A Graph Modification Approach for Finding Core–Periphery Structures in Protein Interaction Networks

Sharon Bruckner[1], Falk Hüffner[2⋆], and Christian Komusiewicz[2]

[1] International Max Planck Research School for Computational Biology and Scientific Computing, Ihnestr. 63-73, 14195 Berlin, Germany
sharonb@mi.fu-berlin.de
[2] Institut für Softwaretechnik und Theoretische Informatik, TU Berlin, Germany
{falk.hueffner,christian.komusiewicz}@tu-berlin.de

Abstract. The core–periphery model for protein interaction (PPI) networks assumes that protein complexes in these networks consist of a dense core and a possibly sparse periphery that is adjacent to vertices in the core of the complex. In this work, we aim at uncovering a global core–periphery structure for a given PPI network. We propose two exact graph-theoretic formulations for this task, which aim to fit the input network to a hypothetical ground truth network by a minimum number of edge modifications. In one model each cluster has its own periphery, and in the other the periphery is shared. We first analyze both models from a theoretical point of view, showing their NP-hardness. Then, we devise efficient exact and heuristic algorithms for both models and finally perform an evaluation on subnetworks of the *S. cerevisiae* PPI network.

1 Introduction

A fundamental task in the analysis of PPI networks is the identification of protein complexes and functional modules. Herein, a basic assumption is that complexes in a PPI network are strongly connected among themselves and weakly connected to other complexes [22]. This assumption is usually too strict. To obtain a more realistic network model of protein complexes, several approaches incorporate the *core–attachment* model of protein complexes [12]: In this model, a complex is conjectured to consist of a stable core plus some attachment proteins, which only interact with the core temporally. In graph-theoretic terms, the core thus is a dense subnetwork of the PPI network. The attachment (or: periphery) is less dense, but has edges to one or more cores.

Current methods employing this type of modeling are based on *seed growing* [18, 19, 23]. Here, an initial set of promising small subgraphs is chosen as cores. Then, each core is separately greedily expanded into cores and attachments to satisfy some objective function. The aim of these approaches was to

⋆ Supported by DFG project ALEPH (HU 2139/1).

predict protein complexes [18, 23] or to reveal biological features that are correlated with topological properties of core–periphery structures in networks [19]. In this work, we use core–periphery modeling in a different context. Instead of searching for *local* core–periphery structures, we attempt to unravel a *global* core–periphery structure in PPI networks.

To this end, we hypothesize that the true network consists of several core–periphery structures. We propose two precise models to describe this. In the first model, the core–periphery structures are disjoint. In the second model, the peripheries may interact with different cores, but the cores are disjoint. Then, we fit the input data to each formal model and evaluate the results on several PPI networks.

Our approach. In spirit, our approach is related to the clique-corruption model of the CAST algorithm for gene expression data clustering [1]. In this model, the input is a similarity graph where edges between vertices indicate similarity. The hypothesis is that the objects corresponding to the vertices belong to disjoint biological groups of similar objects, the clusters. In the case of gene expression data, these are assumed to be groups of genes with the same function. Assuming perfect measurements, the similarity graph is a *cluster graph*, that is, a graph in which each connected component is a clique.

Because of stochastic measurement noise, the input graph is not a cluster graph. The task is to recover the underlying cluster graph from the input graph. Under the assumption that the errors are independent, the most likely cluster graph is one that disagrees with the input graph on a minimum number of edges. Such a graph can be found by a minimum number of edge modifications (that is, edge insertions or edge deletions). This paradigm directly leads to the optimization problem Cluster Editing [3, 4, 21].

We now apply this approach to our hypothesis that there is a global core–periphery structure in the PPI networks. In both models detailed here, we assume that all proteins of the cores interact with each other; this implies that the cores are cliques. We also assume that the proteins in the periphery interact only with the cores but not with each other. Hence, the peripheries are independent sets.

In the first model, we assume that ideally the protein interactions give rise to *vertex-disjoint* core–periphery structures, that is, there are no interactions between different cores and no interactions between cores and peripheries of other cores. Then each connected component has at most one core which is a clique and at most one periphery which is an independent set. This is precisely the definition of a split graph.

