physical Review E*, vol. 81, no. 4, p. 046114, 2010.

Falk Hüffner studied computer science at the Eberhard-Karls-Universität Tübingen and received his PhD at Friedrich-Schiller-Universität Jena. After a postdoctoral stay at Tel Aviv University, he now has a position at TU Berlin. He is interested in the design, analysis, and experimental evaluation of algorithms for hard problems, in particular graph problems and discrete optimization problems, from various areas such as computational biology, VLSI design, and operations research.

Christian Komusiewicz studied bioinformatics at Friedrich-Schiller-Universität Jena and received his PhD from TU Berlin. Currently, he is on a postdoctoral research stay at Université de Nantes. His main research interest lies in algorithmics for hard problems in bioinformatics.

Adrian Liebtrau studied computer science and mathematics at Friedrich-Schiller-Universität Jena. He now works as an SAP consultant for IBM.

Rolf Niedermeier studied computer science at TU München and received his PhD and habilitation from Eberhard-Karls-Universität Tübingen. After chairing for six years the Theoretical Computer Science/Computational Complexity group at Friedrich-Schiller-Universität Jena, since 2010 he chairs the Algorithmics and Complexity Theory group at TU Berlin, Faculty of Electrical Engineering and Computer Science. His research interests include algorithms for NP-hard problems, finding applications in fields such as computational molecular biology, computational social choice, and graph-based data clustering.