A Branch-and-Price Framework for Decomposing Graphs into Relaxed Cliques

Timo Gschwind*,[a], Stefan Irnich[a], Fabio Furini[b], Roberto Wolfler Calvo[c]

[a] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*
[b] *LAMSADE, Université Paris Dauphine, Place du Maréchal de Lattre de Tassigny, F-75016 Paris, France.*
[c] *LIPN, Université Paris 13, F-93430 Villetaneuse, France.*

Abstract

We study the family of problems of partitioning and covering a graph into/with a minimum number of relaxed cliques. Relaxed cliques are subset of vertices of a graph for which a clique-defining property is relaxed, e.g., the degree of the vertices, the distance between the vertices, the density of the edges, or the connectivity between the vertices. These graph partitioning and covering problems have important applications in many areas such as social network analysis, biology, and disease spread prevention. We propose a unified framework based on branch-and-price techniques to compute optimal decompositions. For this purpose, new effective pricing algorithms are developed and new branching schemes are invented. In extensive computational studies, we compare several algorithmic designs, e.g., structure-preserving versus dichotomous branching and their interplay with different pricing algorithms. The finally chosen setup for the branch-and-price produces results that demonstrate the effectiveness of all components of the newly developed framework and the validity of our approach when applied to social network instances.

Key words: Graph decomposition, clique relaxations, branch-and-price algorithm, social networks

1. Introduction

Given a graph $G = (V, E)$, where V is the set vertices and E is the set of edges, we study the problem of decomposing G into the minimum number of *relaxed cliques* (RCs). *Cliques* are subsets $S \subseteq V$ that induce subgraphs $G[S]$ which are complete and therefore have the following structural properties: all vertices have the maximum degree $|S| - 1$ in $G[S]$, the distance between any two vertices is minimal (=1), $G[S]$ has maximum density (=1), and $G[S]$ can not be disconnected by removing vertices. RCs are also subsets S of vertices but one or several of the clique-defining properties are relaxed, i.e., the *degree* of the vertices, the *distance* between the vertices, the *density* of the edges, or the *connectivity* between the vertices (see Section 2 for a formal definition). RCs have been object of intense research in the last decade, see the recent survey of Pattillo *et al.* (2013a). These families of clique relaxations are particularly important in social network analysis where they have been used to identify (large) communities (Balasundaram *et al.*, 2011). RCs are also important in several other domains (see Fortunato, 2010, for a more detailed discussion), such as in disease transmission analysis (Cook *et al.*, 2007), to identify large protein interaction (Yu *et al.*, 2006), and in biology (Almeida and Carvalho, 2012). Other applications were mentioned in (Pattillo *et al.*, 2013a; Fortunato, 2010; Fortunato and Hric, 2016).

Decomposing a graph into the minimum number of cliques is the classical *vertex coloring problem* (VCP) in the complement graph $\overline{G}$ of G, i.e., the problem of coloring the vertices using the minimum number of colors in such a way that adjacent vertices of $\overline{G}$ receive different colors. Vertex coloring has applications

*Corresponding author.
Email address: gschwind@uni-mainz.de (Timo Gschwind)

including scheduling (Lotfi and Sarin, 1986), timetabling (de Werra, 1985), frequency assignment (Gamst, 1986), register allocation (Chow and Hennessy, 1990), and when designing communication networks Woo *et al.* (2002) (see Malaguti and Toth (2010)). The VCP is very challenging from the computational viewpoint. The most effective exact algorithms for VCP are based on set-partitioning formulations where variables are associated to cliques of $\overline{G}$. The recent VCP algorithms by Malaguti *et al.* (2011), Gualandi and Malucelli (2012), and Held *et al.* (2012a) are all highly sophisticated implementations of branch-and-price algorithms.

In the present manuscript, we design an exact framework to effectively tackle new generalizations of the VCP, i.e., the family of problems of decomposing a graph into the minimum number of RCs. These problems arise naturally and directly in social network analysis and community detection. In the companion paper (Gschwind *et al.*, 2017), we have shown that decomposing a social network into RCs constitutes a valid alternative to established community detection methods. Indeed, the usefulness of RC-based decompositions has been validated in (Gschwind *et al.*, 2017) via in-depth analyses of some very prominent real-world social networks. These networks have been discussed and analyzed by dozens of researchers (see, e.g., Fortunato, 2010), because a true decomposition into communities is known in these cases, so that decompositions produced by different community detection methods can be compared in the light of the true community structures.

Our companion paper (Gschwind *et al.*, 2017) presents compact formulations for identifying the minimum number of RCs necessary to decompose a graph. Unfortunately, solving such compact formulations with a MIP solver constitutes a viable approach only for rather small-sized graphs. As for the VCP, it is straightforward to instead write down set-covering and set-partitioning formulations. Herein, each variable represents feasible RC so that already for small graphs these models can only be solved using column-generation techniques (Desaulniers *et al.*, 2005). Both the set-covering and the set-partitioning formulations bear advantages compared to the compact models used in (Gschwind *et al.*, 2017) due to their tighter linear relaxations and the fact that the inherent symmetries are eliminated. Finally, a major difference between the VCP and our decomposition problems should be pointed out: covering and partitioning a graph are two distinct problems in the case of non-hereditary RCs, i.e., when for an arbitrary RC S a subset $S' \subset S$ is not necessarily a RC. In (Gschwind *et al.*, 2017) it is also shown that additional connectivity constraints, which seem rather natural for a meaningful community structure, can make connected RCs non-hereditary.

We design a unified branch-and-price framework to compute optimal solutions for the family of RC-based decomposition problems. In particular, we discuss the most important ingredients of branch-and-price algorithms which have to be tailored to the problem at hand, i.e., the efficient solution of pricing problems and effective branching schemes. As branching decisions can destroy the structure of an original pricing problem, we also discuss and evaluate these algorithmic components when used in combination.

The remainder of this paper is organized as follows: The short Section 2 provides an overview over clique relaxations. The graph decomposition problems are formally stated and their classification into 17 families is provided in Section 3. Related to the branch-and-price solution approach, the pricing problems are identified as maximum-weight RC problems in Section 4. We review the related literature and present two new algorithms for solving specific pricing problems: The first is a new combinatorial branch-and-bound algorithm for the maximum-cardinality and maximum-weight s-club problem. The second is an adapted Russian-doll-search algorithm for hereditary RCs able to handle additional connectivity constraints and negative coefficients in the objective function. In addition, a strengthened model for γ-quasi-cliques is presented. Branching schemes are proposed in Section 5, where we compare branching decisions that preserve the structure of the pricing problems with the classical Ryan-Foster branching scheme. Moreover, a first branching scheme for proper set-covering formulations with non-hereditary subsets is presented. In Section 6, an extensive computational analysis on instances from the literature as well as newly generated networks demonstrates the effectiveness of all the new components of our branch-and-price framework. Final conclusions and an outlook in Section 7 close the paper.

2. Relaxed Cliques

Following (Pattillo *et al.*, 2013a; Gschwind *et al.*, 2017), we briefly summarize the definitions of the different families of RCs considered here. For any subset $S \subseteq V$, the vertex-induced subgraph of S is

$G[S] = (S, E \cap (S \times S))$. A set S is a *clique* if $G[S]$ is complete, i.e., all vertices are adjacent. Vertices adjacent to a vertex $i \in V$ are denoted by $N(i)$. The *vertex degree* of i is $|N(i)|$ and is denoted by $\deg_G(i)$. The *minimum vertex degree* of G is $\delta(G) = \min_{i \in V} \deg_G(i)$. For two vertices $i, j \in V$, $\text{dist}_G(i, j)$ is the minimum *distance* between i and j, i.e., the minimum length of an i-j-path in G (counting the number of edges). Note that $\text{dist}_G(i, j) = \infty$ if i and j are disconnected in G. The maximum distance is the diameter of G given by $\text{diam}(G) = \max_{i,j \in V} \text{dist}_G(i, j)$. For any $S \subseteq V$, the edge set $E(S)$ is the set of edges of G with both endpoints in S. Moreover, the *edge density* of a subgraph $G[S]$ is defined as $\rho(G[S]) = |E(S)|/\binom{|S|}{2}$. A set $C \subset V$ is a *vertex cut* of a connected graph $G = (V, E)$ if $G[V \setminus C]$ is a disconnected graph. The vertex connectivity $\kappa(G)$ is the size of a minimum vertex cut.

Cliques S form extreme subsets, since all vertices have maximum degree $|S| - 1$, the distance between any two vertices is 1, $G[S]$ has the maximum density of 1, and $G[S]$ has vertex connectivity $\kappa(G[S]) = |S| - 1$. Relaxing one of these four properties, one obtains two families of clique relaxations each, either based on degree (*core* and *plex*), distance (*clique* and *club*), density (*quasi-clique* and *defective clique*), and connectivity (*block* and *bundle*), respectively. In Table 1, we give the formal definitions of the eight first-order families of RCs considered in this paper (higher-order RCs result from relaxing several properties simultaneously, see Pattillo *et al.*, 2013a). The defining parameters $k \in \mathbb{N}$ impose an *absolute* lower bound on the degree or connectivity, parameters $s \in \mathbb{N}$ (also $s = 0$ for defective clique) bound the maximum distance from above and bound degree, density, and connectivity relative to the size of S from below, and parameters $\gamma \in (0, 1]$ impose a lower bound on the density.

Type of relaxation	Definition	Based on	Clique	Hereditary	Connected
k-core	$\delta(G[S]) \geq k$	Degree	$k = \lvert S\rvert - 1$	no	$\lvert S\rvert \leq 2k + 1$
s-plex	$\delta(G[S]) \geq \lvert S\rvert - s$	Degree	$s = 1$	yes	$\lvert S\rvert \geq 2s - 1$
s-clique	$\text{dist}_G(i, j) \leq s$ $(i, j \in S)$	Distance	$s = 1$	yes, weakly	$s = 1$
s-club	$\text{diam}(G[S]) \leq s$	Distance	$s = 1$	no	always
γ-quasi-clique	$\rho(G[S]) \geq \gamma$	Density	$\gamma = 1$	no	$\left\lceil \gamma\binom{\lvert S\rvert}{2} - \binom{\lvert S\rvert - 1}{2} \right\rceil \geq 1$
s-defective clique	$\lvert E(G[S])\rvert \geq \binom{\lvert S\rvert}{2} - s$	Density	$s = 0$	yes	$\lvert S\rvert \geq s + 2$
k-block	$\kappa(G[S]) \geq k$	Connectivity	$k = \lvert S\rvert - 1$	no	always
s-bundle	$\kappa(G[S]) \geq \lvert S\rvert - s$	Connectivity	$s = 1$	yes	$\lvert S\rvert \geq s + 1$

Table 1: Definition of different first-order clique relaxations, similar to Table 1 in (Gschwind *et al.*, 2017)
Note: The last column gives sufficient conditions for connectivity (Pattillo *et al.*, 2013a, p. 17).

The second last column of Table 1 refers to the *hereditary* graph property, which becomes important when distinguishing between partitioning and covering solutions in the next section. A graph property Π is *hereditary on vertex induced subgraphs* if for any $S \subseteq V$, where $G[S]$ has property Π, it follows that also $G[S']$ has property Π for any $S' \subset S, S' \neq \varnothing$. The literature distinguishes between "hereditary on induced subgraphs" (in the proper sense) where the property Π can be directly tested on $G[S]$ without knowing G, and "weakly hereditary" where the property refers to the given graph G. We will use "hereditary" in the comprehensive sense because there are no implications for the algorithmic components that we use in the branch-and-price.

By definition, s-club and k-block are always connected, while all other RCs are connected if the (sufficient) connectivity condition, reported in the last column of Table 1, is satisfied. For example, an arbitrarily large s-clique can be disconnected because the removal of the central vertex from a star graph induces an edgeless graph, which is however a 2-clique and therefore also an s-clique for all $s \geq 2$. Also, arbitrarily large disconnected γ-quasi-cliques exist resulting from the addition of an isolated vertex to a clique. In contrast, this phenomenon occurs only for small-sized $S \subset V$ in case of s-plex, s-defective clique, and s-bundle (see again the last column of Table 1). As a consequence, Gschwind *et al.* (2017) define two subclasses of families of RCs: *connected relaxed cliques* for which the connectivity is required and *general relaxed cliques* for which disconnected RCs are allowed. Formally, a RC $S \subseteq V$ is *connected* if the induced subgraph $G[S]$ is connected. Note that for hereditary RCs (s-plex, s-clique, s-defective clique, and s-bundle) the connectivity requirement

	Connected		General		
	Partitioning	Covering	Partitioning	Covering	with
17 interesting	s-plex	s-plex	s-plex		$s \geq 2$
decomposition	s-clique	s-clique			$s \geq 2$
problems:			s-club	s-club	$s \geq 2$
	γ-quasi-clique	γ-quasi-clique	γ-quasi-clique	γ-quasi-clique	$0 < \gamma < 1$
	s-defective clique	s-defective clique	s-defective clique		$s \geq 1$
	s-bundle	s-bundle	s-bundle		$s \geq 2$

Table 2: Families of partitioning and covering problems with RCs, results according to classification presented in (Gschwind *et al.*, 2017, Section 3)

makes the resulting structures non-hereditary. For example, a path with three vertices forms a connected 1-defective clique, which becomes disconnected when the middle vertex is removed. This has important consequences for the applicability of existing algorithms to find maximum-cardinality and maximum-weight RCs, and we discuss this issue in Section 4.