Definition 1. *A graph $G = (V, E)$ is a* split graph *if V can be partitioned into V_1 and V_2 such that $G[V_1]$ is an independent set and $G[V_2]$ is a clique.*

The vertices in V_1 are called *periphery vertices* and the vertices in V_2 are called *core vertices*. Note that the partition for a split graph is not always unique. Split graphs have been previously used to model core–periphery structures in social networks [5]. There, however, the assumption is that the network contains exactly one core–periphery structure. We assume that each connected component is a

split graph; we call graphs with this property *split cluster graphs*. Our fitting model is described by the following optimization problem.

> Split Cluster Editing
> Input: An undirected graph $G = (V, E)$.
> Task: Transform G into a split cluster graph by applying a minimum number of edge modifications.

In our second model, we allow the vertices in the periphery to be attached to an arbitrary number of cores, thereby connecting the cores. In this model, we thus assume that the cores are disjoint cliques and the vertices of the periphery are an independent set. Such graphs are called *monopolar*.

Definition 2. *A graph is* monopolar *if its vertex set can be two-partitioned into V_1 and V_2 such that $G[V_1]$ is an independent set and $G[V_2]$ is a cluster graph. The partition (V_1, V_2) is called* monopolar partition.

Again, the vertices in V_1 are called periphery vertices and the vertices in V_2 are called core vertices. Our second fitting model now is the following.

> Monopolar Editing
> Input: An undirected graph $G = (V, E)$.
> Task: Transform G into a monopolar graph by applying a minimum number of edge modifications and output a monopolar partition.

Clearly, both models are simplistic and cannot completely reflect biological reality. For example, subunits of protein complexes consisting of two proteins that first interact with each other and subsequently with the core of a protein complex are supported by neither of our models. Nevertheless, our models are less simplistic than pure clustering models that attempt to divide protein interaction networks into disjoint dense clusters. Furthermore, there is a clear trade-off between model complexity, algorithmic feasibility of models, and interpretability.

Further related work. The related optimization problem Split Editing asks to transform a graph into a (single) split graph by at most k edge modifications. Split Editing is, somewhat surprisingly, solvable in polynomial time [13]. Another approach of fitting PPI networks to specific graph classes was proposed by Zotenko et al. [25] who find for a given PPI network a close chordal graph, that is, a graph without induced cycles of length four or more. The modification operation is insertion of edges.

Preliminaries. We consider undirected simple graphs $G = (V, E)$ where $n := |V|$ denotes the number of vertices and $m := |E|$ denotes the number of edges. The *open neighborhood* of a vertex u is defined as $N(u) := \{v \mid \{u, v\} \in E\}$. We denote the *neighborhood of a set* U by $N(U) := \bigcup_{u \in U} N(u) \setminus U$. The *subgraph induced by a vertex set* S is defined as $G[S] := (S, \{\{u, v\} \in E \mid u, v \in S\})$.

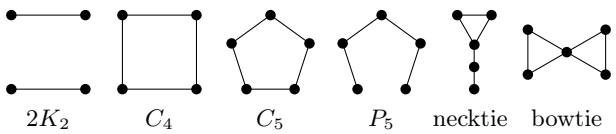


Fig. 1. The forbidden induced subgraphs for split graphs ($2K_2$, C_4, and C_5) and for split cluster graphs (C_4, C_5, P_5, necktie, and bowtie).

2 Combinatorial Properties and Complexity

Before presenting concrete algorithmic approaches for the two optimization problems, we show some properties of split cluster graphs and monopolar graphs which will be useful in the various algorithms. Furthermore, we present computational hardness results for the problems which will justify the use of integer linear programming (ILP) and heuristic approaches.

Split Cluster Editing. Each connected component of the solution has to be a split graph. These graphs can be characterized by forbidden induced subgraphs (see Fig. 1).

Lemma 1 ([10]). *A graph G is a split graph if and only if G does not contain an induced subgraph that is a cycle of four or five edges or a pair of disjoint edges (that is, G is $(C_4, C_5, 2K_2)$-free).*

To obtain a characterization for split cluster graphs, we need to characterize the existence of $2K_2$'s within connected components.

