Fachbereich Mathematik & Informatik

AG Algorithmik

Masterarbeit zum Thema

Finding Optimal Blockmodels

vorgelegt von

Alexander Bille

am 02.05.2022

Betreuer: Prof. Dr. Christian Komusiewicz
M.Sc. Niels Grüttemeier
M.Sc. Nils Morawietz

Hiermit versichere ich, Alexander Bille, dass ich die vorliegende Arbeit mit dem Titel “Finding Optimal Blockmodels” selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.
Die Masterarbeit wurde in der jetzigen oder einer ähnlichen Form noch bei keiner anderen Hochschule eingereicht und hat noch keinen sonstigen Prüfungszwecken gedient.

Marburg, den 02.05.2022 _____

Alexander Bille

Danksagung

Danke

Abstract

In graph clustering, the goal is to partition the vertex set into clusters such that vertices in the same cluster are similar. One approach for defining similarity is to consider vertices as similar if they are structurally equivalent, that is, they have the same neighborhood. In other words, structurally equivalent vertices are in the same neighborhood class.

In this work, we present the new problem Block Modeling. It asks for removal and addition of at most k edges to an input graph such that the resulting graph has at most t neighborhood classes. This means, that the resulting graph is clustered with respect to structural equivalence with at most t clusters. In contrast to previous edge modification based clustering formulations, this problem does not specify the adjacency between or within clusters.

We show NP-Hardness and fixed-parameter tractability for parameter $k + t$. Furthermore, we present several methods to obtain a clustering like branch-and-bound algorithms, heuristics and ILP-formulations. To speed up the algorithms, we analyze properties of solution structure for Block Modeling, and provide lower bounds and reduction rules. Finally, we compare our algorithms with each other on real-world social networks.

Contents

1 Introduction

An important task in the area of unsupervised learning is to compute a clustering of a set of elements only based on the properties of the elements [13]. A clustering is a partition of a set where elements of the same cluster should be similar and elements of different clusters should be dissimilar. Two common approaches for a clustering are k-Means [26] and DBSCAN [11]. These approaches assign each element to at most one cluster and are vector-based. In vector-based approaches, the similarity of two elements is computed by a distance function based on the properties of the elements.
For the clustering of graphs, there are multiple interpretations for the similarity between vertices. One interpretation is that the adjacency between two vertices is equal to their similarity: Two vertices are adjacent to each other if and only if they are similar. Then, in a good clustering, every vertex is adjacent to most of the vertices of its cluster and to few vertices outside its cluster. The Cluster Editing [4] problem demands a transformation into disjoint cliques. Then, every clique is a cluster based on this approach of similarity. Furthermore, the p-Cluster Editing [31] problem demands such a clustering with exactly p cliques.
Another approach to define similarity is via structural equivalence [3, 24]. Two vertices are structurally equivalent if they have the same adjacency to each other vertex. STRUCTURE [7] and CONCOR [6] are well-known heuristics that cluster a graph by structural equivalence. Both algorithms use a vector-based clustering where a vertex is represented as its column of the adjacency matrix. Such methods are called indirect approaches [3] since a solution is not computed with the graph structure.
Note that Cluster Editing can be seen as a transformation into structural equivalent clusters since every two vertices of each clique have the same adjacencies to each other vertex in the resulting graph. This problem specifies a further constraint to the form of the clusters which is the adjacency within a cluster and the non-adjacency between clusters.
In contrast to previous works, we present Block Modeling which puts no constraints to the form of the clustering except the number of clusters. Block Modeling asks for addition and removal of few edges such that the resulting graph can be clustered into at most t clusters where every two vertices of a cluster are structurally equivalent, that is, the resulting graph can be partitioned into at most t sets where every two vertices of a set have the same adjacency to each other vertex. This work focuses mainly on the

development of algorithms for Block Modeling. Still, several theoretical results are shown as well.

1.1 Related Work

A closely related edge modification problem to Block Modeling is H-Bag Editing [9] which resulting graph is also a clustering with respect to structural equivalence. This problem specifies the form of the clusters with H. Every vertex of H represents a cluster in the resulting graph. If a vertex is adjacent to another vertex, then the represented clusters have to be adjacent as well. In addition, every cluster has to be a clique.
H-Bag Editing is subexponential fixed-parameter tractable with respect to the edit parameter k [9]. More precisely, H-Bag Editing can be solved in $2^{\mathcal{O}(\sqrt{k}\cdot\log k)} \cdot n^{\mathcal{O}(1)}$ time [9]. Note that the size of H are constants in this estimation. For several fixed graphs H, this problem is known to be NP-complete [9]. For example, if H is a graph with p isolated vertices, then H-Bag Editing and p-Cluster Editing are equivalent, and p-Cluster Editing is NP-complete for $p \geq 2$ [31].
There are problems that demand a clustering by structural equivalence with more specification to the form. For example, Dense Split Graph Editing [5] asks for a transformation into a universal clique and an independent set. Note that H-Bag Editing does not generalize Dense Split Graph Editing since Dense Split Graph Editing demands an independent set in the resulting graph.
The algorithm STRUCTURE [7] clusters the graph hierarchically by structural equivalence. It computes a dissimilarity matrix for the vertices where the distance is computed on the columns of the adjacency matrix. Different choices for the distance function are possible. The dissimilarity matrix defines a hierarchical clustering using standard methods. To obtain a non-hierarchical clustering, one may set a similarity threshold α or specify the the number of clusters.
Batagelj et al. [3] provided a local search approach to cluster a graph by structural equivalence. For a partition of the vertex set, a cost function is defined that counts the minimal edits such that the graph, applied with these edits, can be clustered with this partition. The algorithm starts with a random initial partition and tries to improve it locally until no further improvement can be made. The allowed improvements are a movement and a swap. The movement puts a vertex into an other cluster if the clustering

improves. The swap changes the clusters of two vertices if the clustering improves. This process is done multiple times and the 10 best clusterings are returned as the output.
The algorithm CONCOR [6] also follows the approach of a heuristic, indirect and hierarchical clustering. Unlike STRUCURE, CONCOR clusters divisive. This means, that all elements are in one cluster at the start. The cluster is split up into two clusters recursively until the desired number of clusters is reached. In the following, we describe a split process of a cluster. From the adjacency matrix M_0 a correlation matrix M_1 is computed where the (i, j)-th entry is the product moment correlation of the columns i and j of M_0. This process is repeated with the last correlation matrix M_ℓ until $M_\ell = M_{\ell+1}$. As argued by Breiger et al. [6], the resulting matrix can almost always be permuted into a matrix of the form $\left[\begin{smallmatrix} 1 & -1 \\ -1 & 1 \end{smallmatrix}\right]$ where a number β represents a submatrix with β as entries. The cluster is split according to this permutation.
To the best of our knowledge, we provide the first approaches to cluster optimally by structural equivalence.

1.2 Our Results

In Section 2, we provide the basic notation that is used in this work.
In Section 3, we observe some properties of Block Modeling and show several characteristics of the solution structure. Furthermore, we provide a polynomial kernel for parameter $k + t$ for Block Modeling and show NP-hardness.
In Section 4, we introduces several algorithms to solve Block Modeling. First, we provide an algorithm that solve Block Modeling in $\mathcal{O}((t^2 + t)^k \cdot n^3)$ time. Then, a vertex-weighted problem Vertex-Weighted Block Modeling is defined that generalizes Block Modeling. For this problem, an algorithm is also given. Afterwards, two greedy heuristics and one local search approach are provided. In Section 4.3, two ILP-formulations are given.
In Section 5, several ideas are listed to improve the running-time of the branch-and-bound algorithms of Section 4.1. These improvements include the storage of additional information, and the update of the graph and of the additional information by applying an edit. This is done to reduce redundant computations. Moreover, two lower bound algorithms, reduction rules and a branching rule are also provided.

In Section 6, we evaluate our algorithms of Section 4 on 21 real-world social networks.

2 Preliminaries

In this section, we define the notation that is used throughout this work.

2.1 Graph Notation

Let $\mathbb{N}$ denote the natural numbers including 0 and let $\mathbb{N}_+ := \mathbb{N} \setminus \{0\}$. For some set S and an integer x we define $\binom{S}{x} := \{T \subseteq S \mid |T| = x\}$. The symmetrical difference for two sets S and T is defined as $S \triangle T := (S \cup T) \setminus (S \cap T)$. 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 $i \neq j$. Let $E(S,T) := \{\{s,t\} \mid s \in S, t \in T\}$ be the set of all vertex pair combinations for two sets S and T.
All *graphs* in this thesis are undirected and simple. A 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 $m := |E|$. The *complement graph* $\overline{G}$ of a graph $G = (V, E)$ is defined as $\overline{G} := (V, E \triangle \binom{V}{2})$. We denote the deletion of a vertex v by $G - v := (V \setminus \{v\}, E \setminus \{\{u, v\} \in E \mid u \in V\})$.
The *neighborhood* of a vertex v is $N_G(v) := \{u \mid \{u, v\} \in E\}$. We call the vertices of $N_G(v)$ the *neighbors* of v. If $\{u, v\} \in E$ we say u and v are *adjacent* or u is adjacent to v in G. If a vertex v is adjacent to every other vertex of V, we call v *universal*. A vertex with no neighbors is called *isolated*. Two disjoint sets, $S_1 \subseteq V$, $S_2 \subseteq V$, are *adjacent* if and only if $E(S_1, S_2) \subseteq E$. We say S_1 and S_2 are *non-adjacent* if $E(S_1, S_2) \cap E = \emptyset$. In the other case we say S_1 and S_2 are *partially adjacent*. 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 u and v have the same adjacency to every vertex of W. We say that a vertex w has the *same adjacencies* to a set W in graph G_1 as in G_2 if every vertex pair of $E(\{w\}, W)$ is either adjacent in G_1 and G_2 or is non-adjacent in G_1 and G_2.
A subset $V' \subseteq V$ is called a *clique* in G if and only if $\binom{V'}{2} \subseteq E$. Also, a subset $V' \subseteq V$ is called an *independent set* in G if and only if $\binom{V'}{2} \cap E = \emptyset$. We omit the reference "in G" or the subscript, if G is clear from context.

2.2 Neighborhood Classes

For the definition of *neighborhood classes* [23] we introduce a relation over vertices:

Definition 2.1. *Let $G = (V, E)$ be a graph and let $u, v \in V$. We denote $u \sim_G v$ if and only if $N(u) \setminus \{v\} = N(v) \setminus \{u\}$.*

The relation $\sim_G$ is an equivalence relation [23].

Definition 2.2. *Let $G = (V, E)$ be a graph. The* neighborhood partition *of G is the partition $\mathcal{W} = \{W_1, W_2, \ldots, W_w\}$ of V such that for all $u, v \in W_i$ $u \sim_G v$, for all $u \in W_i$ and $v \in W_j$ with $i \neq j$ $u \nsim_G v$, and every W_i is not empty.*
The neighborhood graph *is a graph $(\mathcal{W}, E')$ with*

$$\{W_i, W_j\} \in E' \Leftrightarrow \forall u \in W_i, v \in W_j : \{u, v\} \in E.$$

We say G has a neighborhood diversity *of w.*

Note that the neighborhood partition is unique for each graph. Each set of this partition is called *neighborhood class*. Note that a graph has exactly one neighborhood partition. Every neighborhood class is either a clique or an independent set. A neighborhood class C is called *positive* when C is a clique. Analogue, a neighborhood class C is called *negative* if C is an independent set. Note that a neighborhood class with only one element inside is positive and negative at the same time.
A vertex w is a *witness* of a vertex pair $\{u, v\}$ if and only if w is exactly to one vertex of $\{u, v\}$ adjacent. Otherwise, w is a *non-witness*. Let $\text{wit}(p)$ be the set of witnesses of a vertex pair p.

2.3 Graph Problems

A decision problem is a language $L \subseteq \{0, 1\}^*$. Every instance $I \in \{0, 1\}^*$ is called yes-instance if $I \in L$ and no-instance otherwise.
This work studies some graph decision problems that demands a *transformation* of the input graph such that the resulting graph satisfy a property Π. A transformation is done by a set of *edits* $E' \subseteq \binom{V}{2}$ and returns a new graph $G_{\text{res}} = (V, E \triangle E')$. A set of edits is called *solution* for a problem p with property Π when through the transformation the resulting graph satisfy Π.
All problems of this work have a parameter in the input that limits the size of the solution. Usually it is named with k. A solution E_{sol} for instance I is called *optimal* if every instance with the same input as I but with $E_{\text{sol}} - 1$ as the limit parameter is a no-instance.

A vertex is called *affected* if and only if at least one edit in a solution contains this vertex. All other vertices are called *unaffected*. A set of edits is also called *edit set* and a set of vertices is called *vertex set*. To simplify the notation of a transformation to a graph G we overload the operator $\triangle$ with $G \triangle E' := (V, E \triangle E')$.

3 Theoretical Aspects of Block Modeling

In this section, we define Block Modeling and proof some properties of this problem.

Block Modeling
Input: A graph $G = (V, E)$ and integers k and t.
Question: Can G be modified by at most k edits such that the resulting graph has a neighborhood diversity of at most t?

Block Modeling asks for a reduction of the neighborhood diversity and does not force any relation between the resulting neighborhood classes.

Observation 3.1. *An instance I of* Block Modeling *with $t = 1$ is trivial. Either all vertices define a clique or define an independent set in the resulting graph. Therefore, all missing edges have to be added or all edges need to be removed. The instance I is a yes-instance if and only if $k \geq \min(\binom{|E|}{2} - m, m)$.*

Observation 3.2. *An instance (G, t, k) of* Block Modeling *with $t \geq d$ where d is the neighborhood diversity of G is a yes-instance.*

Observation 3.3. *In an optimal solution of* Block Modeling*, vertices of the same neighborhood class can be in separate neighborhood classes in the resulting graph.*

An example for Observation 3.3 is given by Figure 1.

Lemma 3.1. *The neighborhood partition of a graph is also the neighborhood partition of its complementary graph.*

Proof. Let G be an arbitrary graph. We prove this lemma by showing that the relations $\sim_G$ and $\sim_{\overline{G}}$ are equivalent. Since a vertex x changes the adjacency to every other vertex in the complement graph, we have $N_{\overline{G}}(x) = N_G(x) \triangle (V \setminus \{x\})$. Consider any two vertices u and v of V. The

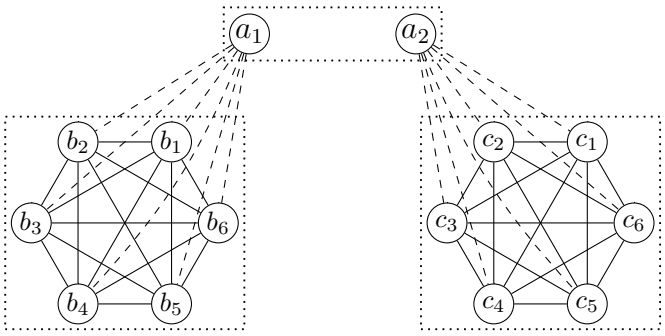


Figure 1: This graph with $t = 2$ is an example where the vertices of one neighborhood class are in distinct neighborhood classes in the resulting graph of an optimal solution. Circles represent vertices and lines are edges. The dotted squares mark the neighborhood classes of the graph. The dashed lines mark the edits of the solution. In this example, the edits are only additions.

equivalence is given by:

$$
\begin{aligned}
& u \sim_{\overline{G}} v \\
\Leftrightarrow \quad & N_{\overline{G}}(u) \setminus \{v\} = N_{\overline{G}}(v) \setminus \{u\} \\
\Leftrightarrow \quad & (N_G(u) \triangle (V \setminus \{u\})) \setminus \{v\} = (N_G(v) \triangle (V \setminus \{v\})) \setminus \{u\} \\
\Leftrightarrow \quad & (N_G(u) \setminus \{v\}) \triangle ((V \setminus \{u\}) \setminus \{v\}) = (N_G(v) \setminus \{u\}) \triangle ((V \setminus \{v\}) \setminus \{u\}) \\
\Leftrightarrow \quad & (N_G(u) \setminus \{v\}) \triangle (V \setminus \{u, v\}) = (N_G(v) \setminus \{u\}) \triangle (V \setminus \{u, v\}) \\
\Leftrightarrow \quad & N_G(u) \setminus \{v\} = N_G(v) \setminus \{u\} \\
\Leftrightarrow \quad & u \sim_G v
\end{aligned}
$$

□

Observation 3.4. *Let $G = (V, E)$ be a graph and let E' be an edit set. The graph $\overline{G} \triangle E'$ is the same as $\overline{G \triangle E'}$.*
This follows immediately by the definitions, and the associativity and the commutativity of the symmetrical difference:

$$
\begin{aligned}
\overline{G} \triangle E' &= (V, E \triangle \binom{V}{2}) \triangle E' = (V, (E \triangle \binom{V}{2}) \triangle E') \\
&= (V, (E \triangle E') \triangle \binom{V}{2}) = \overline{(V, (E \triangle E'))} = \overline{G \triangle E'}
\end{aligned}
$$

Lemma 3.2. *A solution of* Block Modeling *for an instance* (G, k, t) *is a solution of* Block Modeling *for* $(\overline{G}, k, t)$.

Proof. Let E_{sol} be the solution of Block Modeling for G. We prove this lemma by showing that $\overline{G} \triangle E_{\text{sol}}$ has a neighborhood diversity of at most t. Since E_{sol} is a solution, $G \triangle E_{\text{sol}}$ has a neighborhood diversity of at most t. Due to Lemma 3.1 the graph $\overline{G \triangle E_{\text{sol}}}$ has the same neighborhood partition as $G \triangle E_{\text{sol}}$ and hence the same neighborhood diversity. Because of Observation 3.4 the graph $\overline{G} \triangle E_{\text{sol}}$ is the same as $\overline{G \triangle E_{\text{sol}}}$ and therefore $\overline{G} \triangle E_{\text{sol}}$ has a neighborhood diversity of at most t. □

Due to Lemma 3.2 we can assume that the graph of an Block Modeling instance has at most $\binom{m}{2}/2$ edges.

Lemma 3.3. *If there is an optimal solution of* Block Modeling *where two vertices of a positive neighborhood class* P *in the input graph* G *are in distinct and positive neighborhood classes in the resulting graph* G_{res}*, then there is an optimal solution where in the resulting graph* G_{alt} *all vertices of* P *of positive neighborhood classes in* G_{res} *are in the same positive neighborhood class in* G_{alt}.