3. Problem Definition and Formulation

We now formally define the family of graph decomposition problems addressed in this paper. Given a graph $G = (V, E)$, a specific type of RC (including the defining parameter s or γ), the task is to determine a partitioning or a covering of the vertex set V with the minimum number of RCs. In the reminder of the paper, we use the acronyms PGMRC and CGMRC (for Partitioning/Covering a Graph with a Minimum number of RCs), respectively.

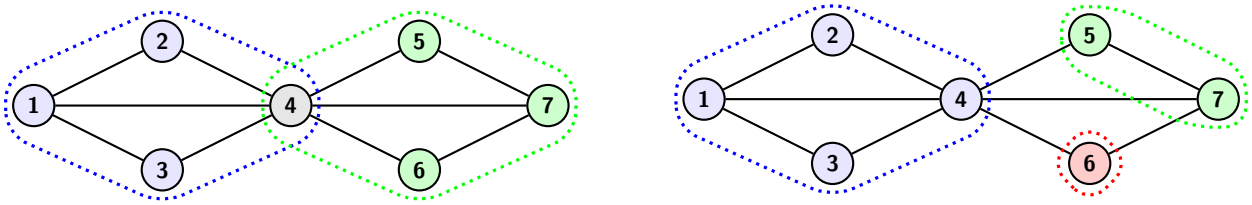

Figure 1: Covering and partitioning into a minimum number of $\frac{5}{6}$-quasi-cliques

Covering is always a relaxation of partitioning and this relaxation is proper for non-hereditary structures. The decomposition into $\frac{5}{6}$-quasi-cliques shown in Figure 1 is an example. The non-hereditary property of a particular structure may either result from the property Π defining the type of RC or from the connectivity requirement. When connectivity is not already ensured by the definition of the RC, the variants double and we analyze variants with and without connectivity requirement. The companion paper (Gschwind *et al.*, 2017, Section 3) has identified 17 computationally interesting variants of PGMRC and CGMRC. Table 2 summarizes this classification. Gschwind *et al.* (2017) also provide additional non-trivial examples where general and connected RCs lead to two different problems. Furthermore, they show that the decomposition problems using k-core and k-block are not interesting: for k-core, some vertices may have a degree smaller than k and cannot belong to any k-core; for k-block, the resulting problem is to determine the k-connected components, for which efficient algorithms exist (Kammer and Täubig, 2005).

Problem Formulation. It has been proven that compact *(Mixed) Integer linear Programming* ((M)IP) formulations for PGMRC and CGMRC are very weak, because the bound provided by their linear-programming relaxation is always 1 (see Gschwind *et al.*, 2017, Theorem 1). Accordingly, MIP solvers using these models exhibit a rather poor computational performance, which is also due to the high degree of symmetry between

feasible solutions. Applying a Dantzig-Wolfe reformulation and aggregation on the compact formulations results in set-partitioning or set-covering formulations with an exponential number of variables, each associated with a RC. Let the collection of all possible feasible RCs of a specific class (e.g., all feasible 3-clubs) be

$$\mathscr{S} = \{S \subseteq V : S \text{ is a relaxed clique }\}.$$

For each $S \in \mathscr{S}$ the partitioning/covering formulations have a binary variable λ_S which takes value one if and only if the RC S is part of the solution, and zero otherwise. The extensive formulations for PGMRC and CGMRC read as follows:

$$\min \quad \sum_{S \in \mathscr{S}} \lambda_S \tag{1a}$$

$$\text{s.t.} \sum_{S \in \mathscr{S}: i \in S} \lambda_S = 1 \quad (\text{or} \geq 1) \qquad i \in V \tag{1b}$$

$$\lambda_S \in \{0,1\} \qquad S \in \mathscr{S}. \tag{1c}$$

The objective function (1a) minimizes the number of RCs in the decomposition, (1b) are the partitioning/covering constraints, and (1c) define the domain of the variables.

Due to the large number of variables (in general exponential in the number of vertices of the graph), already the linear relaxation of model (1) has to be solved via column-generation techniques (cf. Desaulniers *et al.*, 2005). The *Restricted Master Problem* (RMP) is (1a)-(1b) and $\lambda_S \geq 0$ defined over a subset of the variables. At each column-generation iteration, the RMP is solved and the dual prices π_i (for all $i \in V$) associated with constraints (1b) are computed. The pricing subproblem asks for a maximum-weight RCs using the weights $w_i := \pi_i$ for all $i \in V$. If a negative reduced-cost column is found, i.e., a set $S \in \mathscr{S}$ with $1 - \sum_{i \in S} \pi_i < 0$, the corresponding variable λ_S is added to the RMP, the RMP is re-optimized, and the process is iterated. Typically, adding several reduced-cost columns in each iteration accelerates the solution process. Eventually, the linear relaxation of (1) is solved to optimality when no more negative reduced-cost columns exist.

In order to solve the integer model (1), column generation must be embedded into branch-and-bound a.k.a. branch-and-price (Desaulniers *et al.*, 2005; Barnhart *et al.*, 1998). A branch-and-price algorithm is not an out-of-the-box solver for MIPs with many variables, but an algorithmic principle that has to be tailored to the type of problem at hand. The two fundamental components that need to be designed are the subproblem solver (pricer, see next section) and the branching scheme (see Section 5). For the latter, we present alternative branching schemes and discuss the general tradeoff between being structure-preserving for the pricer and being effective in dividing the search space.

4. Pricing Algorithms

The pricing subproblems when solving formulation (1) with column-generation techniques are maximum-weight RC problems. Generally, two types of exact approaches can be found for the solution of these problems in the literature: MIP-based techniques and *combinatorial branch-and-bound* (CB&B) algorithms. In this section, we detail on how they are solved in our branch-and-price by either pointing to the respective literature whenever an existing algorithm is used or describing newly developed approaches in detail. Since it is typically not clear which pricing algorithm is favorable if there is more than one available, we implemented different pricing solvers for some of the RCs and compare the resulting branch-and-price algorithms in Section 6.1.

Different MIP formulations for maximum-cardinality RCs have been suggested for all first-order RCs presented in Table 1. In (Gschwind *et al.*, 2017, Section 2.3), the most recent and successful formulations are reviewed. All of them can straightforwardly be adapted to the maximum-weight case. In our branch-and-price, we use the MIPs presented in (Balasundaram *et al.*, 2011), (Veremyev and Boginski, 2012), (Gschwind *et al.*, 2017), and (Veremyev *et al.*, 2015) for finding maximum-weight s-plexes, s-clubs, s-bundles, and γ-quasi-cliques, respectively. Furthermore, for γ-quasi-cliques we derived a strengthened formulation which

uses the fact that in an integer solution the number of edges in a γ-quasi-cliques is always integer. The new formulation is presented in Section 4.1.

As discussed above, some types of RCs are not necessarily connected so that the addition of additional constraints is necessary to impose connectivity (for details see Gschwind *et al.*, 2017, Section 2.3.3). The connectivity requirements complicate the MIP-based solution, since the additional constraints have to be added in a cutting-plane fashion leading to a branch-and-cut algorithm.

The second type of approaches for maximum-cardinality/weight RCs are CB&B algorithms. The *Russian doll search* (RDS) originally developed for the identification of maximal hereditary structures (Verfaillie *et al.*, 1996) is such an approach. The well-known `cliquer` algorithm for identifying maximum-weight cliques by Östergård (2002) follows the RDS principle without explicitly making the connection to RDS. Another CB&B algorithm was presented in (Held *et al.*, 2012b), where the authors use `cliquer` for relatively sparse graphs and their own new algorithm for denser graphs in order to benefit from the better performance of the respective algorithm. Trukhanov *et al.* (2013) present RDS algorithms for s-plex and s-defective clique. Gschwind *et al.* (2018) revisit these RDS algorithms, suggest several techniques to accelerate the search, and present a first exact algorithm for s-bundle. Overall, these algorithms cover the hereditary cases of s-clique, s-plex, s-defective clique, and s-bundle. The extensive computational tests on 245 graphs from different benchmarks, not only social networks, conducted in (Gschwind *et al.*, 2018) have shown that our computer implementations compare favorably with state-of-the-art methods.

The RDS principle was not designed to handle connectivity and may return a solution that is disconnected (see Gschwind *et al.*, 2017, Figure 1). Also, covering and partitioning with connected s-cliques for $s \geq 2$ renders the indirect solution as a clique partitioning problem in the s-th power graph infeasible. In Section 4.2, we therefore present a new adaptation of RDS, in the following referred to as *modified RDS* (mRDS), to handle connected versions of s-clique, s-plex, s-defective clique, and s-bundle. Note further that negative weights can arise in the partitioning case. Our mRDS is the first variant that has the ability to handle arbitrary weights $w_i \in \mathbb{R}$ for $i \in V$.

Several CB&B algorithms have been proposed for the maximum-cardinality s-club problem (Veremyev and Boginski, 2012; Mahdavi Pajouh and Balasundaram, 2012; Shahinpour and Butenko, 2013; Wotzlaw, 2014). As s-club is not hereditary, negative vertex weights require a careful handling in bounding procedures in the maximum-weight counterpart of the problem. For this purpose, we propose a new CB&B for maximum-weight s-club in Section 4.3. Note that although our s-club pricing algorithm is designed for the general case of weighted graphs, it outperforms the recent implementations listed above. The tests in Section 4.3 with six of the nine networks used by all researchers, indicates that our algorithm is most of the time faster, in some cases by more than factor 100.

In Table 7, we summarize the pricing algorithms used in our generic framework for solving PGMRC and CGMRC (see Table 2). If two different approaches are listed for a RC, both algorithms are implemented in our branch-and-price and their practical performance as pricing solver is analyzed and compared in Section 6.1. The acronym MIP-CP refers to the case in which constraints imposing connectivity need to be added to the corresponding MIP formulations.

	General		Connected	
s-plex	RDS (Gschwind *et al.*, 2018)	MIP (Balasundaram *et al.*, 2011)	mRDS (Sect. 4.2)	MIP-CP
s-clique			mRDS (Sect. 4.2)	
s-club	CB&B (Sect. 4.3)	MIP (Veremyev and Boginski, 2012)		
γ-quasi-clique	MIP (Veremyev *et al.*, 2015)	MIP (Sect. 4.1)		MIP-CP
s-defective clique	RDS (Gschwind *et al.*, 2018)		mRDS (Sect. 4.2)	
s-bundle	RDS (Gschwind *et al.*, 2018)	MIP Gschwind *et al.* (2017)	mRDS (Sect. 4.2)	MIP-CP

Table 3: Alternative pricing approaches for the maximum weight RCs
CB&B=combinatorial branch-and-bound; MIP(-CP)=Mixed Integer Programming solver (with Cutting Plane algorithm); (m)RDS=(modified) Russian Doll Search

4.1. Strengthened Formulation for γ-Quasi-Clique

In the recent paper (Veremyev *et al.*, 2015), four formulations for the γ-quasi-cliques are given and compared against two older ones from (Pattillo *et al.*, 2013b). For the sake of brevity, we present only the one that was identified as giving the most consistent results and best bounds for maximum-cardinality (Veremyev *et al.*, 2015, p. 210ff). The associated formulation uses binary variables x_i and y_{ij} to indicate if vertex $i \in V$ and edge $\{i,j\} \in E$, respectively, are part of the solution. Additional binary indicator variables t_s for $s \in \mathcal{S} := \{1, 2, \ldots, |V|\}$ define the size of the γ-quasi-clique. The formulation of the maximum-cardinality γ-quasi-clique problem is:

$$\max \quad \sum_{i \in V} x_i \tag{2a}$$

$$\text{s.t.} \quad y_{ij} \leq x_i, y_{ij} \leq x_j \qquad \{i,j\} \in E \tag{2b}$$

$$\sum_{e \in E} y_e \geq \gamma \sum_{s \in \mathcal{S}} \frac{s(s-1)}{2} t_s \tag{2c}$$

$$\sum_{i \in V} x_i = \sum_{s \in \mathcal{S}} s t_s \tag{2d}$$

$$\sum_{s \in \mathcal{S}} t_s = 1 \tag{2e}$$

$$t_s \geq 0 \qquad s \in \mathcal{S} \tag{2f}$$

$$x_i \in \{0,1\} \qquad i \in V \tag{2g}$$

$$y_e \geq 0 \qquad e \in E \tag{2h}$$

Herein, the coupling between vertex and edge indicator variables is established via (2b), the γ-related constraint on the number of edges in the induced graph is (2c), the coupling between the x- and t-variables is given by (2d), and the unique cardinality of the induced graph is enforced via (2e). Veremyev *et al.* (2015) show that a smaller formulation results from replacing the possible sizes $\mathcal{S}$ by $\{l, l+1, \ldots, u\}$ when a lower bound l and an upper bound u is known.