Lemma 2. *If a connected graph contains a $2K_2$ as induced subgraph, then it contains a $2K_2 = (V', E')$ such that there is a vertex $v \notin V'$ that is adjacent to at least one vertex of each K_2 of (V', E').*

Proof. Let G contain the $2K_2$ $\{x_1, x_2\}, \{y_1, y_2\}$ as induced subgraph. Without loss of generality, let the shortest path between any x_i, y_j be $P = (x_1 = p_1, p_2, \ldots, p_k = y_1)$. Clearly, $k > 2$. If $k = 3$, then x_1 and y_1 are both adjacent to p_2. Otherwise, if $k = 4$, then $\{x_2, x_1 = p_1\}, \{p_3, p_4 = y_1\}$ is a $2K_2$ and x_1 and p_3 are both adjacent to p_2. Finally, if $k > 4$, then P contains a P_5 as induced subgraph. The four outer vertices of this P_5 induce a $2K_2$ whose K_2's each contain a neighbor of the middle vertex. □

We can now provide a characterization of split cluster graphs.

Theorem 1. *A graph G is a split cluster graph if and only if G is a $(C_4, C_5, P_5,$ necktie, bowtie$)$-free graph.*

Proof. Let G be a split cluster graph, that is, every connected component is a split graph. Clearly, G does not contain a C_4 or C_5. If a connected component of G contains a P_5, then omitting the middle vertex of the P_5 yields a $2K_2$, which

contradicts that the connected component is a split graph. The same argument shows that the graph cannot contain a necktie or bowtie.

Conversely, let G be $(C_4, C_5, P_5, \text{necktie}, \text{bowtie})$-free. Clearly, no connected component contains a C_4 or C_5. Assume for a contradiction that a connected component contains a $2K_2$ consisting of the K_2's $\{a, b\}$ and $\{c, d\}$. Then according to Lemma 2 there is a vertex v which is, without loss of generality, adjacent to a and c. If no other edges between the $2K_2$ and v exist, then $\{a, b, v, c, d\}$ is a P_5. Adding exactly one of $\{b, v\}$ or $\{d, v\}$ creates a necktie, and adding both edges results in a bowtie. No other edges are possible, since there are no edges between $\{a, b\}$ and $\{c, d\}$. □

This leads to a linear-time algorithm for checking whether a graph is a split cluster graph.

Theorem 2. *A forbidden subgraph for a split cluster graph can be found in $O(n+m)$ time.*

Proof. For each connected component, we run an algorithm by Heggernes and Kratsch [14] that checks in linear time whether a graph is a split graph, and if not, produces a $2K_2$, C_4, or C_5. If the forbidden subgraph is a C_4 or C_5, we are done. If it is a $2K_2$, we can find in linear time a P_5, necktie, or bowtie, using the method described in the proof of Lemma 2. □

In contrast, Split Cluster Editing is NP-hard even in restricted cases. We reduce from Cluster Editing which has as input an undirected graph $G = (V, E)$ and an integer k, and asks whether G can be transformed into a cluster graph by applying at most k edge modifications. Cluster Editing is NP-hard even if the maximum degree of the input graph is five [11] and it cannot be solved in $2^{o(k)} \cdot n^{O(1)}$ time assuming the so-called exponential-time hypothesis (ETH) [11, 17]. The reduction simply attaches to each vertex u an additional $\deg_G(v)$ many new degree-one vertices; we omit the correctness proof.

Theorem 3. Split Cluster Editing *is NP-hard even on graphs with maximum degree 10. Further, it cannot be solved in $2^{o(k)} \cdot n^{O(1)}$ or $2^{o(n)} \cdot n^{O(1)}$ time if the exponential-time hypothesis (ETH) [15] is true.*

This hardness result motivates the study of algorithmic approaches such as fixed-parameter algorithms or ILP-formulations. For example, Split Cluster Editing is fixed-parameter tractable for the parameter number of edge modifications k by the following search tree algorithm: Check whether the graph contains a forbidden subgraph. If this is the case, branch into the possibilities to destroy this subgraph. In each recursive branch, the number of allowed edge modifications decreases by one. Furthermore, since the largest forbidden subgraph has five vertices, at most ten possibilities for edge insertions or deletions have to be considered to destroy a forbidden subgraph. By Theorem 2, forbidden subgraphs can be found in $O(n+m)$ time. Altogether, this implies the following.