Proof. Let $G = (V, E)$ be the input graph and let E_{sol} be the optimal solution. Thus, the resulting graph is $G_{\text{res}} = (V, E_{\text{res}} = E \triangle E_{\text{sol}})$. Let P_p be the subset of P where every vertex is in a positive neighborhood class in G_{res}.
First, we show that if two vertices of P_p are non-adjacent in G_{res} the solution E_{sol} is not optimal. Let u and v be two vertices of P_p with $\{u, v\} \notin E_{\text{res}}$. Let C_u be the neighborhood class of u in G_{res} and let C_v be the neighborhood class of v in G_{res}. Let $P_u := C_u \cap P$ be the vertices of P which are in C_u in G_{res}. Let $P_v := C_v \cap P$ be the vertices of P which are in C_v in G_{res}. Note that P_u and P_v are not empty. We count the necessary edits for the vertices of $P_u \cup P_v$ such that they are in P_u or in P_v respectively. First, consider the edits of $E(P_u \cup P_v, V \setminus (P_u \cup P_v))$. A vertex $w \in P_u$ changed $\text{cost}_{C_u} := |E(\{w\}, V \setminus (P_u \cup P_v)) \cap E_{\text{sol}}|$ many adjacencies towards $V \setminus (P_u \cup P_v)$ such that w is in C_u. Observe that cost_{C_u} is the same for each vertex of P and not depends on P_u and P_v since C_u and P are positive. Analogously, a vertex $w \in P_v$ changed $\text{cost}_{C_v} := |E(\{w\}, V \setminus (P_u \cup P_v)) \cap E_{\text{sol}}|$ many adjacencies towards $V \setminus (P_u \cup P_v)$ such that w is in C_v.
Now, we count the edges $E(P_u, P_v)$ between P_u and P_v since they are edited by E_{sol} because u and v are non-adjacent in G_{res}. Altogether, the cost is:

$f(P_u, P_v) := |P_u| \cdot \text{cost}_{C_u} + |P_v| \cdot \text{cost}_{C_v} + |P_u| \cdot |P_v|$. The minimum of f is reached when P_u or P_v equals the empty set. Without loss of generality let $\text{cost}_{C_u} \leq \text{cost}_{C_v}$. We construct a solution where every vertex of $P_u \cup P_v$ is in C_u. This solution has less edits than E_{sol}. Thus, it contradicts the assumption that E_{sol} is optimal, and $|P_u| \geq 1$ and $|P_v| \geq 1$.
Second, we assume that any two vertices of P_p are adjacent. We provide a solution size $|E_{\text{sol}}|$ where all vertices of P_p are in the same neighborhood class. Consider two vertices u and v of P_p of distinct neighborhood classes in G_{res}. Let $E_u := \{\{u, w\} \in E_{\text{sol}} \mid w \in V\}$ be the edits on u and let $E_v := \{\{v, w\} \in E_{\text{sol}} \mid w \in V\}$ be the edits on v. Note that $E_u \cap E_v = \emptyset$. Let $E_{\text{rest}} := E_{\text{sol}} \backslash (E_u \cup E_v)$ be the edits that do not contain u or v. We constructed a solution with size $|E_{\text{alt}}| + 2 \cdot |E_u|$ by undoing the edits on v and applying the analog changes of u to v such that the vertex v is in the same neighborhood class as u. Analogously, we can construct a solution of size $|E_{\text{rest}}| + 2 \cdot |E_v|$ where u is in the same neighborhood class of v. Next, we show that $|E_u| = |E_v|$. Because E_{sol} is optimal, we have

$$|E_{\text{sol}}| = |E_{\text{rest}}| + |E_u| + |E_v| \leq |E_{\text{rest}}| + 2 \cdot |E_u|$$

and

$$|E_{\text{rest}}| + |E_u| + |E_v| \leq |E_{\text{rest}}| + 2 \cdot |E_v|.$$

We simplify the inequations to $|E_v| \leq |E_u|$ and $|E_u| \leq |E_v|$. It immediately follows $|E_u| = |E_v|$. Thus, both alternative solutions are optimal as well. For an arbitrary but fixed vertex u, we can construct sequentially a solution where every vertex of P_p will be in the same neighborhood class as u in G_{alt}. □

The Lemma 3.3 holds for negative neighborhoods as well due to Lemma 3.2.

Lemma 3.4. *Let E_{sol} be a solution of* Block Modeling *for a graph G. Let $G_{\text{res}} = G \triangle E_{\text{sol}}$ be resulting graph.*

1. *If there is a neighborhood class C in G where at least one vertex is unaffected by E_{sol} and some other vertices of C are affected by E_{sol}, then E_{sol} is not optimal.*
2. *If there is a neighborhood class C_{res} in G_{res} which is a real subset of a neighborhood class in G, then E_{sol} is not optimal.*

Proof. Both cases are proven by contradiction. Assume that E_{sol} is optimal. We split the lemma into the two cases.

Case 1: One vertex c in C is unaffected and some other vertices of C are affected by E_{sol}.

The vertex c has the same neighbors in G and in G_{res}. We can construct a solution from E_{sol} such that all vertices of C are in the same neighborhood class as c in the resulting graph. We achieve this by "undoing" the edits of the vertices of C. Let $E_{\text{undo}} := \{\{u, c'\} \in E_{\text{sol}} \mid c' \in C\}$ be the edits of E_{sol} which contain at least one vertex of C. Note that E_{undo} is not empty since at least one vertex of C is affected. The constructed solution is defined by $E_{\text{alt}} := E_{\text{sol}} \setminus E_{\text{undo}}$.

We prove that E_{alt} is a solution for G. Let $G_{\text{alt}} := G \triangle E_{\text{alt}}$. Obviously, E_{alt} has a size of at most k since $|E_{\text{alt}}|$ cannot be greater than $|E_{\text{sol}}|$, and $|E_{\text{sol}}| \leq k$. Moreover, the size is smaller than k because E_{undo} is not empty. It remains to show that G_{alt} has a neighborhood diversity of at most t. Let P be the neighborhood class of c in G_{res}.

Obviously, all vertices of C are in the same neighborhood class in G_{alt} and are unaffected. Now, we show that the vertices of $P \setminus C$ are in the same neighborhood class as c in G_{alt} as well. Let p be a vertex of $P \setminus C$. Since c is unaffected by E_{sol} the vertex p has the same adjacency to C as to the vertex c in G_{res}. Because c is unaffected and $\{c, p\}$ cannot be in E_{sol}, every vertex pair of $E(\{p\}, C)$ is not in E_{sol}, otherwise $p \not\sim_{G_{\text{res}}} c$. Therefore, the neighborhood of p is the same in G_{res} and in G_{alt} since E_{alt} – in comparison with E_{sol} – only undoes the edits on the vertices of C. This implies that p is in the same neighborhood class as c in G_{alt}. This argument applies to every vertex of $P \setminus C$. Thus, the vertices $P \cup C$ are in the same neighborhood class in G_{alt}.

Next, we prove that the vertices of $V \setminus (C \cup P)$ are in at most $t-1$ neighborhood classes in G_{alt}. Consider two vertices u, v from the same neighborhood class in G_{res} with $u \notin (C \cup P)$ and $v \notin (C \cup P)$. We show that u and v are in the same neighborhood in G_{alt}. Note that in general, two vertices that have a different adjacency to C in G have at least $k+1$ witnesses and therefore cannot be in the same neighborhood class in any solution. Thus, u and v have the same adjacency to C in the input graph. Since every vertex of C is unaffected in G_{alt}, u and v have the same adjacency to C in G_{alt} as well.

Since E_{sol} is a solution, and u and v are in the same neighborhood class in G_{res}, we know $N_{G_{\text{res}}}(u) \setminus \{v\} = N_{G_{\text{res}}}(v) \setminus \{u\}$. Therefore, and because u and v have the same adjacency to C in G, the neighborhoods of u and v

in G_{alt} are either $N_{G_{\text{res}}}(u) \cup C$ and $N_{G_{\text{res}}}(v) \cup C$, or $N_{G_{\text{res}}}(u) \setminus C$ and $N_{G_{\text{res}}}(v) \setminus C$. In both cases $u \sim_{G_{\text{alt}}} v$ holds because u and v are not in C. Thus, two vertices of the same neighborhood class in G_{res} are in the same neighborhood class in G_{alt}. Since the vertices of $V \setminus (C \cup P)$ are in at most $t-1$ distinct neighborhood classes in G_{res} they are in at most $t-1$ distinct neighborhood classes in G_{alt}.
Altogether, the neighborhood diversity of G_{alt} is at most t and hence E_{alt} is a solution with less than k edits. This contradicts the assumption that E_{sol} is an optimal solution.
Case 2: There is a neighborhood class C' in the resulting graph G_{res} that is a subset of C**.** We assume that every vertex of C is affected by E_{sol}, otherwise the condition of the first case are fulfilled. This case resolves like Case 1, that is, E_{alt} is a solution for this case.
The proof proceeds analogous with some mentions. As in Case 1 the vertices of C are in the same neighborhood class in G_{alt}. Since $P = C'$ there is no subcase for a vertex $p \in P \setminus C$ because $P \setminus C = \emptyset$. The proof that the vertices of $V \setminus (C \cup P) = V \setminus C$ are in at most $t-1$ distinct neighborhood classes is the same. □

Lemma 3.5. *In every optimal solution of* Block Modeling *every vertex of every neighborhood class with size larger than k in the input graph is unaffected.*

Proof. We prove this property by contradiction. Assume that E_{sol} is a solution where at least one vertex of a neighborhood class C in G with $|C| > k$ is affected. Let G be the input graph and let $G_{\text{res}} = G \triangle E_{\text{sol}}$ be the resulting graph . Due to Lemma 3.4 every vertex of C is affected and there is no neighborhood class in G_{res} that is a real subset of C.
We show $|E_{\text{sol}}| > k$ and thus E_{sol} is no solution by counting at least one edit for every vertex of C. Throughout the proof, we call an edit $\{x, y\}$ *external* if $x \in C$ and $y \notin C$. If x and y are in C, then this edit is called *internal*. We say a vertex x has *external edits* if an external edit exists with x in it. We define has *internal edits* analogously.
The proof is done by a construction of a mapping of disjoint subsets $\{C_1, \ldots, C_q\}$ of C to disjoint subsets $\{E_1, \ldots, E_q\}$ of E_{sol} with $|C_i| \leq |E_i|$ for all $1 \leq i \leq q$. Let $\mathcal{D} = \{D_1, \ldots, D_q\}$ be the collection of neighborhood classes containing at least one vertex of C. We define the subsets of C by $C_i := D_i \cap C$ for all $1 \leq i \leq q$. Since no neighborhood class is a subset of C there is at least one vertex $d_i \in D_i \setminus C_i$ for all $1 \leq i \leq q$. We consider each neighborhood

class in $\mathcal{D}$ and define partially the mapping. The neighborhood classes are distinguished by four cases (from (a) to (d)). First, we define the partial mapping for each $D \in \mathcal{D}$ that applies to Case (a). Then, every neighborhood class is considered that applies to Case (b). Note that a neighborhood class D' of Case (b) does not apply to Case (a). Otherwise, D' would be considered earlier in Case (a). After that, the partial mapping for each neighborhood class that applies to Case (c). At last, we define the partial mapping for neighborhood classes that apply to Case (d).
We name D_i the considered neighborhood class. Let $\ell = |C_i|$ be the number of vertices of C in the current considered neighborhood class D_i. For each case a subcase is illustrated in Figure 2 for a better understanding.

(a) One vertex c of C_i has at least one external edit**.**

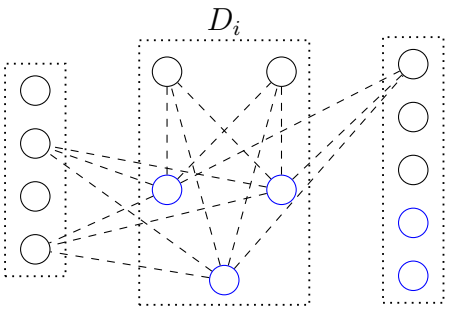
Note that for any two vertices u and v of C of the same neighborhood class, $\{u, w\} \in E_{\text{sol}}$ if and only if $\{v, w\} \in E_{\text{sol}}$ for some vertex $w \in V \setminus \{u, v\}$. Let $y \notin C$ be a vertex of an external edit $\{c, y\}$. Then, for every vertex c' of C_i the vertex pair $\{c', y\}$ is an external edit. Since the set $E(\{y\}, C_i)$ is a subset of E_{sol}, we define $C_i \mapsto E(\{y\}, C_i)$ for the mapping. Two map images of this category are disjoint since for two vertex sets C_i and C_j with $i \neq j$ the subsets $E(\{y\}, C_i)$ and $E(\{x\}, C_i)$ are disjoint with $x \notin C, y \notin C$.

(b) There is a vertex $c_j \notin C_i$ such that $E(\{c_j\}, C_i) \subseteq E_{\text{sol}}$ and $\ell \geq 2$**.**

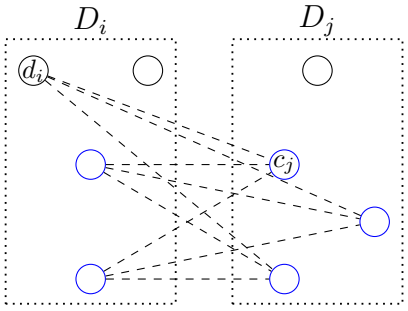
The vertex c_j is in C, otherwise the vertices of C_i have external edits. Let D_j be the neighborhood class of c_j in G_{res} and let C_j be the vertices of C in D_j. Let $E_{\text{between}} := E(C_i, C_j)$ be the vertex pairs containing one vertex of C_i and one vertex of C_j.

First, we consider the case where $\binom{C_i}{2} \cap E_{\text{sol}} = \emptyset$. The vertices of C_i changed their adjacency towards C_j but the vertex pairs $\binom{C_i}{2}$ are no edits in E_{sol}. Moreover, the vertex d_i has the same adjacencies to C in G_{res} as in the input graph G, otherwise the vertices of C_i have external edits. To have a solution, E_{sol} includes either $E(\{d_i\}, C_i)$ or $E(\{d_i\}, C_j)$. Because the vertices of C_i have no external edits, E_{sol} cannot include the edits $E(\{d_i\}, C_i)$. This implies that C_j fulfills Case (a) since the vertices of C_j have external edits and its image is already defined. Therefore, we can define $C_i \mapsto E_{\text{between}}$ since $|E_{\text{between}}| = \ell \cdot |C_j| \geq \ell$.

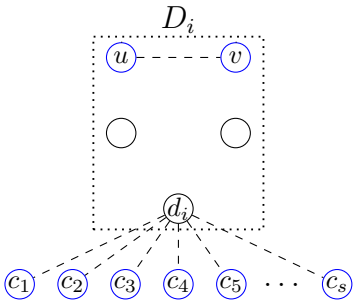
Second, we consider the case where $\binom{C_i}{2} \subseteq E_{\text{sol}}$. With the edits $\binom{C_i}{2} \cup$



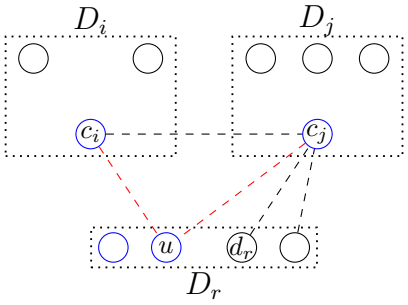
(a) The blue vertices are in C_i. The marked edits are all external edits for vertices of C_i.



(b) Case $\binom{C_i}{2} \cap E_{\text{sol}} = \emptyset$. Because the vertices of C_j have an external edit, the edits $E(C_i, C_j)$ can be counted for the vertices of C_i.



(c) Case $\ell = 2$. The neighborhood classes of the $c_1, \ldots, c_s$ are not illustrated. The size of C is $s + 2$.



(d) Case $|C_j| = 1$. Exactly one of the red marked edits has to be in E_{sol}. In both cases the edit $\{c_j, d_r\}$ has to be in E_{sol} as well.

Figure 2: Visualizations of cases of the proof of Lemma 3.5. Each circle is a vertex. Vertices in a dotted rectangle represent a neighborhood class in G_{res}. Blue vertices are in C and black ones not. Dashed lines represent an edit which is in E_{sol}. All other information like edges are not illustrated because there are unnecessary for the proof.

E_{between} we have enough edits for two images since $|\binom{C_i}{2} \cup E_{\text{between}}| = \binom{\ell}{2} + \ell \cdot |C_j| \geq \ell + |C_j|$. Therefore, we can split $\binom{C_i}{2} \cup E_{\text{between}}$ into two parts S_i and S_j with $|C_i| \leq |S_i|$ and $|C_j| \leq |S_j|$, and define $C_i \mapsto S_i$ and $C_j \mapsto S_j$. Since we defined the image of C_j, we can skip the neighborhood class D_j in the consideration.

The edits E_{between} do not overlap with any subsets of the images of Case (a) because all edits of E_{between} are internal. At last, we show that two map images of this case do not overlap. Consider another image of the neighborhood classes D_r and D_s with $r \neq i$ and $r \neq s$. We show that $A = \binom{C_i}{2} \cup E_{\text{between}} = \binom{C_i}{2} \cup E(C_i, C_j)$ do not intersect with $B = \binom{C_r}{2} \cup E(C_r, C_s)$. Since $r \neq i$ no edit of $\binom{C_i}{2}$ is in B and no edit of $\binom{C_r}{2}$ is in A. The intersection $E(C_i, C_j) \cap E(C_r, C_s)$ is the empty set because $C_i \cap C_r = \emptyset$.

(c) The vertex pairs $\binom{C_i}{2}$ are edits of E_{sol} and $\ell \geq 2$.

First, we consider $\ell \geq 3$. Since $\binom{\ell}{2} \geq \ell$, we define $C_i \mapsto \binom{C_i}{2}$. This image does not overlap with any other images of the previous cases, and this image does not overlap with another image of this case.

For $\ell = 2$ and $C_i = \{u, v\}$ we have to consider d_i. Recall $d_i \in D_i \setminus C_i$. We show that this situation cannot appear. Note that only the edit $\{u, v\}$ is in E_{sol}. No other edit contains u or v, especially the vertex pairs $\{u, d_i\}$ and $\{v, d_i\}$. Since d_i is in D_i and has the same adjacency to C_i and $C \setminus C_i$ in the input graph the edits $E(\{p\}, C \setminus C_i)$ have to be in E_{sol} because $\{u, v\} \in E_{\text{sol}}$. For example, without loss of generality let C be a clique in the input graph. Since $\{u, v\}$ is contained in E_{sol}, the neighborhood class D_i is an independent set. Because u has no external edit, d_i is non-adjacent to C in the input graph. The vertex d_i is adjacent to $C \setminus C_i$ in the resulting graph. Therefore, the edits $E(\{d_i\}, C \setminus C_i)$ are a subset of E_{sol} in order that d_i and u are in the same neighborhood class.

Since $|E(\{d_i\}, C \setminus C_i)| \geq k-1$ we know that at least $|\{u, v\}| + |E(\{d_i\}, C \setminus C_i)| \geq k$ edits are in E_{sol}. Depending on the size of C there are either too many edits such that E_{sol} is a no solution or $|C| = k + 1$ and therefore $E_{\text{sol}} = \{u, v\} \cup E(\{d_i\}, C \setminus C_i)$. With this solution E_{sol} the set $C \setminus C_{\text{sub}}$ is a neighborhood class and a subset of C. Due to Lemma 3.4 this is a contradiction to an optimal solution. Thus, Case (c) with $\ell = 2$ cannot appear.

(d) The size of C_i is 1**.**

The set $C_i = \{c_i\}$ includes only c_i. Since c_i is affected and has no external edits there is a c_j such that the edit $\{c_i, c_j\}$ is in E_{sol} with $c_j \in C$. Let D_j be the neighborhood class of c_j and let C_j be the vertices of C in D_j. If $|C_j| \geq 2$, then a mapping is already done for C_j. We can define $C_i \mapsto \{\{c_i, c_j\}\}$.

Finally, assume $|C_j| = 1$. Because $c_i \nsim c_j$, there is a witness u of $\{c_i, c_j\}$. If $u \notin C$ then C_j has external edits and we define $C_i \mapsto \{\{c_i, c_j\}\}$ for the mapping. If $u \in C$, let D_r be the neighborhood class of u and let C_r be the vertices of C in D_r. Let d_r be some vertex of $D_r \setminus C_r$. Note that d_r is either adjacent or non-adjacent to $\{c_i, c_j\}$ in G. Since d_r is in the same neighborhood class as u in G_{res} and u is a witness of $\{c_i, c_j\}$, the vertex d_r is also a witness of $\{c_i, c_j\}$. Thus, either $\{d_r, c_i\}$ or $\{d_r, c_j\}$ has to be in E_{sol}. Since c_i has only internal edits, the edit $\{d_r, c_j\}$ is in E_{sol}. Therefore, the vertices of C_j have external edits and we define $C_i \mapsto \{c_i, c_j\}$ for the mapping. The edit $\{c_i, c_j\}$ is not contained in any subset of the image space of the mapping since D_j only can fulfill either Case (a) or Case (b). In Case (a), C_j maps to external edits. In Case (b), the other considered neighborhood class either fulfills Case (a) or a mapping for this neighborhood class is already done.