Since the number of edges in a γ-quasi-clique is clearly integer, we propose the following strengthening of the above formulation. For each $s \in \mathcal{S}$, we define the number $\Delta_s := \lceil \gamma s(s-1)/2 \rceil - \lceil \gamma(s-1)(s-2)/2 \rceil$ of additional edges a γ-quasi-clique of size s must comprise compared to one of size $s-1$. We use binary variables t'_s to indicate that the size of the resulting γ-quasi-clique is at least $s \in \mathcal{S}$. The relationship to the above size indicators t_s of Veremyev *et al.* (2015) is

$$t_s = t'_s - t'_{s+1} \qquad \text{for all } s \in \mathcal{S} \setminus \{|V|\}.$$

Then, the resulting polytope is given by the constraints:

$$\text{(2b), (2g), and (2h)} \tag{3a}$$

$$\sum_{e \in E} y_e \geq \sum_{s \in \mathcal{S}} \Delta_s t'_s \tag{3b}$$

$$\sum_{i \in V} x_i = \sum_{s \in \mathcal{S}} t'_s \tag{3c}$$

$$t'_{s-1} \geq t'_s \qquad s \in \mathcal{S} \setminus \{1\} \tag{3d}$$

$$0 \leq t'_s \leq 1 \qquad s \in \mathcal{S} \tag{3e}$$

The difference to (2) lies in the new formulation of constraints (2c)–(2e), where now constraints (3b) ensure that sufficiently many edges are present in the induced subgraph, constraints (3c) couple the vertex selection with the new indicator variables for the size, and constraints (3d) make the new t'-variables non-increasing in s. The system (3) can be reduced if the Δ_s-coefficients are non-decreasing (which is often fulfilled for not too small γ-values). In this case, constraints (3d) are redundant. We use (3) without constraints (3d) whenever possible.

Algorithm 1: RDS for the Maximum-Weight Π Problem

Input: $G = (V, E, w_i)$ with $w_i \in \mathbb{R}_0^+$; Π

1 Order vertices $(v_1, v_2, \ldots, v_n)$
2 Set $LB := 0$ and $S := \varnothing$
3 **for** $i := n, n-1, \ldots, 1$ **do**
4 Set $C := \{v_j : j > i, \{v_i, v_j\}$ satisfies $\Pi\}$
5 Call FindMax(C, $\{v_i\}$)
6 $LB_i := LB$

Output: $S \subseteq V$ inducing a maximum-weight Π subgraph $G[S]$

Algorithm 2: Modified RDS for the Maximum-Weight Connected Π Problem

Input: $G = (V, E, w_i)$ with $w_i \in \mathbb{R}$, Π

1 Order vertices $(v_1, v_2, \ldots, v_n)$
2 Set $LB := 0$, $LB^c := 0$, and $S := \varnothing$
3 **for** $i := n, n-1, \ldots, 1$ **do**
4 Set $C := \{v_j : j > i, \{v_i, v_j\}$ satisfies $\Pi\}$
5 Call FindMaxConnected(C, $\{v_i\}$)
6 $LB_i := LB$

Output: $S \subseteq V$ inducing a maximum-weight connected Π subgraph $G[S]$

4.2. Handling Connectivity Constraints and Negative Weights in RDS

Recall from Section 4 that the RDS is an algorithm for finding maximum-weight RCs, which are defined by a hereditary property Π. We now present the necessary modifications that allow us to find *connected* RCs. The resulting structure of a connected RC is no longer hereditary, which implies that negative vertex weights generate additional complications. Such negative vertex weights result from three facts: (i) the weights (before branching) are equal to the dual prices of the constraints (1b) which can be negative in case of partitioning, (ii) the implementation of separate-constraints imposes large negative weights on the vertices (see Section 5.1.1), and (iii) dual prices of constraints that bound the number of vertex contacts from above are non-positive summands of the weights (see Section 5.3). The modified RDS is applicable to s-clique, s-plex, s-defective clique, and s-bundle. Our description of RDS follows the presentation of Trukhanov *et al.* (2013) and Gschwind *et al.* (2018).

In standard RDS (Algorithm 1), the n vertices V are ordered into a sequence $(v_1, v_2, \ldots, v_n)$. Instead of one depth-first branch-and-bound search, n searches are performed in the main loop of RDS (Steps 3-6 of Algorithm 2). Starting from $i = n$, the ith search determines a maximum-weight Π set for $G[\{v_i, v_{i+1}, \ldots, v_n\}]$ with the initial set $S = \{v_i\}$ by means of the recursion FindMax. In every iteration, i is decreased by 1 so that a sequence of lower bounds $LB_n, LB_{n-1}, \ldots, LB_2, LB_1$ is computed, where LB_i corresponds to the value of a maximum-weight $S \subseteq \{v_i, \ldots, v_n\}$ fulfilling Π. The value of the best solution found so far is retained in the overall lower bound LB. At each stage of the RDS search, the current solution P satisfies Π. Moreover, a set of candidates C with $P \cup \{c\}$ satisfies Π for all $c \in C$ is maintained. Whenever P is enlarged, C has to be adjusted, i.e., candidate vertices not compatible with the new set P are removed from C. We use the shorthand notation $w(P) = \sum_{v \in P} w_v$ and $w^+(P) = \sum_{v \in P} w_v^+$.

The modified RDS (Algorithm 2) uses two types of overall lower bounds instead of just one: LB is, as in the standard RDS, the maximum weight of subset $P \subseteq V$ fulfilling Π either connected or not. LB^c is the maximum weight of subset $P \subseteq V$ fulfilling Π that is connected.

Furthermore, the pruning criteria of the standard RDS and the modified RDS differ. The standard weight-based pruning ($w(C)+w(P) \leq LB$, Step 6 in Procedure FindMax) is adapted so that it takes possibly negative weights into account and compares against the connected bound, i.e., $w^+(C) + w(P) \leq LB^c$ in Step 9 of Procedure FindMaxConnected. The RDS-specific pruning Step 11 of Procedure FindMaxConnected compares $LB_i + w(P)$ against LB^c instead of LB. Note that LB_i is the maximum weight of a general (connected or not) $P \subseteq \{v_i, \ldots, v_n\}$ fulfilling Π. Both modified pruning steps are less effective compared to the pruning steps of the standard RDS.

When the candidate set C becomes empty, the modified RDS does three things: First, the lower bound LB is updated whenever P is improving (Steps 2 and 3 in Procedures FindMax and FindMaxConnected). Second, the connected components R of $G[P]$ are computed. For s-plex, s-defective clique, and s-bundle, this step is obsolete whenever $|P|$ is sufficiently large (see Table 1), while for smaller $|P|$ we use a straightforward enumeration. For s-clique, the connected components are determined with an efficient union-find algorithm (Cormen *et al.*, 2009, Section 21.3). In all cases, the component P^* with largest weight $w(P^*)$ is determined. Third, if $w(P^*)$ improves the connected lower bound LB^c, a new improving connected RC is found and LB^c

Procedure FindMax(C, P)

Input: Candidate set C, current set P

1 **if** $C = \varnothing$ **then**
2 **if** $w(P)$ ¿ LB **then**
3 Set $LB := w(P)$ and $S := P$
4 **return**
5 **while** $C \neq \varnothing$ **do**
6 **if** $w(C) + w(P) \leq LB$ **then return**
7 Set $i := \min\{j : v_j \in C\}$
8 **if** $LB_i + w(P) \leq LB$ **then return**
9 Set $C := C \setminus \{v_i\}$ and $P' := P \cup \{v_i\}$
10 Set $C' := \{v \in C : P' \cup \{v\}$ satisfies $\Pi\}$
11 Call FindMax(C', P')

Procedure FindMaxConnected(C, P)

Input: Candidate set C, current set P

1 **if** $C = \varnothing$ **then**
2 **if** $w(P)$ ¿ LB **then**
3 Set $LB := w(P)$
4 Set $P^* := \underset{R \text{ connected comp. of } G[P]}{\operatorname{argmax}} \sum_{r \in R} w_r$
5 **if** $w(P^*)$ ¿ LB^c **then**
6 Set $LB^c := w(P^*)$ and $S := P^*$
7 **return**
8 **while** $C \neq \varnothing$ **do**
9 **if** $w^+(C) + w(P) \leq LB^c$ **then return**
10 Set $i := \min\{j : v_j \in C\}$
11 **if** $LB_i + w(P) \leq LB^c$ **then return**
12 Set $C := C \setminus \{v_i\}$ and $P' := P \cup \{v_i\}$
13 Set $C' := \{v \in C : P' \cup \{v\}$ satisfies $\Pi\}$
14 Call FindMaxConnected(C', P')
15 Call FindMaxConnected($\varnothing$, P)

as well as S are updated.

Finally, the additional Step 15 of FindMaxConnected is required, since all candidates C may have a negative weight so that the current set P without any vertex additions has the largest weight among all subsets S with $P \subseteq S \subseteq P \cup C$.

Note that it is sufficient to consider only the connected components of $G[P]$ instead of all connected subsets of P in Steps 4-6 of the recursion. The reason is that for given sets P and C the RDS enumerates all subsets of $P \cup C$ as long as no pruning occurs. Indeed, the modified pruning Step 11 guarantees that no subset of $P \cup C$ fulfilling Π (connected or not) and therefore no connected subset of $P \cup C$ fulfilling Π is excluded. Thus, connected subsets of P that are not connected components of $G[P]$ are found as connected components of the induced subgraph of a different current set P' in another iteration of Procedure FindMaxConnected.

4.3. New Combinatorial Branch-and-Bound for Maximum-Weight s-Club

We developed a new CB&B algorithm for maximum-weight s-club which is able to handle arbitrary vertex weights $w_i \in \mathbb{R}$. Since s-club is non-hereditary, we have to cope with negative weights. As before, we assume that a simple graph $G = (V, E)$ with vertex weights $(w_i = \pi_i^*), i \in V$ is given together with some $s \geq 2$.

Throughout the branch-and-bound, we partition the vertices V into three sets, the *included vertices* I with $x_i = 1$ for all $i \in I$, the *free vertices* F with $x_i \in \{0, 1\}$ for all $i \in F$, and the *excluded vertices* X with $x_i = 0$ for all $i \in X$. The algorithm is initialized with $F = V$ and $I = X = \varnothing$. All partitions with $F \neq \varnothing$ are partial solutions, while those with $F = \varnothing$ are complete solutions. We always assume that the partition $V = I + F + X$ must admit that a subset S of the *admissible vertices* $I \cup F$ is a feasible s-club with $S \supseteq I$. In particular, the distance between all admissible vertices $i \in F \cup I$ and the included vertices $j \in I$ must not exceed s, i.e., $\text{dist}_{G[I \cup F]}(i, j) \leq s$, meaning that $I \cup \{j\}$ is an s-clique for all $j \in F$. This is similar to the branch-and-bound algorithm by Mahdavi Pajouh and Balasundaram (2012). However, our approach differs in the computation of upper and lower bounds and it has an additional component for fixing vertices and detecting infeasible partial solutions.

Upper Bounds. A straightforward upper bound is

$$ub_1(I + F) = \pi(I) + \pi^+(F) = \sum_{i \in I} w_i + \sum_{i \in F} w_i^+$$

with the standard shorthand notation $w_i^+ := \max\{0, w_i\}$ for referring to the positive part.

