A Graph-Theoretic Formulation of Exploratory Blockmodeling*

Alexander Bille
Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, Germany

Niels Grüttemeier
System Technologies and Image Exploitation,
Fraunhofer IOSB, Lemgo, Fraunhofer Institute of Optronics, Germany

Christian Komusiewicz
Institute of Computer Science, Friedrich Schiller Universität Jena, Germany

Nils Morawietz
Institute of Computer Science, Friedrich Schiller Universität Jena, Germany

Abstract

We present a new simple graph-theoretic formulation of the exploratory blockmodeling problem on undirected and unweighted one-mode networks. Our formulation takes as input the network G and the maximum number t of blocks for the solution model. The task is to find a minimum-size set of edge insertions and deletions that transform the input graph G into a graph G' with at most t neighborhood classes. Herein, a neighborhood class is a maximal set of vertices with the same neighborhood. The neighborhood classes of G' directly give the blocks and block interactions of the computed blockmodel.

We analyze the classic and parameterized complexity of the exploratory blockmodeling problem, provide a branch-and-bound algorithm, an ILP formulation and several heuristics. Finally, we compare our exact algorithms to previous ILP-based approaches and show that the new algorithms are faster for $t \geq 4$.

2012 ACM Subject Classification Theory of computation → Social networks; Theory of computation → Parameterized complexity and exact algorithms; Theory of computation → Branch-and-bound

Keywords and phrases Clustering, Exact Algorithms, ILP-Formulation, Branch-and-Bound, Social Networks

Digital Object Identifier 10.4230/LIPIcs.SEA.2023.14

Supplementary Material *Software (Source Code and Results)*: `https://git.uni-jena.de/algo-engineering/blockmodeling`
archived at `swh:1:dir:1271064707a641448704346b9ff9dc43cdc799b3`

Funding *Nils Morawietz*: Supported by the Deutsche Forschungsgemeinschaft (DFG), project OPERAH, KO 3669/5-1.

Acknowledgements We would like to thank the anonymous reviewers of SEA for their helpful comments which have improved the presentation of our results.

1 Introduction

In social network analysis, a standard task is to determine which vertices have the same role in the network. One approach for this role assignment problem, called structural equivalence [3], is to assign the same role to vertices if and only if they have the *same* neighborhood. This

* Most of the results of the presented work are also contained in the first author's Master's thesis `https://www.uni-marburg.de/de/fb12/arbeitsgruppen/algorith/paper/master-alex-bille.pdf`

demand is often too strict. In this case, one may instead use blockmodeling. Here, one wants to partition the vertex set of the network into blocks in such a way that vertices with *similar* neighborhoods end up in the same block [3, 25]. The partition of the vertices gives a blockmodel of the input network, which is essentially a graph whose vertices are the blocks and where an edge between two blocks a and b indicates that vertices of block a are likely to interact with vertices of block b. Usually this graph is represented via its adjacency matrix, also called blockmatrix or imagematrix. For an example of a vertex partition of a graph and the corresponding blockmatrix, see Figure 1.

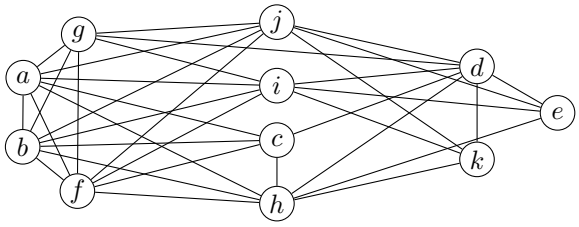
A critical distinction in blockmodeling is whether the blockmatrix is fixed in advance or not [5, 8, 21]. For fixed blockmatrices, the blockmodeling problem essentially asks to confirm whether this blockmatrix is a good explanation for the network; when the blockmatrix is not fixed, then we have an *exploratory* blockmodeling problem, where we aim to identify an unknown model from the given input. In this work, we study a simple graph-theoretic formulation of the exploratory blockmodeling problem. We adopt a graph modification point of view which has already been used for graph clustering [11, 23] and special cases of blockmodeling such as the identification of core/periphery structures [2, 4]. In our formulation, the only prespecified aspect of the blockmodel is the number t of blocks. A starting point is to consider the situation that the graph admits a perfect blockmodel with t (possibly empty) blocks. Informally speaking, this is the case when the number of different neighborhoods of the graph is at most t: In that case, we may simply assign all vertices that have the same neighborhood to the same block. In the resulting blockmodel, each block is either a clique or an independent set in the graph and between two different blocks, either all edges are present or all edges are missing in the input graph.

Naturally, for small values of t, most networks will not admit a perfect blockmodel in this sense. Our aim is to find a new network that can be obtained from the original network by a minimum number of edge additions or deletions and which admits a perfect blockmodel with t blocks. In other words, we aim for a blockmodel with t blocks such that the number of disagreements between input network and blockmodel is minimum. This objective function measures globally how different the neighborhoods inside a block are. Thus, minimizing the objective leads to a blockmodel in which vertices with similar neighborhoods indeed end up in the same block.

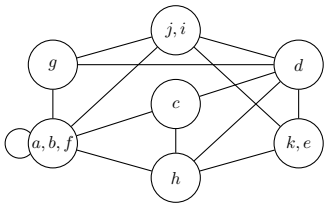
1.1 Related Work

Batagelj et al. [1] also consider the problem of finding the blockmodel with the minimum number of disagreements. There are two main differences: First, in contrast to Batagelj et al. [1], we take a graph-theoretic view. Second, we consider exact algorithms whereas Batagelj et al. [1] use a local search heuristic called TEA. The most closely related exact approaches were used by Brusco and Steinley [5] and Dabkowski et al. [8]. One difference is that these works consider directed networks. A further, algorithmic, difference is that both works solve the exploratory blockmodeling by considering all possible $t \times t$ blockmatrices and selecting one that gives a solution with a minimum number of disagreements. Brusco and Steinley [5] note that their approach is limited to $t \leq 3$ due to the rapidly growing number of possible blockmatrices. Dabkowski et al. [8] show that by considering for example isomorphism classes of blockmatrices, this approach can be extended to $t = 4$. Both works use integer linear programming (ILP) formulations to compute optimal solutions for small networks with up to 20 vertices [5] and 13 vertices [8], respectively.

There are further less closely related formulations. Reichardt and White [21] use a cost function that additionally introduces corrections for the degree distribution of the network. They consider the case with and without prespecified blockmatrix and use local search



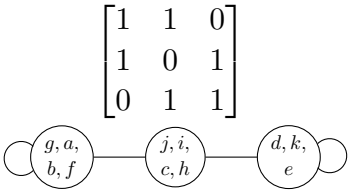
(a) Input graph H.



(b) Neighborhood graph of H. For the sake of clarity, loops are omitted when the corresponding neighborhood class consists of one vertex.

	g	a	b	f	j	i	c	h	d	k	e
g	-	1	1	1	1	1	0	0	1	0	0
a	1	-	1	1	1	1	1	1	0	0	0
b	1	1	-	1	1	1	1	1	0	0	0
f	1	1	1	-	1	1	1	1	0	0	0
j	1	1	1	1	-	0	0	0	1	1	1
i	1	1	1	1	0	-	0	0	1	1	1
c	0	1	1	1	0	0	-	1	1	0	0
h	0	1	1	1	0	0	1	-	1	1	1
b	1	0	0	0	1	1	1	1	-	1	1
k	0	0	0	0	1	1	0	1	1	-	0
e	0	0	0	0	1	1	0	1	1	0	-

(c) Partitioned adjacency matrix of H where red entries are the vertex pairs contained in an optimal set of edge modifications S.



(d) The blockmatrix for H with $t = 3$ obtained by applying S, and the neighborhood graph of $G \triangle S$.

Figure 1 Example instance $(H, t = 3)$ of Blockmodeling with one optimal solution S.

to compute heuristic solutions. Chan et al. [6] use nonnegative matrix factorization to compute blockmodels without prespecified blockmatrix, the factorization problem is solved via a (heuristic) gradient descent method. Jessop [14] and Proll [19] use a model where the objective function rewards large blocks and every block must have a certain minimum density in the input network, both models are solved via ILP formulations.

From the more graph-algorithmic perspective, the most closely related work is due to Damaschke and Mogren [9] who consider graph modification formulations of blockmodeling problems where the blockmatrix is prespecified and each block must become a clique. This problem was shown to be NP-hard for a number of different blockmatrix types [9, 15]; the problem can be solved efficiently when the number of necessary modifications and t are small [9, 13]. Core/periphery problems can be seen as blockmodeling problems for $t = 2$ and a certain fixed blockmatrix [2]. The special case when t equals the number of vertices, each block is fixed to be a clique, and the blockmatrix is fixed to have no edges between different blocks is known as Cluster Editing [23].