Theorem 4. Split Cluster Editing *can be solved in* $O(10^k \cdot (n+m))$ *time.*

This result is purely of theoretical interest. With further improvements of the search tree algorithm, practical running times might be achievable.

Monopolar Graphs. The class of monopolar graphs is hereditary, and thus it is characterized by forbidden induced subgraphs, but the set of minimal forbidden induced subgraphs is infinite [2]; for example among graphs with five or fewer vertices, only the wheel W_4 (⊠) is forbidden, but there are 34 minimal forbidden subgraphs with six vertices. In contrast to the recognition of split cluster graphs, which is possible in linear time by Theorem 2, deciding whether a graph is monopolar is NP-hard [9]. Thus Monopolar Editing is NP-hard already for $k = 0$ edge modifications.

3 Solution Approaches

Integer Linear Programming. We experimented with a formulation based directly on the forbidden subgraphs for split cluster graphs (Theorem 1). However, we found a formulation based on the following observation to be faster in practice, and moreover applicable also to Monopolar Editing: If we correctly guess the partition into clique and independent set vertices, we can get a simpler characterization of split cluster graphs by forbidden subgraphs.

Lemma 3. *Let $G = (V, E)$ be a graph and $C \dot\cup I = V$ a partition of the vertices. Then G is a split cluster graph with core vertices C and periphery vertices I if and only if it does not contain an edge with both endpoints in I, nor an induced P_3 with both endpoints in C.*

Proof. "⇒": Clearly, if there is an edge with both endpoints in I or an induced P_3 with both endpoints in C, then I is not an independent set or C does not form a clique in each connected component, respectively.

"⇐": We again use contraposition. If G is not a split cluster graph with core vertices C and periphery vertices I, then it must contain an edge with both endpoints in I, or $C \cap H$ does not induce a clique for some connected component H of G. In the first case we are done; in the second case, there are two vertices $u, v \in C$ in the same connected component with $\{u, v\} \notin E$. Consider a shortest path $u = p_1, \ldots, p_l = v$ from u to v. If it contains a periphery vertex $p_i \in I$, then p_{i-1}, p_i, p_{i+1} forms a forbidden subgraph. Otherwise, p_1, p_2, p_3 is one. □

With a very similar proof, we can get a simpler set of forbidden subgraphs for annotated monopolar graphs.

Lemma 4. *Let $G = (V, E)$ be a graph and $C \dot\cup I = V$ a partition of the vertices. Then G is a monopolar graph with core vertices C and periphery vertices I if and only if it does not contain an edge with both endpoints in I, nor an induced P_3 whose vertices are contained in C.*

Proof. "⇒": Easy to see as in Lemma 3.

"⇐": If G is not monopolar with core vertices C and periphery vertices I, then it must contain an edge with both endpoints in I, or C does not induce a cluster graph. In the first case we are done; in the second case, there is a P_3 with all vertices in C, since that is the forbidden subgraph for cluster graphs. □

From Lemma 3, we can directly derive an integer linear programming formulation for Split Cluster Editing. We introduce binary variables e_{uv} indicating whether the edge $\{u, v\}$ is present in the solution graph and binary variables c_u indicating whether a vertex u is part of the core. Defining $\bar{e}_{uv} := 1 - e_{uv}$ and $\bar{c}_u := 1 - c_u$, and fixing an arbitrary order on the vertices, we have

$$\text{minimize} \sum_{\{u,v\}\in E} \bar{e}_{uv} + \sum_{\{u,v\}\notin E} e_{uv} \text{ subject to} \tag{1}$$

$$c_u + c_v + \bar{e}_{uv} \geq 1 \ \forall u, v \tag{2}$$

$$\bar{e}_{uv} + \bar{e}_{vw} + e_{uw} + \bar{c}_u + \bar{c}_w \geq 1 \ \forall u \neq v, v \neq w > u. \tag{3}$$