In the course of the algorithm, the induced graph $G[I \cup F]$ can become disconnected with components $C_1, \ldots, C_p$. Since every s-club is connected, the component with the largest weight provides a tighter bound in this case:

$$ub_2(I + F) = \min_{j=1,\ldots,p} ub_1(C_j).$$

A third upper bound results from the fact that every s-club is also an s-clique. This is a (1-)clique in the sth power graph G^s. Therefore,

$$ub_3(I + F) = w(I) + z(w, G^s[F]),$$

where $z(w, G^s[F])$ is the maximum weight of a clique that can be found in the induced subgraph by F in G^s. Any algorithm for computing maximum-weight cliques can be used to compute $z(w, G^s[F])$ (or any algorithm for maximum-weight independent sets in the complement graph of G^s). We used the publicly available code from the paper Held *et al.* (2012b).

Lower Bounds. For computing an initial lower bound, we adapt the `Constellation` heuristic of Bourjolly *et al.* (2002). `Constellation` first determines a start vertex i with maximum value $w_i + \sum_{j \in N(i)} w_j^+$. The start solution is $S = \{i\} \cup \{j \in N(i) : w_j \geq 0\}$. Then, $s - 2$ times additional vertices are added. For all vertices $j \in S$, one with maximum $\sum_{i \in N(j) \setminus S} w_i^+$ is determined and all vertices $N(j) \setminus S$ are added to S. Note that this step increases the diameter of $G[S]$ by at most one so that after $s-2$ iterations the diameter of S cannot exceed s, i.e., $G[S]$ is an s-club. Since no computation of distances in induced subgraphs is needed, the `Constellation` heuristic provides a lower bound very quickly. Preliminary test revealed that the `Drop` heuristic of Bourjolly *et al.* (2002) is in many cases too time consuming, and we do not use it.

The only primal heuristic that we employ at every branch-and-bound node is the check whether or not $I \cup F$ induces a feasible s-club. It requires the computation of a distance matrix for $G[I \cup F]$ (with unit costs), which is implemented as a straightforward breadth-first-search (BFS). It requires not more than $\mathcal{O}(n_{I \cup F} \cdot m_{I \cup F})$ time, where $n_{I \cup F} = |I \cup F|$ is the number of vertices and $m_{I \cup F}$ is the number of edges of $G[I \cup F]$.

Vertex Fixation and Infeasibility Detection. In the branch-and-bound algorithm, branches result from the inclusion or exclusion of a free vertex, i.e., we select a free vertex $v \in F$, remove it from F, and add it to either I or X. Obviously, the inclusion of a free vertex does not change distances in $G[I \cup F]$. In contrast, the exclusion of a free vertex requires the re-computation of the distances in the graph $G[I \cup F]$ using BFS. For the sake of brevity, let d_{ij} be the short notation for $\text{dist}_{G[F \cup I]}(i, j)$. The following cases are interesting:

1. If $d_{ij} > s$ for $i, j \in I$ then the partial solution is infeasible.
2. If $d_{ij} > s$ for $i \in I, j \in F$ then vertex j can be excluded, i.e., $X' = X \cup \{j\}$ and $F' = F \setminus \{j\}$.
3. If $d_{ij} = s$ for $i, j \in I$ and there exists a $0 < d < s$ and a *unique* vertex $k \in F$ with $d_{ik} = d$ and $d_{kj} = s - d$, then vertex k can be included, i.e., $I' = I \cup \{j\}$ and $F' = F \setminus \{j\}$.

If case 2 applies, distances need to be recalculated. If one of the cases 2 or 3 applies, the variable fixation and infeasibility detection procedure is repeated. This can create a longer sequence of fixations.

Branching. None of the bounding procedures can cope precisely with negative weights, and therefore we start branching by first selecting a free vertex $v \in F$ with smallest negative weight $w_v < 0$ (if any). Then, at the second level, the selection of a vertex for branching is based on the idea of choosing a highly influential free vertex $v \in F$. It is a vertex v maximizing the number of included vertices to which the distance is exactly s. Formally, the vertex $v \in F$ maximizes $|\{i \in I : d_{iv} = s\}|$ and ties are broken by preferring larger weights w_v. Finally, the overall branching strategy is depth-first, where the branch in which the free vertex is included is inspected before the branch in which the free vertex is excluded.

The overall branch-and-bound is summarized in Algorithm 3 and the recursion in Procedure Recursion.

We briefly compare our new algorithm with the fastest algorithms from the literature. Wotzlaw (2014) presents two MAX-SAT formulations that are solved by a state-of-the-art SAT solver (`clasp` 2.1.3). His

Algorithm 3: Combinatorial B&B Algorithm

Input: Graph $G = (V, E)$, vertex weights $w = (w_i)$
SET $lb :=$ `Constellation`(G, w), $ub := \infty$, $I = \varnothing$, $F = V$, and $X = \varnothing$
Recursion(I, F, X)
Output: maximum-weight s-club S^* with weight lb

Procedure Recursion(I, F, X)

if $lb > ub_1(I \cup F)$ *or* $lb > ub_2(I \cup F)$ *or* $lb > ub_3(I \cup F)$ **then** RETURN
if $I \cup F$ *induces s-club* ***and*** $w(I \cup F) > lb$ **then** SET $lb := w(I \cup F)$
Select branching vertex $j \in F$
for *branches* $(I \cup \{j\}, F \setminus \{j\}, X)$ *and* $(I, F \setminus \{j\}, X \cup \{j\})$ **do**
    DETECT infeasibility, FIX additional vertices with result (I', F', X')
    if *infeasible* **then** RETURN
    if $F' = \varnothing$ **then**
        if $w(I') > lb$ **then** SET $lb = w(I')$ and $S^* := I'$
        RETURN
    Recursion(I', F', X')

comparison shows that these implementations often outperform the older algorithms presented in Veremyev and Boginski (2012); Mahdavi Pajouh and Balasundaram (2012); Shahinpour and Butenko (2013). Table 4 shows the computation times for computing a maximum-cardinality s-club for $s \in \{2, 3, 4\}$. For the sake of brevity, column *Other* is the shortest runtime obtained with any of the algorithms compared in Wotzlaw (2014), while *New* is our computation time (computed on different machines of comparable performance). It seems that our implementation is competitive, in particular, it scales well for larger instances and $s > 2$.

			Time in seconds					
			$s = 2$		$s = 3$		$s = 4$	
Instance	$\|V\|$	$\|E\|$	Other	New	Other	New	Other	New
`adjnoun`	112	425	0.010	0.003	0.020	0.059	0.340	0.017
`football`	115	613	0.020	0.015	0.320	4.118	0.001	0.003
`jazz`	198	2742	0.060	0.005	1.050	0.125	8.870	0.075
`celegansneutral`	297	2148	0.430	1.060	1.060	1.058	45.300	0.435
`email`	1133	5451	16.100	0.037	TL	TL	TL	TL
`polblogs`	1490	16715	46.900	0.086	TL	TL	TL	TL

Table 4: Computation times of s-club algorithms
Note: Computation time is limited to $TL = 600$ seconds.

5. Branching Schemes

The design of a branching scheme is crucial for the performance of a branch-and-price algorithm (Vanderbeck, 2011). First and foremost, it must ensure that integrality can be imposed in all cases. The perfect branching scheme would be one that lets the algorithm find and prove an optimal solution quickly, i.e., it creates a small search tree, allows a fast solution of each node, and does not require modifications neither on the master problem nor on the subproblem algorithm. The latter means that branching should not alter the structure of the respective pricing problem (structure-preserving) so that the best performing algorithm can be applied during the entire search. Such a perfect branching scheme does not exist for partitioning and covering a graph into RCs as we explain next. We present competing branching schemes that are generic and

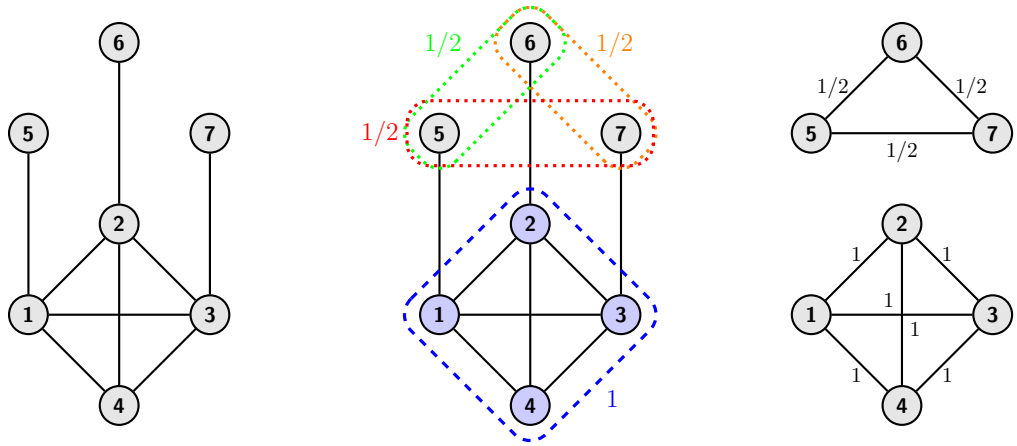

Figure 2: Partitioning with (possibly disconnected) 2-plexes. Example shows (a) the graph G, (b) the fractional solution $\lambda_{\{1,2,3,4\}} = 1$ and $\lambda_{\{5,6\}} = \lambda_{\{5,7\}} = \lambda_{\{6,7\}} = 0.5$, (c) the support graph having binary f_{ij}^{λ} for all $\{i,j\} \in E$

applicable to all 17 variants allowing us to reuse all pricing algorithms presented in the previous section. We first discuss two alternative branching rules for partitioning before we reuse some of the results to introduce branching schemes for covering.

Let $\boldsymbol{\lambda}$ be a solution of the RMP. The *support graph* of $\boldsymbol{\lambda}$ is the weighted undirected graph $G^{\lambda} = (V, E^{\lambda})$ defined by $E^{\lambda} := \{\{i,j\} : i,j \in V, i \neq j, f_{ij}^{\lambda} > 0\}$ with $f_{ij}^{\lambda} := \sum_{S \in \Omega : i,j \in S} \lambda_S$. It follows $f_{ij}^{\lambda} \geq 0$ and, for partitioning, also $f_{ij}^{\lambda} \leq 1$.

5.1. Ryan-Foster Branching for Relaxed Clique Partitioning

Ryan-Foster branching has been proven as one of the most effective branching rules when solving set-partitioning problems with LP-based branch-and-bound Ryan and Foster (1981): When $\boldsymbol{\lambda}$ is a fractional solution to the LP-relaxation $A\boldsymbol{\lambda} = \mathbf{1}, \boldsymbol{\lambda} \geq \mathbf{0}$ (with binary matrix $A = (a_{ik})$), then there exist at least two rows of A, say i and j, such that $0 < \sum_{k : a_{ik} = a_{jk} = 1} \lambda_k < 1$. In any integer solution, however, this value is 0 or 1. For any i-j-pair with fractional value, two branches can be created, the so-called *together-branch* forcing variables λ_k with $a_{ik} + a_{jk} = 1$ to zero, and the *separate-branch* forcing variables λ_k with $a_{ik} = a_{jk} = 1$ to zero. The fractional values f_{ij}^{λ} allow the direct detection of branching opportunities, where the together-branch results in $f_{ij}^{\lambda} = 1$ and the separate-branch in $f_{ij}^{\lambda} = 0$.

For the extensive formulation (1), the subproblems must then respect $x_i = x_j$ and $x_i + x_j \leq 1$, respectively. Such constraints are simple to enforce if the subproblem is solved with a MIP solver. Moreover, instead of considering every possible i-j-pair, Ryan-Foster branching can be implemented by using only edges $\{i,j\} \in E$ in case of connected RCs. Figure 2 shows that the connectivity requirement is crucial. The validity of this statement is straightforward to prove.

Section 5.1.1 shows how to apply it in situations where it is impossible or inconvenient to directly modify pricing algorithms such as CB&B. However, Ryan-Foster branching is not structure-preserving, but it is complete and generic in the sense that it can be used for all types of RCs.

5.1.1. Generic Handling of Together- and Separate-Constraints in Pricing

For vertex coloring (clique partitioning of the complement graph), the Ryan-Foster constraints can be imposed solely by graph modifications. In the together-branch, the vertices i and j are merged, while in the separate-branch the new edge $\{i,j\}$ is added (Mehrotra and Trick, 1998; Held *et al.*, 2012b). However, for RCs, such graph modifications generally change distance, density, and connectivity on many subsets S and, therefore, do not reproduce the original situation with the separate/together constraint added.