A further distinction in blockmodeling approaches is whether they are direct or indirect [3]. Direct approaches compute the blockmodel using the adjacency information of the graph itself. In contrast, indirect approaches follow a two-step procedure where the first step is to compute a distance function for the network vertices based on their neighborhoods and the second step is to compute a clustering of the vertex set with respect to this distance function. In this terminology, our approach and the related ones mentioned above are direct methods.

1.2 Our Results

We first formally define Blockmodeling, a simple edge-modification-based formulation for exploratory blockmodeling with t (possibly empty) blocks that avoids blockmatrices in the problem definition (Section 2). We then show that Blockmodeling is NP-hard for all fixed values of $t \geq 2$ and that it can be solved efficiently when t and the necessary number k of edge modifications are small (Section 3). We develop a branch-and-bound algorithm, an ILP formulation, and several heuristics for Blockmodeling (Section 4). For the branch-and-bound algorithm, we present several speedups based on reduction rules and upper and lower bounds. We evaluate our algorithms experimentally on standard benchmark data sets (Section 5). A comparison with an adaption of the approach of Dabkowski et al. [8] to undirected networks shows that the new algorithm is much faster for $t \geq 4$. For example, for $t = 5$ our new algorithm can find optimal solutions on networks with up to 32 vertices whereas previous approaches can solve only networks with up to 18 vertices. Summarizing, our new approach is competitive with state-of-the-art approaches for exact blockmodeling and paves the way for exact algorithms for larger values of t. In this regard, our approach is a substantial improvement over previous approaches as trying all $t \times t$ blockmatrices becomes clearly infeasible for $t \geq 6$.

Moreover, we find that the heuristics give almost-optimal solutions on the considered instances. Finally, we show that for $t = 4$ our approach finds a reasonable solution for the Karate Club network. Proofs of statements marked with (*) are deferred to the appendix.

2 Preliminaries and Problem Definition

We now introduce some relevant graph-theoretic notation and formally define the exploratory blockmodeling problem.

Notation. For a set S and an integer x, we define $\binom{S}{x} := \{T \subseteq S \mid x = |T|\}$. The symmetrical difference of two sets S and T is denoted by $S \triangle T := (S \cup T) \setminus (S \cap T)$. A collection of sets $(T_1, \ldots, T_x)$ is a *partition* of S if and only if $\bigcup_{i=1}^{x} T_i = S$ and $T_i \cap T_j = \emptyset$ for all $i \neq j$.

A simple undirected graph $G = (V, E)$ consists of a set of *vertices* V and a set of *edges* $E \subseteq \binom{V}{2}$. We set $n := |V|$ and $m := |E|$. Let S be a set of vertices of G, then we denote the deletion of S in G by $G - S := (V \setminus S, \{e \in E \mid S \cap e = \emptyset\})$. The set of edges of G with one endpoint in S and the other in T is denoted by $E_G(S, T) := \{\{s, t\} \in E \mid s \in S, t \in T\}$. Furthermore, let $E_G(S) := E_G(S, S)$ denote the set of edges with both endpoints in S.

The *neighborhood* of a vertex v is defined as $N_G(v) := \{u \mid \{u, v\} \in E\}$. We call the vertices of $N_G(v)$ the *neighbors* of v. If $\{u, v\} \in E$, then we say u and v are *adjacent*. If a vertex v is adjacent to every other vertex of V, then we call v *universal*. A vertex with no neighbors is called *isolated*. Two disjoint vertex sets $S_1 \subseteq V$ and $S_2 \subseteq V$ are *adjacent* if each vertex of S_1 is adjacent to every vertex of S_2. Similarly, S_1 and S_2 are *non-adjacent* if $E_G(S_1, S_2) = \emptyset$. We say the vertices u and v have the same *adjacency* to another vertex w if $\{u, w\} \in E$ if and only if $\{v, w\} \in E$. We say that u and v have the same *adjacency* to a set W of vertices if for each vertex $w \in W$, u and v have the same adjacency to w.

A set of vertices $S \subseteq V$ is a *clique* in G if $\binom{S}{2} \subseteq E$ and an *independent set* in G if $\binom{S}{2} \cap E = \emptyset$. If G is clear from the context, we may omit the subscript.

Problem Definition. Perfect structural equivalence is defined using the following equivalence relation over vertices [17].

▶ **Definition 1.** *Let $G = (V, E)$ be a graph. We let $\sim_G$ denote the relation over V such that $u \sim_G v$ if and only if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

▶ **Definition 2.** *Let $G = (V, E)$ be a graph. The* neighborhood partition *$\mathcal{W}$ of G is the collection of equivalence classes of $\sim_G$. We say that G has a* neighborhood diversity *of $|\mathcal{W}|$.*

The neighborhood partition is unique for each graph and it can be computed in linear time via computing a modular decomposition of the graph [18, 12]. Each set of this partition is called *neighborhood class*. Every neighborhood class is a clique or an independent set. A neighborhood class C is called *positive* if C is a clique and *negative* if C is an independent set. Note that a neighborhood class containing only one element is both positive and negative.

▶ **Definition 3.** *The* neighborhood graph *of a graph with neighborhood partition $\mathcal{W}$ is the graph $(\mathcal{W}, E')$ with $\{W_i, W_j\} \in E'$ if and only if for all $u \in W_i$ and all $v \in W_j$, $\{u, v\} \in E$ or if $W_i = W_j$ and W_i is positive.*

Neighborhood graphs are undirected but not necessarily simple since they may contain loops.

A transformation of a graph G is done by a set of *edits* $S \subseteq \binom{V}{2}$ and results in the graph $G \triangle S := (V, E \triangle S)$. We may now formulate blockmodeling as follows.

Blockmodeling
Input: An undirected graph $G = (V, E)$ and integers k and t.
Question: Is there a set of edits $S \subseteq \binom{V}{2}$ of size at most k such that $G \triangle S$ has neighborhood diversity at most t?

A set of edits that fulfills the above requirements is a *solution* of an instance I, a solution S for I is *optimal* if there is no solution S' for I with $|S'| < |S|$. Given a solution S to an instance of Blockmodeling, we can obtain the blockmodel from $G \triangle S$ as follows: the equivalence classes of $G \triangle S$ are the blocks and the blockmatrix is the adjacency matrix of the neighborhood graph. See Figure 1 for an example instance of Blockmodeling and an optimal solution.

3 NP-Hardness and Kernelization

We now study the complexity of the problem. For the hardness proof, it will be interesting to distinguish vertices whose neighborhoods are changed by the solution from the remaining vertices. Accordingly, a vertex v is called *affected* by a set of edits S if at least one element of S contains v. Otherwise, v is *unaffected* by S.

▶ **Lemma 4 (*).** *If (G, k, t) is a yes-instance of Blockmodeling, then there is a solution such that every vertex of each neighborhood class of size larger than $2k$ in G is unaffected.*

3.1 NP-Hardness

In this section, we show NP-hardness for Blockmodeling for each fixed $t \geq 2$. We first show the NP-hardness for $t = 2$ and afterwards, we extend this result to each fixed $t \geq 2$.

▶ **Lemma 5.** *Blockmodeling is* NP*-hard for $t = 2$.*

Proof. We reduce from Sparse Split Graph Editing[1] which is NP-hard [15].

[1] Note that Sparse Split Graph Editing is a confirmatory formulation of Blockmodeling for $t = 2$ with the fixed block matrix $\begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}$. Hence, the problems are closely related but the NP-hardness of Sparse Split Graph Editing does not directly imply the NP-hardness of Blockmodeling.

Sparse Split Graph Editing
Input: A graph $G = (V, E)$ and an integer k.
Question: Is there an edge set S of size at most k such that $G \triangle S$ is a sparse split-graph, that is, a graph consisting of a clique C and set of isolated vertices P?

Let $I = (G = (V, E), k)$ be an instance of Sparse Split Graph Editing. Moreover, let $G' = (V \cup K, E)$ be the graph obtained by adding an independent set K of size $2k + 1$ to G. We set $I' = (G', k, t = 2)$ and show that I is a yes-instance of Sparse Split Graph Editing if and only if I' is a yes-instance of Blockmodeling.