As we can count for every vertex of C at least one different edit in the mapping the size of E_{sol} is at least $k + 1$. This contradicts that E_{sol} is a solution. □

3.1 Kernelization

In this section, we provide a polynomial kernel for parameter $k+t$ for Block Modeling. Before we give the kernel, we make an observation regarding the difference of the neighborhood diversities by an application of one edit.

Lemma 3.6. *Let $G = (V, E)$ be a graph, let $e = \{u, v\}$ be a vertex pair, let w be the neighborhood diversity of G and let $G' = G \triangle \{e\}$ be the resulting graph by the transformation $\{e\}$. The neighborhood diversity of G' differs from w by at most by 2.*

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 $\mathcal{C}_{\text{rest}} := P \setminus \{C_u, C_v\} =$

$\{C_1, \ldots, C_{w-2}\}$. 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 $\{v_1, \ldots, v_{w-2}\}$ with $v_i \in C_i$ are in $w-2$ distinct neighborhood classes in G' since $v_i \not\sim_G v_j$ for $1 \leq i \neq j \leq w-2$. This implies that G' has at least $w-2$ neighborhood classes.
On the other hand, u and v could be not similar in terms of $\sim_{G'}$ such that they are in single-sized neighborhood classes in G'. This give us an upper bound of $w+2$ for the number of neighborhood classes in G'. □

Based on Lemma 3.5 and Lemma 3.6, we are now able to obtain a polynomial kernel as shown in the following theorem.

Theorem 3.1. Block Modeling *admits a kernel with* $\mathcal{O}(k^2+kt)$ *vertices.*

Proof. Due to Lemma 3.6 the neighborhood diversity can be reduced by at most 2 with a single edit. Hence, a graph with more than $2k+t$ neighborhood classes cannot be solved with k edits. Furthermore, due to Lemma 3.5 we know that there is a solution where all vertices of neighborhood classes greater than k are unaffected. Therefore, all neighborhood classes with a size greater than k can be reduced to $k+1$ vertices. This implies that a non trivial instance has at most $(2k+t) \cdot (k+1) = 2k^2 + 2k + kt + t = \mathcal{O}(k^2 + kt)$ vertices. Since the neighborhood partition can be computed in polynomial time [23] this admits a polynomial kernel. □

3.2 NP-Hardness

In this section, we show NP-Hardness for Block Modeling by reducing from Sparse Split Graph Editing [5] which is NP-hard [21].

Sparse Split Graph Editing
Input: A graph $G = (V, E)$ and an integer k.
Question: Can G be modified by at most k edits, such that V can be partitioned into a clique C and an independent set P such that the vertices of P are isolated in the resulting graph?

First, we show the NP-hardness for $t = 2$ with Lemma 3.7. Let $(G = (V, E), k)$ be an instance of Sparse Split Graph Editing. Consider the graph $G' = (V \cup K, E)$ that consists of G and an independent set K containing $k+1$ new vertices. Let $(G', k, t = 2)$ be the instance for Block Modeling.

Lemma 3.7. *There is a* Sparse Split Graph Editing *solution of* (G, k) *if and only if there is a* Block Modeling *solution of* $(G', k, 2)$*, where* G' *is constructed as described above.*

Proof. ($\Rightarrow$) : Let E_{sol} be the solution of Sparse Split Graph Editing for (G, k). We show that E_{sol} is a solution of Block Modeling for $(G', k, 2)$ as well. Obviously, the size of E_{sol} is at most k. Let $G'_{\text{res}} = G' \triangle E_{\text{sol}}$ be the resulting graph of G' applied with the edits E_{sol}. The vertices of K are unaffected by E_{sol} and therefore are isolated in G'_{res}. The transformation with E_{sol} partitions V into a clique C and an isolated set P. Obviously, the vertices of $P \cup K$ are in the same neighborhood class in G'_{res} since they are all isolated.
The vertices of C define a clique in G'_{res} and have no other neighbors. Thus, the vertex set C defines a neighborhood class in G'_{res}. This implies that G'_{res} has a neighborhood diversity of at most 2.
($\Leftarrow$) : Now, let E_{sol} be the solution of Block Modeling for $(G', k, 2)$. Due to Lemma 3.5 we construct a solution E'_{sol} from E_{sol} such that every vertex of K is unaffected. We show that E'_{sol} is a solution of Sparse Split Graph Editing for (G, k). The size of E'_{sol} is at most k. Let $G'_{\text{res}} = G' \triangle E'_{\text{sol}}$ be the resulting graph of G' applied with the edits E'_{sol}. Since the vertices of K are in the same neighborhood class, let Q_1 be the neighborhood class including K and let Q_2 be the other neighborhood class in G'_{res}. We know that Q_1 is a negative neighborhood class and every vertex of it is isolated. Therefore, the neighborhood class Q_2 has to be positive, otherwise the vertices of are isolated as well and therefore in Q_1. The existence of Q_2 do not affect the following conclusion. If there is no second neighborhood class, Q_2 can be treated as the empty set.
Altogether, through E'_{sol} the vertices V partitions into an isolated independent set $Q_1 \setminus C$ and a clique Q_2. These are the conditions for Sparse Split Graph Editing. Since only K is added to G by construction and every vertex of K is unaffected by E'_{sol}, the edits E'_{sol} fulfill the requirements for a solution of Sparse Split Graph Editing for (G, k). □

Next, we show that Block Modeling is NP-hard for $t \geq 2$.

Theorem 3.2. Block Modeling *is NP-hard for* $t \geq 2$*.*

Proof. For $t = 2$ the NP-hardness is shown by Lemma 3.7. For $t > 2$ we reduce from Sparse Split Graph Editing as well. Let $(G = (V, E), k)$ be

the instance of Sparse Split Graph Editing. Consider the instance $(G^* = (V^*, E^*), k, t)$ of Block Modeling where G^* is defined as G' but in addition with $t-2$ many cliques $\mathcal{S} = \{S_1, \ldots, S_{t-2}\}$ of size $k+1$. Each clique S_i is non-adjacent to every other vertex in $V \setminus S_i$.

First, assume that E_{sol} is a solution of Sparse Split Graph Editing for (G, k). As proven in Lemma 3.7 applying E_{sol} on graph G^* the vertices of $V \cup K$ are in at most two neighborhood classes. Every vertex of any clique of $\mathcal{S}$ is unaffected and therefore these vertices are in at most $t-2$ neighborhood classes. Hence, E_{sol} is a solution of Block Modeling for (G^*, k, t).

Second, assume that E_{sol} is now a solution of Block Modeling for (G^*, k, t). Due to Lemma 3.5 we construct the solution E^*_{sol} where every vertex of $K \cup \bigcup_{S \in \mathcal{S}} S$ is unaffected. Since the cliques of $\mathcal{S}$ are in exact $t-2$ neighborhood classes, the vertices of $V \cup K$ are in at most 2 neighborhood classes. Analogue as in the proof of Lemma 3.7, the vertices of V are partitioned into a clique and an isolated independent set. Therefore, E^*_{sol} is a solution of Sparse Split Graph Editing for (G, k). □

4 Algorithms

In this section, we present some branch-and-bound algorithms, heuristics and ILP-formulations.

4.1 Branch-and-Bound

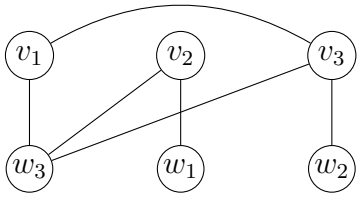
Branch-and-bound algorithms are used to solve discrete or combinatorial problems. Such an algorithm computes the solution by solving subproblems stepwise. A question of a subproblem can be whether to include an edge to the solution or not. A branch-and-bound algorithm can be viewed as a search tree where one branch represent a decision of a solution to a subproblem. Due to lower bounds or other criteria for exclusion, some branches do not need to be considered, they will be *cut off*. For more about branch-and-bound algorithms we refer to [8].

The first algorithm considers $t+1$ many vertices from distinct neighborhood classes. At least two of these vertices have to be in the same neighborhood class in the resulting graph. This gives us $\binom{t+1}{2}$ branch possibilities. But what has to be done to assure that two vertices will be in one neighborhood class? Remember, if two vertices are in one neighborhood class, they have the same adjacency to each other vertices. In other words, the pair of these two vertices has no witnesses in the resulting graph.

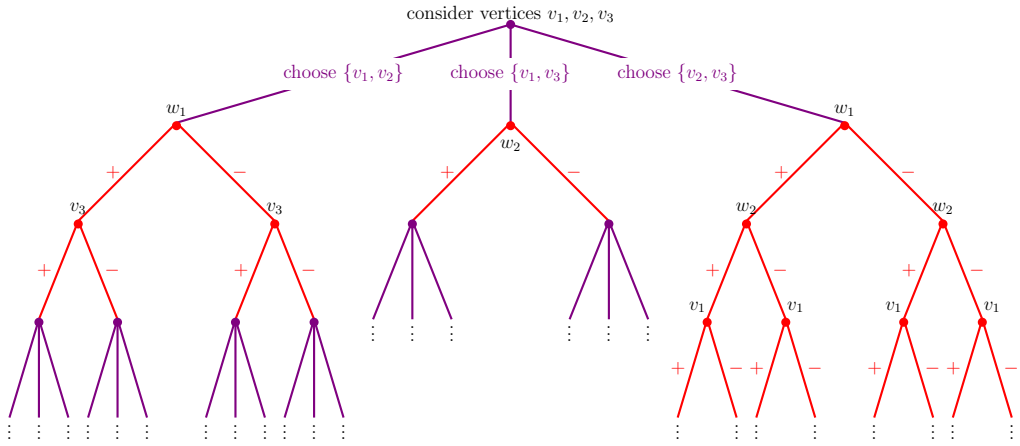
To bring u and v in the same neighborhood class, all witnesses need to be *resolved*. There are two ways to resolve a witness w of the pair $\{u, v\}$. Without loss of generality, let $\{w, u\} \in E$ and $\{w, v\} \notin E$. To resolve w for $\{u, v\}$ we have to either add the missing edge $\{w, v\}$ or delete the present edge $\{w, u\}$. This decision has to be done for each witness independently. Hence, there are 2^r different possibilities to achieve that u and v are int the same neighborhood class, where $r = \text{wit}(\{u, v\})$ is the number of witnesses of $\{u, v\}$. 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. With this, we can write a basic version of a branch-and-bound algorithm shown in Algorithm 1.

Clearly, the algorithm terminates because the vertex pair $\{u, v\}$ have at least one witness and thus a recursive call has a smaller edit budget k.

Note that instead of computing the set T it is possible to branch over the witnesses. In each node, a witness resolves either by addition or by removal. See Figure 3 for more details.



(a) Example graph $G = (V, E)$. Every circle represents a vertex of V. Lines between vertices are edges of E.



(b) Visualization of a possible branching of Algorithm 1. On violet nodes $t + 1$ vertices from distinct neighborhood classes are chosen. Violet lines indicate the choice of a vertex pair. In red nodes a random witness of the earlier chosen vertex pair is selected. The red lines portray the kind how the witness is resolved.

Figure 3: Example branching of Algorithm 1 with $t = 2$.

Algorithm 1: Method `solve`

Input: Instance $I = (G, k, t)$ of Block Modeling
Output: $True$ if and only if I is a yes-instance of Block Modeling

```
1  if k < 0 then return False;
2  if G has neighborhood diversity of at most t then return True;
3  T ← list of t + 1 vertices of distinct neighborhood classes;
4  foreach {u, v} ∈ (T choose 2) do
5  |  T ← collection of all resolve sets of {u, v};
6  |  foreach E_edit ∈ T do
7  |  |  G' ← G △ E_edit;
8  |  |  I' ← (G', k − |E_edit|, t);
9  |  |  if solve(I') then return True;
10 |  end
11 end
12 return False;
```

In the following, we estimate the running-time of Algorithm 1. The algorithm has two kinds of branching, one that chooses a vertex pair which vertices will be in the same neighborhood class in the resulting graph with $\binom{t+1}{2}$ possibilities and the other one that resolves a witness with 2 possibilities. The first one is worse because $\binom{t+1}{2} > 2$ for $t \geq 2$ (the case $t = 1$ is trivial as shown by Observation 3.1). We get the worst branching when the algorithm considers only vertex pairs that have only one witness. In this situation, the branch tree alternates between these two branch cases. Then, the branch tree has the maximum depth of $2k$ because the edit budget decreases by 1 after one witness is resolved. When combining one cycle of the alternation, the combined branch case has $\binom{t+1}{2} \cdot 2$ branches. With the combined branch case the search tree has a depth of k. Therefore, the upper bound of leafs in the branch tree is $(\binom{t+1}{2} \cdot 2)^k = (t^2 + t)^k$. The other computations depend on n, the neighborhood classes can be calculated in polynomial time [23] and the search of a witness can be done in linear time with the adjacency matrix. Altogether, Algorithm 1 runs in $\mathcal{O}((t^2 + t)^k \cdot n^3)$ time.

After resolving all witnesses of a vertex pair, Algorithm 1 can still separate these vertices afterwards if they will be resolved differently for another

vertex pair.
In order to prevent these situation, we present a vertex-weighted variation of Algorithm 1 that *merges* vertices. This variation uses a weight function $\omega : V \to \mathbb{N}_+$ and a label function $\ell : V \to \{\oplus, \ominus, \odot\}$. To merge a vertex pair $\{u, v\}$, the vertices u and v have to be in the same neighborhood class. After the merge, v is not be part of the graph in the recursive call. The weight function states how many vertices of the very first input graph are *represented* by one vertex. The initial value of the weight for each vertex is 1. The weight of u increases by $\omega(v)$ when merging u and v.
Through the deletion of v, we lose the information whether $\{u, v\}$ is an edge or not. Therefore, we make a case distinction whether $\{u, v\}$ is an edge or not in the resulting graph and label u accordingly. The label $\oplus$ of a vertex u states that u has to be in a positive neighborhood class. Analogous, the label $\ominus$ of a vertex u states that u has to be in a negative neighborhood class. The label $\odot$ makes no statement. When one vertex is labeled as $\oplus$ and another vertex is labeled with $\ominus$, these two vertices cannot be in the same neighborhood class by definition, we say they are not *mergeable*. Whether two vertices u, v are mergeable, is determined by the function

$$\text{mergeable}_\ell(u, v) := \ell(u) = \ell(v) \vee \ell(u) = \odot \vee \ell(v) = \odot.$$

The function mergeable needs to be considered in the neighborhood classes as well. Therefore, we define the modified relation $\sim_G^\ell$ as

$$u \sim_G^\ell v := u \sim_G v \wedge \text{mergeable}_\ell(u, v).$$

At last, since vertices may represent more than themselves the cost of an edit differs with Algorithm 1. For this problem, the editing cost of a vertex pair $\{x, y\}$ is defined as $\omega(x) \cdot \omega(y)$.
Altogether, we stated the necessary definitions to introduce:

Vertex-Weighted Block Modeling

Input: A graph $G = (V, E)$, an edit budget k, a positive integer t, a weight function $\omega : V \to \mathbb{N}_+$ and a label function $\ell : V \to \{\oplus, \ominus, \odot\}$.

Question: Can G be modified by a cost budget of k such that the resulting graph has at most t neighborhood classes with respect to $\sim_G^\ell$ where an edit $\{u, v\}$ costs $\omega(u) \cdot \omega(v)$?

Algorithm 2: Method `solve`

Input: Instance $I = (G, k, t, \omega, \ell)$ of Vertex-Weighted Block Modeling

Output: $True$ if and only if I is yes-instance of Vertex-Weighted Block Modeling

```
1  if k < 0 then return False;
2  if G has at most t neighborhood classes with respect to ∼_G^ℓ then
3  |  return True
4  T ← list of t + 1 vertices of distinct neighborhood classes;
5  foreach {u, v} ∈ (T choose 2) do
6  |  T ← collection of all resolve sets of {u, v};
7  |  foreach E_edit ∈ T do
8  |  |  G' ← (G △ E_edit) − v;
9  |  |  r ←total edit cost of E_edit;
10 |  |  if ℓ(u) ≠ ⊖ and ℓ(v) ≠ ⊖ then
11 |  |  |  r_⊕ ← r;
12 |  |  |  if {u, v} ∉ E then r_⊕ ← r+ cost of {u, v} ;
13 |  |  |  I^⊕ ← (G', k − r_⊕, t, ω_{u,v}, ℓ^⊕_{u,v});
14 |  |  |  if solve(I^⊕) then return True;
15 |  |  if ℓ(u) ≠ ⊕ and ℓ(v) ≠ ⊕ then
16 |  |  |  r_⊖ ← r;
17 |  |  |  if {u, v} ∈ E then r_⊖ ← r+ cost of {u, v} ;
18 |  |  |  I^⊖ ← (G', k − r_⊖, t, ω_{u,v}, ℓ^⊖_{u,v});
19 |  |  |  if solve(I^⊖) then return True;
20 |  end
21 end
22 return False;
```

The labeling and the weights changes slightly in the recursive calls. We define the update for ω and ℓ after a merge process of $\{u, v\}$ of Algorithm 2 by:

$$\omega_{u,v}(x) := \begin{cases} \omega(u) + \omega(v) & x = u, \\ \omega(x) & \text{otherwise,} \end{cases}$$

$$\ell^{\oplus}_{u,v}(x) := \begin{cases} \oplus & x = u, \\ \ell(x) & \text{otherwise,} \end{cases}$$

$$\ell^{\ominus}_{u,v}(x) := \begin{cases} \ominus & x = u, \\ \ell(x) & \text{otherwise.} \end{cases}$$

For the label function ℓ, it is important whether the vertices are merged into a clique or into an independent set. Therefore, there are two versions of the update.
In the following, we show that Algorithm 2 is correct for Vertex-Weighted Block Modeling and that Vertex-Weighted Block Modeling generalizes Block Modeling. Therefore, Algorithm 2 can be used to solve an instance of Block Modeling.

Lemma 4.1. *The recursive steps of Algorithm 2 are correct.*

Proof. Let $I = (G, t, k, \omega, \ell)$ be the instance to be considered. This proof splits into two parts. The first part shows that if a recursive call returns $True$, then there is a solution for I. The second part shows that if I is a yes-instance of Vertex-Weighted Block Modeling, then at least one recursive call has to return $True$.

For the first part we consider the recursive call with instance $I^{\oplus} = (G', k' = k - r_{\oplus}, t, \omega_{u,v}, \ell^{\oplus}_u)$, where u and v will be in a positive neighborhood class. Let E_{sol} be the solution of $I^{\oplus}$. We split $E^{\oplus}_{\text{sol}} = E_u \cup E_{\text{rest}}$ in two parts. In E_u are all edits which contain u and E_{rest} consists of the remaining edits. Let $E_v := \{\{v, u\} \mid \{u, u\} \in E_u\}$ be the set that mimics the changes of u to v. We define $E'_{\text{edit}} := E_{\text{edit}}$ and add the edge $\{u, v\}$ to E'_{edit} if $\{u, v\} \notin E$. Now, we show that the edits $E'_{\text{edit}} \cup E_{\text{rest}} \cup E_u \cup E_v$ are a valid solution for G. Intuitively, E'_{edit} ensures that u and v are in the same neighborhood class and adds the missing edge $\{u, v\}$ if necessary. The edits E_u and the "mimic" E_v ensures, that u and v stay in the same neighborhood class. At last, E_{rest} changes the vertices $V \setminus \{u, v\}$ such that the vertices of V are in at most t neighborhood classes.