A single separate-constraint can be implemented by solving two different subproblems, since the removal of one vertex, either i or j, ensures that the two vertices never occur together in a RC. A removal of vertex i is equivalent to assigning a huge negative weight $w_i = -M$ (using a big-M). Also a single together-constraint for i and j can be realized by solving two different subproblems, where in one subproblem both vertices are removed, while in the other one both are enforced by assigning a huge positive weight $w_i = w_j = M$.

Although this approach seems appealing, it has the drawback that when q Ryan-Foster constraints are active up to 2^q different subproblems have to be solved. In order to mitigate this explosion, we propose the following branch-and-bound algorithm. At its root node, no constraints are active. At each node, the relaxed subproblem is solved and if infeasible, a single separate- or together-constraint creates two new branches. A depth-first node selection is applied in order to have lower bounds available at an early stage. Note that this branch-and-bound approach is truly generic as it can be used in combination with any of the subproblem algorithms (see Section 4).

5.2. Generic Branching Rule for Relaxed Clique Partitioning

The here proposed *generic branching rule* (GBR) is a subproblem-structure preserving rule applicable when partitioning with RC variants. The rule is straightforward to implement because it suffices to remove edges from the given graph G. It does neither impose additional constraints nor require a repeated solution of the subproblem, and can be shown to be a complete branching rule for hereditary Π in the presence of connectivity constraints.

Let $P_1, P_2, \ldots, P_p$ be the vertices inducing the connected components of G^λ, that is, $G^\lambda = G^\lambda[P_1] \cup G^\lambda[P_2] \cup \cdots \cup G^\lambda[P_p]$. When all vertex-induced subgraphs $G[P_q]$ for $q = 1, \ldots, p$ fulfill Π no branching is required, since a feasible partition into p RCs has been found. Moreover, when connectivity is required the master problem has objective $\sum_{S \in \mathscr{S}} \lambda_S = p$. Figure 2 shows that, for solutions with disconnected structures, $\boldsymbol{\lambda}$ can be fractional although all component $G[P_q]$ are feasible. In this case, $\sum_{S \in \mathscr{S}} \lambda_S < p$, and the GBR fails.

We now assume that at least one of the sets $P = P_q$ does not induce a feasible RC. We exclude the occurrence of the component $G^\lambda[P]$ via branching by removing its edges one by one. This creates $|E(G^\lambda[P])|$ branches, where in each branch just one edge is removed from G. The following property is helpful for drastically reducing the number of branches.

Property 1. *Let Π be hereditary and connectivity be required. Given a subset $P \subseteq V$ such that $G[P]$ does not fulfill Π, let $T = (P, E_T)$ be an arbitrary tree spanning P. Then, any feasible solution $(\lambda_S)_{S \in \mathscr{S}}$ to (1) fulfills*

$$\sum_{S \in \mathscr{S}} |E(S) \cap E_T| \lambda_S \leq |P| - 2.$$

Proof. Since the tree contains exactly $|P| - 1$ edges, the constraint $\sum_{S \in \mathscr{S}} |E(S) \cap E_T| \lambda_S \leq |P| - 2$ means that at least one of the tree edges is not present in the solution. Otherwise, the simultaneous presence of all edges $e \in E_T$ would imply that there exists a RC P' in the solution with $P' \supseteq P$. Due to the heredity of Π this is impossible. □

If Π is hereditary and connectivity is required, this gives rise to the GBR formalized in Algorithm 7, which guarantees that after a finite number of branchings the solution of the RMP is integral.

Proposition 1. *The GBR is a complete rule for partitioning into RCs with hereditary Π and connectivity constraints.*

Proof. The requirement of $G^\lambda[P]$ to be connected ensures that a spanning tree exists. Thus, by construction, the value f_{ij}^λ is strictly positive for the tree edges, cutting off the current solution λ in each of the resulting branches. Branching can be repeated only up to $|E|$ times because each branch eliminates one edge from G, finally leading to an edgeless graph. Consequently, solutions $\boldsymbol{\lambda}$ must eventually become integral. □

Algorithm 4: Generic Branching Rule (GBR)

Input : Support graph G^λ and weights f_{ij}^λ

1 Determine the connected components $P_1, P_2, \ldots, P_p$ of G^λ.
2 Identify one component P for which $G[P]$ does not fulfill Π.
3 If no such component exists, then stop (the solution is already a union of feasible RCs).
4 (Optional) Replace P by one of its subsets such that
(a) $G[P]$ does not fulfill Π, (b) $G^\lambda[P]$ is connected, and (c) $|P|$ is minimal.
5 Determine a maximum-weight spanning tree $T = (P, E(T))$ of $G^\lambda[P]$ using the weights f_{ij}^λ.

Output : $E(T)$, the set of edges to eliminate one by one

GBR is a non-binary branching scheme because generally the tree T contains more than two edges. The optional Step 4 reduces the number of branches that are created. The computation of the spanning tree in Step 5 is computationally cheap (using Prim's or Kruskal's algorithm). The selection of maximum-weight edges is intended to produce branches that improve the lower bounds as much as possible.

The *reduction problem* in Step 4 is the following: Given a property Π, a graph $H = (S, E(S))$ not fulfilling Π, and a connected subgraph (S, E') spanning S, find a set $S^\star \subset S$ of minimum cardinality such that $G'[S^\star]$ is connected and $G[S^\star]$ does not fulfill Π. (Note that the connected subgraph (S, E') takes the role of $G^\lambda[S]$ in GBR.) We suspect that the reduction problem is $\mathcal{NP}$-hard for arbitrary RCs and, thus, propose the following simple greedy procedure for its resolution: The greedy procedure is inspired by Prim's algorithm for computing a minimum spanning tree. In a first step, the vertices are sorted by increasing degree. Second, we chose a vertices $i \in S$ with smallest vertex degree and set $S^\star = \{i\}$. Iteratively, in the order of increasing degree, vertices $j \in S \setminus S^\star$ are tested whether or not j is adjacent to $S^\star$ in (S, E'). The first vertex j which fulfills the condition is added to $S^\star$. The greedy algorithm stops with the solution $S^\star$ as soon as $H[S^\star]$ does not fulfill Π.

The Non-Hereditary Case. A prerequisite of GBR as presented in Algorithm 7 is that Π is hereditary. However, even for non-hereditary Π a modified version of GBR can be used. Note that heredity of Π has only been exploited in order to ensure that no superset $P' \supsetneq P$ is a feasible RC. If P has no such superset and the optional reduction in Step 4 of GBR is skipped, branching on the edges of the tree spanning $G^\lambda[P]$ is valid.

However, two drawbacks have to be pointed out: On the one hand, the average number of branches created with GBR can be expected to be larger due to the skipped reduction step. On the other hand, for non-hereditary Π, GBR may not be applicable and, thus, it is incomplete. This happens if all connected components which do not fulfill Π have a superset satisfying Π. Figure 3 provides an example when partitioning with 2/3-quasi-cliques.

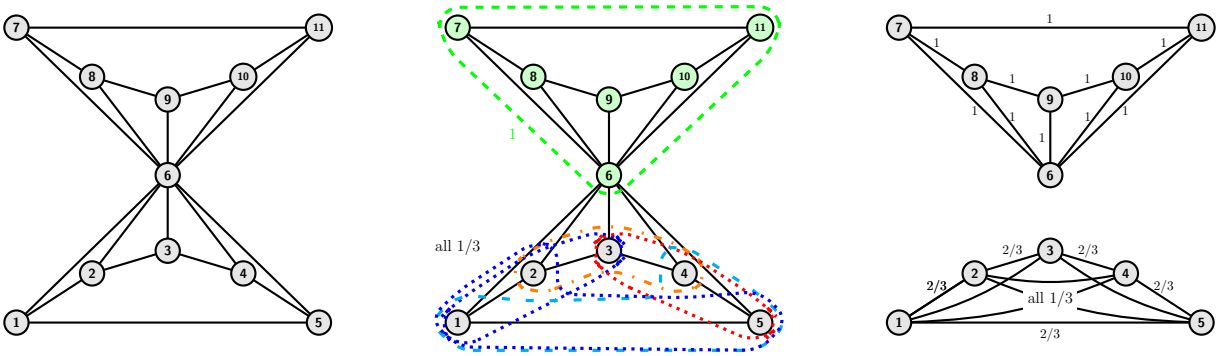

Figure 3: Partitioning with 2/3-quasi-cliques. (a) the graph G, (b) the factional solution $\lambda_{\{6,7,8,9,10,11\}} = 1$ and $\lambda_{\{1,2,3\}} = \lambda_{\{2,3,4\}} = \lambda_{\{3,4,5\}} = \lambda_{\{1,4,5\}} = \lambda_{\{1,2,5\}} = 1/3$, (c) the support graph in which the component $P = \{1, 2, 3, 4, 5\}$ is no 2/3-quasi-clique, but $P' = P \cup \{6\} = \{1, 2, 3, 4, 5, 6\}$ is a 2/3-quasi-clique

For s-club, we can derive a simple branching rule creating exactly $s+1$ branches in each step: If a component P is infeasible, there must exist two vertices $i, j \in V$ with $\mathrm{dist}_{G[P]}(i,j) = s+1$. If these vertices also fulfill $\mathrm{dist}_G(i,j) = s+1$ (recall that generally $\mathrm{dist}_{G[P]}(i,j) \geq \mathrm{dist}_G(i,j)$ holds), then P can be chosen as the vertices of the path with length $s+1$ connecting i with j in G. It remains an open question whether this variant of the GBR is complete. During our experiments, we never found an example (as the one for quasi-clique shown in Figure 3) for s-club and any $s \geq 2$.

The column *Partitioning* of Table 5 summarizes the possible branching schemes presented in this section.

	Partitioning			Covering
Branching scheme	`P1`: | `1.` Ryan Foster	`P2`: | `1.` GBR	`P3`: | `1.` GBR | `2.` Ryan Foster	`C1`–`C3`: | `1.` Vertex contacts fractional | `2.` Vertex contacts fixation | `3.` `P1` or `P2` or `P3` on $V_{=1}$ | `4.` Vertex duplication
Applicable to	s-plex s-clique s-club γ-quasi-clique s-defective clique s-bundle	s-plex, connected s-clique, connected s-defective clique, connected s-bundle, connected	s-plex, general s-clique, general s-club‡ γ-quasi-clique s-defective clique, general s-bundle, general	*all variants*

Table 5: Branching schemes
Branching schemes `C1`, `C2`, and `C3` for covering results from the use of appropriate branching rules `P1`, `P2`, and `P3`, respectively, at level `3` of the scheme. ‡: If GBR is a complete branching rule for s-club, `P2` is applicable instead of `P3`.

5.3. Generic Branching Rules for Relaxed Clique Covering

For covering with RCs, we propose a multi-level branching scheme, where the first two levels decide on so-called vertex contacts and the lower levels assure integrality and apply branching rules known for partitioning, i.e., Ryan-Foster branching or the GBR.

For any subset $T \subseteq V$, the number of *vertex contacts* is defined as $g(T) = \sum_{S \in \mathscr{S}} |T \cap S| \lambda_S$. In order to distinguish between variables and values, we write $g(T)$ for the sum of the variables and $g^\lambda(T)$ for the resulting value. For the sake of convenience, we also define $g_i = g(\{i\})$ and $g_i^\lambda = g^\lambda(\{i\})$ for vertices $i \in V$. Branching on the number of vertex contacts preserves the structure of the subproblem. Indeed, enforcing $g(T) \leq \lfloor g^\lambda(T) \rfloor$ or $g(T) \geq \lceil g^\lambda(T) \rceil$ only changes the weights $w_i, i \in T$ in the subproblem's objective according to the dual price of the constraint. In case of less-or-equal inequalities, weights can become negative.

It may happen that all vertices $i \in V$ have integer vertex contacts g_i^λ, but the solution λ is still fractional. Such a situation is certainly not unusual because in the set-partitioning case all vertex contacts are equal to one and fractional solutions are predominant. Therefore, additional branching actions need to be taken (in the following referred to as *vertex contact fixation*). The intention of our higher-level branching is to fix, for a large subset $P \subseteq V$, the vertex contacts to its minimum, i.e., $g(P) = |P|$, in order to then treat this subset as in the partitioning case. Beside fixation, an alternative branch $g(P) \geq |P| + 1$ must be created, too. Deeper in the tree, branches $g(P) = |P| + 1$ and $g(P) \geq |P| + 2$, and generally $g(P) = |P| + p$ and $g(P) \geq |P| + p + 1$ are created. Note that $g(P) \geq |V| + 1$ is certainly suboptimal so that this process always stops.