($\Rightarrow$) Let S be a solution of I, that is, an edge set $S \subseteq \binom{V}{2}$ of size at most k such that $G \triangle S$ is a sparse split-graph. We show that S is a solution for I'. Let $G'_{\text{res}} = G' \triangle S$. Note that all vertices of K are unaffected by S. Hence, all vertices of K are isolated in G'_{res}. Since $G \triangle S$ consists of a clique C and an isolated set P, all vertices of $P \cup K$ are isolated in G'_{res}, and thus in the same neighborhood class. Moreover, the vertices of C are a clique in G'_{res} and no vertex of C has any neighbors in $P \cup K$. Thus, C is a neighborhood class in G'_{res}. Consequently, G'_{res} has a neighborhood diversity of at most 2.

($\Leftarrow$) Let S be an optimal solution for I'. Due to Lemma 4, we can assume that each vertex of K is unaffected by S, that is, $S \subseteq \binom{V}{2}$. We show that $G_S := G \triangle S$ is a sparse split-graph. Since S is a solution for I', S has size at most k. Let C and P be the two (potentially empty) neighborhood classes of $G'_{\text{res}} := (V \cup K, E \triangle S)$. Assume without loss of generality that $K \subseteq P$. Note that this implies that P is an independent set in G'_{res}. Moreover, since each vertex of K is unaffected, each vertex of $P \supseteq K$ is isolated in G'_{res}.

Consequently, C is either an empty set or a clique with no neighbors outside of C in G'_{res}. Thus, G'_{res} is a sparse split-graph. Since G_S is equivalent to $G'_{\text{res}} - K$, G_S is also a sparse split-graph. ◀

With the NP-hardness of Blockmodeling for $t = 2$ at hand, we can now easily obtain NP-hardness of Blockmodeling for each fixed $t \geq 2$.

▶ **Theorem 6 (*).** *Blockmodeling is* NP*-hard for every fixed $t \geq 2$.*

3.2 Kernelization

In this section, we provide a problem kernel for the parameter $k + t$ for Blockmodeling. Informally, this is a data reduction algorithm that, in polynomial time, transforms every instance into an equivalent one whose size is bounded by a function of $t + k$; for a formal definition, we refer to the standard monographs on parameterized algorithms [7, 10]. First, note that Lemma 4 implies the correctness of the following.

▶ **Reduction Rule 1.** *Let $I = (G = (V, E), k, t)$ be an instance of Blockmodeling and let C be a neighborhood class of size at least $2k + 2$ in G. Then, remove $|C| - (2k + 1)$ arbitrary vertices of C from G.*

Before we give the kernel, we make another observation regarding the difference of the neighborhood diversities by an application of one edit.

▶ **Lemma 7 (*).** *Let $G = (V, E)$ be a graph with neighborhood diversity t and let $e = \{u, v\}$ be a vertex pair. Then, the neighborhood diversity of $G \triangle \{e\}$ differs from t by at most 2.*

Based on Lemma 4 and Lemma 7, we are now able to obtain a polynomial kernel as shown in the following theorem.

▶ **Theorem 8.** *Blockmodeling admits a kernel with $\mathcal{O}(k^2 + kt)$ vertices that can be computed in $\mathcal{O}(n + m)$ time.*

Proof. By Lemma 7, a single edit reduces the neighborhood diversity by at most 2. Hence, a graph with more than $2k+t$ neighborhood classes cannot be solved with k edits. Consequently, our algorithm returns false if the neighborhood diversity of the input graph is larger than $2k+t$. Next, we reduce the size of each neighborhood class to at most $2k + 1$ by applications of Reduction Rule 1. After an exhaustive application of Reduction Rule 1, any instance has at most $(2k + t) \cdot (2k + 1) = 4k^2 + 2k + 2kt + t \in \mathcal{O}(k^2 + kt)$ vertices. Since the neighborhood partition can be computed in linear time [18], and each application of Reduction Rule 1 can be done simultaneously, this whole algorithm runs in linear time. ◀

Informally, this means that large instances where both k and t are small can be replaced by relatively small instances (which can then be solved faster by an exact algorithm of choice).

4 Algorithms

4.1 Branch-and-Bound

Basic Search Tree Algorithm. Our branch-and-bound algorithm considers $t + 1$ many vertices from distinct neighborhood classes in G. At least two of these vertices have to be in the same neighborhood class in the resulting graph. Recall that two vertices are in different neighborhood classes if and only if there is some vertex to which they have a different adjacency.

▶ **Definition 9.** *Let u and v be vertices. A vertex w is a* witness *of a vertex pair $\{u, v\}$ if and only if w is adjacent to exactly one of u and v.*

Let $\text{wit}(\{u, v\})$ denote the set of witnesses of $\{u, v\}$. To bring u and v in the same neighborhood class, all witnesses need to be *resolved*. Consider a witness w of $\{u, v\}$ with $\{w, u\} \in E$ and $\{w, v\} \notin E$. To resolve w, we have to either add the missing edge $\{w, v\}$ or delete the present edge $\{w, u\}$. This decision has to be done for each witness independently. Hence, when $r = |\text{wit}(\{u, v\})|$ is the number of witnesses of $\{u, v\}$, there are 2^r different possibilities to achieve that u and v are in the same neighborhood class. We call an edit set S a *resolve set* of $\{u, v\}$ if the edits of S resolve every witness of $\{u, v\}$ and S is minimal under this property.

▶ **Observation 10.** *Let $G := (V, E)$ and $G' := (V, E')$ be graphs on the same vertex set V and let u and v be distinct vertices of V. If u and v are in the same neighborhood class in G', then there is some resolve set S of $\{u, v\}$ such that $S \subseteq E \triangle E'$. More specific, for each witness w of $\{u, v\}$, $E \triangle E'$ contains either $\{u, w\}$ or $\{v, w\}$.*

With this observation at hand, we can write a basic version of a branch-and-bound algorithm shown in Algorithm 1.

Note that the algorithm is correct, since if I is a yes-instance of Blockmodeling, there is an optimal solution S for I, such that for some pair $\{u, v\} \in \binom{T}{2}$, u and v are in the same neighborhood class in $G \triangle S$. Due to Observation 10, for each witness w of $\{u, v\}$, S contains either $\{u, w\}$ or $\{v, w\}$. In particular, this holds for the vertex w chosen at Line 7. Hence, I is a yes-instance of Blockmodeling if and only if for some instance I' defined in Line 10, I' is a yes-instance of Blockmodeling.

Algorithm 1 `solveBaB`.

```
1: Input Instance I = (G, k, t) of Blockmodeling
2: Output True if and only if I is a yes-instance of Blockmodeling
3: if k < 0 then return False
4: if G has neighborhood diversity of at most t then return True
5: T ← a set of t + 1 vertices of distinct neighborhood classes in G
6: for all {u, v} ∈ (T choose 2) do
7:     w ← a witness of {u, v}
8:     for all e ∈ {{u, w}, {v, w}} do
9:         G' ← G△{e}
10:        I' ← (G', k − 1, t)
11:        if solveBaB(I') then return True
12: return False
```

In the following, we bound the running time of Algorithm 1. Since T contains exactly $t+1$ vertices, Algorithm 1 reaches Line 10 exactly $\binom{t+1}{2} \cdot 2 = t^2 + t$ times. Each instance defined in Line 10 reduces k by exactly one. Hence, the search tree has at most $(\binom{t+1}{2} \cdot 2)^k = (t^2+t)^k$ leaves.

The running time of the other computations depend on n. The neighborhood classes of G can be computed in $\mathcal{O}(n+m) \subseteq \mathcal{O}(n^2)$ time [18]. Moreover, when given the adjacency matrix, for each pair $\{u, v\} \in \binom{T}{2}$, a witness of $\{u, v\}$ can be found in $\mathcal{O}(n)$ time. Since the adjacency matrix can be computed in $\mathcal{O}(n^2)$ time, Algorithm 1 runs in $\mathcal{O}((t^2+t)^k \cdot n^2)$ time. By initially applying the kernelization algorithm presented in Theorem 8, we obtain the following.

▶ **Theorem 11.** *Blockmodeling can be solved in $\mathcal{O}((t^2+t)^k \cdot (k^2+kt)^2 + n + m)$ time.*

Thus, the problem can be solved efficiently when t and k are small.

Heuristic Speedups. To reduce the running time of the branch-and-bound algorithm we develop several speedups. In the following we describe those that have the highest impact.

The first speedup is an improved branching: Once we have branched into the case that two vertices u and v from a set T need to be merged, we consider the witnesses of u and v one by one. That is, after branching on witness w, we check in the recursive calls whether u and v still have some witness w', and if this is the case, we directly branch into the two ways to resolve w'. As a consequence, often the number of created branches is 2 instead of t^2+t. Furthermore, we store all witnesses for each vertex pair and update them whenever an edit is performed. We use the stored witnesses to update the neighborhood partition after an edit is done.

