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. Hartuv and Shamir [8] proposed a clustering algorithm producing so-called *highly connected* clusters. Their method has been successfully used to cluster cDNA fingerprints [9], to find complexes in protein–protein interaction (PPI) data [10], and to find families of regulatory RNA structures [15]. Hartuv and Shamir [8] 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 at least $\lfloor n/2 \rfloor + 1$ [5]. Thus, highly connected graphs are very similar to *0.5-quasi-complete graphs* [11], 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 [8].

The algorithm by Hartuv and Shamir [8] 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

To appear in *Proceedings of the 9th International Symposium on Bioinformatics Research and Applications (ISBRA '13)*, volume 7875 in Lecture Notes in Computer Science, Springer, 2013.

not yet highly connected. 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. Thus, 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 [17], 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 [13], 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 [8], 2-Club Deletion poses a looser demand on intra-cluster connectivity.

It could be expected that the algorithm by Hartuv and Shamir [8] 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 $u_1, \dots, u_n$ and $v_1, \dots, v_n$, 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 [8]. This tendency might introduce systematic bias [12]. Hence, exact algorithms for solving Highly Connected Deletion are desirable.

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. For the notions of fixed-parameter tractability and kernelization, see e. g. [14]. Due to the lack of space, we defer some proofs and details to the full version of this paper.[3]

[3] http://fpt.akt.tu-berlin.de/publications/hcd.pdf

2 Computational Complexity

We can prove the hardness of Highly Connected Deletion by a reduction from Partition Into Triangles on 4-*regular neighborhood-restricted graphs* [19].

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.*

Problem Kernel. We now present four data reduction rules that preserve optimal solvability and whose exhaustive application results in an instance with at most $10 \cdot k^{1.5}$ vertices. The first data reduction rule is obvious.

Rule 1. Remove all connected components from G that are highly connected.

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

Lemma 1. *Let* G *be a highly connected graph and let* u, v *be 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.*

A simple data reduction rule follows directly from Lemma 1.

Rule 2. 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 1 and 2 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.*

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 3. 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 2. *Rule 3 preserves optimal solvability and can be exhaustively applied in* $O(n^2 \cdot mk \log n)$ *time.*

Note that a highly connected graph of size at least $2k$ is an inseparable vertex set. Hence, after exhaustive application of Rule 3, every cluster has bounded size. While Rule 3 identifies clusters that are large with respect to k, Rule 4 identifies clusters that are large compared to their neighborhood.

Rule 4. 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 3. *Rule 4 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 $\{C_i \setminus S \mid 1 \leq i \leq q\}$ 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

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

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 3. ☐

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 1, 3 and 4. 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 3, 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 1 and 4. 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. $\square$

Fixed-Parameter Algorithm. We sketch a fixed-parameter algorithm for Highly Connected Deletion. Since any highly connected graph has diameter at most two, if there is a connected component with diameter three or more, we can find a shortest path $uvwx$ between two vertices u and x, and then branch into three cases according to which edge of this path gets deleted. At the leaves of this search tree, we have a graph where every connected component has diameter at most two. Using Rule 3, we can ensure that each component has at most $4k$ vertices. We can solve an arbitrary Highly Connected Deletion instance by dynamic programming in $O(3^n \cdot m)$ time; applying this to each component yields the following theorem.

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

3 Further Data Reduction and ILP formulation

The fixed-parameter tractability results for Highly Connected Deletion (Theorem 3) 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.

Further Data Reduction. As we demonstrate in the computational experiments presented in Section 4, Rule 2 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 3 and 4 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 5. 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 (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. □

The following rule reduces some low-degree vertices.

Rule 6. 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 (preservation of optimality). 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. □

Integer Linear Programming with 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. 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 that of Aloise et al. [1] 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

$$\text{maximize} \sum_{T \in \mathcal{T}} c_T z_T, \tag{2}$$

$$\text{s.t.} \sum_{\{T \in \mathcal{T} \mid u \in T\}} z_T = 1 \quad \forall u \in V, \tag{3}$$

$$z_T \in \{0, 1\} \quad \forall T \in \mathcal{T}, \tag{4}$$

where c_T is the number of edges in the subgraph induced by t. The objective (2) maximizes the number of edges within clusters, which equivalent to minimizing the number of inter-cluster edges (deletions). The constraints of type (3) 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 (3) in the continuous relaxation of the problem (2)–(4) (see e. g. Aloise et al. [1] 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:

$$\text{maximize} \sum_{\{u,v\} \in E} e_{uv} - \sum_{u \in t} \lambda_u v_u, \tag{5}$$

$$\text{s. t. } d = \sum_{u \in V} v_u, \tag{6}$$

$$e_{uv} \leq v_u, e_{vu} \leq v_v \quad \forall \{u,v\} \in E, \tag{7}$$

$$\text{if } v_u \text{ then} \sum_{v \in N(u)} e_{uv} > d/2 \quad \forall u \in V, \tag{8}$$

where the constraint (8) 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 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.