If $g(P)$ is fixed as well as all vertex contacts of vertices in P, i.e., all g_i for all $i \in P$ are fixed to specific values, then one can proceed as follows. All vertices with their vertex contacts fixed to 1 form the set $V_{=1} = \{i \in V : g_i \text{ is fixed to } 1\}$. Ryan-Foster branching or the GBR can be applied to $V_{=1}$ for which the smaller support graph $G^\lambda[V_{=1}]$ must be considered. If no branching is possible, a final graph modification procedure can always be applied (referred to as *vertex duplication*). The remaining vertices $V_{>1} = \{i \in V : g_i \text{ is fixed to a value} > 1\}$ are replaced by exactly g_i clones $i^1, \ldots, i^{g_i}$. The clones are adjacent to exactly the same vertices as i. Moreover, the additional separate-constraints that no two clones i^k and i^ℓ for $k \neq \ell$ occur together in a RC are imposed. In all experiments, it was never necessary to apply vertex duplication, since solutions were already integral.

The column *Covering* of Table 5 summarizes the branching schemes for the covering variants.

6. Computational Results

The results reported in this section were obtained using a single thread of a standard PC with an Intel(R) Core(TM) i7-4790 3.6 GHz processor and 8 GB of main memory. The algorithms were coded in `C++` and compiled with MS Visual Studio 2010. The callable library of CPLEX 12.5 was used for solving all LPs and MIPs.

According to Table 2, there exist 17 variants of partitioning and covering problems. These decomposition problems must be parameterized with values for s or γ. Similar to previous works (e.g., Veremyev and Boginski, 2012; Trukhanov *et al.*, 2013; Pattillo *et al.*, 2013b), we use values $s \in \{2, 3, 4, 5\}$ (for defective clique we use $s-1$) and $\gamma = \{.95, .9, .85, .8, .75\}$. Since the number of combinations is large ($13 \cdot 4 + 4 \cdot 5 = 72$), we have restricted our computational analysis to the nine networks from the 10th DIMACS challenge with less than 300 vertices (available at `http://dimacs.rutgers.edu/Challenges/`). This gives rise to an overall of 648 instances.

Table 6 lists the nine networks $G = (V, E)$ and their characteristics: edge density $\rho(G)$, minimum degree $\delta(G)$, maximum independent set size $\alpha(G)$, maximum clique size $\omega(G)$, chromatic number $\chi(G)$ and chromatic number of the complement graph $\chi(\bar{G})$. Note that unlike for maximum-cardinality RCs, no graph reduction by a peeling procedure (cf. Abello *et al.*, 1999) is possible when decomposing the entire network. For all experiments, the computation time is limited to 600 seconds.

$G = (V, E)$	—V—	—E—	$\rho(G)$	$\delta(G)$	$\alpha(G)$	$\omega(G)$	$\chi(G)$	$\chi(\bar{G})$
`karate`	34	78	0.1390	1	20	5	5	20
`chesapeake`	39	170	0.2294	3	17	5	5	17
`dolphins`	62	159	0.0841	1	28	5	5	28
`lesmis`	77	254	0.0868	1	35	10	10	35
`polbooks`	105	441	0.0808	2	43	6	6	43
`adjnoun`	112	425	0.0684	1	53	5	5	55
`football`	115	613	0.0935	7	21	9	9	22
`jazz`	198	2742	0.1406	1	40	30	30	40
`celegansneural`	297	2148	0.0489	1	110	8	[8,9]	115

Table 6: Instance features

Recall that for hereditary Π (not necessarily connected), partitioning and covering are equivalent. Hence, we solve a set-covering master program in which the dual values are more stable and we post-process the solution to obtain a feasible partitioning. In order to stabilize the column-generation process also in the proper partitioning cases, we replace partitioning by covering constraints (1b) and add the additional constraint that the number of vertex contacts must not exceed $n = |V|$, i.e., $\sum_{S \in \mathscr{S}} |S| \lambda_S \leq n$, to the master program (1). The effect is that all dual prices π_i for $i \in V$ are non-negative. The resulting weight for vertex $i \in V$ is then $w_i := \pi_i + \mu$, where μ is the (non-positive) dual price of the additional constraint.

Moreover, we use a multi-column pricing strategy: For MIPs, all integer feasible solutions with negative reduced cost found by CPLEX are added to the master. Similarly, all different solutions found in the main loop of (m)RDS are added. For s-club, we use a different acceleration strategy, i.e., the CB&B prematurely stops as soon as a solution with reduced cost smaller than -0.1 is found.

6.1. Linear Relaxation Results

In a first series of experiments, we analyze the performance of alternative pricing algorithms (see Section 4). For s-plex, we compare the IP formulation of Balasundaram *et al.* (2011) with the (m)RDS (see Section 4.2). For s-club, we compare the IP formulation of Veremyev and Boginski (2012) with our CB&B presented in Section 4.3. For s-bundle, we compare our MIP formulation presented in the companion manuscript Gschwind *et al.* (2017) with the (m)RDS. Finally, for γ-quasi-clique, we compare the MIP formulation of Veremyev *et al.* (2015) with the strengthened one presented in Section 4.1.

Table 7 presents aggregated results over all nine benchmark networks. The numbers are ratios of the average times that a single pricing iteration consumes. Whenever the linear relaxation of (1) is not completely solved within the time limit, the average is taken over the iterations solved up to this point. For s-bundle, CPLEX is only able to solve the pricing problem for the two smallest instances `karate` and `chesapeake` (the others caused an out-of-memory exception). When comparing MIP with CB&B, a ratio $t_{PP}^{MIP}/t_{PP}^{CB\&B}$ of more than 1 indicates that CPLEX needs more time than the respective CB&B, while for γ-quasi-clique a ratio $t_{PP}^{Ver}/t_{PP}^{Own}$ greater than 1 means that CPLEX takes longer for solving the MIP of Veremyev *et al.* (2015) (`Ver`) than for the strengthened MIP (`Own`).

			$s=2$	$s=3$	$s=4$	$s=5$	
s-plex	Partitioning	Connected	405.5	150.1	35.4	4.3	
	Covering	Connected	319.5	103.6	26.0	6.3	
	Part./Cover.	General	269.7	36.4	4.0	0.6	
s-club	Partitioning	General	57.3	124.1	153.7	215.6	
	Covering	General	45.9	79.4	116.8	326.8	
s-bundle	Partitioning	Connected	525425.1	89556.1	18944.2	2772.4	
	Covering	Connected	1374606.2	154631.5	15714.0	3112.4	
	Part./Cover.	General	390207.8	124293.4	14491.8	2169.8	
			$\gamma=0.95$	$\gamma=0.90$	$\gamma=0.85$	$\gamma=0.80$	$\gamma=0.75$
γ-quasi-clique	Partitioning	Connected	1.9	2.0	2.0	1.8	3.1
	Covering	Connected	1.8	1.9	1.1	1.8	3.2
	Partitioning	General	1.8	1.9	2.6	1.9	2.1
	Covering	General	2.0	2.0	2.1	2.4	2.7

Table 7: Comparison of pricing algorithms
Note: Factors are the average ratios $t_{PP}^{MIP}/t_{PP}^{CB\&B}$ and $t_{PP}^{Ver}/t_{PP}^{Own}$

Overall, the CB&B algorithms perform better than the MIPs and with increasing s the effect becomes less pronounced for s-plex and s-bundle. In contrast, the results for s-club show that the MIP-based approach becomes less attractive when s increases. For γ-quasi-clique, the strengthened formulation of Section 4.1 consistently outperforms the MIP of Veremyev *et al.* (2015). The results also seem to indicate that for general s-plex and $s \geq 5$ pricing with MIP is superior to RDS. For consistency among different s values, however, all following results are computed with RDS.

In Table 8, we present absolute computation times for solving the linear relaxations of (1). Numbers is brackets show the number of instances for which the linear relaxation is solved within the time limit; (*) means that all nine instances are solved. If an instance is not solved, it contributes to the presented average with 600 seconds. In all cases, pricing consumes more than 99% of the computation time.

The column-generation algorithm for covering with s-club is able to solve all 36 linear relaxations. For the other problem variants, the algorithms are not able to solve all instances for all values of s or γ. The hardest variants are those for γ-quasi-clique and partitioning with connected s-cliques, while for the hereditary structures almost all instances are solved with $s=2$ and $s=3$. Larger values of s and smaller values of γ lead to harder to solve pricing problems, larger generated RCs, more degenerate master programs typically requiring more iterations, and thereby to longer computation times for the linear relaxation. An exception are the distance-based relaxations s-clique and s-club, where for larger s the decomposition becomes trivial because the given graph is already an s-clique/club. Among the other hereditary structures, the linear relaxation for s-defective clique is solved faster than for s-plex and s-bundle, which seem to be similar. The latter result is somewhat unexpected when comparing with the results of Gschwind *et al.* (2018) where maximum-cardinality s-bundle was harder than s-plex.

The only variants for which partitioning is much more time consuming than covering are connected s-clique and s-club: the presence of some negative weights seems to substantially complicate the pricing. We observe that single instances of the pricing problem require significantly more time than the average. The results for the other variants show that covering is slightly easier than partitioning, but the differences

			$s=2$	$s=3$	$s=4$	$s=5$	
s-plex	Partitioning	Connected	0.7 (*)	47.8 (*)	168.3 (7)	360.4 (4)	
	Covering	Connected	0.5 (*)	39.9 (*)	161.6 (7)	351.3 (4)	
	Part./Cover.	General	0.5 (*)	38.9 (*)	168.2 (7)	350.9 (4)	
s-clique	Partitioning	Connected	267.1 (5)	375.0 (4)	400.0 (3)	339.1 (4)	
	Covering	Connected	67.4 (8)	66.9 (8)	0.1 (*)	0.1 (*)	
s-club	Partitioning	General	214.1 (7)	372.8 (5)	335.7 (4)	95.9 (8)	
	Covering	General	3.5 (*)	8.7 (*)	0.1 (*)	0.1 (*)	
$(s-1)$-defective	Partitioning	Connected	0.5 (*)	3.1 (*)	41.3 (*)	86.2 (8)	
clique	Covering	Connected	0.4 (*)	2.2 (*)	39.4 (*)	76.2 (8)	
	Part./Cover.	General	0.4 (*)	2.4 (*)	32.0 (*)	78.9 (8)	
s-bundle	Partitioning	Connected	0.8 (*)	53.7 (*)	175.3 (7)	362.7 (4)	
	Covering	Connected	0.6 (*)	43.4 (*)	172.4 (7)	347.5 (4)	
	Part./Cover.	General	0.6 (*)	40.4 (*)	166.2 (7)	355.9 (4)	
			$\gamma=0.95$	$\gamma=0.90$	$\gamma=0.85$	$\gamma=0.80$	$\gamma=0.75$
γ-quasi-	Partitioning	Connected	215.6 (6)	222.1 (6)	223.4 (6)	228.9 (6)	296.7 (5)
clique	Covering	Connected	215.3 (6)	220.9 (6)	220.2 (6)	227.9 (6)	291.8 (5)
	Partitioning	General	223.4 (6)	227.8 (6)	232.9 (6)	243.6 (6)	232.7 (6)
	Covering	General	218.3 (6)	232.8 (6)	228.9 (6)	252.5 (6)	260.1 (6)

Table 8: Linear programming relaxation average computation times

are not substantial. We also observe that pricing consumes more time for partitioning due to some negative weights, but the multiple-pricing strategy at the same time produces more RCs leading to a comparable number of pricing iterations. Comparing covering with connected RCs and decomposing with general RCs (both formulated as a set-covering master) shows that computation times are strongly correlated.

6.2. Integer Results

For s-plex, s-club, s-bundle, and γ-quasi-clique we tried to decompose the networks using the compact formulation presented in the companion manuscript Gschwind *et al.* (2017), but the results were disappointing. Only the very smallest networks could be decomposed for some of the RCs even after providing the optimal number $rc(G)$ of necessary relaxed cliques. The poor performance can be attributed to the weak lower bound provided by the linear relaxation (see Gschwind *et al.*, 2017, Theorem 1) and the inherent symmetry of the compact formulation. Another study with more networks is presented in Section 6.3.

We next analyze the performance of the branching rules of Section 5 for partitioning into RCs. Depending on the problem variant, we compare `P1` (Ryan Foster) against `P2` (GBR) or `P3` (GBR followed by Ryan Foster). The node-selection strategy is depth-first in order to find upper bounds early in the search.