This branch-and-bound algorithm uses the solution size k^* of any of our heuristics (described in Section 4.3) as an initial upper bound for any optimal solution. The algorithm then searches for a solution S of size at most k^*-1. If such a solution is found, the algorithm decreases the value of k^* by 1 and continues to search for a solution of size at most k^*-1. This is done until no solution of size k^*-1 can be found, that is, if the size of each optimal solution is k^*.

When an edge e is added or removed, we label the vertex pair e as *permanent* in the corresponding child branches. The algorithm forbids that a permanent vertex pair can be edited again. Whenever all witnesses of vertex pair $\{u, v\}$ of the for-loop in Line 6 are resolved, we label this vertex pair as *merged* for its recursive calls. Each two vertices of a merged vertex pair should be in the same neighborhood class in the resulting graph. If each recursive call with u and v merged returns false, we label $\{u, v\}$ as *apart* for the remaining recursive calls. Based on this labeling, we introduce the following reduction rule.

▶ **Reduction Rule 2.** *Let $\{u, v\}$ be a merged vertex pair and let w be a witness of $\{u, v\}$. a) If both vertex pairs $\{u, w\}$ and $\{v, w\}$ are permanent, then return false. b) If the vertex pair $\{u, w\}$ is permanent, then edit $\{v, w\}$.*

The labeling can also be used as follows: any vertex pair that is labeled apart can be skipped in the for-loop of Line 6. Another improvement is the use of a lower bound algorithm (LBA) which computes a number that underestimates the optimal solution size. Our LBA computes disjoint sets of $t + 1$ many vertices of distinct neighborhood classes in the current graph. We call such a set *pack* and a collection of packs is called a *packing*. We define the cost of a pack P as $\min_{p \in \binom{P}{2} \setminus A} |\operatorname{wit}(p)|$ where A are the vertex pairs labeled as apart. The cost is the minimum number of edits that are incident with vertices of P. For a packing $\mathcal{P}$, the sum over the cost of each pack in $\mathcal{P}$ divided by 2 is a lower bound for an instance; division by 2 is necessary since one edit may resolve witnesses in two packs. Furthermore, we try to increase the lower bound by a local search that swaps vertices of a pack with vertices of another pack or with vertices that are in no pack. The packing is updated after each edit; after a fixed time the packing is deleted and recomputed.

4.2 ILP Formulation

The idea of our ILP is derived from Observation 10. Let $G = (V, E)$ be the input graph. The editing of one vertex pair $\{u, v\}$ is represented by the *edit variable* $e_{\{u,v\}} \in \{0, 1\}$. The vertex pair $\{u, v\}$ is in the solution if and only if $e_{\{u,v\}} = 1$. Hence, the objective is to minimize $\sum_{\{u,v\} \in \binom{V}{2}} e_{\{u,v\}}$.

We introduce a *merge variable* $m_{\{u,v\}} \in \{0, 1\}$ for each vertex pair $\{u, v\} \in \binom{V}{2}$. If a merge variable $m_{\{u,v\}}$ equals 1, all witnesses of $\{u, v\}$ must be resolved. Let w be a witness of $\{u, v\}$. The constraints $m_{\{u,v\}} \leq e_{\{u,w\}} + e_{\{v,w\}}$ and $m_{\{u,v\}} \leq 2 - e_{\{u,w\}} - e_{\{v,w\}}$ guarantee that exactly one edit variable equals 1 and thus, in the solution w is resolved for $\{u, v\}$. Next, we introduce constraints for ensuring that there will be at most t neighborhood classes: For every vertex set V' of size $t + 1$, at least two vertices have to be in the same neighborhood class in the resulting graph. Thus, there is at least one vertex pair $\{x, y\}$ with $x \in V'$ and $y \in V'$ such that $m_{\{x,y\}} = 1$ for every solution. To model the transitivity of $\sim_G$, we add further constraints. Consider the vertices u, v, and w. If $m_{\{u,w\}} = m_{\{v,w\}} = 1$, then $m_{\{u,v\}}$ has to be 1 as well. This is equivalent to the constraint $m_{\{u,w\}} + m_{\{v,w\}} - m_{\{u,v\}} \leq 1$. The ILP is given by

$$\begin{aligned}
\min \quad & \sum_{\{u,v\} \in \binom{V}{2}} e_{\{u,v\}}, \\
\text{s.t.} \quad & m_{\{u,v\}} \leq e_{\{u,w\}} + e_{\{v,w\}} && \forall \{u, v\} \in \binom{V}{2}, \forall w \in V, && \text{(a)} \\
& m_{\{u,v\}} \leq 2 - e_{\{u,w\}} - e_{\{v,w\}} && \forall \{u, v\} \in \binom{V}{2}, \forall w \in V, && \text{(b)} \\
& 1 \leq \sum_{\{u,v\} \in \binom{X}{2}} m_{\{u,v\}} && \forall X \in \binom{V}{t+1}, && \text{(c)} \\
& 1 \geq m_{\{u,w\}} + m_{\{v,w\}} - m_{\{u,v\}} && \forall \{u, v, w\} \in \binom{V}{3}, && \text{(d)} \\
& e_{\{u,v\}} \in \{0, 1\},\ m_{\{u,v\}} \in \{0, 1\} && \forall \{u, v\} \in \binom{V}{2}.
\end{aligned}$$

Now, we analyze the number of variables and constraints. For each vertex pair, the ILP has an edit variable and a merge variable. In total, there are $2 \cdot \binom{n}{2} \in \mathcal{O}(n^2)$ variables. Two constraints are constructed for each witness of a vertex pair. A vertex pair can have up to $n-2$ witnesses. Thus, the ILP has $\binom{n}{2} \cdot (n-2) \in \mathcal{O}(n^3)$ *witness constraints* (a) and (b). There are $\mathcal{O}(n^3)$ many *transitivity constraints* (d) as well, one for each vertex triple. The largest number of constraints is taken by the *merge constraints* (c). There are $\binom{n}{t+1} \in \Theta(n^{t+1})$ such constraints. To improve the running time of this algorithm, we add the transitivity and merge constraints in a lazy way. That is, initially, the ILP contains only the witness constraints. Whenever a solution is found for the current constraints, then we construct the graph $G' = G \triangle S$ where S is the set of vertex pairs p with $e_p = 1$ in the current solution. If the neighborhood diversity of G' is larger than t, we find a vertex set X of size $t+1$ of pairwise distinct neighborhood classes in G'. Then, we add a merge constraint for X to the ILP together with the transitivity constraints for all vertex triples of $\binom{X}{3}$. Finally, a current solution may create a new witness w for some vertex pair $\{u, v\}$, for example by editing the edge $\{u, w\}$ and not the edge $\{v, w\}$. To model that this new witness either needs to be resolved by an additional edit or that the current solution must be changed, we add the constraint $m_{\{u,v\}} \leq (1 - e_{\{u,w\}}) + e_{\{v,w\}}$ in this case.

4.3 Heuristics

We now present two greedy heuristic algorithms and a local search approach.

Block-Framework. Let $G = (V, E)$ be the input graph. All heuristics maintain some partition $\mathcal{B} = \{B_0, B_1, \ldots, B_b\}$ of V during the computation, each set of this partition is called a *block*. In each step, a greedy heuristic computes a new partition with a decreased number of blocks. This is repeated until the partition consists of t blocks. To choose the next partition, we need to compute the cost of a partition as follows. For each block, we compute the minimal cost for putting all vertices of this block into the same neighborhood class by considering two aspects: First, a block has to be a clique or independent set in the resulting graph G'. Second, two blocks have to be adjacent or non-adjacent in G'. We define the cost functions ω_1 and ω_2 that compute the minimal cost for both aspects. The cost functions are defined as

$$\omega_1(B_i) := \min(\binom{|B_i|}{2} - |E(B_i)|, |E(B_i)|),$$
$$\omega_2(B_i, B_j) := \min(|B_i| \cdot |B_j| - |E(B_i, B_j)|, |E(B_i, B_j)|).$$

Then, the cost of a partition is the sum of the cost of each block and of each pair of blocks:

$$\omega(\mathcal{B}) := \sum_{B_i \in \mathcal{B}} \omega_1(B_i) + \sum_{\{B_i, B_j\} \in \binom{\mathcal{B}}{2}} \omega_2(B_i, B_j).$$

Merge-Heuristic. The initial partition $\mathcal{B}$ is the neighborhood partition of G. In each step, the *Merge-Heuristic* searches the best partition that can be obtained from the current one by merging two blocks. Algorithm 2 shows the pseudocode of the Merge-Heuristic, herein the function $\text{merge}_{\mathcal{B}}(B_i, B_j)$ returns the partition where the two blocks B_i and B_j of $\mathcal{B}$ are merged, that is, $\text{merge}_{\mathcal{B}}(B_i, B_j) := (\mathcal{B} \setminus \{B_i, B_j\}) \cup \{B_i \cup B_j\}$. This process is repeated until the partition consists of t blocks and the cost ω of the final partition is returned.