Herein, Eq. (2) forces that the periphery vertices are an independent set and Eq. (3) forces that core vertices in the same connected component form a clique. For Monopolar Editing, we can replace Eq. (3) by

$$\bar{e}_{uv} + \bar{e}_{vw} + e_{uw} + \bar{c}_u + \bar{c}_v + \bar{c}_w \geq 1 \ \forall u \neq v, v \neq w > u \tag{4}$$

which forces that the graph induced by the core vertices is a cluster graph.

Data Reduction. Data reduction (preprocessing) proved very effective for solving Cluster Editing optimally [3, 4]. Indeed, any instance can be reduced to one of at most $2k$ vertices [7], where k is the number of edge modifications. Unfortunately, the data reduction rules we devised for Split Cluster Editing were not applicable to our real-world test instances. However, a simple observation allows us to fix the values of some variables of Eqs. (1) to (3) in the Split Cluster Editing ILP: if a vertex u has only one vertex v as neighbor and $\deg(v) > 1$, then set $c_u = 0$ and $e_{uw} = 0$ for all $w \neq v$. Since our instances have many degree-one vertices, this considerably reduces the size of the ILPs.

Heuristics. The integer linear programming approach is not able to solve the hardest of our instances. Thus, we employ the well-known *simulated annealing* heuristic, a local search method. For Split Cluster Editing, we start with a clustering where each vertex is a singleton. As random modification, we move a vertex to a cluster that contains one of its neighbors. Since this allows only a decrease in the number of clusters, we also allow moving a vertex into an empty cluster. For a fixed clustering, the optimal number of modifications can be computed in linear time by counting the edges between clusters and computing for each cluster a solution for Split Editing in linear time [13]. For Monopolar Editing, we also allow moving a vertex into the independent set. Here, the optimal number of modifications for a fixed clustering can also be calculated in linear time: all edges in the independent set are deleted, all edges between clusters are deleted, and all missing edges within clusters are added.

Table 1. Network statistics. Here, n is the number of proteins, without singletons, and m is the number of interactions; n_{lcc} and m_{lcc} are the number of proteins and interactions in the largest connected component; C is the number of CYC2008 complexes with at least 50% and at least three proteins in the network, p is the number of network proteins that do not belong to these complexes, and A_C is the average complex size. Finally, i_g is the number of genetic interactions between proteins without physical interaction.

	n	m	n_{lcc}	m_{lcc}	C	p	A_C	i_g
cell cycle	196	797	192	795	7	148	21.8	1151
transcription	215	786	198	776	11	54	28.0	1479
translation	236	2352	186	2351	5	88	29.8	174

4 Experimental Results

We test exact algorithms and heuristics for Split Cluster Editing (SCE) and Monopolar Editing (ME) on several PPI networks, and perform a biological evaluation of the modules found. We use two known methods for comparison. The algorithm by Luo et al. [19] ("Luo" for short) produces clusters with core and periphery, like SCE, but the clusters may overlap and might not cover the whole graph. The SCAN algorithm [24], like ME, partitions the graph vertices into "clusters", which we interpret as cores, and "hubs" and "outliers", which we interpret as periphery.

4.1 Experimental setup

Data. We perform all our experiments on subnetworks of the *S. cerevisiae* (yeast) PPI network from BioGRID [6], version 3.2.101. Our networks contain only physical interactions; we use genetic interactions only for the biological evaluation. From the complete BioGRID yeast network with 6377 vertices and 81549 edges, we extract three subnetworks, corresponding to three essential processes: cell cycle, translation, and transcription. These are important subnetworks known to contain complexes. To determine the protein subsets corresponding to each process, we select all yeast genes annotated with the relevant GO terms: GO:0007049 (cell cycle), GO:0006412 (translation), and GO:0006351 (DNA-templated transcription). Table 1 shows some properties of these networks.