Now, we discuss the cost of the edits $E'_{\text{edit}} \cup E_{\text{rest}} \cup E_u \cup E_v$ with the weight function ω. To improve the readability, we overload the operator ω and define for an edit set E_S the summed edit cost as $\omega(E_S) := \sum_{e \in E_S} w(e)$. Through the algorithm, we know that

$$\omega^{\oplus}_{u,v}(E_{\text{rest}} \cup E_u) \leq k - r_{\oplus} = k' \text{ and}$$
$$\omega(E'_{\text{edit}}) = r_{\oplus}.$$

The cost of each edit in E_{rest} is the same for ω and $\omega^{\oplus}_{u,v}$ since neither u nor v are in an edit of E_{rest}. Because of the additional cost for u with $\omega^{\oplus}_{u,v}$, we "pay" for E_v:

$$\omega^{\oplus}_{u,v}(E_u) = \omega(E_u) + \omega(E_v). \tag{1}$$

With this, we can estimate the cost of $E'_{\text{edit}} \cup E_{\text{rest}} \cup E_u \cup E_v$:

$$\begin{aligned}
&\omega(E'_{\text{edit}}) + \omega(E_{\text{rest}}) + \omega(E_u) + \omega(E_v) \\
&= r_{\oplus} + \omega^{\oplus}_{u,v}(E_{\text{rest}}) + \omega^{\oplus}_{u,v}(E_u) \\
&\overset{(*)}{=} r_{\oplus} + \omega^{\oplus}_{u,v}(E_{\text{rest}} \cup E_u) \\
&\leq r_{\oplus} + k - r_{\oplus} = k.
\end{aligned}$$

Note that the equation marked with $(*)$ only holds because E_{rest} and E_u are disjoint. The cost of the solution is bounded by k and hence it is a valid solution. Second, we consider $I^{\ominus} = (G', k' = k - r_{\ominus}, t, \omega_{u,v}, \ell^{\ominus}_u)$. This case with an independent set verifies analogously.

In the second part, we assume that I is a yes-instance of Vertex-Weighted Block Modeling. We know that at least two vertices of T have to be in the same neighborhood in the resulting graph. Let u and v be those two vertices. without loss of generality let u and v be in a positive neighborhood class in the resulting graph. Let E^I_{sol} be a solution of I. Let E^I_{edit} be an edit set of $\mathcal{T}$ that is a subset of E^I_{sol}. Let $I^{\oplus}$ be the instance when the second foreach-loop of Algorithm 2 considers E^I_{edit}.
We show that $I^{\oplus}$ is a yes-instance of Vertex-Weighted Block Modeling by constructing a solution for $I^{\oplus}$. The construction proceeds similar to construction of part one. Let $E_{u,v} := E^I_{\text{sol}} \cap \{u, v\}$ the set containing $\{u, v\}$ if it is in E^I_{sol}. We split E^I_{sol} in $E^I_{\text{edit}} \cup E_{u,v} \cup E_u \cup E_v \cup E_{\text{rest}}$. Similar to part one, in E_u is every edit $e \in E^I_{\text{sol}}$ where $u \in e$ and $e \notin E^I_{edit} \cup E_{u,v}$. The edit set

E_v is defined analogously. The set E_{rest} consists of the remaining edits. The edit set $E_{\text{sol}} := E_{\text{rest}} \cup E_u$ is a valid solution for instance $I^{\oplus}$. The explanation is the same as in part one.
At last, we have to show that $\omega(E_{\text{sol}}) \leq k' = k - r_{\oplus}$. Lets denote what we know by the algorithm:

$$\omega(E^I_{\text{sol}}) \leq k,$$
$$\omega(E^I_{edit} \cup E_{u,v}) = r_{\oplus}.$$

This implies $\omega(E_u) + \omega(E_v) + \omega(E_{\text{rest}}) \leq k - r_{\oplus}$. Because E_{rest} does not contain an edit which contains u or v, the cost of E_{rest} is the same with $\omega_{u,v}$ as with ω. Thus, $\omega(E_{\text{rest}}) = \omega_{u,v}(E_{\text{rest}})$. The equation (1) applies here as well. Now, we insert these equations in the upper implication and we are done:

$$\begin{aligned} & \omega(E_u) + \omega(E_v) + \omega(E_{\text{rest}}) \leq k - r_{\oplus} \\ \Leftrightarrow \quad & \omega_{u,v}(E_u) + \omega_{u,v}(E_{\text{rest}}) \leq k - r_{\oplus} \\ \Leftrightarrow \quad & \omega_{u,v}(E_{\text{sol}}) \leq k - r_{\oplus}. \end{aligned}$$

□

Theorem 4.1 shows that Vertex-Weighted Block Modeling generalize Block Modeling.

Theorem 4.1. *Any instance $I = (G = (V, E), k, t)$ of* Block Modeling *is a yes-instance if and only if the instance $I' = (G, k, t, \omega, \ell)$ of* Vertex-Weighted Block Modeling *is a yes-instance with $\omega(x) = 1$ and $\ell(x) = \odot$ for each $x \in V$.*

Proof. With such a configuration of ω and ℓ, every edit has cost 1 and every vertex pair is mergeable, hence $\sim^{\ell}_G \Leftrightarrow \sim_G$. Thus, I is a yes-instance if and only if I' is a yes-instance. □

4.2 Heuristics

In this section, we present two greedy algorithms and a local search approach. A heuristic finds a feasible solution in polynomial time but without a guarantee of an optimal solution. The greedy algorithm presented here build a solution stepwise. In each step, the unfinished solution expands locally in an optimal way.

4.2.1 Block-Framework

This section explains the framework of the heuristics. Let $G = (V, E)$ be the input graph. The approaches compute the cost of a partition $\mathcal{B} = \{B_0, B_1, \ldots, B_b\}$ of V. Each set of this partition is called *block*. In one step, the heuristic computes greedily a new partition with a decreased number of blocks. This step repeats until the partition consists of t blocks. A start partition may be the neighborhood partition.
For each block, we compute the minimal cost such that all vertices of this block will be in the same neighborhood class. We achieve this by two aspects. First, a block has to be a clique or independent set in the resulting graph $G' = (V, E')$. Second, two blocks have to be adjacent or non-adjacent in G'. We define the cost functions ω_{inner} and ω_{outer} that compute the minimal cost for both aspects. Let $E_{\text{edges}}(B_i) := \binom{B_i}{2} \cap E$ be the edges within the block B_i and let $E_{\text{edges}}(B_i, B_j) := E(B_i, B_j) \cap E$ be the edges between B_i and B_j. The cost functions are defined as:

$$\omega_{\text{inner}}(B_i) := \begin{cases} \min(\binom{|B_i|}{2} - E_{\text{edges}}(B_i),\ E_{\text{edges}}(B_i)) & |B_i| > 1, \\ 0 & \text{otherwise,} \end{cases}$$

$$\omega_{\text{outer}}(B_i, B_j) := \min(E(B_i, B_j) - E_{\text{edges}}(B_i, B_j),\ E_{\text{edges}}(B_i, B_j)).$$

Finally, the cost of a partition is the sum of the cost of each block and of the cost of each unordered pair of $\binom{\mathcal{B}}{2}$:

$$\omega_{\text{partition}}(\mathcal{B}) := \sum_{B_i \in \mathcal{B}} \omega_{\text{inner}}(B_i) + \sum_{\{B_i, B_j\} \in \binom{\mathcal{B}}{2}} \omega_{\text{outer}}(B_i, B_j).$$

4.2.2 Merge-Heuristic

The start partition is the neighborhood partition of G. Based on the partition, the first heuristic *Merge-Heuristic* searches the best partition by joining two blocks. The function $\text{merge}_B(B_i, B_j)$ returns the partition where two blocks B_i and B_j of B are joined:

$$\text{merge}_{\mathcal{B}}(B_i, B_j) := (\mathcal{B} \setminus \{B_i, B_j\}) \cup \{B_i \cup B_j\}$$

A pseudocode for Merge-Heuristic can be found in Algorithm 3.

Algorithm 3: Method `Merge-Heuristic`

Input: Instance $I = (G, t)$
Output: upper bound for a solution of I

```
1 B ← neighborhood partition of G;
2 while |B| > t do
3 |   find B_i ∈ B and B_j ∈ B such that ω_partition(merge_B(B_i, B_j)) is
  |     minimal;
4 |   B ← merge_B(B_i, B_j);
5 end
6 return ω_partition(B);
```

4.2.3 Split-Heuristic

The second heuristic *Split-Heuristic* decreases the partition by one in each step as well. Again, the start partition is the neighborhood partition of G. In each step, the algorithm determines a block which vertices are put in other blocks. We say the block is *split*. To determine a block to split, we define a function τ which describes the cost increase when augmenting a block by a vertex. For example, let $\mathcal{B}$ 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 an 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 $\omega_{\text{partition}}(\mathcal{B}'') - \omega_{\text{partition}}(\mathcal{B}')$.
Because of canceling of terms where the augmentation does not affected ω_{inner} or ω_{outer}, we can simplify the term to:

$$\begin{aligned}\tau_{\mathcal{B}'}(B_i, v) := \omega_{\text{inner}}(B_i \cup \{v\}) - \omega_{\text{inner}}(B_i) \\ + \sum_{B_i \neq B_j \in \mathcal{B}'} \omega_{\text{outer}}(B_i \cup \{v\}, B_j) - \omega_{\text{outer}}(B_i, B_j)\end{aligned}$$

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 splitted, the algorithm searches a block B^* such that sum of the minimum cost increase of all vertices of B^* is minimal.
When spitting a block, the block is first 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 pseudocode of the Split-Heuristic is shown in Algorithm 4.
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 can only be higher than the estimation.

Algorithm 4: Method `Split-Heuristic`

Input: Instance $I = (G, t)$
Output: upper bound for a solution of I

```
1  B ← neighborhood partition of G;
2  while |B| > t do
3  |  find B_v ∈ B such that Σ_{v∈B_v} min_{B_i∈B'} τ_{B'}(B_i, v) is minimal with
   |    B' = B \ {B_v};
4  |  B ← B';
5  |  foreach v ∈ B_i do
6  |  |  find B_j ∈ B such that τ_{B''}(B_j, v) with
   |  |    B'' = B \ {B_j} ∪ {B_j ∪ {v}} is minimal;
7  |  |  B ← B'';
8  |  end
9  end
10 return ω_partition(B);
```

4.2.4 Local Search

Our local search algorithm receives the graph and a block partition as input. The algorithm tries to improve the solution by small changes until it no more improvement can be done.
Our approach has three kinds of changes. If no change can decrease the cost of the partition, the local search stops. Otherwise, the algorithm tries to improve the partition further.
The first change removes a vertex from its block and puts the vertex to a block where the cost increases is minimal. The second change swaps the

blocks of two vertices if it reduces the cost. The third change puts all vertices of the same neighborhood class of a block to another block if this reduces the cost. For the last change, we define the *neighborhood class distribution* of a block B. Let $\mathcal{W} = \{W_1, \ldots, W_w\}$ be the neighborhood partition of a graph G. Then, the neighborhood class distribution of B is $\{W_1 \cap B, \ldots, W_w \cap B\}$. The local search heuristic is shown in Algorithm 5.

Algorithm 5: Method `Local Search`

Input: Graph G and block partition $\mathcal{B}$
Output: locally improved block partition

```
repeat
    foreach v ∈ V do
        remove v from its block and update B;
        find B_i ∈ B such that τ_B(B_i, v) is minimal;
        put v in B_i and update B;
    end
    foreach {u, v} ∈ (V choose 2) do
        B_u ← block of u;
        B_v ← block of v;
        put u to B_v and put v to B_u if the updated B has a lower cost;
    end
    foreach B ∈ B do
        S ← neighborhood class distribution of B;
        forall S ∈ S do
            remove all vertices of S from B and put them to any
              block if the updated B has a lower cost;
        end
    end
until ω_partition(B) does not change;
return B;
```

4.3 ILP-Formulations

In this section, we introduce two Integer Linear Program-formulations for Block Modeling. An Integer Linear Program (ILP) consists of a set of variables, a linear objective function that is to minimize or maximize and the

so-called *constraints*. A constraint is a linear equation or a linear inequation. The goal of the ILP is to find one assignment of the variables that all constraints are satisfied and the objective function is optimal. For more theory about ILPs and formal definition, we refer to [29]. We implemented the ILP with the Gurobi[1] solver.

The concept of our first ILP is derived from Algorithm 1. Let $G = (V, E)$ be the input graph.
The editing of one vertex pair $\{u, v\}$ is represented by the *edit variable* $e_{\{u,v\}} \in \{0, 1\}$. If $e_{\{u,v\}} = 1$, then $\{u, v\}$ is in the solution, and not otherwise. Hence, the function to minimize is $\sum_{\{u,v\} \in \binom{V}{2}} e_{\{u,v\}}$.
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 $\{u, v\}$ are resolved. Let w be a witness of $\{u, v\}$. The constraints

$$m_{\{u,v\}} \leq e_{\{u,w\}} + e_{\{v,w\}},$$
$$m_{\{u,v\}} \leq 2 - e_{\{u,w\}} - e_{\{v,w\}}$$

guarantee that exactly one edit variable equals 1 and thus, in the solution w is resolved correctly for $\{u, v\}$.
Among $t + 1$ vertices $V' \subset V$ at least two vertices have to be in the same neighborhood class in the resulting graph. Therefore, among $\binom{V'}{2}$ there is at least one vertex pair $\{x, y\}$ such that the merge variable $m_{\{x,y\}}$ equals to 1 for a solution.
Due to 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 fulfilled by the constraint

$$m_{\{u,w\}} + m_{\{v,w\}} - m_{\{u,v\}} \leq 1.$$

[1] see `https://www.gurobi.com/`

The ILP is given by:

$$\begin{aligned}
\min \quad & \sum_{\{u,v\} \in \binom{V}{2}} e_{\{u,v\}}, \\
\text{s. t.} \quad & \text{\# witness constraints} \\
& m_{\{u,v\}} \leq e_{\{u,w\}} + e_{\{v,w\}} && \forall \{u,v\} \in \binom{V}{2}, \forall w : w \text{ is witness of } \{u,v\}, \\
& m_{\{u,v\}} \leq 2 - e_{\{u,w\}} - e_{\{v,w\}} \\
& \text{\# merge constraints} \\
& \sum_{1 \leq i < j \leq t+1} m_{\{v_i,v_j\}} \geq 1 && \forall \{v_1, v_2, \ldots, v_{t+1}\} \in \binom{V}{t+1}, \\
& \text{\# transitivity constraints} \\
& m_{\{u,w\}} + m_{\{v,w\}} - m_{\{u,v\}} \leq 1 && \forall \{u,v,w\} \in \binom{V}{3}, \\
& e_{\{u,v\}} \in \{0,1\}, m_{\{u,v\}} \in \{0,1\} && \forall \{u,v\} \in \binom{V}{2}.
\end{aligned}$$

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 $2 \cdot \binom{n}{2} = \mathcal{O}(n^2)$ 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 $\binom{n}{2} \cdot (n-2) = \mathcal{O}(n^3)$ *witness constraints*. There are $O(n^3)$ many *transitive constraints* as well, one for each vertex triple. The largest number of constraints is taken by the *merge constraints*. There are $\binom{n}{t+1} = \mathcal{O}(n^{t+1})$ many of them.

The Gurobi solver offers the possibility to add constraints via callbacks. Sometimes, not all constraints are necessary to find the optimal solution for an ILP. An ILP can start with a subset of the constraints and receives more constraints during the computation via callbacks. This technique can decrease the running-time. Callbacks can interrupt the ILP on many events. We use the callbacks exclusively when the ILP finds an optimal solution for its (incomplete) set of constraints. In Section 6.3, it is described which constraints of this ILP are added via callbacks for our evaluation.

The second ILP creates its constraints only via callbacks. Let $G = (V, E)$

be the input graph. As same as in the first ILP, the editing of one vertex pair $\{u, v\}$ is represented by the edit variable $e_{\{u,v\}} \in \{0, 1\}$. If $e_{\{u,v\}} = 1$, then $\{u, v\}$ is in the solution, and not otherwise. The function to minimize is $\sum_{\{u,v\} \in \binom{V}{2}} e_{\{u,v\}}$.
Consider $t + 1$ many vertices $V' = \{v_1, \ldots, v_{t+1}\}$ of distinct neighborhood classes in G. Every vertex pair of $\binom{V'}{2}$ has at least one witness. For one vertex pair $\{x, y\}$, let $\chi^G_{x,y}$ be an arbitrary but fixed witness of $\{x, y\}$ in G. We know that at least one witness of a vertex pair in $\binom{V'}{2}$ has to resolved. This property is fulfilled by this constraint

$$\sum_{\{u,v\} \in \binom{V'}{2}} e_{\{u,\chi^G_{u,v}\}} + e_{\{v,\chi^G_{u,v}\}} \geq 1.$$

This constraint is correct for G but not for the callbacks. Consider a callback: The ILP has a solution E_{sol} that may be incomplete or wrong. We have to search the $t + 1$ many vertices of distinct neighborhood classes in $G \triangle E_{\text{sol}}$ since similarities and witnesses may change by the application of E_{sol}. Furthermore, we have to adjust our constraint to consider already edited vertex pairs of E_{sol}.
Let $\{x, y\}$ be a vertex pair and let z be a witness of $\{x, y\}$. Let $\{x, z\} \in E_{\text{sol}}$ be an edited vertex pair and let $\{y, z\} \notin E_{\text{sol}}$ be unedited. To resolve z for $\{x, y\}$ either $\{y, z\}$ has to be edited or the edition of $\{x, z\}$ must be undone. Hence, we set the term $1 - e_{\{x,z\}}$ instead of $e_{\{x,z\}}$ when $\{x, z\}$ is edited.
For this, we define a function

$$\kappa_{E_{\text{sol}}}(\{x, y\}) := \begin{cases} e_{\{x,y\}} & \{x, y\} \notin E_{\text{sol}}, \\ 1 - e_{\{x,y\}} & \{x, y\} \in E_{\text{sol}}. \end{cases}$$

Altogether, let E_{sol} be the solution of a callback and let $G' = G \triangle E_{\text{sol}}$. For a vertex set V' of size $t+1$ containing vertices of distinct neighborhood classes in G' we add

$$\sum_{\{u,v\} \in \binom{V'}{2}} \kappa_{E_{\text{sol}}}(\{u, \chi^{G'}_{u,v}\}) + \kappa_{E_{\text{sol}}}(\{u, \chi^{G'}_{u,v}\}) \geq 1$$

to the constraints.

5 Speed-Ups

In this section, we provide some lower bounds, reduction rules and other improvements for the branch-and-bound algorithms. To store more information, we present annotated versions for both problems. Generally, the improvements are described for the annotated version of Block Modeling. Afterwards, we describe the adaptations to the annotated version of Vertex-Weighted Block Modeling.

5.1 Basic Improvements

We modify the graph for the recursive calls instead of creating a copy and then modifying it. To facilitate this, we decided to implement the neighbors of a vertex with a slightly modified Partitionable Set [32]. This set consists of two arrays. Consider the neighborhood of the vertex u. One array A_1 stores the neighbors. The first $|N(u)|$ entries in A_1 are the neighbors of u. The other array A_2 stores the index in A_1 for each vertex. If a vertex is not in the set, its index in A_2 is -1.
The adjacency between two vertices can be checked in constant time and the iteration through the neighbors of a vertex v can be done in $\mathcal{O}(|N(v)|)$ time. The insertion and removal of neighbors is also done in constant time.

5.2 Annotations

Since the speed-ups store additional information, we introduce the annotated versions of Block Modeling and of Vertex-Weighted Block Modeling. First, we describe which annotations are in each problem and then we define the annotated problems.
We store vertex pairs that cannot be in the solution of an instance in a set B in both annotated versions. If a vertex pair is in B, we say it is *blocked*.
Only for Block Modeling, we store the information whether a vertex pair is *merged* in a set M. Note that within a merge process in Vertex-Weighted Block Modeling a vertex is deleted and therefore, this information cannot be stored.
For both annotated problems, we store a set A where vertex pairs, which vertices are not in the same neighborhood class in the resulting graph, are stored. Vertex pairs of A are called *apart*.

