Mixed-Integer Programming Techniques for the Connected Max-k-Cut Problem

Christopher Hojny[1], Imke Joormann[2],
Hendrik Lüthen[1], Martin Schmidt[3]

Abstract. We consider an extended version of the classical Max-k-Cut problem in which we additionally require that the parts of the graph partition are connected. For this problem we study two alternative mixed-integer linear formulations and review existing as well as develop new branch-and-cut techniques like cuts, branching rules, propagation, primal heuristics, and symmetry breaking. The main focus of this paper is an extensive numerical study in which we analyze the impact of the different techniques for various test sets. It turns out that the techniques from the existing literature are not sufficient to solve an adequate fraction of the test sets. However, our novel techniques significantly outperform the existing ones both in terms of running times and the overall number of instances that can be solved.

1. Introduction

In this paper we study a special version of the graph partitioning problem in which all parts of the partition have to be connected. The objective is to maximize the number of edges between the parts. We call it the connected Max-k-Cut problem or C-Max-k-Cut for short.

On the one hand, Max-k-Cut is a classical graph theoretical problem and on the other hand, connectivity is a commonly studied restriction for graph problems; see also Section 1.1. Furthermore, our reason to study the C-Max-k-Cut stems from an application: it is a subproblem in computing a market splitting for electricity markets; see, e.g., [3, 21, 22, 34]. The C-Max-k-Cut problem is NP-hard (see Section 2), and it is also hard to solve with out-of-the-box mixed-integer linear programming (MILP) solvers; see Section 8. One reason for the latter is the strong inherent symmetry of the problem. Hence, we focus on developing specialized branch-and-cut techniques to speed up the solution process.

C-Max-k-Cut emerges in other applications as well. For example, forest planning problems are studied in [8], where an important constraint is that, for a number of different planning problems, old tree populations must stay connected. Another related problem is that of finding a coloring of a graph such that each color induces a connected subgraph. However, in this context typically other objectives are used than the one described above. A further application is phylogenetics, where phylogenetic trees are used to model the relationship between different species based upon their similarities and differences [37]. Biological constraints for the characteristics then demand connectivity [10]. Additionally, image segmentation can be done via graph cuts. In this field, imposing connectivity constraints also seems to be beneficial [49].

In the last couple of years, the focus on Max-k-Cut was on approximation algorithms, especially using (complex) semidefinite programming; see, e.g., [19, 42].

Date: June 6, 2019.
2010 *Mathematics Subject Classification.* 90C11, 90C35, 90C57.
Key words and phrases. Max-cut, Connectivity, Branch-and-cut, Mixed-integer programming.

In contrast, our interest is on exact methods that deliver globally optimal solutions for C-Max-k-Cut. However, due to both the theoretical and computational hardness of the problem, out-of-the-box MILP solvers typically cannot produce such solutions within an acceptable amount of time. Our contribution is the following. We augment a general-purpose MILP solver by (i) tweaking known techniques from the existing literature for related problems and by (ii) developing new MILP-techniques that are tailored for the C-Max-k-Cut problem. Regarding the adapted techniques from the literature, we consider cutting planes for MaxCut and Max-k-Cut, standard concepts for primal heuristics like rounding heuristics, and symmetry handling. For a general treatment of such MILP-techniques we refer to [29]. Up to this point, the connectivity constraints are not yet exploited. Hence, we also develop methods specifically for C-Max-k-Cut—namely branching rules, a propagation algorithm, and further cuts. As it turns out, the novel techniques presented in this paper yield a tailored branch-and-cut algorithm that significantly outperforms the method obtained by using the techniques from the existing literature.

In the remainder of this section, we further discuss related literature. Afterward, we give the main definitions and two MILP models—a flow- and a cut-based formulation—for the C-Max-k-Cut in Section 2. In Section 3–6, we describe the details of our tailored MILP-techniques regarding cuts, propagation, branching rules, and primal heuristics. Another focus is on symmetry breaking, which is discussed in Section 7. The computational results are presented in Section 8 and we give our conclusions in Section 9.

1.1. **Related Literature.** Especially for $k = 2$, C-Max-k-Cut has been studied before. In [23], the authors show how the connectivity constraints may change MaxCut: The fraction of the edges in the cut can be arbitrarily close to zero. Furthermore, they prove that C-Max-2-Cut is NP-hard even for planar graphs. In [24], NP-hardness of a slightly different problem is shown. The authors consider MaxCut where only one side has to be connected. For this problem, they present an $\Omega(1/\log n)$ approximation algorithm. Moreover, for bounded genus graphs, the existence of an $\mathcal{O}(1/2-\varepsilon)$ approximation algorithm for a fixed ε is given in [36]. For series-parallel graphs, a linear time algorithm for C-Max-2-Cut is presented in [9].

Coming from the application of phylogenetic trees, [10] describes another interesting approach. They search for a "convex coloring", where convex means that the induced subgraph for each color is connected. After starting with a partial coloring of the nodes, they try to find a minimal number of nodes that must be redyed so that a convex coloring can be obtained. Due to their application, the authors' study is restricted to trees. C-Max-k-Cut with a nonlinear objective function is considered in [44], where different local search methods are used for solving problems from political districting.

As mentioned above, connectivity constraints arise in a number of different graph problems. We now discuss a small selection of them. In [30] sufficient conditions are derived under which a connected partition of the edges exists such that every part has the same number of edges. The general modeling of connectivity constraints is discussed in [17]. The authors compare a compact flow formulation with a formulation using an exponential number of cuts, which can be separated efficiently. Their suggestion is a formulation with variables on nodes and constraints on node-cut sets. Furthermore, they show how to strengthen the cut inequalities. The polyhedral properties of the connectivity cuts are analyzed in [50]. A different approach is to use edge variables and model connectivity with a tailored objective function. We refer to the survey article [51] for this case, where the connection to Steiner trees is discussed as well.

In [8], the above mentioned forest planning is studied. However, the main focus is on connected subgraphs instead of graph partitioning. The connectivity constraints only use node variables and special node-cut sets.

The related connected subgraph problem is considered in [14], where a maximum-weight connected subgraph with some additional characteristics is searched. For example, this has an application in wildlife conservation. To model the connectivity, the authors use single- and multi-commodity flow techniques as well as directed Steiner trees. The multi-commodity formulation is larger than the single-commodity formulation and the authors observe that the larger formulation results in a stronger LP relaxation of the problem. Moreover, reduction techniques and heuristics for the maximum connected subgraph problem are investigated in [43].

Finally, in [46], the hardness of C-Max-2-Cut with the additional requirement that the supplies in both parts are balanced is analyzed. This has an application in power grid islanding, which is used to soften propagating failures.

2. Problem Statement

Let $G = (V, E)$ be an undirected simple and connected graph and let $k \geq 2$ be an integer. The connected Max-k-Cut problem can then be formulated as the following mixed-integer linear problem:

$$\max_{x,y} \quad \sum_{\{u,v\}\in E} w_{uv} y_{uv} \tag{1a}$$

$$\text{s.t.} \quad \sum_{i\in[k]} x_{vi} = 1, \qquad v \in V, \tag{1b}$$

$$x_{ui} - x_{vi} \leq y_{uv}, \qquad \{u,v\} \in E,\ i \in [k], \tag{1c}$$

$$x_{vi} - x_{ui} \leq y_{uv}, \qquad \{u,v\} \in E,\ i \in [k], \tag{1d}$$

$$x_{ui} + x_{vi} + y_{uv} \leq 2, \qquad \{u,v\} \in E,\ i \in [k], \tag{1e}$$

$$x_{vi} \in \{0,1\}, \qquad v \in V,\ i \in [k], \tag{1f}$$

$$y_{uv} \in \{0,1\}, \qquad \{u,v\} \in E, \tag{1g}$$

$$x \in \mathcal{C}, \tag{1h}$$

where we use $[k] \coloneqq \{1, \dots, k\}$ as an abbreviation throughout the paper. Here and in what follows, the abstract constraint $x \in \mathcal{C}$ in (1h) models connectivity of the parts of the partition that is given by the vector $x = (x_{vi})_{v\in V, i\in[k]}$. The details are discussed below. The binary variables x_{vi} model whether vertex v is located in part i of the partition ($x_{vi} = 1$) or not ($x_{vi} = 0$). Furthermore, cut edges $\{u, v\}$ have $y_{uv} = 1$ and $y_{uv} = 0$ holds if and only if the vertices u and v are located in the same part. The objective function aims to maximize the total weight $w^\top y$ of the cut.

This problem is NP-hard. Considering the weighted case, the hardness follows since our problem contains MaxCut ($k = 2$), which is NP-hard even for the complete graph [33] (both for positive or free weights), where the connectivity is trivially fulfilled. For the unweighted case, i.e., $w_{uv} = 1$ for all $\{u, v\} \in E$, the hardness is shown in [23] even for planar graphs.

If all weights are positive and instead of maximizing the objective, we minimize it, the connectivity constraint is not relevant: suppose there is a solution in which a part decomposes into two connected components V_1 and V_2. Then there must be (at least) two arcs connecting V_1 and V_2 to other parts, since we assume that the graph is connected. Hence, there are less arcs in the cut if we add V_1 or V_2 to another part, i.e., we obtain a better solution. Therefore, the minimization version is equivalent to the min-k-cut problem, which is known to be NP-hard for arbitrary k and solvable

in $\mathcal{O}(|V|^{k^2})$ for constant k [20]. If the weights are allowed to be negative, we can transform every minimization instance into the corresponding C-Max-k-Cut with negated objective function. Hence, this version is also NP-hard.