Table 9 shows average computation times for solving the integer model. As before, numbers in brackets indicate the number of instances solved to proven optimality. Furthermore, for those instances solved to optimality with both branching schemes, Table 10 gives the size of the branch-and-bound tree (minimum, average, and maximum over the instances).

It can be seen from Tables 9 and 10 that the pure Ryan-Foster branching compares favorably: Average computation times of `P1` are always smaller than those of `P2` and `P3`. There are, however, a few instances for which `P1` takes longer. Scheme `P1` is superior also with respect to the number of optima. All instances solved with `P2` or `P3` are also solved with `P1`. An explanation for this outcome is that GBR-based rules create, with a few exceptions, many more branches than the pure Ryan-Foster rule, see Table 10. Analyzing times and tree sizes together reveals that for GBR-based rules a single branch-and-bound node is solved faster. This is intuitive because GBR is a structure-preserving rule as opposed to the Ryan-Foster rule which requires the use of less effective pricing algorithms (see Section 5.1.1).

Based on these findings, the final series of experiments applies branching scheme `P1` for partitioning and the corresponding scheme `C1` for covering problems. Since linear-relaxation bounds are generally tight, the

		$s=2$		$s=3$		$s=4$		$s=5$	
		P1	P2/P3	P1	P2/P3	P1	P2/P3	P1	P2/P3
s-plex	Connected	133.8 (7)	210.4 (6)	230.1 (6)	357.5 (4)	344.7 (4)	541.5 (1)	443.2 (3)	546.2 (2)
	General	133.9 (7)	200.2 (6)	217.7 (6)	297.0 (5)	345.7 (4)	413.8 (3)	473.4 (2)	475.4 (2)
s-clique	Connected	347.1 (4)	477.1 (2)	452.9 (3)	533.3 (1)	400.0 (3)	400.0 (3)	339.0 (4)	338.7 (4)
s-club	General	335.5 (4)	337.9 (4)	401.3 (3)	401.3 (3)	335.6 (4)	335.6 (4)	94.6 (8)	94.7 (8)
$(s-1)$-defective	Connected	76.3 (8)	335.0 (4)	155.0 (7)	268.0 (5)	214.7 (6)	400.8 (3)	278.7 (5)	351.4 (4)
clique	General	72.7 (8)	218.1 (6)	169.8 (7)	268.7 (5)	219.4 (6)	339.6 (4)	241.0 (6)	412.2 (3)
s-bundle	Connected	133.8 (7)	210.9 (6)	184.0 (7)	401.2 (3)	347.7 (4)	472.9 (2)	444.7 (3)	521.4 (2)
	General	133.9 (7)	200.3 (6)	258.4 (6)	336.5 (4)	358.2 (4)	419.0 (3)	477.3 (2)	498.8 (2)

Table 9: Comparison of branching schemes for RC partitioning: Average computation time and number of optimal solutions

		$s=2$		$s=3$		$s=4$		$s=5$	
		P1	P2/P3	P1	P2/P3	P1	P2/P3	P1	P2/P3
s-plex	Connected	5/11/31	3/1007/5701	1/12/19	1/972/3345	19/19/19	537/537/537	10/15/20	256/729/1201
	General	1/9/20	1/16/37	9/17/22	7/491/2359	9/17/21	13/468/1343	23/25/26	63/81/99
s-clique	Connected	106/237/367	5/5/5	1/1/1	1/1/1	1/1/1	1/1/1	1/1/1	1/1/1
s-club	General	1/2/2	1/34/48	1/1/1	1/1/1	1/1/1	1/1/1	1/1/1	1/1/1
$(s-1)$-defective	Connected	5/12/18	8/1086/4241	2/6/9	2/109/365	5/523/1550	27/121/294	5/14/25	4/758/2675
clique	General	11/3059/18151	32/2275/11734	7/17/24	11/62/217	4/16/21	20/51/79	11/22/39	38/46/51
s-bundle	Connected	5/11/31	3/1007/5701	1/4/6	1/29/83	15/17/18	214/259/304	20/21/22	291/333/375
	General	1/9/20	1/16/37	7/13/26	2/115/423	21/28/42	41/61/91	14/20/26	48/91/133

Table 10: Comparison of branching schemes for RC partitioning: Tree size (min/avg/max)

overall performance of our algorithms very much depends on the ability to find good feasible decompositions (upper bounds) fast. Therefore, we solve the master program as an integer model with CPLEX at every branch-and-bound node. Pre-tests have shown that such a heuristic is particularly helpful for covering variants which often require massive branching before reaching an integer solution. In order to avoid long MIP runs, CPLEX is limited to 10 seconds. Moreover, we change the node-selection rule in our branch-and-price algorithms to best-first search.

Table 11 is organized as Table 9 and displays the average computation times and the number of optima for the branch-and-price. With the help of the upper bounds provided by CPLEX, 429 instances are solved to proven optimality compared to only 405 optima without using the upper bounds. The upper bounds seem to be particularly helpful for larger values of s. This is also true for variants with intricate subproblems such as s-bundle and γ-quasi-cliques. Overall, computation times are also slightly reduced.

In summary, decomposing into RCs is a computationally challenging problem. The difference is more between different types of RCs than between partitioning and covering and between general and connected RCs. A main indicator for the hardness of the decomposition is the hardness of the corresponding pricing problem. It is therefore not surprising that the decomposition with s-clique and s-defective clique works better than with s-bundle.

6.3. Scalability Analysis on Randomly Generated Networks

The nine real-world social networks that we considered in the previous experiments do not allow statements about how our algorithms scale. In order to analyze the impact of network size and density, we have created a larger instance set with the help the social-network generator of Lancichinetti and Fortunato (2009). The generator allows to configure the network and its desired characteristics by specifying the number of vertices, the distribution of the vertex degrees and the community size, and the percentage of internal edges. We vary the number n of vertices between 100 and 1,000 in steps of 100 and fix the community size to 25. Moreover, the vertex degree is fixed using values between 5 and 25 in steps of size 5. This implicitly determines the density $\rho(G)$ of the network G which cannot be specified directly in the instance generator. The proportion of inner and outer community edges is set to 90 % and 10 %, respectively. For each parameter

			$s=2$	$s=3$	$s=4$	$s=5$	
s-plex	Partitioning	Connected	136.9 (7)	233.2 (6)	328.8 (5)	395.4 (4)	
	Covering	Connected	133.5 (7)	333.6 (4)	289.1 (5)	348.1 (4)	
	Part./Cover.	General	133.4 (7)	139.1 (7)	219.0 (6)	452.5 (3)	
s-clique	Partitioning	Connected	414,1 (3)	476,3 (2)	400 (3)	338,2 (4)	
	Covering	Connected	67.2 (8)	66.8 (8)	0.1 (*)	0.1 (*)	
s-club	Partitioning	General	336.4 (4)	401.4 (3)	335.8 (4)	97.6 (8)	
	Covering	General	85.2 (8)	9.9 (*)	0.1 (*)	0.1 (*)	
$(s-1)$-defective	Partitioning	Connected	145.3 (7)	135.1 (7)	201.0 (6)	154.4 (7)	
clique	Covering	Connected	179.4 (7)	133.7 (7)	137.0 (7)	141.7 (7)	
	Part./Cover.	General	136.5 (7)	136.0 (8)	200.8 (6)	78.7 (8)	
s-bundle	Partitioning	Connected	136.9 (7)	180.7 (7)	285.5 (5)	435.9 (3)	
	Covering	Connected	133.5 (7)	335.0 (5)	232.6 (7)	403.8 (3)	
	Part./Cover.	General	133.4 (7)	247.6 (6)	280.7 (5)	350.3 (4)	
			$\gamma=0.95$	$\gamma=0.90$	$\gamma=0.85$	$\gamma=0.80$	$\gamma=0.75$
γ-quasi-	Partitioning	Connected	215.8 (6)	222.2 (6)	225.5 (6)	235.7 (6)	296.9 (5)
clique	Covering	Connected	215.3 (6)	220.8 (6)	220.4 (6)	348.1 (4)	290.3 (5)
	Partitioning	General	223.5 (6)	227.8 (6)	237.4 (6)	254.0 (6)	233.7 (6)
	Covering	General	217.9 (6)	232.5 (6)	229.4 (6)	365.9 (4)	269.9 (6)

Table 11: Branch-and-price average computation time and number of optimal solutions found

set, ten networks are randomly generated. Note that instances with similar characteristics were also used by Girvan and Newman (2002).

We conduct experiments with partitioning/covering the networks with s-plexes for $s = 2$ and 3. One reason for this choice is that s-plex clique relaxations have been widely used in SNA. Moreover, the presented results of the Sections 6.1 and 6.2 clearly show that the RDS-based subproblem solver is sufficiently mature so that larger networks can be tested.

The computational results of a first series of experiments are depicted in Figure 4. It shows for $s = 2$ and $s = 3$, respectively, the impact of the size n and the average vertex degree $\deg_{avg}$ on average computation times for the linear relaxation. As expected, with increasing size n of the network the computation times grow. For a fixed set of parameters, computation times are very comparable so that either all or none of the ten instances were solved. In the latter case there is no data point in the figure. The only exception is for $s = 3$, $n = 300, 400, 600$, and $\deg_{avg} = 25$ where one of the ten instances could not be solved, and for $s = 3$, $n = 700$, and $\deg_{avg} = 25$ where two could not be solved. Generally, instances with smaller average vertex degree are easier to solve. An exception is the case when the average degree equals the community size 25. Our interpretation is that these networks have a rather pronounced community structure making it easier to find the correct communities in the very first column-generation iterations. Summarizing, for s-plex and $s = 2, 3$ our column-generation algorithms provide lower bounds reasonably fast.

Recall that the linear-relaxation bounds of the compact MIP formulation (see Gschwind *et al.* (2017)) for s-plex is 1. It is however known that modern MIP solvers can generally improve such dual bounds by adding various types of valid inequalities. Therefore, we analyze the root node lower bound provided by CPLEX. In order to reduce redundancy from the compact MIP formulation (see Gschwind *et al.* (2017)), we set $\bar{rc}(G)$ to the best known solution for the respective instance. Nevertheless, for most instances that could be solved (the compact model becomes huge when $rc(G)$ is large causing an "out of memory" exception), the root node lower bound is 1. Some exceptions occur for $s = 3$, where the best bound is 1.97 compared to $rc(G) = 4$ or 5. This shows that a direct solution of our graph decomposition problems using a MIP solver is impossible for the networks we generated.

In the next series of experiments, we analyze the strength of the column-generation lower bounds, which are the linear-relaxation bounds and the tree bounds. Since a reliable statement about the gap is only possible for instances solved to optimality, we extended the time limit to 1 hour (3,600 seconds) for the branch-and-

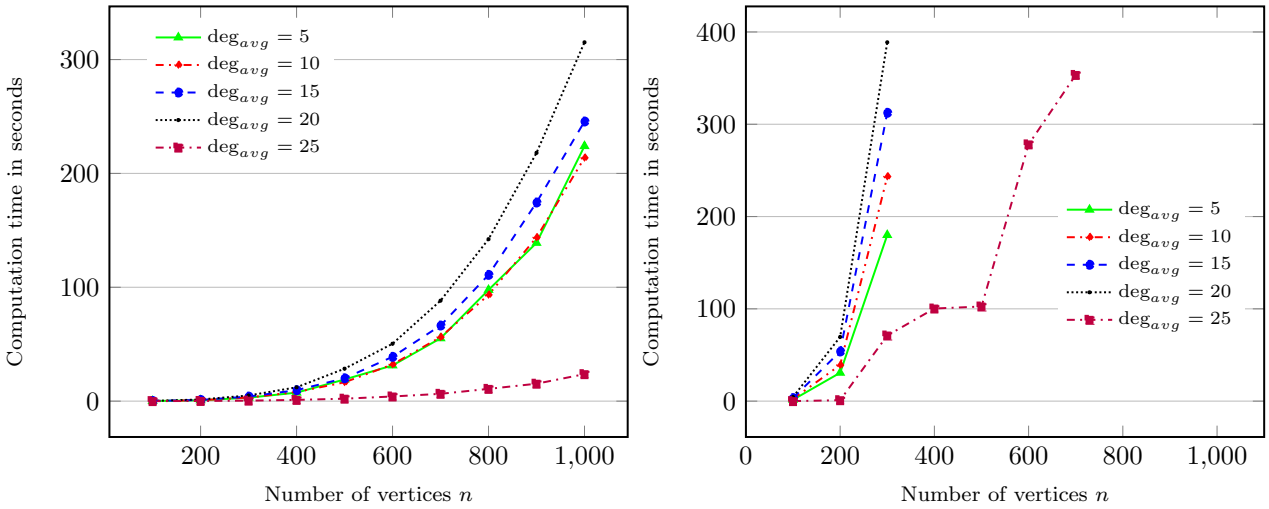

Figure 4: Scalability analysis, computation time for linear relaxation of extensive model: (left) 2-plex partitioning, (right) 3-plex partitioning

price algorithm. As a result, 38 and 103 instances (out of 500 each) were solved to proven optimality for $s = 2$ and 3, respectively. For these instances, the absolute optimality gap $opt - lb$ and relative optimality gap $100\,\%(opt - lb)/opt$ is 1.17 and 7 % for the linear-relaxation bound lb in case of $s = 2$. The respective numbers for $s = 3$ are 0.53 and 3 %.