At last, we define a function $\psi : V \to \mathbb{N}$ that states the number of edges *inside* a vertex for the Vertex-Weighted Block Modeling. The use of ψ is discussed later.
Since the annotations of the annotated problems serve for algorithmic purposes, the problem question is unchanged.

Annotated Block Modeling

Input: A graph $G = (V, E)$, an integer k, a positive integer t and three sets of vertex pairs B, A and M.

Question: Is there a solution E_{sol} of Block Modeling for (G, k, t) such that

1. the vertices u and v with $\{u, v\} \in A$ are in distinct neighborhood classes,
2. the vertices x and y with $\{x, y\} \in M$ are in the same neighborhood class, and
3. vertex pairs of B are not in E_{sol}?

Annotated Vertex-Weighted Block Modeling

Input: A graph $G = (V, E)$, an integer k, a positive integer t, a weight function $\omega : V \to \mathbb{N}$, a label function $\ell : V \to \{\oplus, \ominus, \odot\}$, two sets of vertex pairs B and A, and a function $\psi : V \to \mathbb{N}$.

Question: Is there a solution E_{sol} of Vertex-Weighted Block Modeling for (G, k, t, ω, ℓ) such that

1. the resulting graph has at most t neighborhood classes with respect to $\sim_G^{\ell}$ and λ,
2. the vertices u and v with $\{u, v\} \in A$ are in distinct neighborhood classes with respect to $\sim_G^{\ell}$ and λ, and
3. vertex pairs of B are not in E_{sol}?

For the transformation of a Block Modeling instance or a Vertex-Weighted Block Modeling instance to the corresponding annotated problem, let every set of $\{A, B, M\}$ be the empty sets and we set $\psi(v) = 0$ for each vertex $v \in V$.
In the later sections, we use a synonym to update the sets A, B and M. For

example, if we say that we *mark* a vertex pair p as blocked, we put p in B. This is analogous for the other sets and their denotations.
In this chapter, we refer to Block Modeling and Annotated Block Modeling if we speak of the unweighted problems. Analogously, with the weighted problems are Vertex-Weighted Block Modeling and Annotated Vertex-Weighted Block Modeling meant.

5.3 Branching

A basic improvement for both annotated problems is to edit a vertex pair at most once from the root to the leaf in the branch tree. To ensure this, we mark a vertex pair as blocked in the recursive calls once it is edited.
In Annotated Block Modeling, when the algorithm declares one vertex pair to merge and resolves all witnesses, we mark the vertex pair as *merged*. This annotation is used for Reduction Rule 2.
Next, we describe the use of the apart vertex pairs for both annotated problems. Recall the situation of a vertex pair which is chosen to be merged. Let T be a set of $t+1$ vertices of distinct neighborhood classes. At least one vertex pair of $\binom{T}{2}$ has to be merged in the solution graph. After no solution is found for the merge of one vertex pair $\{x, y\}$, we mark $\{x, y\}$ as apart for the recursive calls since no solution exists where x and y are in the same neighborhood class in the resulting graph. Apart vertex pairs can be skipped in the foreach-loop of $\binom{T}{2}$ since the solution would have been found in an earlier branch where the vertex pair was merged. The meant foreach-loop is in line 4 in Algorithm 1, and in line 5 in the Algorithms 2 and 6. Note that for Annotated Vertex-Weighted Block Modeling two vertices which are not mergeable can be viewed as apart as well and hence skipped in the foreach-loop.
Therefore, in the selection of the $t+1$ vertices of distinct neighborhood classes, it is advantageous when many vertex pairs are already apart. Let $\text{apart}(v)$ be the number of vertex pairs that are apart and contain v. We sort the vertices of V by $\text{apart}(v)$ of each vertex v in descending order to find a good set T' such that many vertex pairs of $\binom{T'}{2}$ are apart. At first, we start with an empty set T'. We iterate through the sorted vertices and put a vertex v in T' if no vertex of T' is in the same neighborhood class as v. This is done until $|T'| = t + 1$.
In the following, we modify the branching in Annotated Vertex-Weighted Block Modeling to decrease the number of nodes in the branch tree.

The decision of the label of a vertex can be delayed in some cases. Then, instead of branching in two cases it is possible to branch into one. To accomplish this, we make use of the function $\psi : V \to \mathbb{N}$ that represents how many edges are inside a vertex. When merging two vertices u and v, where v is deleted in the recursive calls, the updated function $\psi_{u,v}$ is defined as

$$\psi_{u,v}(x) := \begin{cases} \psi(x) & x \neq u, \\ \psi(u) + \psi(v) + \omega(u) \cdot \omega(v) & \{u, v\} \in E, \\ \psi(u) + \psi(v) & \{u, v\} \notin E. \end{cases}$$

If $\psi(x) = 0$, then x represents an independent set. If $\psi(x) = \binom{\omega(x)}{2}$, then x represents a clique. Consider now that the algorithm resolved all witnesses for $\{u, v\}$. As long as $\psi_{u,v}(u) = 0$ or $\psi_{u,v}(u) = \binom{\omega_{u,v}(u)}{2}$, setting the label $\ell(u)$ can be delayed since no decision is necessary such that u represents a clique or an independent set. Thus, this gives us the condition for the single branch. In the other cases where $0 < \psi(u) < \binom{\omega(u)}{2}$, the normal branching has to be done where the label $\ell(u)$ is set in the recursive calls.
In the normal branching the costs $r_\oplus$ and $r_\ominus$ need an adjustment since it is necessary to pay for edits such that the merged vertex represents a clique or an independent set. We define the inner cost $\text{cost}_I := V \to \mathbb{N}$ for a vertex as

$$\text{cost}_I(x) := \begin{cases} \binom{\omega(x)}{2} - \psi(x) & \ell(x) = \oplus, \\ \psi(x) & \ell(x) = \ominus, \\ 0 & \ell(x) = \odot. \end{cases}$$

This function states how many edits are necessary for a vertex so that x represents a clique ($\ell(x) = \oplus$) or x represents an independent set ($\ell(x) = \ominus$). If the label ot the vertex x is $\odot$, then x represents a clique or an independent set with no payment needed. In the recursive call, the inner cost of a merged vertex u can be larger than the summed inner costs of u and v before the recursive call. The difference of the costs has to be considered in the edit budget. Therefore, we define the cost of a merged $\text{cost}_I : V \times V \times \{\oplus, \ominus\} \to \mathbb{N}$ as

$$\text{cost}_I(x, y, \otimes) := \begin{cases} \binom{\omega_{x,y}(x)}{2} - \psi_{x,y}(x) & \otimes = \oplus, \\ \psi_{x,y}(x) & \otimes = \ominus. \end{cases}$$

Algorithm 6: Method `solve`

Input: Instance $I = (G, k, t, \omega, \ell, B, A, \psi)$ of Annotated Vertex-Weighted Block Modeling

Output: *True* if and only if I is yes-instance of Annotated Vertex-Weighted Block Modeling

if $k < 0$ **then return** *False*;
if *G has at most t neighborhood classes with respect to $\sim_G^{\ell}$ and λ* **then return** *True*;
$T \leftarrow t + 1$ vertices of distinct neighborhood classes;
foreach $\{u, v\} \in \binom{T}{2}$ **do**
    $\mathcal{T} \leftarrow$ collection of all resolve sets of $\{u, v\}$;
    foreach $E_{\text{edit}} \in \mathcal{T}$ **do**
        $G' \leftarrow (G \triangle E_{\text{edit}}) - v$;
        $r \leftarrow$ total edit cost of E_{edit};
        if $\psi_{u,v}(u) = 0$ or $\psi_{u,v}(u) = \binom{\omega_{u,v}(u)}{2}$ **then**
            $I' \leftarrow (G', k - r, t, \omega_{u,v}, \ell, B, A, \psi_{u,v})$;
            if `solve`(I') **then return** *True*;
        else
            if $\ell(u) \neq \ominus$ and $\ell(v) \neq \ominus$ **then**
                $r_{\oplus} \leftarrow r + \text{costIncrease}_I(u, v, \oplus)$;
                $I^{\oplus} \leftarrow (G', k - r_{\oplus}, t, \omega_{u,v}, \ell^{\oplus}_{u,v}, B, A, \psi_{u,v})$;
                if `solve`$(I^{\oplus})$ **then return** *True*;
            if $\ell(u) \neq \oplus$ and $\ell(v) \neq \oplus$ **then**
                $r_{\ominus} \leftarrow r + \text{costIncrease}_I(u, v, \ominus)$;
                $I^{\ominus} \leftarrow (G', k - r_{\ominus}, t, \omega_{u,v}, \ell^{\ominus}_{u,v}, B, A, \psi_{u,v})$;
                if `solve`$(I^{\ominus})$ **then return** *True*;
    end
end
return *False*;

To get the cost increase, we subtract the cost of the merged vertex with the previous costs of the vertices:

$$\text{costIncrease}_I(x, y, \otimes) := \text{cost}_I(x, y, \otimes) - \text{cost}_I(x) - \text{cost}_I(y)$$

In Algorithm 6, the changes for the branching are included. Note that the updates for the sets B and A are missing for overview purposes.
At last, we analyze how ψ affects the neighborhood classes. Because of ψ, the relation $\sim_G^\ell$ has to be adjusted once more. For example, let u and v be two vertices which are adjacent to each other with $\omega(u) \geq 2$, $\omega(v) \geq 2$, and $\psi(u) = \psi(v) = 0$. Moreover, let $\text{wit}(\{u, v\}) = \emptyset$. In this example, u and v represent independent sets of size at least 2 which are adjacent. Even though $\{u, v\}$ has no witnesses, the vertices are currently not in the same neighborhood class. Since the explanation whether two vertices with no witnesses are in the same neighborhood class has many special cases, we provide a table for a better understanding. First, we define a second label $\lambda : V \to \{\oplus, \ominus, \odot\}$ that states what a vertex x represents if $\ell(x) = \odot$. The label function λ is defined as:

$$\lambda(x) := \begin{cases} \ell(x) & \ell(x) = \oplus \vee \ell(x) = \ominus, \\ \odot & \omega(x) = 1, \\ \oplus & \psi(x) = \binom{\omega(x)}{2}, \\ \ominus & \psi(x) = 0. \end{cases}$$

Note that λ is not well-defined since the case $\ell(x) = \odot$ with $0 < \psi(x) < \binom{\omega(x)}{2}$ is missing. This is no problem because, when $0 < \psi(x) < \binom{\omega(x)}{2}$, the label ℓ is set to $\oplus$ or $\ominus$ by the algorithm. At last, we mark an vertex x with $\oplus$ if $x \in E$ and with $\ominus$, otherwise. Consider the Table 1 with two vertices u and v with $\text{wit}(\{u, v\}) = \emptyset$. Table 1 contains some examples for the ℓ and λ of u and v, and the result whether u and v are in the same neighborhood class. Essentially, if the row contains both symbols $\oplus$ and $\ominus$, then u and v are not in the same neighborhood class.

5.4 Witnesses

In this subsection, we explain how to update the witnesses by an application of an edit in the unweighted problems. The adaption for the weighted problems has to mention one more detail. This is described at the end. We store

Table 1: Examples of the labels ℓ and λ of two vertices u and v with $\text{wit}(\{u, v\}) = \emptyset$. The last column states whether u and v are in the same neighborhood.

$\ell(u)$	$\ell(v)$	$\lambda(u)$	$\lambda(v)$	mark of $\{u, v\}$	Are u and v in the same neighborhood class?
$\odot$	$\odot$	$\ominus$	$\ominus$	$\oplus$	False
$\odot$	$\odot$	$\odot$	$\ominus$	$\ominus$	True
$\oplus$	$\odot$	$\oplus$	$\odot$	$\oplus$	True
$\odot$	$\ominus$	$\odot$	$\ominus$	$\oplus$	False

for each vertex pair the witnesses in a Partitionable Set. Let $G = (V, E)$ be a graph. A naive computation to obtain every witnesses of a vertex pair $\{u, v\}$ can be done in $\mathcal{O}(|N(u)|+|N(v)|)$ time by iterating through $N(u) \cup N(v)$ and check if the vertex is only adjacent to either u or v. Doing this computation for every vertex pair in each branch is time-consuming.

To improve the running-time, we present how to update the witnesses in $\mathcal{O}(n)$ time after an edit $e = \{u, v\}$ is performed. Let $G' = G \triangle \{e\}$. Only vertex pairs which contain exact one vertex of e are affected by the application of e. For each other vertex pair, its witnesses are the same as before the application of e. Now, consider the vertex pair $\{u, w\}$ with $w \in V \setminus \{u, v\}$. First, assume that v is a witness of $\{u, w\}$ in G, that is, v is either adjacent to u or to w. In G', the vertex v is either adjacent to $\{u, w\}$ or non-adjacent to $\{u, w\}$. Hence, v is a non-witness of $\{u, w\}$ in G'. Second, assume now that v is a non-witness of $\{u, w\}$ in G. This means, the vertex v is either adjacent to $\{u, w\}$ or non-adjacent to $\{u, w\}$ in G. After the application of $e = \{u, v\}$, v is adjacent to exactly one vertex of $\{u, w\}$ in G' and therefore a witness. The vertex v is *flipped* in terms of being a witness of $\{u, w\}$ after a application of $\{u, v\}$. This is analogous with u and the vertex pair $\{v, w\}$. In Algorithms 7 and 8 the update of the witnesses is shown as pseudocode. Since Algorithm 8 iterates through almost every vertex and the operations on the Partitionable Set can be done in constant time, the update is computed in $\mathcal{O}(n)$ time. Unfortunately, $\mathcal{O}(n^3)$ space is required because every vertex pair has a Partitionable Set which has up to $n - 2$ witnesses.

Finally, we outline the adaption for the weighted problems. Algorithm 2 and Algorithm 6 merge two vertices and with this process, a vertex v is removed from V in the recursive call. With the removal of v, every entry where v is a

Algorithm 7: Method `flipWitness`

Input: Vertex pair $p = \{x, y\}$ and vertex z

```
1 if z is witness of p then
2 |   remove z from wit(p)
3 else add z to wit(p);
```

Algorithm 8: Method `UpdateWitnesses`

Input: Graph $G = (V, E)$ and vertex pair $\{u, v\}$

```
1 foreach w ∈ V \ {u, v} do
2 |   flipWitness({u, w}, v);
3 |   flipWitness({v, w}, u);
4 end
```

witness has to be removed as well. This means a $\mathcal{O}(n^2)$ computation effort for a merge and its undoing since v could be a witness of almost every vertex pair. Instead of iterating through every vertex pair, it is possible to store a hash table [34] for each vertex z containing all vertex pairs where z is a witness of. This hash table has to be updated in Algorithm 7.

5.5 Neighborhood Classes

In this subsection, we explain how to update the neighborhood classes by an application of an edit in the unweighted problems. The adaption for the weighted problems is described at the end. The naive computation for the neighborhood partition claims $\mathcal{O}(n^3)$ time [23]. Since two vertices u and v are in the same neighborhood class if the vertex pair $\{u, v\}$ has no witnesses, the computation of the neighborhood partition can be done in $\mathcal{O}(n^2)$ time if the witnesses for every vertex pair are precomputed.

Consider a graph $G = (V, E)$. Let d be the neighborhood diversity of G. In the following, we show that with precomputed witnesses for each vertex pair the neighborhood partition can be updated in $\mathcal{O}(d)$ time after an application of an edit $e = \{u, v\}$.

Let $\mathcal{W} = \{W_1, \ldots, W_d\}$ be the neighborhood partition of G. Let W_u be the neighborhood class of u and let W_v be the neighborhood class of v in G. The relation between two vertices of $V \setminus \{u, v\}$ is the same since their neighborhoods are the same in G' as in G. Let $\mathcal{W}' = (\mathcal{W} \setminus \{W_u, W_v\}) \cup$

$\{W_u \setminus \{u\}, W_v \setminus \{v\}\}$ be the modified and unfinished partition for G' where u and v are removed from their neighborhood class. Since $\mathcal{W}'$ can contain empty sets we remove them from $\mathcal{W}'$ in $\mathcal{O}(d)$ time. Note that $\mathcal{W}'$ is the neighborhood partition of $(G' - u) - v$.
We construct the neighborhood partition for $G' - v$, and then the neighborhood partition for G'. First, the neighborhood class for u will be "searched", afterwards for v. We outline the scheme of the search for u. In $G' - v$ the vertex u is either in a neighborhood class with another vertex or in a neighborhood class with no other vertices. At the beginning, we iterate through $\mathcal{W}'$ and check whether $\{u, w\}$ has no witnesses with an arbitrary vertex w of the considered neighborhood class W in G'. If there is no witness, then add u to W_i and we are done since $W \cup \{u\}$ is a neighborhood class in $G' - v$. Otherwise, the search continues. If there is no such vertex w in $\mathcal{W}'$, we add $\{u\}$ to $\mathcal{W}'$. Afterwards, we search analogously in the updated $\mathcal{W}'$ for v a neighborhood class with G' instead of $G' - v$. At the end, the partition $\mathcal{W}'$ is the neighborhood partition of G'.
Note that for the weighted problems the conditions that two vertices are in the same neighborhood class are more complex. They have to be mergeable and it is necessary to consider λ in Annotated Vertex-Weighted Block Modeling. This augments the check whether two vertices are in the same neighborhood class by a constant but it does not affect the running-time bound.

5.6 Lower Bounds

In this section, we present two lower bound algorithms. Let $I = (G, k, t)$ the considered instance. A lower bound algorithm computes a number. We call this number *lower bound*. A lower bound states how big k has to be such that I can be a yes-instance. In other words, if one lower bound is greater than k, then I is a no-instance.

The first lower bound algorithm is described for the unweighted problems. Afterwards, an adaption for the weighted problems is given. This lower bound algorithm estimates the minimal number of vertices that need to be affected by a solution. Let d be the neighborhood diversity of G. Let ξ be the sum of the size of the $d - t$ smallest neighborhood classes in G.

Proposition 5.1. *The number* $\lceil \frac{\xi}{2} \rceil$ *is a lower bound for an instance* (G, k, t)

of Block Modeling.

Proof. We assume the optimal scenario where, with an application of an edit $\{u, v\}$, the neighborhoods of vertices u and v need no further adjustment in the resulting graph. The neighborhood diversity has to be reduced by $d-t$ to get a graph with a neighborhood diversity of t. Therefore, at least the vertices of $d-t$ many neighborhood classes have to be affected by a solution. The minimum of vertices to affect is given by ξ. Since an edit affects two vertices, at least $\lceil \frac{\xi}{2} \rceil$ edits are necessary to affect the vertices of the $d-t$ smallest neighborhood classes. This implies that $\lceil \frac{\xi}{2} \rceil$ is a lower bound. □

Note that for the weighed problems the number ξ is the minimal sum of the weights of $d-t$ many neighborhood classes. The weight of a neighborhood class is the sum of the weights of the vertices: This can be done since the edit $\{u, v\}$ costs $\omega(u) \cdot \omega(v)$ which is at least $\frac{\omega(u)+\omega(v)}{2}$.
To find the desired neighborhood classes, we create an array containing the neighborhood classes. We partition this array such that the t "biggest" neighborhood classes are in the first t entries. In the remaining entries are the desired neighborhood classes for ξ. Note that "biggest" has a different meaning in the weighted and unweighted problems. The partition is done with Hoare's selection algorithm [18] which is a variation of Hoare's Quicksort [17].