Algorithm 2 Method `Merge-Heuristic`.

Input Instance $I = (G, t)$
Output upper bound for the cost of a solution of I
$\mathcal{B} \leftarrow$ neighborhood partition of G
while $|\mathcal{B}| > t$ **do**
    find $B_i \in \mathcal{B}$ and $B_j \in \mathcal{B}$ with $i \neq j$ such that $\omega(\text{merge}_{\mathcal{B}}(B_i, B_j))$ is minimal
    $\mathcal{B} \leftarrow \text{merge}_{\mathcal{B}}(B_i, B_j)$
return $\omega(\mathcal{B})$

Now we discuss the running time. We use a matrix where the number of edges between two blocks are stored. The matrix for the initial partition can be computed in $\mathcal{O}(m + |\mathcal{B}|^2)$ time. In each iteration of the while-loop the algorithm computes the cost increase for each block pair. In total, we compute the cost increase for $\sum_{i=t}^{|\mathcal{B}|} \binom{i}{2} \in \mathcal{O}(|\mathcal{B}|^3)$ merges. Since several summands of $\omega(\mathcal{B})$ are unaffected by a merge, we only have to update costs for pairs that contain the merged block. Since the merged block is involved in $\mathcal{O}(|\mathcal{B}|)$ pairs, we can compute the cost increase of a merge and update the matrix after performing a merge in $\mathcal{O}(|\mathcal{B}|)$ time. Therefore, the total running time of the while loop is $\mathcal{O}(|\mathcal{B}|^4)$. Note that the cost of the current partition can be stored in a variable and can be updated according to the cost increase of a merge. Overall, the Merge-Heuristic runs in $\mathcal{O}(|\mathcal{B}|^4 + m)$ time.

Split-Heuristic. This heuristic also starts with the neighborhood partition and decreases the number of blocks by greedily choosing some block which *splits* and whose vertices are then added to other blocks. To estimate the cost increase incurred by splitting a block, we define a function $\tau_{\mathcal{B}}$ which describes the cost increase when a vertex is added to a block. For example, let $\mathcal{B}$ be the current block partition and we want to compute the cost increase for a vertex $v \in B_v$ by putting v in B_i. For a naive approach, let $\mathcal{B}' := \mathcal{B} \setminus \{B_v\}$ be the partition without B_v and let $\mathcal{B}'' := (\mathcal{B}' \setminus \{B_i\}) \cup \{B_i \cup \{v\}\}$ be the partition without B_v where B_i is augmented by v. The cost difference is computed by $\omega(\mathcal{B}'') - \omega(\mathcal{B}')$. Due to canceling of terms where the addition does not affect ω_1 or ω_2, we can simplify the term to

$$\tau_{\mathcal{B}'}(B_i, v) := \omega_1(B_i \cup \{v\}) - \omega_1(B_i) + \sum_{B_i \neq B_j \in \mathcal{B}'} \omega_2(B_i \cup \{v\}, B_j) - \omega_2(B_i, B_j).$$

The minimal cost increase for a vertex $v \in B_v$ is computed by $\min_{B_i \in \mathcal{B}'} \tau_{\mathcal{B}'}(B_i, v)$.

To determine the block which should be split, the algorithm searches a block B^* such that the sum of the minimum cost increase over all vertices of B^* is minimal.

During the split process, the chosen block B^* will first be removed from the partition. Then, each vertex of B^* will be put sequentially in another block such that the cost increase is minimal. After the distribution of the vertices, one step is complete. The Split-Heuristic is shown in Algorithm 3.

Note that the order of the vertices during the splitting process can affect the distribution. Furthermore, the precomputed cost for such a block B^* of the block partition $\mathcal{B}$ is not always the actual cost after the split. This is due to the fact that the cost increase for each vertex v is computed with the partition $\mathcal{B} \setminus \{B^*\}$. The other vertices of $B^* \setminus \{v\}$ are not considered in $\tau_{\mathcal{B}\setminus\{B^*\}}(B_i, v)$ for some $B_i \in \mathcal{B} \setminus \{B^*\}$. Note that the actual cost of a split is never smaller than the estimation.

Now we discuss the running time. As in the Merge-Heuristic, we use the adjacency matrix and an additional matrix for the number of edges from each vertex to each block. With these matrices, we can compute $\tau_{\mathcal{B}}$ in $\mathcal{O}(|\mathcal{B}|)$ time. Therefore, we need $\mathcal{O}(|\mathcal{B}|^2)$ time to find the

Algorithm 3 Method `Split-Heuristic`.

1: **Input** Instance $I = (G, t)$
2: **Output** upper bound for the cost of a solution of I
3: $\mathcal{B} \leftarrow$ neighborhood partition of G
4: **while** $|\mathcal{B}| > t$ **do**
5: $\quad B_v \leftarrow \operatorname{argmin}_{B \in \mathcal{B}}(\sum_{v \in B} \min_{B' \in \mathcal{B} \setminus B} \tau_{\mathcal{B} \setminus B}(B', v))$
6: $\quad \mathcal{B} \leftarrow \mathcal{B} \setminus B_v$
7: $\quad$ **for all** $v \in B_v$ **do**
8: $\quad\quad B^* \leftarrow \operatorname{argmin}_{B' \in \mathcal{B}} \tau_{\mathcal{B} \setminus \{B'\} \cup \{B' \cup \{v\}\}}(B', v)$
9: $\quad\quad \mathcal{B} \leftarrow \mathcal{B} \setminus \{B^*\} \cup \{B^* \cup \{v\}\}$
10: **return** $\omega(\mathcal{B})$

block with the minimal cost increase for one vertex. In Line 5 the algorithm iterates over every block and computes its cost increase. Since every vertex is considered exactly once, this takes $\mathcal{O}(n|\mathcal{B}|^2)$ time. Afterwards, the split process starts. The determination of the best block is repeated for each vertex of this block and the vertex will be moved to another block. The latter part demands an update for both matrices. This update requires $\mathcal{O}(n)$ time since only the entries of adjacent vertices and their blocks need to be updated. Therefore, Line 7 can be computed in $\mathcal{O}(n(|\mathcal{B}|^2 + n))$. Hence, for the while-loop the running time is $\mathcal{O}(|\mathcal{B}|(n|\mathcal{B}|^2 + n(|\mathcal{B}|^2 + n))) = \mathcal{O}(|\mathcal{B}|^3 n + |\mathcal{B}|n^2)$.

Local Search. Our local search algorithm receives the graph and a block partition with t blocks as input. The algorithm tries to improve the solution by small changes until a locally optimal solution is found. The following three kinds of changes are considered. The first kind of change allows to move a vertex from one block and to another block. The second kind of change exchanges the blocks of two vertices. The third kind of change removes all vertices of the same neighborhood class from one block and puts these vertices to another block. Each of these changes is applied only if it reduces the total cost. The first two kinds of changes are the same changes local search approach TEA by Batagelj et al. [1]. We use our local search to improve the partitions obtained by the greedy heuristics.

5 Experimental Evaluation

In this section, we compare our algorithms with each other and with an adapted ILP-formulation of Dabkowski et al. [8] which is also described in Section 5.1. Furthermore, we discuss the quality of our heuristics. Finally, we analyze the structure of an optimal Blockmodeling solution with $t = 4$ for Zachary's Karate Club graph [26].

5.1 Matrix-Based ILP

In this section, we recall the ILP of Dabkowski et al. [8]. This ILP calculates for a specific blockmatrix B how well the input graph $G = (V, E)$ fits B. Naively, to find the optimal blockmatrix we need to solve an integer linear program for every quadratic matrix with at most t columns. Fortunately, it is possible to exploit isomorphism properties to reduce the number of matrices to consider.