Implementation details. The integer linear program and simulated annealing heuristic were implemented in C++ and compiled with the GNU g++ 4.7.2 compiler. As ILP solver, we used CPLEX 12.6.0. For the ILP, we use the heuristic solution found after one minute as MIP start, and initially add all independent set constraints (2). In a cutting plane callback, we add the 500 most violated constraints of type (3) or (4).

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 Debian GNU/Linux 7.0.

Biological evaluation. We evaluate our results using the following measures. First, we examine the coherence of the GO terms in our modules using the semantic similarity score calculated by G-SESAME [8]. We use this score to test the hypothesis that the cores are more stable than the peripheries. If the hypothesis is true, then the pairwise similarity score within the core should be higher than in the periphery. We test only terms relating to process, not function, since proteins in the same complex play a role in the same biological process. Since Monopolar Editing and SCAN return multiple cores and only a single periphery, we assign to each cluster C its neighborhood $N(C)$ as periphery. We consider only clusters with at least two core vertices and one periphery vertex.

Next, we compare the resulting clusters with known protein complexes from the CYC2008 database [20]. Since the networks we analyze are subnetworks of the larger yeast network, we discard for each network the CYC2008 complexes that have less than 50% of their vertices in the current subnetwork, restrict them to proteins contained in the current subnetwork, and then discard those with fewer than three proteins. We expect that the cores mostly correspond to complexes and that the periphery may contain complex vertices plus further vertices.

Finally, we analyze the genetic interactions between and within modules. Ideally, we would obtain significantly more genetic interactions outside of cores than within them. This is supported by the *between pathways model* [16], which proposes that different complexes can back one another up, thus disabling one would not harm the cell, but disabling both complexes would reduce its fitness or kill it. Here, when counting genetic interactions, we are interested only in genetic interactions that occur between proteins that do not physically interact.

4.2 Results

Our results are summarized in Table 2. For Split Cluster Editing, the ILP approach failed to solve the cell cycle and transcription network, and for Monopolar Editing, it failed to solve the transcription network, with CPLEX running out of memory in each case. The fact that for the "harder" problem ME more instances were solved could be explained by the fact that the number k of necessary modifications is much lower, which could reduce the size of the branch-and-bound tree. For the three optimally solved instances, the heuristic also finds the optimal solution after one minute for two of them, but for the last one (ME transcription) only after several hours; after one minute, it is 2.9% too large. This indicates the heuristic gives good results, and in the following, we use the heuristic solution for the three instances not solvable by ILP.

Table 3 gives an overview of the results. We say that a cluster is *interesting* if it contains at least two vertices in the core and at least one in the periphery. In the cell cycle network (see Fig. 2), the SCE solution identifies ten interesting clusters, along with four clusters containing only cores, and some singletons. Only for one of the ten clusters is the GO term coherence higher in the periphery than in the core, as expected (for two more the scoring tool does not return a result).

Table 2. Experimental results. Here, K is the number of clusters with at least two vertices in the core and at least one in the periphery, p is the size of the periphery, k is the number of edge modifications, and c_t, c_c, and c_p is the average coherence within the cluster, core, and periphery, respectively.

	cell-cycle						transcription						translation					
	K	p	k	c_t	c_c	c_p	K	p	k	c_t	c_c	c_p	K	p	k	c_t	c_c	c_p
SCE	10	108	321	0.60	0.64	0.40	13	112	273	0.54	0.54	0.57	6	94	308	0.63	0.73	0.69
ME	24	75	126	0.46	0.58	0.39	26	78	106	0.55	0.61	0.54	11	129	240	0.52	0.58	0.53
SCAN	28	48	—	0.42	0.62	0.34	26	58	—	0.53	0.51	0.47	2	25	—	0.59	0.59	0.76
Luo	16	84	—	0.34	0.50	0.31	12	125	—	0.40	0.52	0.38	4	137	—	0.72	0.84	0.67

Table 3. Experimental results for the complex test. Here, D is the number of detected complexes ($\geq 50\%$ of core contained in complex and $\geq 50\%$ of complex contained in cluster), $\text{core}_{\%}$ is among the detected complexes the median percentage of core vertices that are in this complex and $\text{comp}_{\%}$ is the median percentage of complex proteins that are in the cluster.