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

4 Experimental 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 [8] (called *min-cut method* below), a highly optimized implementation in C was used [6]. 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.

We used protein interaction networks available at the BIOGRID repository [18]. 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 Fig. 2, 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 [3] with *A. thaliana* annotation data from the TAIR database [2]. 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, 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

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

S. pombe with 1541 vertices and 3036 edges; this is probably because this is a denser network, making data reduction less effective.

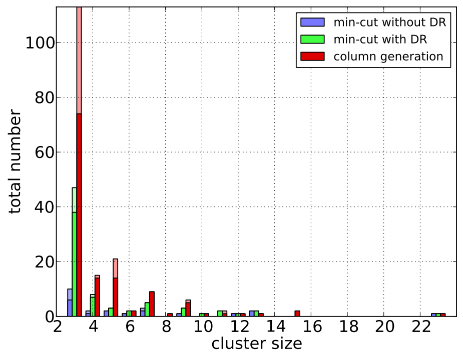
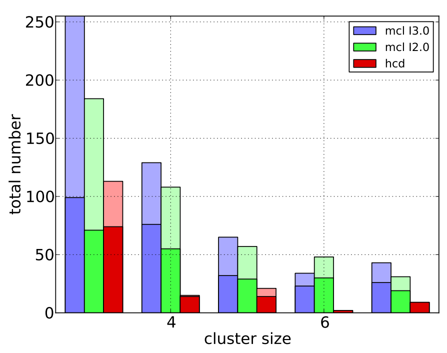


Fig. 1. Clusters in the *A. thaliana* network with all interactions. The brighter part of each bar shows the fraction of clusters without significant enrichment of biological process annotation terms.

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 1. 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. 1, 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 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.

Comparison with Markov Clustering. Next, we compare our clustering algorithm with a popular clustering algorithm for protein interaction networks. As compari-



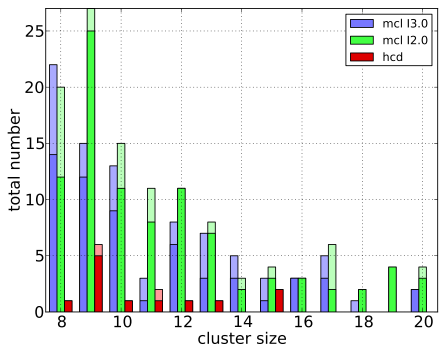


Fig. 2. Clusters in the *A. thaliana* networks produced by the MCL algorithm and our algorithm (HCD) for small clusters (left) and medium-size clusters (right).

son, we choose the so-called Markov Clustering Algorithm (MCL). For details concerning MCL refer to [7]; 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 2, 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.

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 the semantic similarity score for the GO annotations proposed by Wang et al. [20]. 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. We therefore further examined the effect of the cluster size on the average semantic score for protein pairs in the same cluster. We found that across all cluster sizes, the clusters produced by HCD show better similarity than those produced by MCL. 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.

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. First, 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. Comparisons with the min-cut algorithm show that with a time limit of one hour, this heuristic variant can find 120 additional clusters compared to the min-cut algorithm.

Another intrinsic problem of demanding highly-connected clusters is the fact that biological networks contain many low-degree vertices: these vertices cannot be contained in any highly connected cluster and HCD computes a clustering of the dense core of the network. Similar to a post-processing suggested by Hartuv and Shamir [8], we used the following simple post-processing to "readd" the proteins not included in any cluster returned by HCD: add each unclustered protein to some cluster if its interactions are exclusively with proteins of this cluster. A first examination of the enrichment statistics indicates that this version of HCD produces better clusters than MCL concerning cluster quality while clustering a significantly larger number of proteins than the pure HCD approach.

5 Outlook

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 [16]. 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 4. 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 [4]. 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] 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.

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

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

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

[9] 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.

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

[11] 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.

[12] 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.

[13] 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.

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

[15] 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.

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

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

[18] 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.

[19] 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.

[20] 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.