In this paper, we discuss two different possibilities to model the connectivity requirement of Constraint (1h): a flow and a cut formulation. For the former, we follow the modeling of [48] and [21, 34] that works on the bi-directed graph $G' \coloneqq (V, A)$ with arc set A consisting of arcs $a_e^1 = (u, v)$ and $a_e^2 = (v, u)$ for all $e = \{u, v\} \in E$. Moreover, we introduce flow variables f_{uv} for every arc $(u, v) \in A$. Connectivity of the parts of the partition can then be modeled using the constraints

$$\sum_{u \in V} z_{ui} = 1, \qquad i \in [k], \tag{2a}$$

$$z_{ui} \leq x_{ui}, \qquad u \in V,\ i \in [k], \tag{2b}$$

$$f_{uv} + f_{vu} \leq M(1 - y_{uv}), \qquad \{u, v\} \in E, \tag{2c}$$

$$\sum_{(u,v) \in \delta_u^{\text{out}}} f_{uv} - \sum_{(v,u) \in \delta_u^{\text{in}}} f_{vu} \geq 1 - M \sum_{i \in [k]} z_{ui}, \qquad u \in V, \tag{2d}$$

$$f_{uv}, f_{vu} \in \mathbb{R}_{\geq 0}, \qquad \{u, v\} \in E, \tag{2e}$$

$$z_{ui} \in \{0, 1\}, \qquad u \in V,\ i \in [k], \tag{2f}$$

where we used the standard δ-notation

$$\delta_u^{\text{in}} \coloneqq \{a \in A : \text{there exists } v \in V \text{ with } a = (v, u)\},$$
$$\delta_u^{\text{out}} \coloneqq \{a \in A : \text{there exists } v \in V \text{ with } a = (u, v)\}$$

for in- and outgoing arcs of a node u. Here and in what follows, M is a sufficiently large constant. In our specific setting, we can choose $M = |V| - k + 1$.

The idea of the flow formulation is to declare one node of a part as an "artificial sink" and to route flow from every other node of the part to the sink by using only non-cut edges to ensure connectivity. The newly introduced variables z_{ui} model whether vertex u is the artificial sink of the part i. The Constraints (2a) and (2b) state that every part has exactly one sink. Constraint (2c) models that flow is only allowed on arcs that are not in the cut. Finally, Constraint (2d) models that if u is not an artificial sink, it has to have an outflow of at least 1. Otherwise the constraint is redundant. Note that Constraints (2a) and (2b) ensure that every part $i \in [k]$ contains at least one vertex. Thus, every part is connected and non-empty.

Besides the flow formulation (2) further approaches to enforce connectivity of the parts of the partition are discussed in the literature. On the one hand, one can model connectivity via multi-commodity flow formulations; see, e.g., [14, 21, 34]. On the other hand, connectedness can also be modeled directly without using additional variables but by using additional constraints [8]. In this paper, we combine the concepts of adding additional variables and additional constraints to enforce connectivity. We decide to use the single-commodity flow formulation (2) instead of the multi-commodity flow formulation because both formulations model connectivity, but the single-commodity version introduces fewer auxiliary variables.

Next, we introduce the cut formulation, for which we need the concept of induced subgraphs. Given a graph $G = (V, E)$ and a vertex subset $V' \subseteq V$, the *induced subgraph* on V' is $G[V'] = (V', E')$ with $E' \coloneqq \{\{u, v\} \in E : u, v \in V'\}$.

Definition 2.1. Let $G = (V, E)$ be an undirected graph. For two distinct vertices $u \neq v \in V$ with $\{u, v\} \notin E$ the set $S \subset V$ is called an *u-v-separator* if the induced graph $G[V \setminus S]$ contains no path from u to v. A separator S is called *minimal* if no proper subset of S is a separator. The set of all minimal u-v-separators is denoted by $\mathcal{S}_{uv}$.

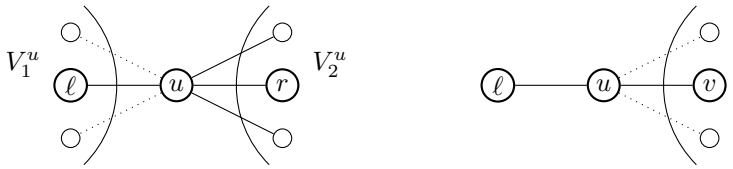

FIGURE 1. Articulation-vertex cuts (left) and leaf cuts (right).

Using this definition, connectivity can also be modeled by imposing the inequalities

$$\sum_{w \in S} x_{wi} \geq x_{ui} + x_{vi} - 1, \quad S \text{ is a } u\text{-}v\text{-separator}, \ i \in [k]. \tag{3}$$

Moreover, only imposing the minimal separators

$$\sum_{w \in S} x_{wi} \geq x_{ui} + x_{vi} - 1, \quad S \in \mathcal{S}_{uv}, \ i \in [k], \tag{4}$$

suffices, since they dominate the remaining separator inequalities. The latter inequalities can be separated by maximum-flow calculations, but the authors of [16] refrain from using this procedure because it is too time consuming.

As we want to partition the graph into exactly k parts, we also add the non-emptiness constraints

$$\sum_{u \in V} x_{ui} \geq 1, \quad i \in [k].$$

Note that these are not necessary if we assume all edge weights to be strictly positive or if we use the flow formulation due to (2a) and (2b).

3. CUTS

In this section, we discuss cuts for the considered problem. We start with the novel ones derived from the connectivity condition (Section 3.1–3.3) and then proceed with the review of known cuts from the MAXCUT or MAX-k-CUT literature (Section 3.4–3.6). For an overview of cuts for MAX-k-CUT see, e.g., [4]. Furthermore, we state some implementation details, e.g., about separation routines, and give a brief comparison in Table 1 at the end of this section. Later, in Section 8 we evaluate the computational gain of the presented cuts in a detailed numerical study.

3.1. Articulation-Vertex Cuts.

To state the first cut, we first have to give another definition.

Definition 3.1. An *articulation vertex* is a vertex $u \in V$ such that the graph without u is no longer connected, i.e., $G[V \setminus \{u\}]$ decomposes into at least two components. We denote the set of all articulation vertices by $\mathcal{A}$.

In particular, every articulation vertex u is a minimal separator. For an articulation vertex u, we denote the resulting c connected components by V_j^u, $j \in [c]$, and we identify a component with its vertex set. Let ℓ and r be two vertices in different components. Then the following special case of (4) holds:

$$x_{ui} \geq x_{\ell i} + x_{ri} - 1, \quad u \in \mathcal{A}, \ \ell \in V_j^u, \ r \in V_p^u, j \neq p, \ i \in [k], \tag{5}$$

see Figure 1 (left).

In principle, one can add all articulation-vertex cuts at the beginning of the branch-and-bound algorithm. However, our numerical experiments revealed that for many graphs of our test sets there are simply too many to be efficient. Note that for separating these cuts it suffices to find a violated inequality for the maximal values $x_{\ell i}$ and x_{ri}, respectively, which has a running time of $\mathcal{O}(k|\mathcal{A}||V|)$ after a preliminary

calculation of the articulation vertices and the corresponding connected components. Testing all possible inequalities of type (5) is thus possible in $\mathcal{O}(k|\mathcal{A}||V|^2)$. This procedure turned out to be beneficial in our case and was implemented in all cases where separator inequalities are used.

Since there were initially too many of these cuts, we tried an even more restricted form of cuts, which is explained next. To this end, we denote for a vertex v of an undirected graph the set of incident edges by $\delta(v)$. The set of all neighbors of v, i.e., the nodes that are adjacent to v, is denoted by $N(v)$.

3.2. **Leaf Cuts.** Let $\ell \in V$ be a leaf of G, i.e., $|\delta(\ell)| = 1$. Furthermore, let u be the unique neighbor of ℓ, i.e., $N(\ell) = \{u\}$. This, in particular, means that u is an articulation vertex and a minimal separator. Hence, the following inequality holds:

$$x_{\ell i} + x_{vi} - x_{ui} \leq 1, \quad \ell, u \in V : N(\ell) = \{u\},\ v \in V \setminus \{\ell, u\},\ i \in [k],$$

see also Figure 1 (right) for an illustration. As for the articulation-vertex cuts, the number of these inequalities is still too large for many of our examples, because we have to consider for each part all pairs of leaves and (basically) all remaining nodes of G. We added another separation routine for these inequalities, which iterates over all possible leaves and checks whether the leaf cut is violated. This routine has a running time of $\mathcal{O}(k|V|^2)$.

3.3. **Bounded-Edge Cuts.** A simple cut involving all y-variables is given by

$$\sum_{\{u,v\} \in E} y_{uv} \leq |E| - |V| + k. \tag{6}$$

This inequality is valid since

$$\sum_{\{u,v\} \in E} y_{uv} \leq |E| - \sum_{i=1}^{k} (n_i - 1) = |E| - |V| + k,$$

where n_i is the number of nodes in part i of the partition. Observe that the first estimation is valid because each connected part contains at least $n_i - 1$ edges. The last equality follows from Constraint (1b).