The second lower bound algorithm is first described for the unweighted problems. Afterwards, two version of this lower bound algorithm are presented for the weighted problems. Let $G = (V, E)$ be a graph. This algorithm considers $t+1$ vertices of distinct neighborhood classes. We call such a set a *pack*. As already known, at least two vertices of one pack have to be in the same neighborhood class in the resulting graph. To achieve this, all witnesses of this vertex pair have to be resolved. For a pack P, let $\text{cost}(P) = \min_{p \in \binom{P}{2}} |\operatorname{wit}(p)|$ be the smallest number of witnesses of a vertex pair $\{u, v\} \in \binom{P}{2}$. The function $\text{cost}(P)$ states the minimal cost needed to *resolve* P. This number is called *cost* for its pack.
A collection of packs is called *packing*. Furthermore, every vertex can be in at most one pack of a packing. Let $\gamma(\mathcal{P}) = \sum_{P \in \mathcal{P}} \text{cost}(P)$ be the sum of the minimal costs of to resolve every pack of the packing $\mathcal{P}$.

Proposition 5.2. *The number* $\lceil \frac{\gamma(\mathcal{P})}{2} \rceil$ *of a packing* $\mathcal{P}$ *is a lower bound for an instance* (G, k, t) *of* Block Modeling.

Proof. Obviously, every pack has to resolved by a solution. To resolve a pack, edits of $E(P, V \setminus P)$ are necessary. In the worst case, an edit of $E(P, V \setminus P)$ is also necessary for at most another pack $P' \subseteq V \setminus P$. Therefore, an edit can affect at most two packs since every vertex is in at most one pack. Thus, $\lceil \frac{\gamma(\mathcal{P})}{2} \rceil$ is a lower bound. □

In the annotated problems the computation of $\text{cost}(P)$ can ignore vertex pairs of $\binom{P}{2}$ that are marked as apart since they will not be merged in the further branches. Moreover, in the weighted problems vertex pairs of $\binom{P}{2}$ that are not mergeable can be ignored in the computation of $\text{cost}(P)$ as well.

Note that there are generally multiple packings for G and they can have different lower bounds. Finding the optimal packing $\mathcal{P}$ such that $\gamma(\mathcal{P})$ is maximal is not trivial. We present a greedy approach to find a packing. Let S be the vertices that are not in a pack. Let $P = \emptyset$ be the pack that we construct. Now, we describe the process. At first, we search the vertex pair $\{u, v\}$ in $\binom{S}{2}$ such that $\text{wit}(\{u, v\})$ is maximal where and u and v are in distinct neighborhood classes, and $\{u, v\}$ is mergeable and not apart. Add u and v to P. Now, we add a vertex of $S \setminus P$ to P such that $\text{cost}(P)$ is maximal until $|P| = t + 1$. At last, we remove all vertices of P from S. Note that we can only create a pack if the vertices of S are in at least $t + 1$ many distinct neighborhood classes.
Next, we present a local approach to improve a packing. This swaps vertices of two distinct packs if the cost of both packs can increase. What means "can increase" in this context? Consider a pack P with cost c. We call the vertex pairs in a pack $\binom{P}{2}$ with a resolving cost equals the current cost of P *bottleneck vertex pairs*. Consider a swap of a vertex $u \in P$ and another vertex $v \in P' \neq P$. For the exchange, we demand that u is in at least one bottleneck vertex pair of P. At the other hand, each vertex pair p of $E(\{v\}, P \setminus \{u\})$ should have a higher resolving cost than c, be apart, or be not mergeable. The analog conditions hold for P' and v. For each pack of $\{P, P'\}$ either the number of bottleneck vertex pair reduces or the minimal resolve cost increases.
A swap can be also done with a vertex of a pack and another vertex that is no pack. In this case, only the cost pack should increase by the swap.

For the weighted problems we have two variants of the second lower bound algorithm. The first variant is the same as for the unweighted problems but

with the exception that a vertex v can be in at most $\omega(v)$ packs. This lower bound is still correct since a vertex v represents as many vertices of the initial branch call as its weight $\omega(v)$.
The other variant uses of the weight function ω in the computation of $cost(P)$ for a pack P. Since in the weighted versions the resolving of a witness w of $\{u, v\}$ costs at least $\min(\omega(u), \omega(v)) \cdot \omega(w)$, we redefine function $\text{cost}(P)$. Let $\omega(\{u, v\}) = \min(\omega(u), \omega(v)) \cdot \sum_{w \in \text{wit}(\{u,v\})} \omega(w)$ be the minimal cost to resolve all witnesses of $\{u, v\}$. We redefine $\text{cost}(P) := \min_{p \in \binom{P}{2}} \omega(p)$ as the minimal resolve cost of a vertex pair in $\binom{P}{2}$.

In the following we describe how to improve the running-time computation for the second lower bound algorithm in Annotated Vertex-Weighted Block Modeling where a vertex can be at most in one pack. At the end, we describe what needs to be adapted for the other lower bound algorithm version where vertices may be in more than one pack. Instead of a complete recomputation of the lower bound in each branch node, we update the packing and search for new packs. We store for each pack its cost in an integer to avoid redundant computation. This integer has to be updated in some cases as well. Let $\mathcal{P}$ be a packing for a graph G. The update of $\mathcal{P}$ has to consider the following cases:

1. Two vertices are merged or the undo of a merge is done**.** In this cases, we set a flag such that the lower bound recomputes the packing in the next branch node.
2. A witness is flipped**.** We distinguish in two cases for the flip of a witness of a vertex pair $\{u, v\}$. Note that only an update is necessary when u and v are in the same pack P. Therefore, we assume that u and v are in the same pack P. Let c' be cost for resolving the witnesses of $\{u, v\}$. If a witness is added to $\text{wit}(\{u, v\})$, then the cost to resolve P can increase. Thus, we recompute the cost for P. If a witness is removed from $\text{wit}(\{u, v\})$, then the cost to resolve P can decrease or even be 0. Let c be the cost of P before the removal of the witness. If $c' < c$, we set the cost of P to c. Moreover, if $\text{wit}(\{u, v\}) = \emptyset$ then u and v may be in the same neighborhood class. In this case, we remove P from the packing.
3. The apartness of a vertex pair $\{u, v\}$ changes**.** This case is very similar to the last one, we describe it nevertheless. An update is only necessary

if u and v are in the same pack. Hence, we assume that u and v are in the same pack P. Let c be the cost of P before the change. Let c' be cost for resolving all witnesses of $\{u, v\}$. If $\{u, v\}$ turns apart, then the cost of P can increase if $c' = c$. In this case, we recompute the cost for P. If $\{u, v\}$ turns not apart, then the cost of resolving P decreases when $c' < c$. In this case, we set the cost of P to c'. Moreover, if $c' = 0$, we remove P from $\mathcal{P}$.

4. The mergeability of a vertex pair $\{u, v\}$ changes**.** This update is analog to the last case since the characteristics not mergeable and apart are the equal for the packing.
5. A label is set for a vertex v**.** If v is in no pack, the packing $\mathcal{P}$ needs no update. Otherwise, some vertex pairs can change their mergeablility. Let P_v be the pack of v. We check whether a vertex pair of $E(\{v\}, P_v \setminus \{v\})$ changed the mergeability and update is accordingly.
6. The vertex pair $\{u, v\}$ is edited**.** Witnesses are flipped through an edit. This update is already stated. Moreover, this edit can change the mergeability of $\{u, v\}$. If the mergeability changes, then the corresponding update described in Case 4 has to be done.

Now, we describe the adaption for the lower bound algorithmm for Annotated Vertex-Weighted Block Modeling where a vertex can be in multiple packs. In this situation, a change on a vertex pair $\{u, v\}$ (Cases 2-4 and 6) can affect multiple packs since u and v can be in more than one common packs. The update can be done separately on the common packs. Note that in Case 5 multiple packs can contain v. In each pack of these packs, the search for a changed label has to be done.
Since the updates on the packing can prevent to find a better branching, we still recompute the packing in a fixed cycle like in every x-th branch node. Note that the update-process is also applicable for the unweighted problems. In Block Modeling only Case 2 has to be considered and in Annotated Block Modeling the Case 2 and Case 3 have to be considered.

5.7 Reduction Rules

In this section, we present some reduction rules for the annotated problems. A reduction rule transforms the current instance I to another I' such that I and I' are equivalent. Recall that every vertex pair of B cannot be edited and every vertex pair in A cannot be merged.

Reduction Rule 1. *In any instance of the annotated problems, mark every vertex pair where the resolving of all witnesses $\{u, v\}$ costs at least $k + 1$ as apart.*

Proof. Let $\{u, v\}$ be such a vertex pair. This rule is correct since it is not possible that u and v can be in one neighborhood class by a solution because the summed resolving cost of all witnesses is higher than k. □

The following reduction rule concerns merged vertex pairs in Annotated Block Modeling. After resolving all witnesses for a vertex pair, every merged vertex pair of M should have no witnesses in the resulting graph. In the resolving process it is possible that a merged vertex pair has witnesses in a recursive call. Consider the following example. Let u and v be two vertices where $\{u, v\}$ is already marked as merged. Let $\{x, y\}$ be the vertex pair to be merged and $\text{wit}(\{x, y\}) = \{u, v\}$. Let x be adjacent to $\{u, v\}$. In the algorithm it is possible that u is resolved by addition and v is resolved by deletion for $\{x, y\}$. Afterwards, $\{u, v\}$ has x and y as witnesses.

Reduction Rule 2. *Resolve every witness of every vertex pair that is merged in a graph of an instance of in* Annotated Block Modeling.

Proof. A vertex pair $\{u, v\}$ is marked if every witness is resolved in the algorithm. Therefore, every witness w in a recursive call is created by an edit of $E(w, \{u, v\})$. Without loss of generality, let $\{u, w\}$ be edited. Hence, the witness w can be resolved in at most one way since $\{u, w\}$ is already edited. If $\{v, w\}$ is also edited, the branch can be cut off. □

Reduction Rule 3. *In an instance of an unweighted problem, return false if there are $t + 1$ many neighborhood classes with a size of at least $k + 1$.*

Proof. This rule is a conclusion of Lemma 3.5. If there are $t + 1$ many neighborhood classes in the current graph with a size greater than k, then, in the resulting of the optimal solution E_{sol}, the vertices of these neighborhood classes are unaffected. Hence, there are at least $t + 1$ neighborhood classes in the resulting graph. This contradicts that E_{sol} is a solution. Therefore, the instance is a no-instance. □

The next reduction rule is only applicable for instances of the weighted problems. Moreover, the rule requires information of the *initial graph* G^*. The initial graph means the graph of the root of the branch tree.

Reduction Rule 4. *Merge every two vertices u and v of the same positive neighborhood class in the initial graph if $\ell(u) = \ell(v) = \oplus$.*
Analogue, merge every two vertices u and v of the same negative neighborhood class in the initial graph if $\ell(u) = \ell(v) = \ominus$.

Proof. This reduction is a conclusion from Lemma 3.3 and the note afterwards. This lemma states only for instances of Block Modeling. The proof for Vertex-Weighted Block Modeling is analog, it needs only few more arguments. First, note that if a vertex v is in a positive neighborhood class, then $\ell(v) \neq \ominus$. In the second part of the proof, the cost of the edit sets has to be considered instead of the number of edits. With these adjustments Lemma 3.3 is also applicable for the weighted problems.
Now, we prove the this reduction rule. Since two vertices u and v with $\ell(u) = \ell(v) = \oplus$ are in positive neighborhood classes in the resulting graph, the condition for this lemma are satisfied. Hence, there is an optimal solution where u and v are in a neighborhood class and they can be merged.
The argument for two vertices u and v with $\ell(u) = \ell(v) = \ominus$ is analog. □

5.8 Branching Rule

In the following, we present a branching rule for the unweighted problems. This branching rule is derived from Lemma 3.4 and can only be applied when the conditions of Lemma 3.4 are fulfilled. Again, we need the information of the initial graph G^*. Let $G = (V, E)$ be the considered graph of the current branch.

Branching Rule 1. *Let v be a vertex of V and let C_v be the neighborhood class of v. One vertex pair of $E(\{v\}, V \setminus C_v)$ has to be merged if*

1. *there is a neighborhood class C in G^* where some vertices are affected by the constructed and unfinished solution of the current branch and v of C is unaffected, or*
2. *the neighborhood class C_v is a real subset of a neighborhood class C in G^*.*

Proof. We proof this rule with a case distinction.
First, consider Case 1. Due to Lemma 3.4, we know that v has be to affected, otherwise the solution is not optimal. To affect v it has to be merged with

some other vertex of a different neighborhood class. Thus, among $E(\{v\}, V \setminus C_v)$ at least one vertex pair has to be merged.
Second, consider Case 2. Again, due to Lemma 3.4 the vertex v has to be in a neighborhood class with another vertex of $V \setminus C'$, otherwise the solution is not optimal. Thus, among $E(\{v\}, V \setminus C')$ at least one vertex pair has to be merged. □

For comparison, normally one vertex pair among $\binom{t+1}{2}$ vertex pairs has to be merged. Hence, this reduction rule reduces the number of branch nodes if $\binom{t+1}{2} > |E(\{v\}, V \setminus C_v)| = n - |C_v|$. We simplify the term $\binom{t+1}{2} > n - |C_v|$ to $|C_v| > n - \binom{t+1}{2}$. Thus, the branching rule improves the running-time if $|C_v|$ is big or if t is so big that $\binom{t+1}{2} \approx n$.

6 Evaluation

In this section, we evaluate our exact algorithms and our heuristics. We want to find the size of an optimal solution for a graph G and the desired number of neighborhood classes t.

6.1 Implementation Details

Our implementation[2] is written in Java with OpenJDK 14.0.1 using IntelliJ IDEA (Community Edition) in version 2020.1. To construct and solve our ILPs we used the Gurobi Optimizer[3] in version 9.5.0 under an academic license. The evaluation was done on an Intel(R) Xeon(R) Silver 4116 CPU 2.10GHz machine with 128GB RAM under Debian GNU/Linux 11 operating system.

We tested our algorithms on real-world social networks. We obtained the graph data of Mastrandrea et al. [27] from sociopatterns.org[4]. All other graphs are obtained through konect.cc[5] [22]. Table 2 shows the list of graphs we were using. For the graphs we use their abbreviations as noted in this table instead of their names. Note that the references are also stated in Table 2.

If a graph has selfloops, we ignore them. We also ignored the weight of edges in graphs. Directed graphs were transformed into undirected graphs by replacing direct edges with undirected ones. In a signed graph, each edge is labeled either with -1 or $+1$. The two labels represents mostly opposite relations. For example, $+1$ means the friendship of two vertices and -1 indicates an antipathy. Hence, we construct three graphs from a signed graph. One graph has only the positive edges, the other graph contains all negative edges and the third graph has all edges and ignores the labels. We added a "pos" or "neg" after the graph name to indicate the version. In bipartite graphs, the vertex set may not be homogeneous. For example, in the test-graph SC, a vertex represents either a person or a company. Some additional constraints can be set, like the restriction that a person and a company are not allowed to be in one neighborhood class. For our evaluation,

[2]The source code and the results can be found at `https://www.uni-marburg.de/en/fb12/research-groups/algorith/blockmodeling.zip`.

[3]see `https://www.gurobi.com/`

[4]`www.sociopatterns.org`

[5]`www.konect.cc/networks/`

we ignored the fact that a graph is bipartite and therefore, set no additional constraints.
Each algorithm has for a time limit of 30 minutes per instance. We test every graph of Table 2 with every $t \in \{2, 3, 4, 5, 10\}$.

6.2 Branch-and-Bound

In this section, we discuss our choices for the branch-and-bound algorithms. One way to obtain an optimal solution with a branch-and-bound algorithm, is to apply the algorithm with $I = (G, k', t)$ as input. Every time I is a no-instance, we increment k' until the updated I is a yes-instance. Later in this section, another way is described.
The first algorithm is the annotated version of Algorithm 1 for Block Modeling with Reduction Rule 2 and with both lower bound algorithms. To distinguish the algorithms we name this algorithm `bb` (**B**ranch and **B**ound). The second algorithm is the annotated version of Algorithm 2 for Vertex-Weighted Block Modeling with Reduction Rule 4, modified branching as described in Section 5.3 and with both lower bound algorithms. We name this algorithm `bbm` (**B**ranch and **B**ound **M**erge).
The second lower bound algorithm uses the version where a vertex can be in more than one pack. Recall that a pack P consists of $t + 1$ many vertices of distinct neighborhood classes and it is used to compute a lower bound since the witnesses of at least one vertex pair of $\binom{P}{2}$ have to resolved by a solution. This version has given better results in some individual tests. Even though in the other version, higher resolve cost can be achieved for a pack P, a vertex pair of $\binom{P}{2}$ with small resolve cost determines the resolve cost of P. We suppose that this is the reason why the version where a vertex can be in more than one vertex is better in many cases.
We name an algorithm `bbmnb` (**B**ranch and **B**ound **M**erge **N**ormal **B**ranching) that is like `bbm` but without the modified branching. To see the impact of the modified branching, we compare `bbm` with `bbmnb`. A comparison of the running-times and the recursive calls of these algorithms is shown in Figure 4. With increasing running-time, algorithm `bbm` is faster than `bbmnb` except for 5 instances and `bbm` is mostly twice as fast as `bbmnb`. This is also reflected in the number of recursive calls. Moreover, `bbm` solved one more instance within the time-limit. Therefore, we will not consider `bbmnb` in any other evaluations.
In the following, we will adapt `bb` and `bbm` by using heuristics.

Table 2: List of the graphs used for the evaluation. The number of vertices is stated in n. The number of edges is stated in m. The neighborhood diversity is stated in d.

graph	abbreviation	n	m	d
Club Membership [12]	BM	25	90	25
Contact Network Data [27]	CND	120	348	117
Contiguos USA [20]	CU	49	107	49
Corporate Leadership [2]	BC	24	86	22
Dolphins [25]	DO	62	159	60
Facebook known Pairs [27]	FP	156	4515	84
Facebook known Pairs neg	FPn	156	3078	152
Facebook known Pairs pos	FPp	156	1437	153
Friendship Network [27]	FN	134	406	123
Highland Tribes [28]	HT	16	58	16
Highland Tribes neg	HTn	16	29	16
Highland Tribes pos	HTp	16	29	12
HIV [1]	HI	40	41	34
Iceland [16]	IC	75	114	43
Kangaroos [14]	MK	17	91	14
Karate Klub [35]	ZA	34	78	29
South Africa Companies [19]	SC	6	8	6
Southern Women large [10]	SW	18	64	16
Southern Women small [10]	Sw	5	8	3
Taro Exchange [15, 30]	MT	22	39	21
Zebra [33]	MZ	27	111	17

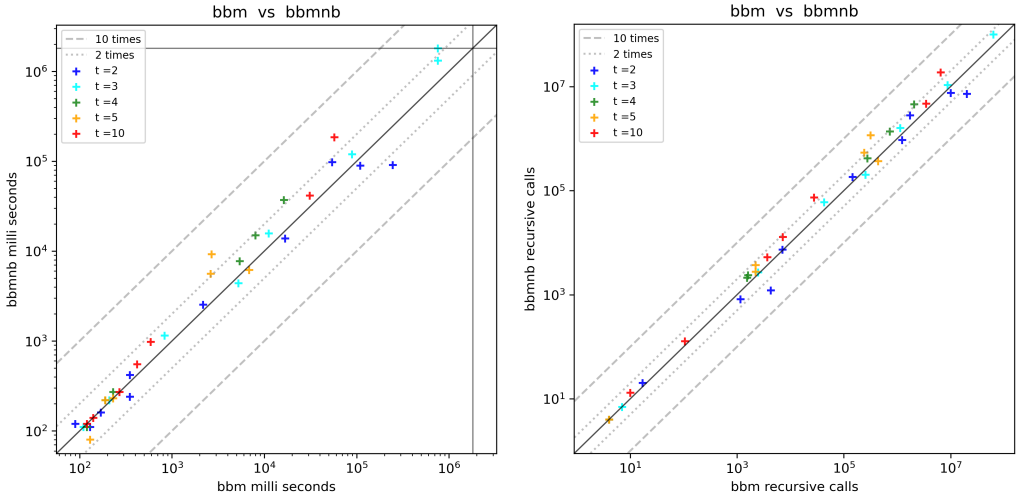


Figure 4: Comparison between `bbm` and `bbmnb`. The left plot compares the running-time and the right plot compares the number of recursive calls. Every data point represents the instance (G, t). The color of the data point indicates t. The axes are scaled logarithmically. Only instances that are solved within the time-limit of at least one algorithm are illustrated.
We describe on basis of the left plot how to read it. The x-axis indicates the time `bbm` needed to solve the instance. The y-axis indicates the time `bbmnb` needed to solve the instance. The vertical and horizontal black lines mark the time-limit. Instances on the black middle line are solved equally fast. The dashed and pointed lines show the proportion of the running-times. For example, a data point on the dotted line above the black line was solved 2 times faster by `bbm` in comparison to `bbmnb`.
The right plot reads analogously except that the axes indicate the number of recursive calls instead of the running-time.