Finally, we also analyze absolute gaps $gap = ub - lb_{tree}$ where ub is the best known integer solution and lb_{tree} is the tree lower bound resulting from branch-and-price. Also here, the maximum computation time is extended to 1 hour. Figure 5 summarizes the results: The comparison of the left and right figure reveals that partitioning into a minimum number of 3-plexes is easier than in 2-plexes. This results comes unexpectedly because computation times for solving the linear relaxation (see Figure 4) are opposed to this. The explanation for this behavior is that on average optimality gaps for $s = 3$ are smaller than for $s = 2$. Note that the large-scale instances which can be solved with a small integrality gap are those with average vertex degree $\deg_{avg} = 25$. These instances have a very clear community structure so that very good integer solutions can be found quickly. We also see that for the other instances the larger gap results from missing or inferior primal solutions.

Concluding, the branch-and-price algorithm for partitioning into a minimum number of s-plexes scales reasonably well with acceptable precision.

7. Conclusions

Decomposing a graph into RCs is a computationally challenging family of problems that are generalizations of the classical VCP in the complement graph. We designed a unified framework based on branch-and-price techniques for their solution. The column-generation pricing problems of our approach are maximum-weight RCs. While for some types of RCs effective pricing algorithms were available from the literature, we derived new and effective pricing algorithms for several other RCs, namely a strengthened MIP formulation for γ-quasi-clique, a CB&B algorithm for s-club, and a modified version of RDS for hereditary RCs that is able to handle connectivity and negative weights. For branching, we proposed different schemes based on Ryan-Foster branching and/or a newly developed structure-preserving branching rule applicable to all considered variants of RCs. In extensive computational results, we identified the preferred pricing algorithms for the different RCs whenever there was more then one available. Moreover, a comparison of the branching schemes revealed that Ryan-Foster branching is superior although it is not structure-preserving for the pricing problem. Overall, we showed that our exact framework is capable of solving to proven optimality

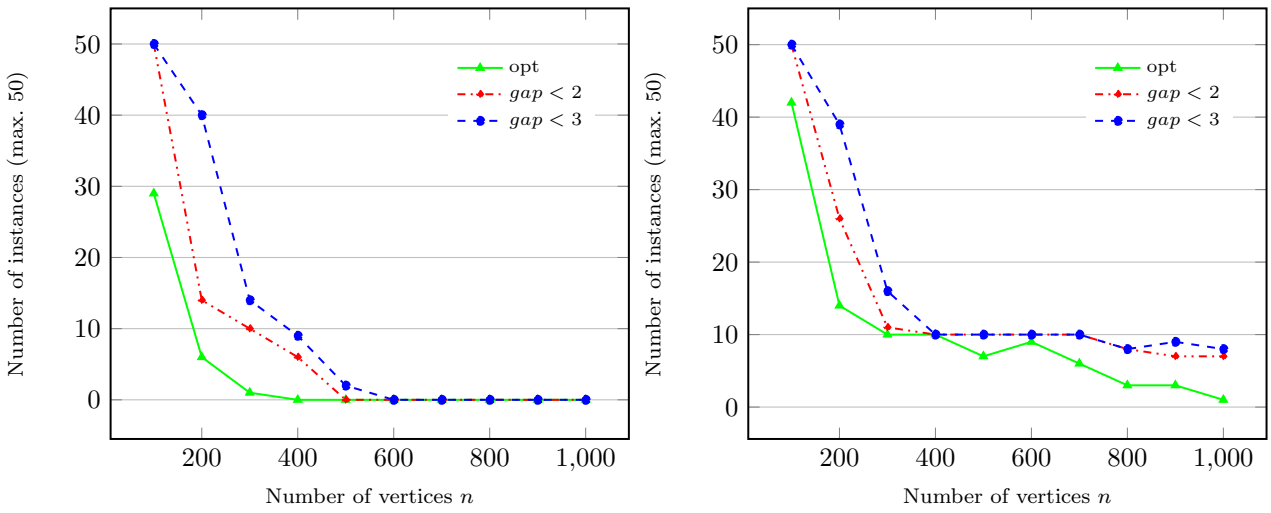

Figure 5: Precision analysis, integer solutions using branch-and-price with a maximum computation time of 1 hour: (left) 2-plex partitioning, (right) 3-plex partitioning

real-world instances with up to 300 vertices and it is able to effectively tackle randomly generated social networks with up to 1000 vertices.

We can identify several lines of future research: For all variants, the pricing problem is a maximum-weight RC problem which is $\mathcal{NP}$-hard and also computationally challenging. Effective (meta)heuristics for the solution of these problems are not available in the literature. The integration of such algorithms would certainly accelerate the column-generation process and, as a consequence, the overall performance of the proposed framework. Finally, the performance could also be improved by the design of new and effective primal heuristic to determine a good-quality heuristic partitioning or covering of the vertex set with the minimum number of relaxed cliques.

References

Abello, J., Pardalos, P., and Resende, M. G. C. (1999). On maximum clique problems in very large graphs. In J. M. Abello and J. S. Vitter, editors, *External Memory Algorithms and Visualization. DIMACS Series on Discrete Mathematics and Theoretical Computer Science*, pages 119–130. American Mathematical Society, Boston, MA.

Almeida, M. T. and Carvalho, F. D. (2012). Integer models and upper bounds for the 3-club problem. *Networks*, **60**(3), 155–166.

Balasundaram, B., Butenko, S., and Hicks, I. V. (2011). Clique relaxations in social network analysis: The maximum k-plex problem. *Operations Research*, **59**(1), 133–142.

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., and Vance, P. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, **46**(3), 316–329.

Bourjolly, J.-M., Laporte, G., and Pesant, G. (2002). An exact algorithm for the maximum k-club problem in an undirected graph. *European Journal of Operational Research*, **138**(1), 21–28.

Chow, F. C. and Hennessy, J. L. (1990). The priority-based coloring approach to register allocation. *ACM Transactions on Programming Languages and Systems*, **12**(4), 501–536.

Cook, V. J., Sun, S. J., Tapia, J., Muth, S. Q., Argüello, D. F., Lewis, B. L., Rothenberg, R. B., and McElroy, P. D. (2007). Transmission network analysis in tuberculosis contact investigations. *Journal of Infectious Diseases*, **196**(10), 1517–1527.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms*. MIT Press, Cambridge, MA and London, 3 edition.

de Werra, D. (1985). An introduction to timetabling. *European Journal of Operational Research*, **19**(2), 151–162.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Fortunato, S. (2010). Community detection in graphs. *Physics Reports*, **486**(3–5), 75–174.

Fortunato, S. and Hric, D. (2016). Community detection in networks: A user guide. *Physics Reports*, **659**(Supplement C), 1 – 44. Community detection in networks: A user guide.

Gamst, A. (1986). Some lower bounds for a class of frequency assignment problems. *IEEE Transactions on Vehicular Technology*, **35**(1), 8–14.

Girvan, M. and Newman, M. E. J. (2002). Community structure in social and biological networks. *Proceedings of the National Academy of Sciences*, **99**(12), 7821–7826.

Gschwind, T., Irnich, S., Furini, F., and Wolfler Calvo, R. (2017). Social network analysis and community detection by decomposing a graph into relaxed cliques. Technical Report LM-2017-06, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

Gschwind, T., Irnich, S., and Podlinski, I. (2018). Maximum weight relaxed cliques and Russian doll search revisited. *Discrete Applied Mathematics*, **234**(Supplement C), 131–138.

Gualandi, S. and Malucelli, F. (2012). Exact solution of graph coloring problems via constraint programming and column generation. *INFORMS Journal on Computing*, **24**(1), 81–100.

Held, S., Cook, W., and Sewell, E. (2012a). Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, **4**(4), 363–381.

Held, S., Cook, W., and Sewell, E. C. (2012b). Maximum-weight stable sets and safe lower bounds for graph coloring. *Mathematical Programming Computation*, **4**(4), 363–381.

Kammer, F. and Täubig, H. (2005). Connectivity. In U. Brandes and T. Erlebach, editors, *Network Analysis*, volume 3418 of *Lecture Notes in Computer Science*, pages 143–177. Springer.

Lancichinetti, A. and Fortunato, S. (2009). Benchmarks for testing community detection algorithms on directed and weighted graphs with overlapping communities. *Physical Review E*, **80**(1).

Lotfi, V. and Sarin, S. (1986). A graph coloring algorithm for large scale scheduling problems. *Computers & Operations Research*, **13**(1), 27–32.

Mahdavi Pajouh, F. and Balasundaram, B. (2012). On inclusionwise maximal and maximum cardinality k-clubs in graphs. *Discrete Optimization*, **9**(2), 84–97.

Malaguti, E. and Toth, P. (2010). A survey on vertex coloring problems. *International Transactions in Operational Research*, **17**, 1–34.

Malaguti, E., Monaci, M., and Toth, P. (2011). An exact approach for the vertex coloring problem. *Discrete Optimization*, **8**(2), 174–190.

Mehrotra, A. and Trick, M. (1998). Cliques and clustering: A combinatorial approach. *Operations Research Letters*, **22**, 1–12.

Östergård, P. R. (2002). A fast algorithm for the maximum clique problem. *Discrete Applied Mathematics*, **120**(1-3), 197–207.

Pattillo, J., Youssef, N., and Butenko, S. (2013a). On clique relaxation models in network analysis. *European Journal of Operational Research*, **226**(1), 9–18.

Pattillo, J., Veremyev, A., Butenko, S., and Boginski, V. (2013b). On the maximum quasi-clique problem. *Discrete Applied Mathematics*, **161**(1–2), 244–257.

Ryan, D. and Foster, B. (1981). An integer programming approach to scheduling. In A. Wren, editor, *Computer Scheduling of Public Transport: Urban Passenger Vehicle and Crew Scheduling*, chapter 17, pages 269–280. Elsevier, North-Holland.

Shahinpour, S. and Butenko, S. (2013). Algorithms for the maximum k-club problem in graphs. *Journal of Combinatorial Optimization*, **26**(3), 520–554.

Trukhanov, S., Balasubramaniam, C., Balasundaram, B., and Butenko, S. (2013). Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations. *Computational Optimization and Applications*, **56**(1), 113–130.

Vanderbeck, F. (2011). Branching in branch-and-price: a generic scheme. *Mathematical Programming*, **130**(2), 249–294.

Veremyev, A. and Boginski, V. (2012). Identifying large robust network clusters via new compact formulations of maximum k-club problems. *European Journal of Operational Research*, **218**(2), 316–326.

Veremyev, A., Prokopyev, O. A., Butenko, S., and Pasiliao, E. L. (2015). Exact MIP-based approaches for finding maximum quasi-cliques and dense subgraphs. *Computational Optimization and Applications*, **64**(1), 177–214.

Verfaillie, G., Lemaître, M., and Schiex, T. (1996). Russian doll search for solving constraint optimization problems. In *Proceedings of the thirteenth national conference on Artificial intelligence*, volume 1, pages 181–187. AAAI Press.

Woo, T., Su, S., and Wolfe, R. N. (2002). Resource allocation in a dynamically partitionable bus network using a graph coloring algorithm. *IEEE Transactions on Communications*, **39**(12), 1794–1801.

Wotzlaw, A. (2014). On solving the maximum k-club problem. Technical Report arXiv:1403.5111v2, Institut für Informatik, Universität zu Köln, Köln, Germany.

Yu, H., Paccanaro, A., Trifonov, V., and Gerstein, M. (2006). Predicting interactions in protein networks by completing defective cliques. *Bioinformatics*, **22**(7), 823–829.