One drawback of (6) might be that all edge variables are involved; a cut linking decisions about fewer edges could be more significant. Hence, if we can identify certain subsets of vertices where less than k different parts are possible, this might lead to stronger cuts. To this end, we want to utilize the additional information from vertices that are already assigned to a certain part. This means going from the globally valid cut (6) to locally valid cuts, i.e., cuts that are only valid in certain sub-trees of the branch-and-bound tree. Herewith, we can generalize the idea of (6): We denote by $F_i \coloneqq \{u \in V : x_{ui} = 1\}$ the set of vertices that are fixed to be in part i of the partition; see also Figure 2. Let G_i be the graph defined by removing all nodes that are fixed to part i, i.e., $G_i \coloneqq G[V \setminus F_i]$. Assume that G_i consists of the connected components $C_1, \dots, C_q$. For every connected component C_ℓ we define $\tilde{C}_\ell \coloneqq C_\ell \cup \{u \in F_i : \exists v \in C_\ell \text{ with } \{u, v\} \in E\}$ and $\tilde{G}_\ell \coloneqq G[\tilde{C}_\ell]$. For $\ell \in [q]$, let

$$k_\ell \coloneqq \left| \left\{ j \in [k] : \text{there exists } v \in \bigcup\nolimits_{r \in [q] \setminus \{\ell\}} C_r \text{ with } x_{vj} = 1 \right\} \right|,$$

i.e., k_ℓ denotes the number of parts that contain at least one node from a connected component C_r, $r \neq \ell$.

Since we know that if a node from C_r is in part j, no node $u \in C_\ell$, $r \neq \ell$, can be in part j, we can extend (6) to

$$\sum_{\{u,v\} \in E[\tilde{G}_\ell]} y_{uv} \leq |E[\tilde{G}_\ell]| - |V[\tilde{G}_\ell]| + k - k_\ell \tag{7}$$

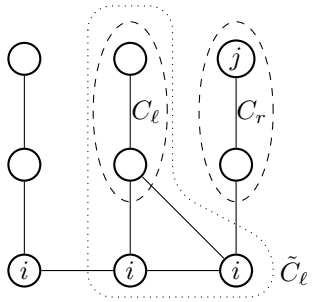

FIGURE 2. Local bounded-edge cuts. Vertices are inscribed with the number of their part if they are already fixed.

for every $\ell \in [q]$. Note that (7) is (6) applied to the subgraph induced by $\tilde{C}_\ell$ with the additional subtraction of k_ℓ. In particular, if $k = 2$, the global and local cuts coincide: If there exists $r \neq \ell$ such that $u \in C_r$ is in part j, then no node in C_ℓ can be contained in part j. Hence, all nodes in C_ℓ can be assigned to the other part and every y_{uv}, $\{u, v\} \in E[\tilde{G}_\ell]$, has value 0. Consequently, the global bounded-edge cut reduces to the local cut on $E[\tilde{G}_r]$.

Note that the global cut is completely dense, while the local cuts may be much sparser if $E[\tilde{G}_\ell]$ is small. Thus, because dense constraints may slow down the solution process [40], it might be favorable to separate the local cuts instead of adding the global cut initially. These cuts can be separated by computing the graph $\tilde{G}_\ell$ as well as the corresponding value k_ℓ, which can be done in $\mathcal{O}(k(|V|+|E|))$ time, and adding the locally valid cut if (7) is violated.

3.4. **Odd-Cycle Cuts.** In the remainder of this section we review some known cuts from the literature that we later also use to enhance our branch-and-cut framework.

For $k = 2$, any cycle crosses a cut in the graph an even number of times, which implies that the so-called odd-cycle cuts [5] are feasible:

$$\sum_{\{u,v\}\in B} y_{uv} - \sum_{\{u,v\}\in Z\setminus B} y_{uv} \leq |B| - 1, \qquad B \subseteq Z,\ |B| \text{ odd}.$$

Here, $Z \subseteq E$ is a cycle in G. We separate these inequalities in polynomial time using a shortest-path calculation in an auxiliary graph G'; see [5]. Let $G' = (V', E')$ be derived from $G = (V, E)$ by having two nodes u' and u'' for each $u \in V$. Every edge $\{u, v\} \in E$ gives rise to two edges $\{u', v'\}$ and $\{u'', v''\}$ in E' with weights y_{uv} and two edges $\{u', v''\}$ and $\{u'', v'\}$ with weights $1 - y_{uv}$. For every vertex $u \in V$, we calculate a shortest path from u' to u''. The minimum length of all these paths gives rise to the required cycle. The running time of this separation procedure is $\mathcal{O}(|V|(|V|\log|V| + |E|))$ by using Dijkstra's algorithm.

3.5. **Triangle Cuts.** For $k = 2$, no cut can contain exactly one edge from any triangle. Therefore, if the nodes u, v, w form a triangle, the following cuts are also valid [11]:

$$y_{vw} - y_{wu} - y_{uv} \leq 0, \quad -y_{vw} + y_{wu} - y_{uv} \leq 0, \quad -y_{vw} - y_{wu} + y_{uv} \leq 0.$$

We separate these cuts by first calculating all triangles in G and then iterating over all of them to check if there is a violated one.

3.6. **Clique Cuts.** For any subset V' with $|V'| = k+1$ vertices, at least two vertices have to be in the same part. Hence, if $G[V']$ forms a clique, at least one edge is not

TABLE 1. Overview of discussed cutting planes.

cutting plane	k	separation complexity	reference
articulation vertex cut	arbitrary	$\mathcal{O}(k\|\mathcal{A}\|\|V\|^2)$	Section 3.1
leaf cut	arbitrary	$\mathcal{O}(k\|V\|^2)$	Section 3.2
bounded-edge cut	arbitrary	$\mathcal{O}(\|E\|)$	Section 3.3
local bounded-edge cut	arbitrary	$\mathcal{O}(k(\|V\|+\|E\|))$	Section 3.3
odd cycle cut	$\{2\}$	$\mathcal{O}(\|V\|(\|V\|\log\|V\|+\|E\|))$	[5]
triangle cut	$\{2\}$	$\mathcal{O}(\|V\|^3)$	[11]
clique cut	arbitrary	$\mathcal{O}(\|V\|^{k+1})$	[11]

in the cut. Thus, the so-called clique inequalities hold [11]:

$$\sum_{u,v\in V'} y_{uv} \leq \binom{k+1}{2} - 1, \quad V' \subseteq V,\ |V'| = k+1,\ G[V'] \text{ clique.}$$

To separate these cuts, we compute, in the beginning of the branch-and-cut algorithm, a list containing all cliques of size $k+1$ by a brute-force method, which runs in $\mathcal{O}(|V|^{k+1})$ time. In each separation round of the branch-and-cut method, we iterate over the list of cliques and check whether a corresponding inequality is violated. Since the initialization is very costly and the size of the list may grow exponentially in k, we use this routine only for $k \leq 3$.

Table 1 summarizes the cutting planes discussed in this section. The column "cutting plane" contains the names of the considered cuts and column "k" specifies for which values of k the corresponding cutting planes are defined. Column "separation complexity" reports on the complexity of separating the cuts and column "reference" provides references for the discussed families of inequalities.

4. Propagation Algorithm

In this section we state an algorithm for propagating the information of vertices that are already assigned to a part of the partition to non-assigned ones. Hence, we consider the situation in which we are given a partial solution x, i.e., some variables are set to 1, some to 0, and the remaining ones are still fractional or unfixed. Given a partial solution, the propagation algorithm 1 assigns vertices to a certain part if this assignment is necessary for connectedness. Moreover, if an unassigned vertex cannot be reached from vertices of part i, then that vertex cannot be in this part. In the following, we denote by $F_i := \{u \in V : x_{ui} = 1\}$ the set of vertices that are fixed to be in part i of the partition.

The first aim of Algorithm 1 is to detect vertices $u \in V$ that cannot be assigned to a part i. If such a vertex u is detected, the algorithm sets $x_{ui} = 0$. To find such vertices for part i, we remove all nodes from G which have already been assigned to a part $i' \neq i$, which results in a graph $\tilde{G}$ (Line 2). Afterward, we select a vertex $q \in F_i$ and compute all nodes in $\tilde{G}$ that are reachable from q by a breadth-first search (BFS). Let T be the obtained BFS tree. If there exist vertices u in $\tilde{G}$ that are not contained in T, every path from q to u in G has to traverse a vertex assigned to another part. Hence, assigning both q and u to part i results in a disconnected part. Consequently, if there exists a vertex assigned to part i that is not contained in T, the partial solution cannot be extended to a connected solution—it is infeasible; see Line 4. Otherwise, x_{ui} can be set to 0 for all unassigned vertices in $\tilde{G}$ that are not in T; see Line 5.

The second idea incorporated in Algorithm 1 is to assign a vertex u to part i if every extension of the partial solution satisfies $x_{ui} = 1$. This is the case if there exists an unassigned articulation vertex u in $\tilde{G}$ that separates two vertices contained in F_i. Thus, such articulation vertices need to be assigned to the same part as well in order to ensure connectedness; see Line 8.