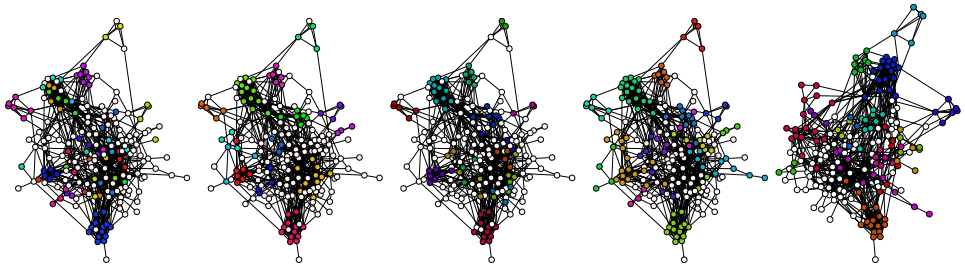
	cell-cycle			transcription			translation		
	D	$\text{core}_{\%}$	$\text{comp}_{\%}$	D	$\text{core}_{\%}$	$\text{comp}_{\%}$	D	$\text{core}_{\%}$	$\text{comp}_{\%}$
SCE	4	100	100	7	89	100	4	100	96
ME	5	100	100	11	100	100	4	100	96
SCAN	4	91	100	8	84	100	0	—	—
Luo	5	81	100	6	87	100	4	92	96

Following our hypothesis, we say that a complex is *detected* by a cluster if at least 50% of the core belongs to the complex and at least 50% of the complex belongs to the cluster. Out of the seven complexes, three are detected without any error, and one is detected with an error of two additional proteins in the core that are not in the complex. The periphery contains between one and eight extra proteins that are not in the complex (which is allowed by our hypothesis).

The Monopolar Editing result contains more interesting clusters than SCE (24). Compared to SCE, clusters are on average smaller and have a smaller core, but about the same periphery size (recall that a periphery vertex may occur in more than one cluster). ME detects the same complexes as SCE, plus one additional complex.

SCAN identifies 7 hubs and 41 outliers, which then comprise the periphery. SCAN fails to detect one of the complexes ME finds. It also has slightly more errors, for example having three extra protein in the core for the anaphase-promoting complex plus one missing. Luo identifies only large clusters (this is true for all subnetworks we tested). It detects the same complexes as ME, but also finds more extra vertices in the cores.

In the transcription network, for GO-Term analysis, we see a similar pattern here that Luo has worse coherence, but all methods show less coherence in the



(a) Complexes (b) SCE cores (c) Monopolar (d) SCAN (e) Luo

Fig. 2. Results of the four algorithms on the cell-cycle network. The periphery is in white, remaining vertices are colored according to their clusters.

peripheries than in the cores. The ME method comes out a clear winner here with detecting all 11 complexes and generally fewer errors.

In the translation network, SCE and ME find about the same number of interesting clusters (22 and 24) and detect the same four complexes. The SCAN algorithm does not seem to deal well with this network, since it finds only two interesting clusters and does not detect any complex. Luo finds only four interesting clusters, corresponding to the four complexes also detected by SCE and ME; this might also explain why it has the best coherence values here.

Counting genetic interactions. Since the identified clusters largely correspond to known protein complexes, it is not surprising that we identify a higher than expected number of genetic interactions between these complexes. For SCE, the binomial test to check whether the frequency of genetic interactions within the periphery is higher than the frequency in the entire network gives p-values lower than $4 \cdot 10^{-7}$ for all networks, thus the difference is significant.

Experiments conclusion. The coherence values for cores and peripheries indicate that a division of clusters into core and periphery makes sense. In detecting complexes, the ME method does best (20 detected), followed by SCE and Luo (15 each), and finally SCAN (12). This indicates that the model that peripheries are shared is superior. Note however that SCE is at a disadvantage in this evaluation, since it can use each protein as periphery only once, while having large peripheries makes it easier to count a complex as detected.