Now we describe the formulation in detail. For each vertex pair $\{i,j\} \in \binom{V}{2}$, the constant $s_{i,j}$ equals 1 if $\{i,j\} \in E$ and 0, otherwise. Furthermore, the constant $b_{p,\ell}$ corresponds to the entry in the pth row and ℓth column of B. The binary variable $x_{i,p}$ indicates whether vertex i is in block p. Obviously, every vertex should be assigned to exactly one block and each block should have at least one vertex, otherwise another blockmatrix fits this graph better. This formulation models the error for the adjacency of two vertices i and j that are assigned to the blocks p and ℓ, respectively, (that is, $x_{i,p} = x_{j,\ell} = 1$) using the term $b_{p,\ell} + s_{i,j} - 2b_{p,\ell}s_{i,j}$. Altogether, the ILP is given by:

$$\begin{aligned}
\min \quad & \sum_{p\in\{1,\dots,t\}} \sum_{\ell\in\{1,\dots,t\}} \sum_{i\in V} \sum_{j\in V\setminus\{i\}} x_{i,p}x_{j,\ell}(b_{p,\ell} + s_{i,j} - 2b_{p,\ell}s_{i,j}) && \\
\text{s.t.} \quad & \sum_{p\in\{1,\dots,t\}} x_{i,p} = 1 && \forall i \in V, \\
& \sum_{i\in\{1,\dots,n\}} x_{i,p} \geq 1 && \forall p \leq t, \\
& x_{i,p} \in \{0,1\} && \forall i \in V, \forall p \leq t.
\end{aligned}$$

Since the objective function is not linear, they introduced new variable $y_{i,j,p,\ell}$ to replace $x_{i,p}x_{j,\ell}$ for every $1 \leq p,\ell \leq t$, $i \in V$, and $j \in V \setminus \{i\}$ and added several constraints ensuring $y_{i,j,p,\ell} = x_{i,p}x_{j,\ell}$ [8].

Since Blockmodeling is defined only on undirected graphs and the ILP of Dabkowski et al. [8] is designed to work with directed graphs, we adapt their formulation slightly by omitting some variables that are redundant in the case of undirected graphs. This improves the running time and thus allows for a more fair comparison to our methods.

Let *MB-ILP* be the algorithm that searches a best blockmatrix for a given graph G and an integer t using the above ILP. MB-ILP applies the adapted ILP formulation for symmetric block matrices of increasing size and stops when all symmetric block matrices with up to t columns have been considered.

MB-ILP can be improved by adding an upper bound. Then, MB-ILP has to find a solution where the objective function is smaller than the upper bound. We use the minimum of the heuristic upper bound (as described in Section 5.2) and the best solution of any previously solved blockmatrix as the upper bound.

5.2 Running Time

The algorithms[2] are implemented in Java with OpenJDK 14.0.1. To solve the ILPs we used the Gurobi Optimizer in version 10.0.0. Each experiment was run on a single thread of an Intel(R) Xeon(R) Silver 4116 CPU 2.10GHz machine with 128GB RAM under Debian GNU/Linux 11 operating system.

For the experiments we used social networks obtained from KONECT [16]. For each graph of Table 1, we constructed four instances, one for each $t \in \{2,3,4,5\}$ giving a total of 48 instances. Each algorithm had a time limit of 3 hours for each instance. The three graphs whose names include "pos" were obtained from directed and weighted graphs. In the appendix, we describe in detail how this was done.

All algorithms used the best solution of 10 runs of the Merge-Heuristic with local search improvement and 10 runs of the Split-Heuristic with local search improvement as initial upper bound. Table 1 shows for each graph G, for which values of t at least one of our algorithms

[2] `https://git.uni-jena.de/algo-engineering/blockmodeling`

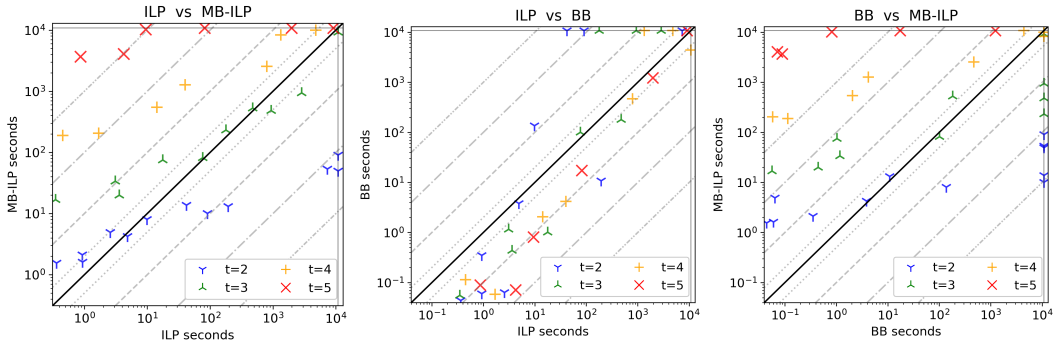


Figure 2 Each data point represents an instance, the color indicates the value of t. The gray lines mark a relative running time difference of 2, 10, 100, and 1000, respectively. The thin black lines mark the time limit. Points on these lines are not solved by the corresponding algorithm.

Table 1 Overview of the graphs used for the running time evaluation. Numbers in the middle columns denote the discovery of the optimal solution size for $t \in \{2, 3, 4, 5\}$ for each graph. Numbers in the last columns denote the maximal t the corresponding algorithm solved within the time limit.

Name	n	m	k_2	k_3	k_4	k_5	BB	ILP	MB-ILP
Highland Tribes pos	16	29	18	12	9	8	5	5	5
Kangaroo	17	91	20	16	9	8	5	5	5
Crisis in a Cloister pos	18	26	22	19	16	13	5	5	5
Taro Exchange	22	39	35	31	27	24	5	5	4
Zebra	27	111	32	26	20	17	5	5	4
Dutch College pos	32	87	61	51	45	41	2	5	4
Karate Club	34	78	65	57	44	–	–	4	4
Chesapeake Bay	39	170	118	104	–	–	–	3	3
HIV	40	41	38	33	30	–	4	3	3
Dolphins	62	159	146	–	–	–	–	–	2
Train Bomb	64	243	185	152	–	–	–	2	3
Iceland	75	114	105	–	–	–	–	–	2

found an optimal solution, and for each algorithm this table includes the maximal t-value for which the algorithm solved the instance (G, t). The branch-and-bound algorithm (BB) solved 24 instances and both ILPs solved 32. Every algorithm solved at least one instance which could not be solved by any other algorithm. For example, BB solved (HIV, $t = 4$), our ILP solved (Dutch College, 5) and MB-ILP solved (Iceland, 2). Altogether, for 36 out of the 48 instances at least one of these algorithms found an optimal solution.

Figure 2 shows a running time comparison between BB, our ILP, and MB-ILP and Figure 5 shows how many instances each algorithm solved depending on the time. While BB solved many instances faster than both ILPs, the ILPs solved more instances in total. As expected, the running time of MB-ILP depends very strongly on t; compared to our algorithms, MB-ILP solved more instances with $t \in \{2, 3\}$ and less with $t \in \{4, 5\}$, even if the optimal solution size is small. For larger t, our ILP is faster than MB-ILP by a factor up to more than 1000. In our opinion, the worse running time for $t = 2$ is not critical, since this special case can be solved much faster with tailored algorithms.

5.3 Heuristics

Now we evaluate the quality of Merge-Heuristic and Split-Heuristic with subsequent local search compared to optimal solutions. In particular, we examine the dependence of the relative error of the best found heuristic solution to the number of repetitions of these

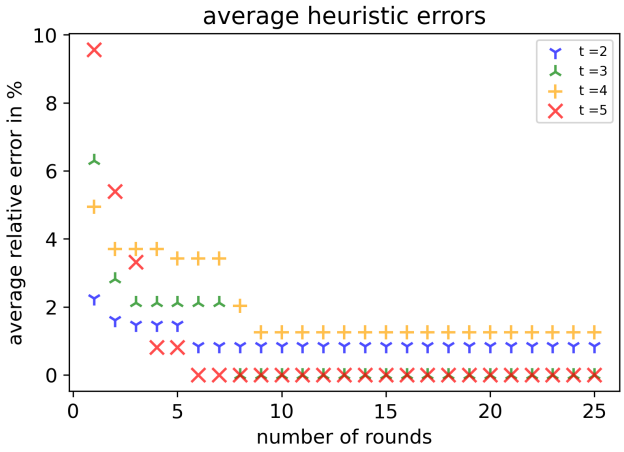


Figure 3 Relative difference between optimal solutions and the heuristics after r repetitions. One repetition includes a run of Merge-Heuristic and Split-Heuristic, each with subsequent local search.