Algorithm 1 Propagation.

Input: A graph $G = (V, E)$, a partial solution x, and the corresponding sets F_i

1: **for all** $i \in [k]$ **do**
2: Set $\tilde{G} = (\tilde{V}, E[\tilde{V}])$ with $\tilde{V} \leftarrow V \setminus \bigcup_{j \neq i \in [k]} F_j$.
3: Let q be some arbitrary vertex in F_i and set $T \leftarrow$ BFS-T$(\tilde{G}, q)$.
4: **if** there exists $u \in F_i \setminus T$ **then** partial solution is infeasible, **terminate**
5: **for all** unassigned $u \notin T$ **do** $x_{ui} = 0$
6: Compute the set of all articulation vertices $\mathcal{A}$ of $\tilde{G}$.
7: **for all** vertices $u \in \mathcal{A}$ with $u \notin F_i$ **do**
8: **if** u separates two vertices from F_i **then** $x_{ui} = 1$

9: **function** BFS-T(G,u)
10: Compute a spanning tree T of G starting from u using BFS.
11: **return** T

The running time of this algorithm is $\mathcal{O}(k|V||E|)$, which can be seen as follows: Line 1 needs $\mathcal{O}(k)$ time. The part inside the for-loop is dominated by the inner for-loops over the articulation vertices. Computing them in Line 6 can be done in time $\mathcal{O}(|V| + |E|)$ by one breadth-first-search call and some linear checking [26]. Iterating over the articulation vertices in Line 7 is in time $\mathcal{O}(|V|)$ and checking if a vertex separates parts containing at least one assigned vertex in Line 8 can be realized by, e.g., one BFS call, and linear checking in time $\mathcal{O}(|V| + |E|)$.

As the algorithm depends only on components rather than specific vertices, one idea to improve the running time is to contract edges if both endpoints are assigned to the same part of the partition. This leads to smaller graphs on which the propagation might be faster. However, since the algorithm can be applied in every node of the branch-and-bound tree in principle, it is very costly to keep the contracted graph up to date: Suppose we are given two nodes b and b' of the branch-and-bound tree that are treated consecutively and let p be their first common predecessor. To provide the correct contracted graph in b', we basically have two choices. On the one hand, we can expand the edges that have been contracted along the path from p to b to find the contracted graph at p and then contract the edges along the path from p to b'. One the other hand, we can compute the contracted graph from scratch at each node of the branch-and-bound tree. Consequently, the cost of contracting and expanding as well as the handling of the corresponding data structures may exceed the time saved by a faster propagation routine. Of course, to keep the changes in the contracted graph as small as possible, one can use a depth first search node selection strategy in branch-and-bound. Except for feasibility problems, however, a pure depth first search strategy is not beneficial in general; see [1]. Thus, we refrained from implementing this idea.

5. Branching Rules

A central component of a branch-and-bound algorithm are branching rules. A *variable-based* branching rule typically selects a variable x_i with violated integrality constraint and creates two subproblems by either adding $x_i \leq \lfloor \bar{x}_i \rfloor$ or $x_i \geq \lceil \bar{x}_i \rceil$ to

its child nodes, where $\bar{x}$ is the relaxation's solution at the current node. *Constraint-based* branching rules, on the contrary, create two subproblems by choosing a hyperplane $a^\top x = \beta$ and adding either $a^\top x \leq \beta$ or $a^\top x \geq \beta$ to a child of an open node. For more details on branching rules, we refer to, e.g., [2, 38].

In the following, we describe three variable-based and one constraint-based branching rule for the C-Max-k-Cut problem. To this end, we denote the set of all x-variables that are fixed to value α in a given node b of τ by F_b^α.

5.1. **Articulation-Vertex Branching.** By definition, a graph splits into at least two connected components $C_1, \dots, C_s$ if an articulation vertex u is removed from the graph. For C-Max-k-Cut, this implies that if u is assigned to a specific part $i \in [k]$, every remaining part in $[k] \setminus \{i\}$ contains vertices from exactly one connected component C_r, $r \in [s]$; see Section 4. Thus, once an articulation vertex is assigned to a part, the number of possible assignments of vertices to parts drastically reduces since at most one part may contain vertices from more than one of the connected components. Based on this observation, the idea of the *articulation-vertex branching rule* is to assign articulation vertices to a part first.

In our implementation, we compute all articulation vertices of G, which can be done in $\mathcal{O}(|V| + |E|)$ time, see [26], when the branch-and-bound algorithm is initialized and we store the set $\mathcal{A}$ of articulation vertices for the entire algorithm. Whenever the branching rule is called in a node b of τ, we select a vertex $u \in \mathcal{A}$ and check whether u has not yet been assigned. In this case and for each part $i \in [k]$ with $x_{ui} \notin F_b^0$ we generate a child node of b in which we fix $x_{ui} = 1$. Otherwise, if u is already assigned to a part, we analogously proceed with another articulation vertex, until we either created child nodes or we processed all vertices in $\mathcal{A}$. In the latter case, the articulation-vertex branching rule unsuccessfully terminates and another branching rule is called. Thus, the running time (neglecting the initialization of $\mathcal{A}$) of the branching rule is $\mathcal{O}(k|\mathcal{A}|)$.

5.2. **Infeasibility Branching.** Consider now a vertex $u \in V$ for which only few assignments $x_{ui} = 1$ exist such that $F_b^1 \cup \{x_{ui}\}$ can be extended to a feasible solution of the C-Max-k-Cut problem. If u is chosen for branching and for each $i \in [k]$ with $x_{ui} \notin F_b^0$ a child node is generated by setting $x_{ui} = 1$, there is hope that many of these child nodes are infeasible, and thus can be pruned. Of course, using this *infeasibility branching rule* may produce an unbalanced branch-and-bound tree. But if infeasibility of a child node can be detected early, we can potentially rule out many branching decisions that would end up in an infeasible node of τ in a later stage if we had chosen another branching rule.

To find a candidate vertex u for branching, we select an unassigned vertex $u \in V$ with the smallest number of unassigned neighbors. To break ties, we choose u such that the remaining possibilities for assignments of u to a part is minimal. As another tie break we choose a vertex with maximal degree. Performing these steps requires to compute the number of remaining assignments to a part for each vertex, which is possible in $\mathcal{O}(k|V|)$ time. If we store these numbers, the remaining steps of the branching rule can be implemented to run in $\mathcal{O}(\Delta|V|)$ time, where Δ is the maximum degree of a vertex in G. This leads to an overall running time of $\mathcal{O}((k + \Delta)|V|)$.

5.3. **Objective Branching.** In this section, we assume that all objective coefficients w_{uv} are 1. The idea of the *objective branching rule* is to incorporate this objective into branching decisions. Since we aim to find a partition with the maximum number of edges between different parts, this branching rule selects an unassigned vertex u with as many already assigned neighbors as possible. The child nodes are generated via the different possibilities for u to be assigned to a

part. Since many neighbors of u were already assigned, assigning u to a part that was not used often by its neighbors hopefully increases the objective in this child node. However, using the same argumentation as for the infeasibility branching rule, shows that the objective branching rule may produce infeasible child nodes with increased probability for some nodes. The time to find a branching candidate u is in $\mathcal{O}((k+\Delta)|V|)$.

5.4. **Path Branching.** In contrast to the previous branching rules, the *path branching rule* does not change the bounds of a single variable, but adds a specific constraint to the subproblems of the generated child nodes. To find this inequality, the branching rule selects an index $i \in [k]$ such that the subgraph G_i of G induced by the vertices $u \in V$ with $x_{ui} \in F_b^1$ is disconnected. If the assignments of vertices to parts in b can be extended to a feasible solution of C-Max-k-Cut, there exists a path $P = (V_P, E_P)$ in G that connects two vertices in different connected components of G_i such that the vertices of P are unassigned or assigned to part i. To incorporate this observation into a branching decision, the path branching rule chooses such a path P and creates two child nodes of b by adding either of the inequalities

$$\sum_{u \in V_P'} x_{ui} \leq |V_P'| - 1 \quad \text{or} \quad \sum_{u \in V_P'} x_{ui} \geq |V_P'|,$$

where $V_P' \coloneqq \{v \in V_P : x_{vi} \notin F_b^1\}$. In particular, the inequality added in the latter case ensures that the number of connected components of G_i decreases by one and thus tries to enforce finding a partition with connected parts. Since the connected components of G_i and a path between two of these connected components can be found by two calls of a (modified) BFS algorithm, a branching decision can be made in $\mathcal{O}(k(|V| + |E|))$ time.

6. Primal Heuristics

In this section we present different primal heuristics for the C-Max-k-Cut problem. We start by describing a relaxation-based construction heuristic and an improvement heuristic in Section 6.1. Afterward, we present a root-node heuristic in Section 6.2 that is based on a spanning tree computation.

Algorithm 2 A relaxation-based rounding heuristic.

Input: A vector $\hat{x} = (\hat{x}_{ui})_{u \in V, i \in [k]} \in [0,1]^{k|V|}$.