5 Outlook

There are many further variants of our models that could possibly yield better biological results or have algorithmic advantages. For instance, one could restrict the cores to have a certain minimum size. Also, instead of using split graphs as a core–periphery model, one could resort to dense split graphs [5] in which every periphery vertex is adjacent to all core vertices. Finally, one could allow some limited amount of interaction between periphery vertices.

References

[1] A. Ben-Dor, R. Shamir, and Z. Yakhini. Clustering gene expression patterns. *Journal of Computational Biology*, 6(3-4):281–297, 1999.

[2] A. J. Berger. Minimal forbidden subgraphs of reducible graph properties. *Discussiones Mathematicae Graph Theory*, 21(1):111–117, 2001.

[3] S. Böcker and J. Baumbach. Cluster editing. In *Proc. 9th CiE*, volume 7921 of *LNCS*, pages 33–44. Springer, 2013.

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

[5] S. P. Borgatti and M. G. Everett. Models of core/periphery structures. *Social Networks*, 21(4):375–395, 1999.

[6] A. Chatr-aryamontri et al. The BioGRID interaction database: 2013 update. *Nucleic Acids Research*, 41(D1):D816–D823, 2013.

[7] J. Chen and J. Meng. A $2k$ kernel for the cluster editing problem. *Journal of Computer and System Sciences*, 78(1):211–220, 2012.

[8] Z. Du, L. Li, C.-F. Chen, P. S. Yu, and J. Z. Wang. G-SESAME: web tools for GO-term-based gene similarity analysis and knowledge discovery. *Nucleic Acids Research*, 37(suppl. 2):W345–W349, 2009.

[9] A. Farrugia. Vertex-partitioning into fixed additive induced-hereditary properties is NP-hard. *The Electronic Journal of Combinatorics*, 11(1): R46, 2004.

[10] S. Foldes and P. L. Hammer. Split graphs. *Congressus Numerantium*, 19: 311–315, 1977.

[11] F. V. Fomin, S. Kratsch, M. Pilipczuk, M. Pilipczuk, and Y. Villanger. Subexponential fixed-parameter tractability of cluster editing. *CoRR*, abs/1112.4419, 2011.

[12] A.-C. Gavin et al. Proteome survey reveals modularity of the yeast cell machinery. *Nature*, 440(7084):631–636, 2006.

[13] P. L. Hammer and B. Simeone. The splittance of a graph. *Combinatorica*, 1(3):275–284, 1981.

[14] P. Heggernes and D. Kratsch. Linear-time certifying recognition algorithms and forbidden induced subgraphs. *Nordic Journal of Computing*, 14(1–2): 87–108, 2007.

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

[16] R. Kelley and T. Ideker. Systematic interpretation of genetic interactions using protein networks. *Nature Biotechnology*, 23(5):561–566, 2005.

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

[18] H. C. Leung, Q. Xiang, S.-M. Yiu, and F. Y. Chin. Predicting protein complexes from PPI data: a core-attachment approach. *Journal of Computational Biology*, 16(2):133–144, 2009.

[19] F. Luo, B. Li, X.-F. Wan, and R. Scheuermann. Core and periphery structures in protein interaction networks. *BMC Bioinformatics*, (Suppl 4):S8, 2009.

[20] S. Pu, J. Wong, B. Turner, E. Cho, and S. J. Wodak. Up-to-date catalogues of yeast protein complexes. *Nucleic Acids Research*, 37(3):825–831, 2009.

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

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

[23] M. Wu, X. Li, C.-K. Kwoh, and S.-K. Ng. A core-attachment based method to detect protein complexes in PPI networks. *BMC Bioinformatics*, 10(1): 169, 2009.

[24] X. Xu, N. Yuruk, Z. Feng, and T. A. J. Schweiger. SCAN: a structural clustering algorithm for networks. In *Proc. 13th KDD*, pages 824–833. ACM, 2007.

[25] E. Zotenko, K. S. Guimarães, R. Jothi, and T. M. Przytycka. Decomposition of overlapping protein complexes: a graph theoretical method for analyzing static and dynamic protein associations. *Algorithms for Molecular Biology*, 1(7), 2006.