Another way to obtain an optimal solution is to start with an upper bound k^* for the edit budget. Run the algorithm with $I = (G, k^*, t)$ as input. Decrement k^* if the algorithm returns *True* and repeat the process until the updated I is a no-instance. Then, $k^* + 1$ is the number of edits of an optimal solution.
For every instance, we compute 10 locally improved Split-Heuristics and 10 locally improved Merge-Heuristics and store the smallest upper bound in a lookup table.
To see the improvement through a heuristic, we create two more algorithms `bbh` and `bbmh` that start at the upper bound of the lookup table.
Since the first lower bound algorithm `lb1` runs faster than the second lower bound algorithm `lb2`, we first compute the lower bound r of `lb1`. If r is higher than the edit budget we cut the branch off. Otherwise, we compute the lower bound of `lb2`.

6.3 ILP-Formulations

In this section, we state which ILP-formulations we used for the evaluation. One algorithm is based on the first ILP-formulations as stated in Section 4.3. We name this algorithm `ilp1`. In `ilp1`, the transitivity constraints and the merge constraints are added via callbacks. We add every transitivity constraint that is violated in a callback. If no transitivity constraint is added, then we add one violated merge constraint in the callback. This combination had the best results in preliminary experiments.
The next algorithm `ilp2` is based on the second ILP-formulation of Section 4.3. In `ilp2`, we add a maximum of 9 constraints in a callback. The running-time is very sensitive to this number. Just small adjustments can worsen the running-time by a factor of 2 or 3. Moreover, the maximal number of added constraints may be different for different instances. Maybe the added number of constraints should depend on other criteria. Still, for us seem 9 to be a valid compromise based on preliminary experiments.
We created two algorithms `ilp1h` and `ilp2h` where the upper bound of a heuristic is given as a constraint. In Gurobi it is possible to set the initial values of the variables. Since the heuristics partition the vertices into at most t blocks, we use this information and set the start variables such that every two vertices of a block would be in a neighborhood class.
The heuristic for `ilp1h` and `ilp2h` is computed by taking the best partition of 5 locally improved Merge-Heuristic results and 5 locally improved results

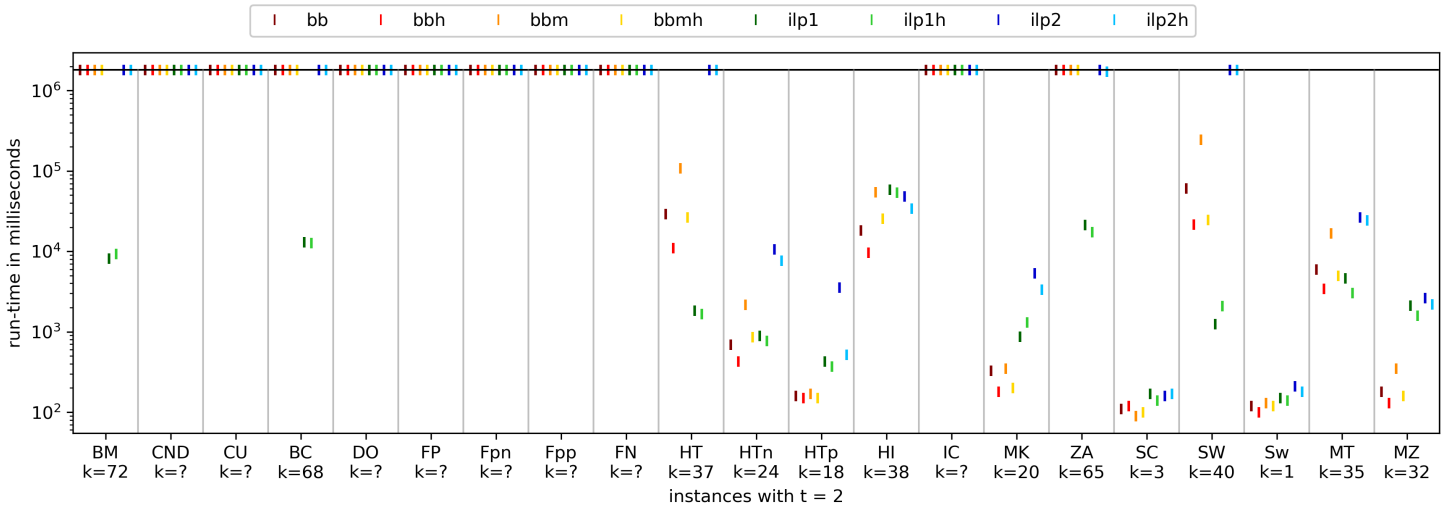
Table 3: Table of solved instances for each algorithm. An entry indicates the number how many instances an algorithm solved with a specific t. The last row is the sum of the above numbers. Note that trivial instances are also counted. The test-graph SC is trivial with $t = 10$ and the test-graph Sw is trivial with $t \geq 3$.

algorithm	`bb`	`bbh`	`bbm`	`bbmh`	`ilp1`	`ilp1h`	`ilp2`	`ilp2h`
$t = 2$	10	10	10	10	13	13	8	9
$t = 3$	10	10	9	9	12	12	5	5
$t = 4$	8	8	7	8	9	10	4	4
$t = 5$	7	8	7	7	8	9	2	4
$t = 10$	10	10	9	9	7	7	3	4
sum	45	46	42	43	49	51	22	26

of Split-Heuristic.

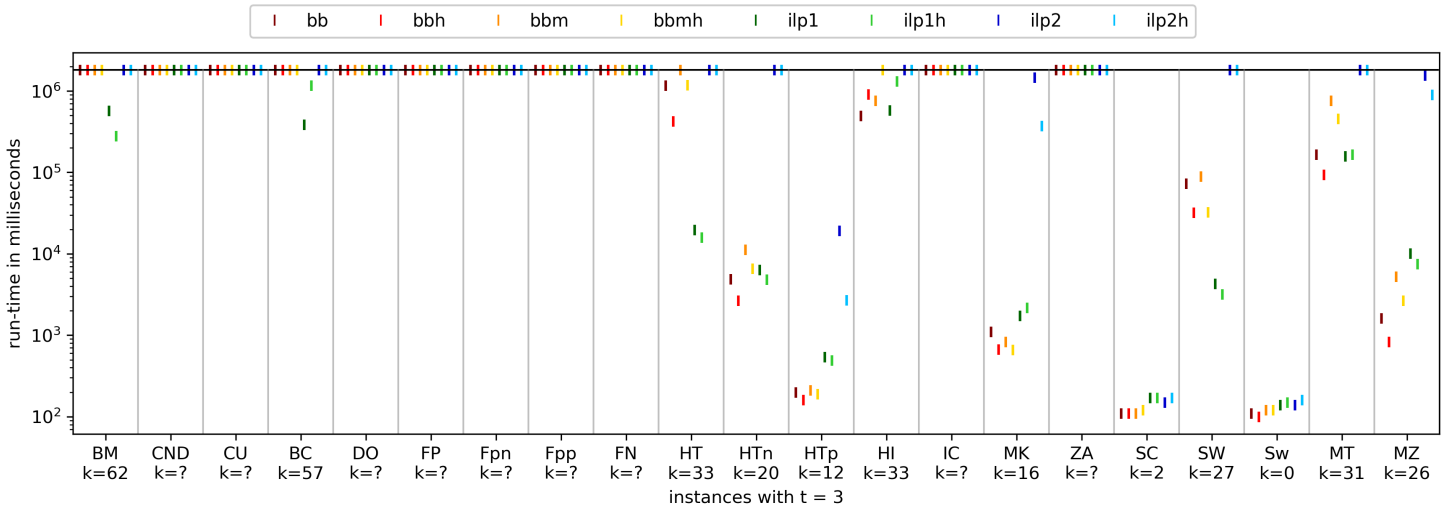
6.4 Algorithm Comparison

In this section, we compare the exact algorithms. We tried to find the size of the optimal solution for every $t \in \{2, 3, 4, 5, 10\}$ and for every graph of Table 2. The time-limit was set to 30 minutes for each instances and each algorithm. Note that some instances are trivial since the test-graph Sw has a neighborhood diversity of 3 and the test-graph SC has a neighborhood diversity of 6.
Table 3 gives an overview how many instances each algorithm could solve within 30 minutes. In Figure 5, the running-time comparison for each $t \in \{2, 3, 4, 5, 10\}$ is illustrated.
In the following, we describe our observations. In general, the algorithms `ilp2` and `ilp2h` have the worst performance since they could not solve a significant number of instances in comparison to the other algorithms.
The use of the heuristics improved the running-time of each algorithm in most cases. One exception is graph HI with $t = 3$. The branch-and-bound algorithms have worse running-times on this graph when using a heuristic. In this case, the upper bound was 40 which is percentually significant above the size of the optimal solution 33. Another exception is the graph MT with $t = 4$.

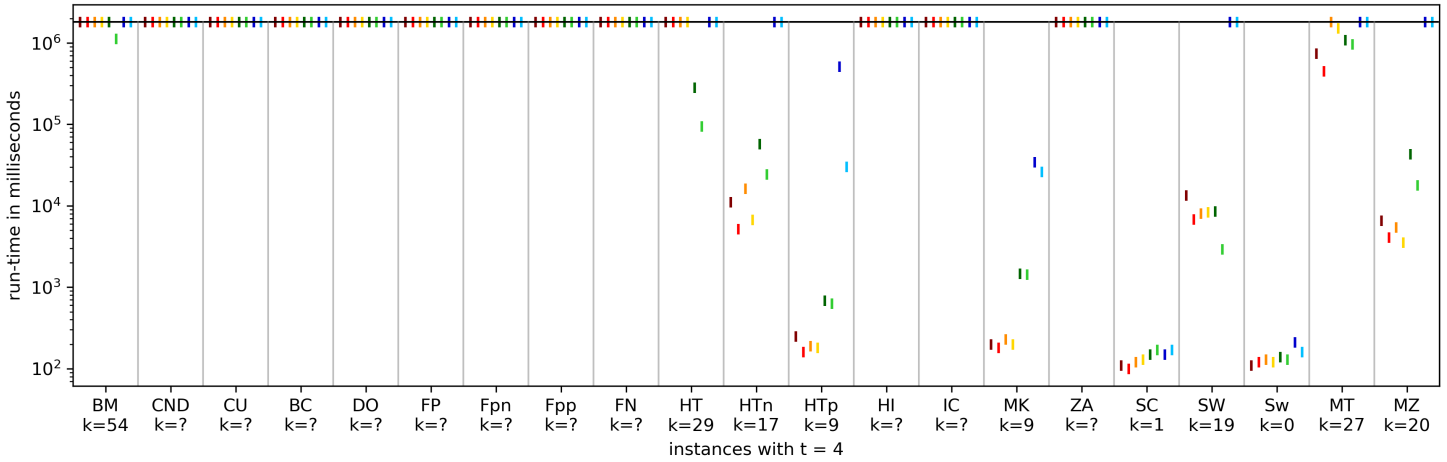


(a) Running-time comparison with $t = 2$

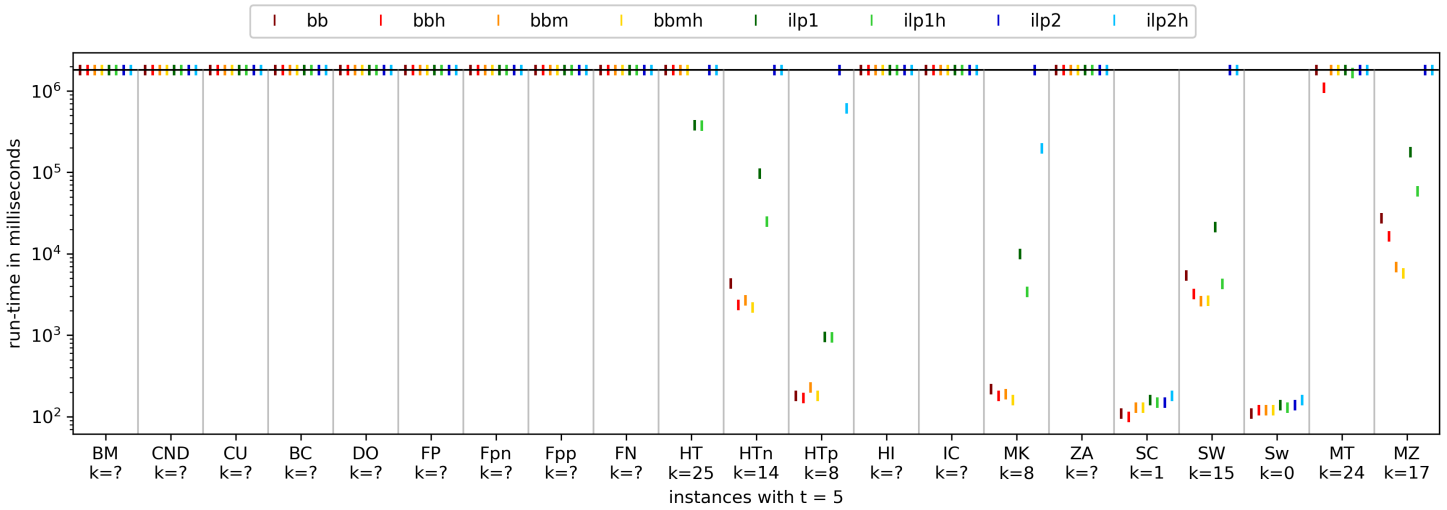
Figure 5: These five plots compare the algorithms for each t. Each vertical line on the graph is the running-time result of an algorithm. The color indicates the algorithm. The instances, represented by their abbreviations, are on the x-axis. Below an abbreviation, the size of the optimal solution is written if it is known. The y-axis indicates the running-time in milliseconds. Note that the y-axes is scaled logarithmically. The upper black thin line marks the time-limit.



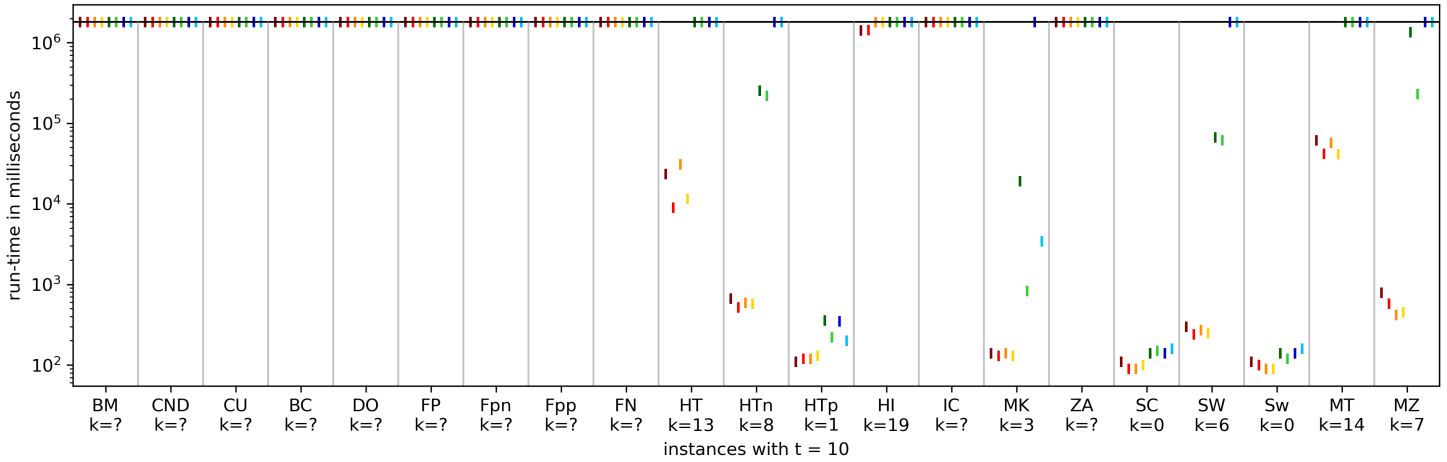
(b) Running-time comparison with $t = 3$



(c) Running-time comparison with $t = 4$



(d) Running-time comparison with $t = 5$



(e) Running-time comparison with $t = 10$

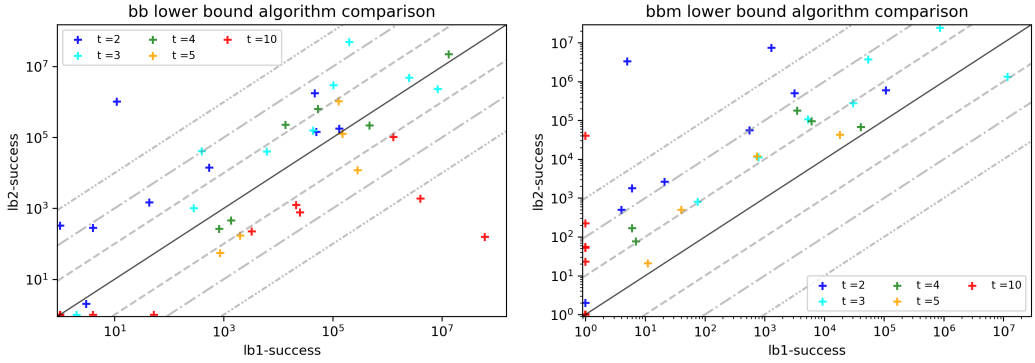


Figure 6: Comparison of the number of cut offs due to a lower bound algorithm. Each point represents an instance. The x-value of an instance indicates how often `lb1` cut off a branch The y-value of an instance indicates how often `lb2` cut off a branch Both axes are scaled logarithmically. The color of an instance indicates t. In instances on the black middle line, both lower bound algorithm cut off the same number of branches. The gray lines mark the proportions of the number of cut offs. The first gray lines mark a proportion of factor 10. The second gray lines mark a proportion of factor 100. The third gray lines mark a proportion of factor 1000. If some lower bound algorithm did not cut off any branch, we set its success number to 1 such that the relation can still be illustrated. For example, with $t = 10$ the lower bound algorithm `lb1` in `bbm` did not cut off a branch in any instance.

The running-times of the ILP algorithms scale stronger with increasing t on the same instance in comparison to the branch-and-bound algorithms. This can be seen in the running-time comparison and with the number of solved instances in Table 3 as well.
The algorithms `bb` and `bbh`, that are based on Block Modeling, benefit more from the first lower bound algorithm `lb1`. Let `lb1`-success be the number of times when `lb1` cut a branch off and let `lb2`-success be this number for `lb2`. We illustrated these numbers for every solved instance for `bb` and `bbm` in Figure 6. We omit the illustration for `bbh` and `bbmh` since they are very similar. With $t \in \{2, 3\}$ `lb2`-success is larger than `lb1`-success for algorithm `bb`. But with increasing t, `lb1`-success is larger than `lb2`-success for algorithm `bb`. The number `lb2`-success of algorithm `bbm` is in nearly all instances larger than `lb1`-success.
Altogether, `ilp1h` solves the most instances, especially if t is small. With

increasing t, the branch-and-bound algorithms are usually faster. Moreover, it seems like they can solve more instances with a bigger t. Overall, `ilp2` and `ilp2h` have the worst performance. The unweighted algorithms `bb` and `bbh` are slightly better than their weighted variants.