Sort the vector $\hat{x} = (\hat{x}_{ui})_{u \in V, i \in [k]}$ in descending order, yielding the vector $(x_\ell)_{\ell \in [k|V|]}$ with $\nu(\ell) = u \in V$ and $\pi(\ell) = i \in [k]$.
Set $P_i = \emptyset$ for all $i \in [k]$, $\ell = 1$, and $M = \emptyset$.
while $\exists\, i \in [k]$ with $P_i = \emptyset$ **do**
    if $P_{\pi(\ell)} = \emptyset$ **then**
        Set $P_{\pi(\ell)} = \{\nu(\ell)\}$ and $M \leftarrow M \cup \{\nu(\ell)\}$.
    Set $\ell \leftarrow \ell + 1$.
while $M \neq V$ **do**
    for all $\ell = 1, \ldots, k|V|$ **do**
        if $\nu(\ell) \notin M$ **then**
            if $\nu(\ell) \in N(P_{\pi(\ell)})$ **then**
                Set $P_{\pi(\ell)} \leftarrow P_{\pi(\ell)} \cup \{\nu(\ell)\}$ and $M \leftarrow M \cup \{\nu(\ell)\}$.
return new partition $(P_i)_{i \in [k]}$.

6.1. **A Relaxation-Based Rounding Heuristic.** The relaxation-based rounding algorithm is formally stated in Algorithm 2. It is mainly taken from [21], where it

is used in a multilevel framework for electricity market splitting. However, it can be directly used for the more general problem considered in this paper. Let $\hat{x}_{ui} \in [0,1]$ for $u \in V$, $i \in [k]$, be part of a relaxation solution of Problem (1). We interpret these relaxation solutions as the probability that vertex u should be in part i. The idea is now as follows. First, we sort the vector $\hat{x} \in [0,1]^{k|V|}$ in descending order in Line 1. While the indicies u and i of the entries of the vector $\hat{x}$ clearly put every entry in relation to a node and a part, this is obviously not the case anymore after sorting. For still being able to relate an entry of the sorted vector to the corresponding node and part, we introduce the mappings ν and π to encode the information to which pair of vertex and part each entry of the sorted vector belongs. For instance, assume entry $\hat{x}_{ui}$ has index $\alpha \in \{1, \dots, k|V|\}$ after sorting. Then, $\nu(\alpha) = u$ and $\pi(\alpha) = i$ holds. Next, we assign to every part the vertex with the highest probability of being assigned to that part (first while-loop). The set M used in the algorithm collects all vertices that have already been associated to a certain part of the partition. Afterward, we again iterate over the sorted vector of relaxation solutions and assign every vertex that is not yet assigned to a part if this assignment does not violate connectivity (for-loop).

It is possible that in early iterations of Algorithm 2, a vertex cannot be assigned to a favorable part because it is not yet connected to any other vertex of that part. Thus, we subsequently apply local improvement steps in which we iteratively check for each vertex whether it should be assigned to a different part than to the one to which it is currently assigned to. The method is given in Algorithm 3. More formally, suppose a vertex u is assigned to part i and let us denote the corresponding objective function value by φ_{ui}. Consider now the situation in which node u is assigned to another part $j \neq i$ and everything else stays unchanged. Let φ_{uj} be the corresponding objective function value. If the objective function value φ_{ui} is smaller than φ_{uj} then u is moved from part i to j if this leaves all parts non-empty and if the resulting parts are still connected.

Algorithm 3 A 1-opt improvement heuristic.

Input: A connected graph partition $V = \cup_{i=1}^{k} P_i$.

1: **for all** $u \in V$ **do**
2: Let $i \in [k]$ be the part with $u \in P_i$.
3: **if** $\exists\, i \neq j \in [k]$ such that $\emptyset \neq P_i \setminus \{u\}$ and $P_j \cup \{u\}$ are connected **then**
4: Let φ_{ui} be the objective function value with $u \in P_i$.
5: Let φ_{uj} be the objective function value with $u \in P_j$.
6: **if** $\varphi_{uj} > \varphi_{ui}$ **then**
7: Set $P_i \to P_i \setminus \{u\}$, $P_j \to P_j \cup \{u\}$.
8: **return** new partition $(P_i)_{i \in [k]}$.

One crucial aspect of Algorithm 3 is the choice of the part j to which vertex u should be assigned. We implemented a brute-force strategy to search for possible candidate parts and found that this is appropriate in our numerical experiments. However, more involved strategies are given in the literature; see, e.g., [6].

6.2. **Spanning Tree Heuristic.** We now describe a greedy-based root-node heuristic, which uses a spanning tree of the graph to enforce the connectedness of the resulting parts. The idea is to initialize the components with the root node of the spanning tree and $k-1$ leaves and then decide for each branch where it is chopped, i.e., which part is added to the respective leaf component and which part is added to the root component.

Unfortunately, there are some cases that need to be considered; see Algorithm 4. Since the objective function is to find a maximum-weight cut, we start by computing

Algorithm 4 Spanning Tree Heuristic.

Input: A connected graph G with edge-weights w and the number of parts k.

1: Compute a minimum spanning tree T of G with starting/root node r and let L be the set of leaves of T.
2: Set $\bar{k} \leftarrow k$.
3: **if** $k-1 > |L|$ **then**
4: Set $j \leftarrow k$.
5: **while** $j > |L|$ **do**
6: Let $\ell \leftarrow$ be the largest leaf in T w.r.t. $w(\delta_G(\ell))$.
7: Set $P_j \leftarrow \{\ell\}$, $T \leftarrow T[V(T) \setminus \{\ell\}]$, $j \leftarrow j-1$, and recompute leaves L.
8: Set $\bar{k} \leftarrow |L|$.
9: Set $P_1 \leftarrow \{r\}$ and sort $\ell_i \in L$ decreasing w.r.t. $w(\delta_G(\ell_i))$.
10: **for all** $i = 1, \dots, \bar{k}$ **do** $P_{i+1} \leftarrow \{\ell_i\}$
11: **for all** $i = \bar{k}+1, \dots |L|$ **do**
12: $P_1 \leftarrow P_1 \cup \{v \in V : v$ lays on the (unique) r-ℓ_i-path in $T\}$
13: **for** $i = 2, \dots, \bar{k}$ **do**
14: Set $w_1 \leftarrow w(\delta_G(P_i))$ and $S_1 \leftarrow P_i$.
15: $p \leftarrow \{v \in V : v$ lays on the unique P_i-r-path in $T\}$ ▷ $|P_i| = 1$ and $p(1)$ should contain this node
16: Set $j \leftarrow 2$.
17: **while** $p(j) \neq \{r\}$ and $p(j) \notin P_1 \cup P_{\bar{k}+1} \cup \dots \cup P_L$ **do**
18: Set $S_j \leftarrow S_{j-1} \cup \{p(j)\}$, $w_j \leftarrow w(\delta_G(S_j))$, and $j \leftarrow j+1$.
19: Set $\hat{\jmath} \leftarrow \arg\max\{w_j\}$ and $P_i \leftarrow S_{\hat{\jmath}}$.
20: Set $P_1 \leftarrow P_1 \cup \{v \in V : v \notin \bigcup P_i\}$.
21: **return** new partition $(P_i)_{i \in [k]}$.

a minimum spanning tree T. Typically, the edges of the spanning tree tend to end up inside one part of the generated partition. One component is always initialized with the tree's root node r, but we have to distinguish between the number of leaves L of T compared to k. If we do not have enough leaves to initialize all remaining $k-1$ components (see Lines 3–7), we look for the leaf ℓ with the largest $w(\delta(\ell))$ and set the last uninitialized component to contain ℓ and only ℓ. Afterward, we remove ℓ from T and iterate this process until there are enough leaves in the recent spanning tree to cover the remaining components. Note that the components $P_{|L|+1}, \dots, P_k$ will not change in the second part of the algorithm due to the usage of $\bar{k}$.

Afterward—or if there are enough leaves to start with—every component is initialized with one leaf. If there are more leaves than components (see Line 11), the remaining leaves together with their complete branch are put in the root-component.

Then, we can start the actual heuristic in Line 13. For every remaining branch of the tree, we successively compute the weights for the vertex sets that are obtained by starting with the leaf and adding the respective next vertex on the unique path to the root (or the first node that was already assigned in a previous step). We choose the largest of such weights and set the component to contain the corresponding set. The remainder of the branch is (theoretically) added to the root-component. Since later leaves and their branches should be able to claim them, we do not mark these vertices directly, but wait until after the for-loop to put all remaining uncleared vertices in the root-component. Due to this construction, every component will be connected.

7. Symmetry Handling

Assume we are given a solution of C-Max-k-Cut, i.e., an assignment x_{vi} of the vertices $v \in V$ to the parts $i \in [k]$. Then, we can transform this solution into another solution with the same objective value by permuting the partitioning labels. That is, if $x \in \{0,1\}^{V\times[k]}$ is a solution and γ is a permutation of $[k]$, then $\bar{x}$ with $\bar{x}_{vi} = x_{v\gamma(i)}$ is a feasible solution with the same objective value. Similarly, if there exists an automorphism $\tilde{\gamma}\colon V \to V$ of G, an equivalent solution of C-Max-k-Cut is given by $\tilde{x}$ with $\tilde{x}_{vi} = x_{\tilde{\gamma}(v)i}$. In the following, we refer to the first kind of symmetries as *partitioning symmetries* and to the latter kind as *graph symmetries*.