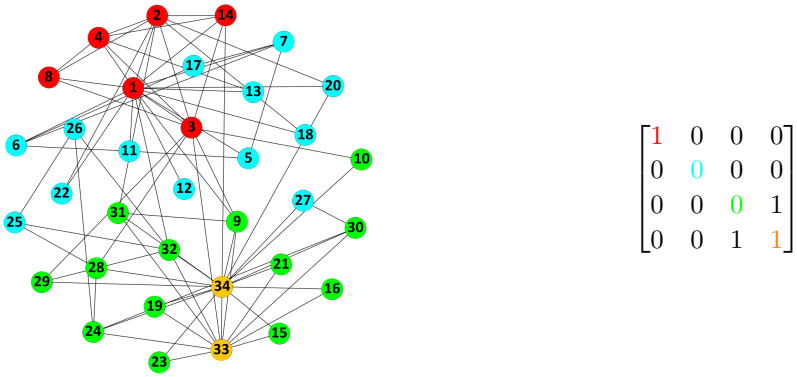


Figure 4 The Karate Club graph and its blockmatrix of an optimal clustering for $t = 4$; the vertex colors indicate the block in the optimal computed blockmodel.

heuristics. The maximum number of repetitions was set to 25. The optimal solution size was found for 32 of 36 instances. In the four cases where the optimal solution was not found, the relative difference was 1.37 % (Dolphins, $t = 2$), 7.89 % (HIV, $t = 2$), 10% (HIV, $t = 4$), and 0.95 % (Iceland, $t = 2$). The relative errors, averaged over all instances, are shown in Figure 3. After 9 rounds no progress is made. In preliminary experiments we could not deduce that one heuristic is in general better than the other. Therefore, we used both for the computation of the upper bound.

5.4 Karate Club

We now discuss one optimal blockmodel of the well-known Karate Club [26] for $t = 4$. This graph consists of members of a karate club. An edge represents an interaction between two actors outside the karate club. The club split into two new clubs, headed by the members 1 and 34, respectively. The clustering and the blockmatrix of an optimal solution for $t = 4$ is shown in Figure 4. The four blocks of the computed blockmodel correspond approximately to the split into two clubs: The red and cyan blocks correspond to the new club headed by vertex 1, the orange and green blocks correspond to the new club headed by 34. The orange

and red blocks (which contain the new leaders 1 and 34, respectively) are the two clique blocks and can be interpreted as the two cores of the corresponding new clubs. The orange and green blocks form together a dense split graph, the red and cyan clusters form together a sparse split graph. Hence, the blockmodel recovers a core/periphery structure within the clubs. In these terms, the vertices which are in a block that does not correspond to their new club (9, 25, 26, and 27) are periphery vertices. In other words, the cores are correctly separated. Finally, note that there is another optimal solution that adds 27 to the green cluster. This solution deletes fewer edges; it could be advantageous to favor such optimal solutions, to avoid producing too many isolated vertices.

6 Conclusion

We presented a new formulation of exact exploratory blockmodeling in undirected networks as graph-modification problem and developed exact and heuristic algorithms for this approach. Our algorithms are competitive with previous state-of-the-art exact approaches in terms of running times. More crucially, our algorithms enable solving the blockmodeling problem for larger values of t for which previous approaches based on enumerating all candidate blockmatrices become prohibitively slow.

There are many opportunities for future work. First, further improvements of our algorithms are desirable and likely possible. The most promising direction here seems to be the development of better lower bounds. Further extending the range of tractable instances could then allow for an empirical study of the quality of optimal solutions, beyond the anecdotical evidence discussed here. Moreover, an adaption to directed networks seems promising. Finally, one could extend our formulation also to blockmodeling with more complicated objective functions such as the one of Reichardt and White [21].

References

1 Vladimir Batagelj, Anuška Ferligoj, and Patrick Doreian. Direct and indirect methods for structural equivalence. *Social Networks*, 14(1):63–90, 1992. Special Issue on Blockmodels.

2 Stephen P. Borgatti and Martin G. Everett. Models of core/periphery structures. *Social Networks*, 21(4):375–395, 2000.

3 Stephen P. Borgatti, Martin G. Everett, and Jeffrey C. Johnson. *Analyzing social networks*. SAGE, 2013.

4 Sharon Bruckner, Falk Hüffner, and Christian Komusiewicz. A graph modification approach for finding core-periphery structures in protein interaction networks. *Algorithms for Molecular Biology*, 10:16, 2015.

5 Michael J. Brusco and Douglas Steinley. Integer programs for one- and two-mode blockmodeling based on prespecified image matrices for structural and regular equivalence. *Journal of Mathematical Psychology*, 53(6):577–585, 2009.

6 Jeffrey Chan, Wei Liu, Andrey Kan, Christopher Leckie, James Bailey, and Kotagiri Ramamohanarao. Discovering latent blockmodels in sparse and noisy graphs using non-negative matrix factorisation. In *22nd ACM International Conference on Information and Knowledge Management (CIKM '13)*, pages 811–816. ACM, 2013.

7 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms*. Springer, 2015.

8 Matthew Dabkowski, Neng Fan, and Ronald L. Breiger. Exploratory blockmodeling for one-mode, unsigned, deterministic networks using integer programming and structural equivalence. *Social Networks*, 47:93–106, 2016.

9 Peter Damaschke and Olof Mogren. Editing simple graphs. *Journal of Graph Algorithms and Applications*, 18(4):557–576, 2014.

10 Rodney G. Downey and Michael R. Fellows. *Fundamentals of Parameterized Complexity*. Texts in Computer Science. Springer, 2013.

11 Michael R. Fellows, Jiong Guo, Christian Komusiewicz, Rolf Niedermeier, and Johannes Uhlmann. Graph-based data clustering with overlaps. *Discrete Optimization*, 8(1):2–17, 2011.

12 Michel Habib and Christophe Paul. A survey of the algorithmic aspects of modular decomposition. *Computer Science Review*, 4(1):41–59, 2010.

13 Falk Hüffner, Christian Komusiewicz, and André Nichterlein. Editing graphs into few cliques: Complexity, approximation, and kernelization schemes. In *Proceedings of the 14th International Symposium on Algorithms and Data Structures (WADS '15)*, volume 9214 of *Lecture Notes in Computer Science*, pages 410–421. Springer, 2015.

14 Alan Jessop. Blockmodels with maximum concentration. *European Journal of Operational Research*, 148(1):56–64, 2003.

15 Ivan Kovác, Ivana Selecéniová, and Monika Steinová. On the clique editing problem. In *Proceedings of the 39th International Symposium on Mathematical Foundations of Computer Science (MFCS '14)*, volume 8635 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2014.

16 Jérôme Kunegis. KONECT: the koblenz network collection. In *Proceedings of the 22nd International World Wide Web Conference (WWW '13)*, pages 1343–1350. International World Wide Web Conferences Steering Committee / ACM, 2013.

17 Michael Lampis. Algorithmic meta-theorems for restrictions of treewidth. *Algorithmica*, 64(1):19–37, 2012.

18 Ross M. McConnell and Jeremy P. Spinrad. Modular decomposition and transitive orientation. *Discrete Mathematics*, 201(1-3):189–241, 1999.

19 Les G. Proll. ILP approaches to the blockmodel problem. *European Journal of Operational Research*, 177(2):840–850, 2007.

20 K. E. Read. Cultures of the central highlands, new guinea. *Southwestern Journal of Anthropology*, 10(1):1–43, 1954.

21 Jörg Reichardt and Douglas R White. Role models for complex networks. *The European Physical Journal B*, 60(2):217–224, 2007.

22 Samuel F Sampson. *Crisis in a cloister*. PhD thesis, Ph. D. Thesis. Cornell University, Ithaca, 1969.

23 Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. *Discrete Applied Mathematics*, 144(1-2):173–182, 2004.

24 Gerhard G Van de Bunt, Marijtje AJ Van Duijn, and Tom AB Snijders. Friendship networks through time: An actor-oriented dynamic statistical network model. *Computational & Mathematical Organization Theory*, 5(2):167–192, 1999.

25 Stanley Wasserman and Katherine Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

26 Wayne W. Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33(4):452–473, 1977.

A Supplementary Material

A.1 Deferred Proofs

A.1.1 Proof of Lemma 4

Proof. Let $I := (G = (V, E), t, k)$ be a yes-instance of Blockmodeling. Assume towards a contradiction, that for each optimal solution S, there is a at least one neighborhood class C of G with $|C| > 2k$ where some vertices of C is affected by S.