6.5 Heuristics Evaluation

In this section, we compare the heuristic upper bounds with the size of optimal solutions and later, we compare Split-Heuristic with Merge-Heuristic. We computed 100 Merge-Heuristic results and 100 Split-Heuristic results for each instance. Then, we locally optimized them.
There are possibly multiple block pairs that have the same cost increase in a merge process of Merge-Heuristic. Analogously, there are possibly multiple blocks that have the same cost increase for the split cost computation. Both heuristics are randomized when a tie occurs. Note that in Split-Heuristic the precomputated cost increase is not always the actual cost increase. This is described in Section 4.2.3.
Table 4 shows the best result and the *average result* shown for each instance. The average result is the average over the 200 locally improved results. Moreover, there are also the optimal solution sizes listed if any algorithm solved the instance within the time-limit. For every instance with $t \in \{3, 4, 5, 10\}$ the optimal solution size was found with a heuristic. And for the instances with $t = 2$ the optimal solution was found in 92.31% of the cases.
In the following, we average the heuristic upper bounds and compare them to the optimal solution size. The average percentual difference between the optimal solution size and the average solution of non-trivial instances is 4.75% for $t = 2$, 5.24% for $t = 3$, 6.27% for $t = 4$, 3.75% for $t = 5$, and 8.75% for $t = 10$. The individual differences between the average heuristic result size versus the optimal solution size is illustrated in Figure 7. As seen in Figure 7, most of the average heuristic results are at most 10% away from the optimal solution size.
At last, we compare Merge-Heuristic with Split-Heuristic. For this, we use the 100 results of the heuristics without the local optimization. In Figure 8, the average results of Merge-Heuristic and Split-Heuristic are illustrated. Based on the average upper bounds, a clear winner cannot be identified since the difference lies within 10% most of the time. Within the result size of $(500, 3500)$, it seems that Split-Heuristic yields generally smaller upper bounds. On the other hand, within the range $(250, 40)$ Merge-Heuristic has

Table 4: Overview of the instances with their optimal solution size, the best heuristic upper bound and the average upper bound of our heuristic experiment.

t	2			3			4		
graph	opt	best	avg	opt	best	avg	opt	best	avg
BM	72	72	77.74	62	62	65.01	54	54	59.09
CND	-	334	347.44	-	320	339.06	-	307	326.3
CU	-	101	103.22	-	96	98.69	-	91	94.91
BC	68	68	70.89	57	57	57.11	-	52	53.36
DO	-	148	157.46	-	135	148.12	-	122	136.48
FP	-	2878	2901.37	-	2230	2282.0	-	2017	2025.02
Fpn	-	2542	2783.12	-	2230	2236.24	-	2074	2140.96
Fpp	-	1280	1295.7	-	1133	1196.86	-	991	1007.51
FN	-	368	396.01	-	343	366.88	-	321	342.54
HT	37	37	38.38	33	33	34.3	29	29	30.2
HTn	24	24	27.03	20	20	21.2	17	17	18.04
HTp	18	18	18.0	12	12	12.0	9	9	9.8
HI	38	40	40.95	33	33	39.92	-	30	33.43
IC	-	106	111.84	-	90	107.66	-	85	91.05
MK	20	20	20.0	16	16	17.83	9	9	9.0
ZA	65	65	71.81	-	57	60.99	-	44	44.1
SC	3	3	3.16	2	2	2.04	1	1	1.19
SW	40	40	40.1	27	27	27.0	19	19	19.04
Sw	1	1	1.0	0	0	0.0	0	0	0.0
MT	35	35	38.28	31	31	33.57	27	27	28.92
MZ	32	32	32.0	26	26	26.0	20	20	20.3

t	5			10		
graph	opt	best	avg	opt	best	avg
BM	-	51	55.51	-	36	37.95
CND	-	300	313.25	-	259	266.06
CU	-	87	90.88	-	68	71.78
BC	-	48	50.44	-	33	35.01
DO	-	116	124.05	-	96	100.27
FP	-	1688	1696.66	-	1129	1154.0
Fpn	-	1971	2051.9	-	1616	1647.74
Fpp	-	913	913.91	-	718	725.75
FN	-	301	321.37	-	251	259.79
HT	25	25	26.83	13	13	13.74
HTn	14	14	15.06	8	8	8.18
HTp	8	8	8.0	1	1	1.36
HI	-	29	31.94	19	19	21.13
IC	-	77	81.72	-	57	58.48
MK	8	8	8.4	3	3	3.07
ZA	-	38	38.35	-	25	25.38
SC	1	1	1.0	0	0	0.0
SW	15	15	15.0	6	6	6.43
Sw	0	0	0.0	0	0	0.0
MT	24	24	25.19	14	14	14.03
MZ	17	17	17.88	7	7	7.36

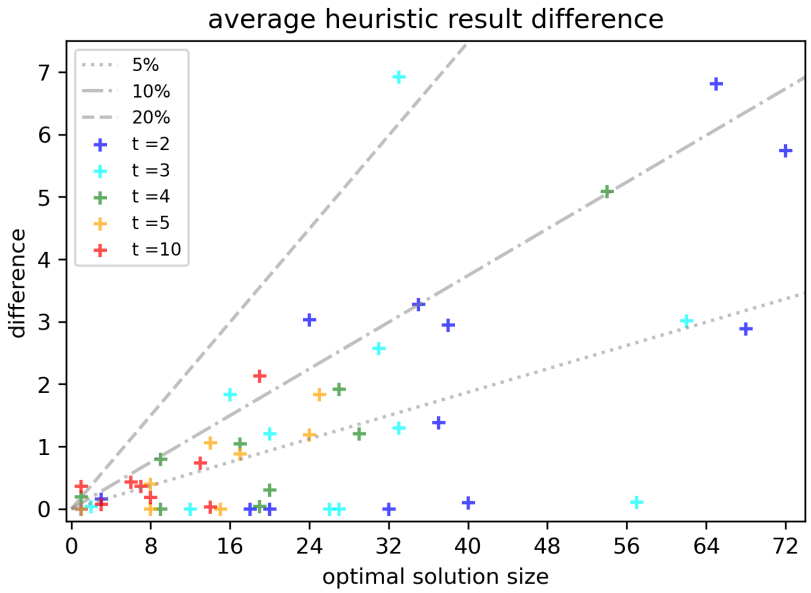


Figure 7: Each data point represents an instance. The color indicates t of the instance. The x-value of an instance is its known optimal solution size and the y-value is the difference between the average heuristic result and the optimal solution size. The gray lines mark the percentage difference. For example, assume that $(40|4)$ is a data point in this plot. This point would be on the 10% difference line. The average heuristic result would be $40+4=44$.

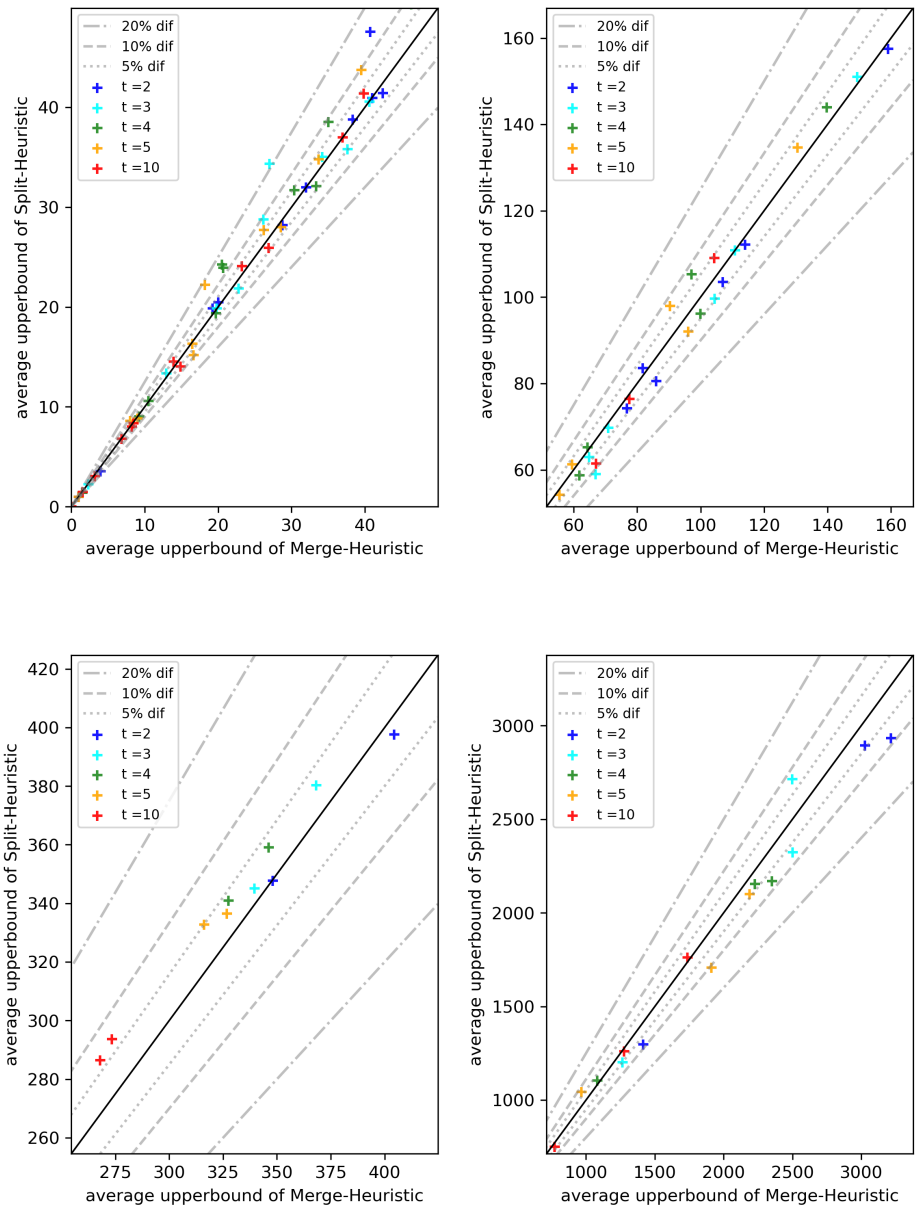


Figure 8: These four plots show the difference of the average upper bound over 100 Split-Heuristic results with the average upper bound over 100 Merge-Heuristic results for each instance in different ranges. Each data point represents an instance. The color indicates the parameter t. The lines mark the percentage difference for an instance.

the smaller average result. Since both heuristics are not time-consuming, both upper bounds can be computed in practice for an instance.

7 Future Work

The first part of this section is occupied with the improvements ideas for the branch-and-bound algorithms. The second part provides further research ideas.
Currently, the algorithm `ilp1h` solves the most instances. We think that the branch-and-bound algorithms have the potential to solve as many instances as the ILP-formulations or even more. The best branch-and-bound algorithms solves 10% less instances than `ilp1h`. We think that the branch-and-bound algorithms can be improved significantly with better heuristics for the choice of the branching vertices and for the packing.
In the following, we describe two possible ways to improve the running-time of the branch-and-bound algorithms. Recall, that `lb2` creates a packing which consists of disjoint vertex sets called packs. A pack consists of vertices of distinct neighborhood classes. For each pack, we can charge costs for the lower bound. In our experiments, the lower bound algorithm `lb2` causes a large part of the running-time. Without the dynamic update of the packing by an edit, the search of the new packs causes more than 90% of the running-time. To speed this up, an algorithm could consider a graph G' that has the same vertices as the input graph G. In G', two vertices are adjacent if they are apart or not mergeable in G. The search time for new packs could decrease by using G': When constructing a pack P, vertices that are apart or not mergeable with some vertex of P can be prioritized. These vertices can be found in quickly using the adjacency lists G'.
An idea for a new lower bound would be to use a packing where the *witnesses* of each pack are disjoint. The witnesses of a pack P are defined as $\bigcup_{p \in \binom{P}{2}} \text{wit}(p)$. The correctness is not shown yet. The idea of its proof is given by the following argumentation. Since every edit that is necessary to resolve a pack is disjoint with every edit that resolves an other pack, the lower bound of this approach is the sum of the resolving costs. Note that in the second lower bound algorithm `lb2`, the sum is divided by 2. This division has not to be done with the new lower bound. Hence, the new lower bound algorithm could give better results than `lb2` in some cases.

We evaluated our algorithms on real-world social networks. A point of interest is to analyze the goodness of a clustering obtained by Block Modeling. In general, it is interesting to find new approaches to solve Block Model-

ING.

In this work, we dealt with undirected and simple graphs. Variants of Block Modeling for directed, edge-weighted or bipartite graphs are also possible. In these variants, the notion of neighborhood class has to be redefined. It may be possible to adapt some of our results to these variants.

Another point of interest would be to find the optimal block modeling without specifying the number of clusters. This means we do not demand a maximum neighborhood diversity of t in the definition of the problem. If no upper bound for the neighborhood diversity is set, then the optimal solution would edit nothing since the input neighborhood diversity is also acceptable. This would give a useless clustering. One way to fix this issue is to introduce a penalty function f that charges costs for large neighborhood diversities. In this variant, the edit cost plus the penalty of f of the resulting graph have to be minimized.

Some clustering procedures, like DBSCAN [11], detect outliers. An outlier is an element that does not fit in any cluster of the clustering. The condition for an outlier is defined for each clustering method individually. It would be of interest to find such a definition of outliers for Block Modeling. The problem definition could then be adapted to disregard the outliers when computing.

References

[1] David M. Auerbach, William W. Darrow, Harold W. Jaffe, and James W. Curran. Cluster of cases of the acquired immune deficiency syndrome: Patients linked by sexual contact. *The American Journal of Medcine*, 76(3):487–492, 1984.

[2] Roy Barnes and Tracy Burkett. Structural redundancy and multiplicity in corporate networks. *International Network for Social Network Analysis*, 30(2), 2010.

[3] Vladimir Batagelj, Anuška Ferligoj, and Patrick Doreian. Direct and indirect methods for structural equivalence. *Social Networks*, 14(1-2):63–90, 1992.

[4] Sebastian Böcker and Jan Baumbach. Cluster editing. In Paola Bonizzoni, Vasco Brattka, and Benedikt Löwe, editors, *The Nature of Computation. Logic, Algorithms, Applications - 9th Conference on Computability in Europe, CiE 2013, Milan, Italy, July 1-5, 2013. Proceedings*, volume 7921 of *Lecture Notes in Computer Science*, pages 33–44. Springer, 2013.

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

[6] Ronald L. Breiger, Scott A. Boorman, and Phipps Arabie. An algorithm for clustering relational data with applications to social network analysis and comparison with multidimensional scaling. *Journal of Mathematical Psychology*, 12(3):328–383, 1975.

[7] Ronald S. Burt. Positions in networks. *Social Forces*, 55(1):93–122, 1976.

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

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

[10] Allison Davis, Burleigh B. Gardner, and Mary R. Gardner. *Deep South; a Social Anthropological Study of Caste and Class*. The University of Chicago Press, 1941.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, pages 226–231. AAAI Press, 1996.

[12] Katherine Faust. Centrality in affiliation networks. *Social Networks*, 19(2):157–191, 1997.

[13] Zoubin Ghahramani. Unsupervised learning. In Olivier Bousquet, Ulrike von Luxburg, and Gunnar Rätsch, editors, *Advanced Lectures on Machine Learning, ML Summer Schools 2003, Canberra, Australia, February 2-14, 2003, Tübingen, Germany, August 4-16, 2003, Revised Lectures*, volume 3176 of *Lecture Notes in Computer Science*, pages 72–112. Springer, 2003.

[14] T. R. Grant. Dominance and association among members of a captive and a free-ranging group of grey kangaroos (*Macropus giganteus*). *Animal Behaviour*, 21(3):449–456, 1973.

[15] Per Hage and Frank Harary. *Structural Models in Anthropology*. Cambridge Univ. Press, 1983.

[16] Sigridur Haraldsdottir, Sunetra Gupta, and Roy M. Anderson. Preliminary studies of sexual networks in a make homosexual community in Iceland. *Journal of Acquired Immune Deficiency Syndromes*, 5(4):374–381, 1992.

[17] Charles A. R. Hoare. Algorithm 64: Quicksort. *Communications of the ACM*, 4(7):321, 1961.

[18] Charles A. R. Hoare. Algorithm 65: Find. *Communications of the ACM*, 4(7):321–322, 1961.

[19] John Atkinson Hobson. *The Evolution of Modern Capitalism: A Study of Machine Production*. Walter Scott, 1919.

[20] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 0: Introduction to Combinatorial and Boolean Functions*. Addison-Wesley, 2008.

[21] Ivan Kovác, Ivana Selecéniová, and Monika Steinová. On the clique editing problem. In Erzsébet Csuhaj-Varjú, Martin Dietzfelbinger, and Zoltán Ésik, editors, *Mathematical Foundations of Computer Science 2014 - 39th International Symposium, MFCS 2014, Budapest, Hungary, August 25-29, 2014. Proceedings, Part II*, volume 8635 of *Lecture Notes in Computer Science*, pages 469–480. Springer, 2014.

[22] Jérôme Kunegis. KONECT – The Koblenz Network Collection. In *Proc. Int. Conf. on World Wide Web Companion*, pages 1343–1350, 2013.

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

[24] Francois Lorrain and Harrison C. White. Structural equivalence of individuals in social networks. *The Journal of Mathematical Sociology*, 1(1):49–80, 1971.

[25] David Lusseau, Karsten Schneider, Oliver J. Boisseau, Patti Haase, Elisabeth Slooten, and Steve M. Dawson. The bottlenose dolphin community of Doubtful Sound features a large proportion of long-lasting associations. *Behavioral Ecology and Sociobiology*, 54:396–405, 2003.

[26] James MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on Mathematical Statistics and Probability*, volume 1, pages 281–297. Oakland, CA, USA, 1967.

[27] Rossana Mastrandrea, Julie Fournet, and Alain Barrat. Contact patterns in a high school: a comparison between data collected using wearable sensors, contact diaries and friendship surveys. *PLOS ONE*, 10(9):e0136497, 2015.

[28] Kenneth E. Read. Cultures of the Central Highlands, New Guinea. *Southwestern Journal of Anthropology*, 10(1):1–43, 1954.

[29] Alexander Schrijver. *Theory of Linear and Integer Programming*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1999.

[30] Eric Schwimmer. *Exchange in the social structure of the Orokaiva: traditional and emergent ideologies in the Northern District of Papua*. Angus & Robertson, 1973.

[31] Ron Shamir, Roded Sharan, and Dekel Tsur. Cluster graph modification problems. In Ludek Kucera, editor, *Graph-Theoretic Concepts in Computer Science, 28th International Workshop, WG 2002, Cesky Krumlov, Czech Republic, June 13-15, 2002, Revised Papers*, volume 2573 of *Lecture Notes in Computer Science*, pages 379–390. Springer, 2002.

[32] Brandon Sorg. A new graph data structure. Technical report, The Ohio State University, 2010.

[33] Siva R. Sundaresan, Ilya R. Fischhoff, Jonathan Dushoff, and Daniel I. Rubenstein. Network metrics reveal differences in social organization between two fission–fusion species, grevy's zebra and onager. *Oecologia*, 151(1):140–149, 2007.

[34] Jay Wengrow. *A Common-Sense Guide to Data Structures and Algorithms*. Pragmatic Bookshelf, 2020.

[35] Wayne Zachary. An information flow model for conflict and fission in small groups. *Journal of Anthropological Research*, 33:452–473, 1977.