Since computing symmetric solutions within branch-and-bound typically increases the solution time, we use two symmetry handling approaches that are discussed in the literature for handling partitioning and graph symmetries. The aim of both approaches is to cut off solutions that are not contained in a representative class of symmetric solutions—which hopefully decreases the solution time.

7.1. **Partitioning Symmetries.** In Formulation (1), an assignment of vertices to parts is given by a matrix $x \in \{0,1\}^{V\times[k]}$ with exactly one 1-entry per row. Thus, a partitioning symmetry, i.e., a relabeling of the parts, permutes the columns of x and keeps the number of 1-entries per row invariant. To handle such column symmetries, [32] introduced the concept of partitioning orbitopes. The *partitioning orbitope* $\mathrm{O}_{m,n}$ is the convex hull of all binary $(m \times n)$-matrices with one 1-entry per row whose columns are sorted in a lexicographically non-increasing way. In [32], a facet description of partitioning orbitopes has been developed and the authors proved that it can be separated in $\mathcal{O}(|V|k)$ time. Moreover, in [31] it is shown that partitioning orbitopes can be propagated in $\mathcal{O}(|V|k)$ time. In our experiments, we use an implementation of both the separation and propagation routine to handle partitioning symmetries.

7.2. **Graph Symmetries.** Let $\gamma\colon V \to V$ be an automorphism of G. By the above discussion, we can associate a permutation $\tilde{\gamma}\colon V \times [k] \to V \times [k]$, $\tilde{\gamma}(v,i) = (\gamma(v), i)$ with γ that reorders the rows of x according to γ. To handle graph symmetries $\tilde{\gamma}$, we use the concept of symresacks that was introduced in [25]. The symresack w.r.t. a permutation $\tilde{\gamma}$ contains all binary vectors that are lexicographically not smaller than their permutation w.r.t. $\tilde{\gamma}$. Thus, similarly to orbitopes, valid inequalities for symresacks can be used to cut off symmetric solutions. The authors of [25] show that there exists an IP formulation of $\mathrm{P}_{\tilde{\gamma}}$ with left-hand side coefficients in $\{0, \pm 1\}$ that can be separated in $\mathcal{O}(N\alpha(N))$ time, where α is the inverse Ackermann function. Moreover, a linear time propagation algorithm for $\mathrm{P}_{\tilde{\gamma}}$ is described in [25]. Both the separation and propagation algorithm for symresacks are used in our implementation to handle graph symmetries in C-Max-k-Cut.

Note that while different symmetry handling techniques cannot be used simultaneously in general, orbitopes and symresacks can be applied on the same instance because both are based on a lexicographic order. Thus, if we guarantee the same underlying variable order, no conflicts can arise.

8. Computational Study

In the Sections 3–7, we presented enhancements of a branch-and-cut framework for solving the C-Max-k-Cut problem. In this section, we now discuss the numerical results that we obtained using these enhancements. As we have mentioned throughout the paper, some of these enhancements are new and some of them can be found in the literature. Our goal in this section is to compare the novel techniques with a reference branch-and-cut algorithm that involves all the ingredients from the literature.

In order to obtain credible results, we tested the techniques on different test sets from the literature and on an additional test set of randomly generated instances. The entire test set and the general computational setup is described in Section 8.1. The large test set allows for a detailed numerical study that (i) evaluates the benefit of different (combinations of) techniques applied to different test sets and (ii) shows the improvement due to the novel techniques compared to the current state-of-the-art. Moreover, the broadness of the numerical study allows for drawing conclusions why a certain technique leads to computational enhancements for different instances or not. Both is carried out in Sections 8.2–8.4, where we analyze the results for $k \in \{2, 5, 10\}$ in order to also shed light on the computational benefit of the techniques for different values of k. In Section 8.5, we collect and discuss the general observations and insights. Finally, we use these insights to set up a problem-tailored parameterization of a branch-and-cut method for solving the C-MAX-k-CUT on realistic gas and power transport networks in Section 8.6.

8.1. **Test Sets and Computational Setup.** The test sets that we use in our computational study are the following:

- **Color02:** 51 of the smallest instances from the Color02 symposium on coloring problems [12]. The density of the considered instances ranges from 3.4 % to 89.6 % of possible edges; see Table 16. This test set contains the largest graphs (with 11 to 282 nodes; except for one outlier with 2368 nodes) and the graphs that vary the most w.r.t. their density.
- **Random:** 150 randomly generated instances with sizes in the range of 50 to 100 vertices and with densities that range from 4.7 % to 21.3 %; see Table 17. These instances were created with Mathematica [27] employing different edge probabilities.
- **Steiner-80:** 81 instances containing graphs with 80 vertices from the SteinLib I080 library [35, 47] of Steiner tree problems. Since the original test set contains 20 Steiner tree instances on the complete graph, which are identical for our problem, we removed 19 of these instances. The density of the considered instances varies between 3.8 % to 11.1 % with the exception of one complete graph; see Table 18.
- **Steiner-160:** 81 instances containing graphs with 160 vertices from the SteinLib I160 library [35, 47] of Steiner tree problems. Similar to Steiner-80, we removed 19 out of the 20 instances on the complete graph. The density of the considered instances varies between 1.9 % to 6.4 % with the exception of one complete graph; see Table 19. Both SteinLib test sets contain the graphs with the largest number of articulation vertices.

Recall from the definition of C-MAX-k-CUT that we require all graphs to be simple. Thus, we replace parallel edges in the above instances by a single edge in a preprocessing step. Detailed information about the (preprocessed) test sets are given in the tables in Appendix A.

We use SCIP 5.0.1 [18] with the LP solver CPLEX 12.7.1 [13] to solve the instances and to implement the techniques discussed in Sections 3–6. To handle symmetries as described in Section 7, we use the implementation of these methods that is already available in SCIP. Note, however, that SCIP itself is not able to detect symmetries automatically because of our additional plug-ins. Thus, we add partitioning orbitopes by hand. Moreover, we use Nauty [41] to detect graph automorphisms of the underlying graphs and add for each of the automorphism group's generators a symresack to the problem formulation. All graph algorithms (like computing connected components or articulation vertices) have been implemented using the Boost graph library [7]. All computations were run on a Linux cluster with Intel

TABLE 2. Summary of activated components in the reference branch-and-cut method.

cutting planes	heuristics	symmetry handling
clique cuts (Sect. 3.6)	rounding heuristic (Alg. 2)	part. symmetries
separator cuts (Ineq. (3))	1-opt heuristic (Alg. 3)	graph symmetries
art.-vertex cuts (Sec. 3.1)	spanning tree heuristic (Alg. 4)	

Xeon E5 3.5 GHz quad core processors and 32 GB memory. The code was executed using a single thread. The time limit of all computations is 1 h per instance.

Before we start analyzing the specific results, we briefly describe the reference branch-and-cut algorithm to which we compare the novel techniques in the following. We will see in this section that the novel techniques discussed in this paper significantly outperform the reference branch-and-cut algorithm. This is especially the case for larger values of k, where the reference method is almost never able to compute a global optimal solution. During our preliminary numerical tests it turned out that the situation is even more drastic for the general-purpose solver SCIP without any additional problem-specific components. Since a comparison of new techniques with a setting that is almost never able to compute solutions will not lead to useful insights, we thus decided to use SCIP extended by computational useful existing techniques as a reference branch-and-cut method.

To decide which of the cutting planes, heuristics, and symmetry handling methods from the literature discussed in the preceding sections are activated in the reference algorithm, we ran preliminary experiments. The experiments show that for both the flow and cut formulation it is favorable to handle graph and partitioning symmetries as well as to activate all discussed heuristics. Moreover, the results show that separating clique cuts and separator inequalities during branch-and-bound has a positive impact on the performance, whereas odd cycle and triangle cuts have an adverse effect. Since articulation-vertex cuts and leaf cuts are special cases of separator inequalities, we also evaluated the impact of these cutting planes. Our experiments show that the leaf cuts perform worse in comparison with separator inequalities, whereas the articulation-vertex cuts perform well. For this reason, the leaf cuts are deactivated in the reference method and the articulation-vertex cuts are activated. Table 2 provides a summary of the activated components in the reference branch-and-cut method.

Regarding the presented novel techniques, we did preliminary numerical experiments before we obtained the results that are discussed in the following sections. First, for the branching rules introduced in Section 5, it turned out that only the articulation-vertex branching rule yields improved results reliably. All other branching rules may lead to an improved method for some instances but also harm the solution process on other instances. Thus, we focus on the articulation-vertex branching rule (cf. Section 5.1) in what follows.