Let S be an optimal solution for I and let C be a neighborhood class in G with $|C| > 2k$ such that at least one vertex of C is affected. We can assume without loss of generality, that C is an independent set in G, as otherwise, we can simply consider the complement

graph $G' = (V, \binom{V}{2} \setminus E)$. Since S is a solution for I, $|S| \leq k$. Hence, there is at least one vertex $c \in C$ which is unaffected by S, that is, c has the same neighbors in G and in $G_{\text{res}} = (V, E \triangle S)$.

Let $P \subseteq V$ be the neighborhood class of c in G_{res}. We show that P is an independent set in G_{res}. Note that this is the case if $P = \{c\}$. If P contains a second vertex of C, then P is an independent set in G_{res} since c is unaffected. Otherwise, c is the only vertex of C in P and there is at least one vertex p in $P \setminus \{c\}$. Hence, if P is a clique, $\{c, p\} \in E$ which implies that for each $c' \in C \setminus \{c\}$, $\{c', p\} \in E$ since c is unaffected and C is a neighborhood class in G. Since P is a neighborhood class in G_{res}, C is an independent set in G, and c is unaffected, for each $c' \in C \setminus \{c\}$, some edge $\{p, v\}$ is in S. This would imply that $|S| > k$. Since by assumption $|S| \leq k$, P is an independent set in G_{res}. Note that this also implies that there is no edge between any vertex of C and any vertex of $P \setminus C$ in E since P is an independent set in G_{res} and c is unaffected.

Let $S' := \{\{u, v\} \in S \mid u \notin C, v \notin C\}$. We show that S' is a solution for I. Note that each vertex of C is unaffected by S'. Since by the above, P is an independent set in G_{res} with $N(p) = N(c)$ for each $p \in P \setminus C$, and there is no edge between any vertex of $P \setminus C$ and C in E, $P \cup C$ is a neighborhood class in $G_{\text{alt}} := (V, E \triangle S')$. It remains to show that $G_{\text{alt}} - (P \cup C)$ has neighborhood diversity at most $t - 1$.

Let u and v be two distinct vertices of $V \setminus (C \cup P)$ of the same neighborhood class in G_{res}. We show that u and v are in the same neighborhood class in G_{alt}. Since u and v are in the same neighborhood class in G_{res} and $|S| \leq k$, both u and v are either adjacent to each vertex of C in G or non-adjacent to each vertex of C in G. Hence, both u and v are either adjacent to each vertex of $C \cup P$ in G_{alt} or non-adjacent to each vertex of $C \cup P$ in G_{alt}. By the fact that S' contains all vertex pairs of S that do not contain any vertex of C, u and v are in the same neighborhood class in G_{alt}.

Since S is a solution for I, $G_{\text{res}} \setminus P$ has neighborhood diversity at most $t - 1$. By the above, this implies that $G_{\text{alt}} \setminus (C \cup P)$ has neighborhood diversity at most $t - 1$. Hence S' is a solution for I. Since S' is a proper subset of S, S is not an optimal solution for I, a contradiction. ◀

A.1.2 Proof of Theorem 6

Proof. For $t = 2$ the NP-hardness is shown by Lemma 5. For $t > 2$, we reduce from Blockmodeling. Let $I = (G = (V, E), k, 2)$ be an instance of Blockmodeling. Moreover, let $G' = (V', E')$ be the graph obtained from G by adding for each $i \in \{1, \ldots, t-2\}$ a clique C_i of size $2k+1$ to G such that each vertex of C_i has no neighbors outside of C_i in G'. Finally, we set $I' = (G', k, t)$ and show that I is a yes-instance of Blockmodeling if and only if I' is a yes-instance of Blockmodeling.

Note that for each $i \in \{1, \ldots, t-2\}$, the clique C_i is a neighborhood class in G'. Let $S \subseteq \binom{V'}{2}$ be an optimal solution for I'. Hence, due to Lemma 4, every vertex of $V' \setminus V$ is unaffected by S. That is, every optimal solution for I' is a subset of $\binom{V}{2}$ and no vertex of V is in any neighborhood class some vertex of $V' \setminus V$.

Let $S \subseteq \binom{V}{2}$ be a set of size at most k. Hence, for each $i \in \{1, \ldots, t-2\}$, the clique C_i is a neighborhood class in $G'_{\text{res}} := (V', E' \triangle S)$. As a consequence G'_{res} has neighborhood diversity at most t if and only if $G_{\text{res}} = (V, E \triangle S)$ has neighborhood diversity at most 2. Hence, I is a yes-instance of Blockmodeling if and only if I' is a yes-instance of Blockmodeling. ◀

Table 2 Overview of the graphs and their links.

Graph	Link
Highland Tribes	http://www.konect.cc/networks/ucidata-gama/
Kangaroos	http://www.konect.cc/networks/moreno_kangaroo/
Cloister	http://www.konect.cc/networks/moreno_sampson/
Taro Exchange	http://www.konect.cc/networks/moreno_taro/
Zebra	http://www.konect.cc/networks/moreno_zebra/
Dutch College	http://www.konect.cc/networks/moreno_vdb/
Karate Club	http://www.konect.cc/networks/ucidata-zachary/
Chesapeake Bay	http://www.konect.cc/networks/dimacs10-chesapeake/
HIV	http://www.konect.cc/networks/hiv/
Dolphins	http://www.konect.cc/networks/dolphins/
Train Bomb	http://www.konect.cc/networks/moreno_train/
Iceland	http://www.konect.cc/networks/iceland/

A.1.3 Proof of Lemma 7

Proof. Let C_u and C_v be the neighborhood classes of u and v respectively, let $\mathcal{C}$ be the neighborhood partition of G and let $\mathcal{C}_{\text{rest}} := \mathcal{C} \setminus \{C_u, C_v\} = \{C_1, \ldots, C_{t-2}\}$. For two vertices x and y of $V \setminus \{u, v\}$ the relation $\sim_G$ is the same as $\sim_{G'}$ because x and y have the same neighbors in G as in G'. Therefore, the vertices $\{v_1, \ldots, v_{t-2}\}$ with $v_i \in C_i$ are in $t-2$ distinct neighborhood classes in G' since $v_i \not\sim_G v_j$ for $1 \leq i \neq j \leq t-2$. This implies that G' has at least $t-2$ neighborhood classes.

On the other hand, both u and v might be in single-sized neighborhood classes in G'. Hence, G' has neighborhood diversity at most $t+2$. ◀

A.2 Data acquisition

We describe how we obtained undirected and unweighted graphs out of the given data sets for the three "pos"-graphs. The weight of an edge in these data sets represents how much one actor likes/dislikes the other. The undirected graph Highland Tribes [20] consists of edges of weights 1 and -1. We removed the edges with a negative weight. The remaining two graphs are Crisis in a Cloister [22] and Dutch College [24]. Both are directed and have weights $\{-1, 0, 1\}$ and $\{-1, 0, 1, 2, 3\}$ respectively. In the undirected graphs, there is an undirected

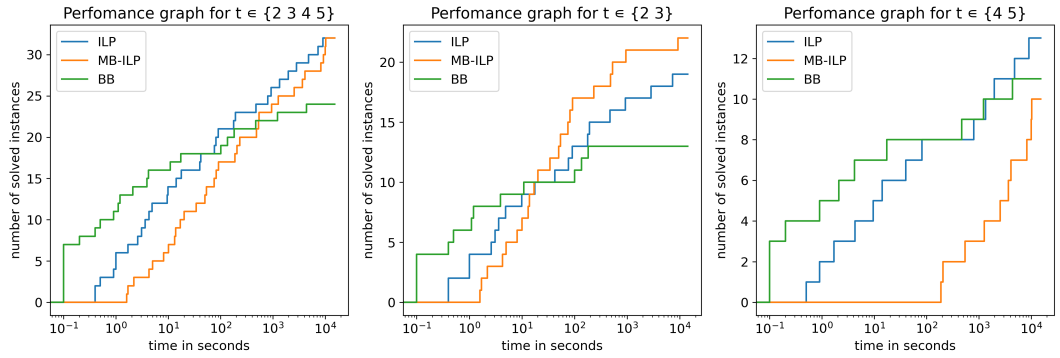


Figure 5 These graphs indicates how many instances with selected values for t each algorithm solved depending on the time.

edge $\{u, v\}$ if and only if there is an edge form u to v and an edge from v to u, the sum of weights of these edges is at least 2, and none of the weights of these edges is negative. The Dutch College data consists of 7 graphs among the same actors. For our experiment we used the graph (timestamp: 924217200) with the most edges obtained by the above-mentioned method.