Next, we report on our experiments on the four test sets and for $k \in \{2, 5, 10\}$. All reported results on the number of nodes in the branch-and-bound trees ("#nodes") and running times ("time") are given in shifted geometric mean

$$\prod_{i=1}^{n} (t_i + s)^{1/n} - s$$

of all n instances within a test set, where we use a shift of $s = 10$ for running times and of $s = 100$ for nodes to reduce the impact of very easy instances. Moreover, we denote the total number of instances that can be solved by a setting within

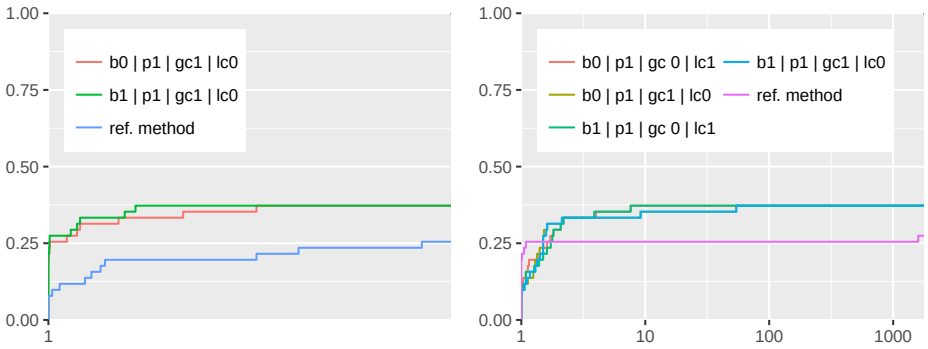

FIGURE 3. Selected combinations for the Color02 test set and $k = 2$ for the flow formulation (left) and the cut formulation (right).

the time limit by "#opt". The columns "branch.", "prop.", "glo. cut", and "loc. cut" encode whether the articulation-vertex branching rule, the propagation algorithm, the global bounded-edge cut, and the local bounded-edge cut, respectively, are used in a specific setting.

To gain further insight, we also evaluate the performance of selected parameterizations of the different settings using performance profiles with running times as the performance measure [15]. In these profiles we declare that one parameterization dominates another one if the corresponding curve is above the dominated one. Here, "b" denotes the branching rule, "p" the propagation, "gc" the global cut, and "lc" the local cut. If the corresponding method is used, this is denoted by "1", otherwise by "0".

8.2. **Discussion of the Numerical Results for $k = 2$.**

8.2.1. *The Color02 Test Set.* We start with the Color02 test set. As it can be seen in Table 16, there are only very few articulation vertices. This explains that turning on or off the articulation-vertex branching rule changes almost nothing in the results; see the Color02 block in Table 3. There is no difference in the number of solved instances and the running times as well as the number of required branch-and-bound nodes also do not differ significantly. This holds both for the flow and the cut formulation. However, the branching rule is not costly, cf. Section 5.1, and thus running times are not affected significantly. It can also be seen that the cut formulation solves, on average, more instances. Using the flow formulation, additionally applying the propagation algorithm 1 and the global bounded-edge cut (6)—without using the separation of the local bounded-edge cut (7) and irrespective of using the branching rule or not—leads to the most successful methods. Using the cut formulation, the most important components are the local and global bounded-edge cut. While the propagation algorithm 1 used alone deteriorates the performance of the cut formulation both in terms of solved instances and running time, applying a combined approach of using either variant of the bounded-edge cut and propagation leads to the best results. Comparing the global and local variant of the bounded-edge cut, it can be seen that the local one performs better if the global cut is not used, which also leads to the best results on this test set.

Choosing the most successful methods w.r.t. this criterion and again comparing them using a performance profile, we obtain the results given in Figure 3, where we also integrated the reference branch-and-cut method. First, we see that all combinations of novel techniques outperform the reference approach both for the flow and the cut formulation. Moreover, the figure emphasizes that for the cut

TABLE 3. Node counts, solution time, and number of solved instances for $k = 2$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#nodes	time	#opt	#nodes	time	#opt
Color02 (51):									
0	0	0	0	13 756.8	1323.0	13	19 270.5	1039.4	15
0	0	0	1	11 600.0	1080.3	15	16 581.9	883.8	18
0	0	1	0	10 007.0	937.7	17	14 636.1	852.2	16
0	0	1	1	13 148.8	1166.2	17	16 596.6	885.5	18
0	1	0	0	13 831.9	1270.9	14	20 159.5	1063.5	14
0	1	0	1	12 337.2	1156.6	15	14 481.3	796.4	20
0	1	1	0	10 088.1	921.1	19	15 288.2	821.1	20
0	1	1	1	11 523.2	1055.0	18	16 077.8	855.2	19
1	0	0	0	13 829.3	1329.8	13	19 382.9	1044.2	15
1	0	0	1	11 493.2	1077.4	15	16 990.4	895.8	18
1	0	1	0	9823.4	928.3	17	14 728.9	856.8	16
1	0	1	1	13 042.7	1161.2	17	17 021.0	898.2	18
1	1	0	0	13 802.4	1264.9	14	20 025.5	1062.3	14
1	1	0	1	12 431.5	1161.1	15	14 508.9	803.8	20
1	1	1	0	9975.4	918.9	19	15 185.5	820.6	20
1	1	1	1	11 476.0	1056.2	18	16 137.1	860.9	19
Random (150):									
0	0	0	0	10 972.6	356.3	123	11 976.6	224.1	125
0	0	0	1	9036.7	312.4	139	8622.6	166.8	141
0	0	1	0	6229.1	206.4	140	8004.4	149.5	142
0	0	1	1	9197.4	316.5	138	9763.8	186.5	141
0	1	0	0	10 939.8	354.9	123	11 902.6	223.3	125
0	1	0	1	9716.2	345.7	133	8234.9	159.6	142
0	1	1	0	6155.8	204.8	141	6934.2	134.9	144
0	1	1	1	9557.6	323.8	136	10 038.0	195.0	137
1	0	0	0	10 940.3	356.4	123	12 004.2	224.4	125
1	0	0	1	9350.2	315.8	140	8823.1	169.6	144
1	0	1	0	6737.8	222.1	141	6988.4	135.3	144
1	0	1	1	9688.0	334.8	135	8166.6	159.9	144
1	1	0	0	10 926.3	356.9	123	11 937.2	224.1	125
1	1	0	1	10 065.6	338.6	137	7704.3	150.5	144
1	1	1	0	6454.8	211.4	141	7404.5	143.4	144
1	1	1	1	8840.0	302.0	137	8110.9	158.8	144
Steiner-80 (81):									
0	0	0	0	53 779.6	678.8	41	84 846.7	509.8	41
0	0	0	1	10 941.1	157.4	75	8153.5	57.4	78
0	0	1	0	5702.9	83.7	77	10 947.2	72.4	76
0	0	1	1	11 631.1	169.6	73	10 392.8	74.9	74
0	1	0	0	53 847.1	677.6	41	86 434.9	511.9	41
0	1	0	1	9318.5	128.8	77	8244.4	61.6	77
0	1	1	0	4703.3	78.2	76	7202.6	52.2	76
0	1	1	1	7536.3	120.7	76	7879.9	62.5	75
1	0	0	0	53 772.2	678.8	41	88 377.4	509.1	41
1	0	0	1	6626.3	97.7	80	7309.1	53.9	77
1	0	1	0	5717.1	87.3	75	7657.7	60.4	76
1	0	1	1	5644.6	83.6	79	5468.5	46.8	77
1	1	0	0	53 669.5	676.4	41	88 915.2	511.4	41
1	1	0	1	6913.4	103.2	77	7004.8	53.6	77
1	1	1	0	6533.2	90.8	76	12 085.7	83.6	75
1	1	1	1	6009.5	86.9	77	5495.5	52.7	76
Steiner-160 (81):									
0	0	0	0	53 664.3	3509.8	1	110 746.4	3474.1	1
0	0	0	1	63 533.8	2827.7	9	58 722.0	1508.6	28
0	0	1	0	29 776.0	1180.6	29	80 023.3	1834.0	18
0	0	1	1	72 116.8	2834.3	10	61 562.2	1464.1	22
0	1	0	0	52 272.8	3514.5	1	110 871.4	3480.4	1
0	1	0	1	69 460.5	2874.5	10	62 782.8	1601.1	20
0	1	1	0	41 601.7	1480.3	26	79 336.4	1758.7	18
0	1	1	1	61 771.4	2406.3	13	47 681.5	1214.8	23
1	0	0	0	47 511.7	3509.9	1	107 665.7	3474.0	1
1	0	0	1	59 077.9	2915.9	9	68 776.9	1794.0	32
1	0	1	0	32 255.0	1168.3	28	76 146.2	1804.7	18
1	0	1	1	74 694.6	3151.9	6	57 551.8	1428.2	26
1	1	0	0	47 840.3	3514.5	1	108 959.2	3480.3	1
1	1	0	1	68 520.0	3188.7	11	55 029.2	1465.5	29
1	1	1	0	38 951.8	1471.7	26	71 930.6	1617.7	20
1	1	1	1	64 603.1	2664.2	9	54 469.5	1335.2	26

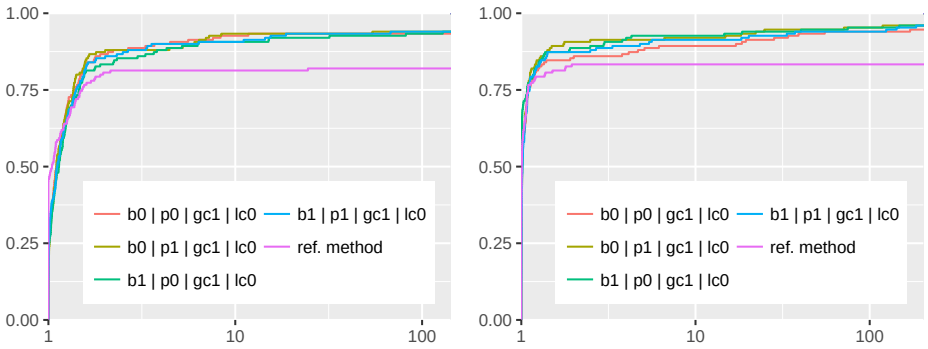

FIGURE 4. Selected combinations for the Random test set and $k = 2$ for the flow formulation (left) and the cut formulation (right).

formulation (right figure) using the propagation algorithm and the local bounded-edge cut leads to the best approach, whereas the articulation-vertex branching rule is not important for both formulations as explained above. After having chosen these parameterizations, the cut formulation clearly outperforms the flow formulation; see Table 3.

Finally, we see that the Color02 test set for $k = 2$ contains quite hard instances, since the best method obtained still only solves 20 instances out of the 51 Color02 instances.

8.2.2. *The Random Test Set.* First of all, the second block in Table 3 reveals that the cut formulation outperforms the flow formulation both in terms of the number of solved instances and running time. Thus, we focus on the cut formulation in the following. Using the articulation-vertex branching rule yields better results on average: The maximum number of solved instances without using the branching rule is 144, which is obtained for 1 (out of 8) parameterization, whereas the same number of solved instances (144) is obtained for 6 (again out of 8) parameterizations if the problem-specific branching rule is activated. Thus, in contrast to the Color02 test set, the articulation-vertex branching rule has a positive effect on the branch-and-bound mechanism. This behavior is comprehensible because the Random test set contains 21 (out of 150) instances that contain articulation vertices, whereas only 2 instances (out of 51) from the Color02 test set have articulation vertices. Thus, using the branching rule the problem decomposes into subproblems, which can be solved easily since the graphs are relatively sparse. Regarding running times, however, a difference between the variants using the branching rule or not is not obvious. Only considering the combinations using the branching rule, one can see that the most efficient methods use the global bounded-edge cut. The usage of propagation algorithm 1 has a slightly positive effect but does not influence the remaining parameterizations significantly. Hence, the most important choices are to use the cut formulation and to activate the articulation-vertex branching rule as well as the global bounded-edge cut. Again, all novel techniques dominate the reference approach and among the novel ones—compare Figure 4—it can also be seen that the cut formulation is performing better than the flow formulation. The statistical tool of performance profiles, i.e., of distribution functions, supports the conclusions that can be drawn from Table 3.

Finally, since the randomly generated instances are on average smaller and sparser than the Color02 instances, they are much easier to solve than the Color02 instances. Here, we solve 144 out of 150 instances in the best cases, which corresponds to 96.0 % in comparison to less than 40 % for the Color02 test set.

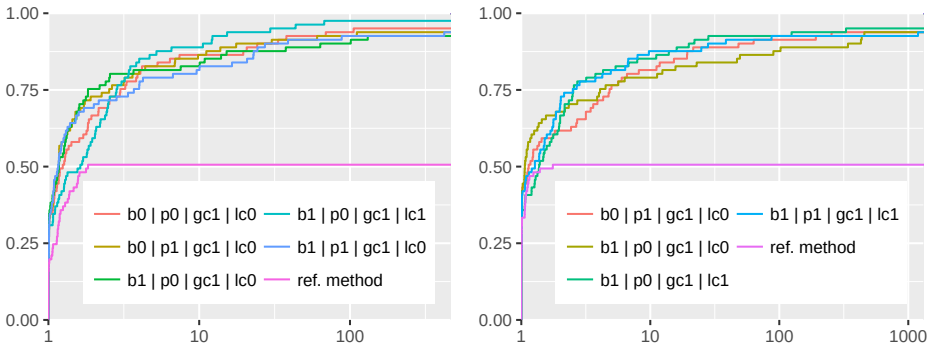

FIGURE 5. Selected combinations for the Steiner-80 test set and $k = 2$ for the flow formulation (left) and the cut formulation (right).

8.2.3. *The Steiner-80 Test Set.* Table 3 shows that, on average, using the articulation-branching rule again yields slightly better results both in terms of the maximum number of solved instances and in terms of running times. The larger impact can be seen for running times. Moreover, using the bounded-edge cuts (in its global or local variant) clearly speeds up the solution process and yields a larger number of solved instances. One might thus ask whether both the global and the local bounded-edge cut should be used or not. Interestingly, the answer strongly depends on whether the articulation-vertex branching rule is activated or not. If the branching rule is not used and the propagation is activated, it is better to use the global cut (6) and to deactivate its local variant (7). Deactivating the propagation routine and using the local bounded-edge cut, however, leads to the best result in terms of solved instances and almost the best performance w.r.t. running time for the cut formulation.

If the articulation-vertex branching rule is used, the global bounded-edge cut still has the greatest impact on performance. In contrast to the case of the deactivated branching rule, it seems better to use both the global and the local variant of the cut. A more general pattern additionally reveals that the local bounded-edge cut yields drastic improvements in those cases, in which the global variant is not used and leads on average to the best results in the cut formulation. Again, some of the best combinations for the flow and the cut formulations are compared in the performance profiles in Figure 5. One can see that all novel techniques drastically outperform the reference approach and that the results for the flow and the cut formulation are quite comparable. The above discussion suggests to use the global bounded-edge cut. Thus, we only included combinations in which this cut is used. For the flow formulation, the branching rule and both variants of the bounded-edge cut lead to the most reliable method (which is in line with the analysis above), whereas deactivating the local variant of the cut yields a faster method. The same applies for the cut formulation but with less clear differences between the different combinations of techniques. Moreover, Figure 5 shows that for the easy instances it is better to deactivate the separation of the local bounded-edge cut, whereas separating these inequalities improves the running time on the harder instances—a conclusion that cannot be drawn from the average numbers of Table 3 alone.

8.2.4. *The Steiner-160 Test Set.* We now turn to the Steiner-160 test set that contains larger graphs. It is obvious from Table 3 that omitting the global and local bounded-edge cut leads to very bad results: only 1 out of 81 instances is solved. Thus, this unfavorable result is also obtained with the reference method. For the flow formulation we observe that using the global or local cuts gives much better results than the reference method. However, using the global variant alone is clearly the

best choice regarding these cuts. An interesting pattern of Table 3 is that using the articulation-vertex branching rule does not reduce the number of solved instances in the cut formulation, whereas it almost always (except for one case) leads to less solved instances when used in combination with the flow formulation. For the flow formulation, the most important technique—both in terms of overall solved instances and running times—is the global bounded-edge cut. Additionally using the propagation algorithm then seems to be too costly, yielding longer running times and less solved instances (26 vs. 29 for deactivated branching rule and 26 vs. 28 for activated branching rule). Next, we discuss the results for cut formulation and focus on the combinations that use the articulation-vertex branching rule. Using only the local bounded-edge cuts, our implementation is able to solve the largest number of instances. In terms of running times, however, it is necessary to additionally activate the propagation algorithm and the global bounded-edge cut to achieve the best results on average. Unfortunately, the trade-off for the faster average running time is that some instances become unsolvable (26 vs. 32). Nevertheless, the propagation algorithm leads to a more successful method on average. One additional observation is that, when using the cut formulation, it is always preferable (both w.r.t. running times and the number of solved instances) to use the local bounded-edge cuts. When using the flow formulation, however, this is not the case.

8.3. **Discussion of the Numerical Results for $k = 5$.** We now discuss the numerical results for the case of $k = 5$. For this situation, our preliminary experiments revealed that, without the global bounded-edge cut, only very few instances can be solved within the time limit. The bounded-edge cut, however, significantly improves the solution process. Thus, we only discuss the cases in which this cut is used. An overview of all results is given in Table 4. In contrast to the results for $k = 2$, for larger k it turned out that it is very helpful to split the separate test sets further depending on the hardness of the contained instances for obtaining a better analysis of the results. This splitting is given in Table 5 for the Color02 test set. For the grouping of the instances of our test sets into difficulty classes, we use the following notation. The class $[\ell, u)$ contains all instances that need at least ℓ seconds and less than u seconds for every setting to be solved.

8.3.1. *The Color02 Test Set.* First of all, Table 5 shows that the usage of the local bounded-edge cuts hampers the solution process. This holds for all instances except for the very easy instances in class $[0, 100)$, for which the local bounded-edge cut leads to a speed-up between 30 % to 50 % in the flow formulation. It can also be seen that the cut formulation is faster than the flow formulation on the easy instances and that the flow formulation is superior on the medium and hard instances. Moreover, Table 4 shows that the number of solved instances is larger using the flow formulation whereas the cut formulation is usually faster. Finally, the propagation algorithm improves the solution process both w.r.t. the overall number of solved instances and the running times if the flow formulation is used. Using the cut formulation, however, this behavior can only be observed if the local bounded-edge cuts are not separated. The articulation-vertex branching rule has almost no impact because the graphs of the Color02 test set contain almost no articulation vertices. In Figure 6 we compare the most successful settings, i.e., both the cut and flow formulation with activated global bounded-edge cut and propagation. Since the reference method only solves a single instance for both formulations, we refrain from integrating it into the figure. Figure 6 confirms the above analysis: We solve more instances using the flow formulation. However, if the cut formulation solves an instance, it always solves it faster than the flow formulation.

TABLE 4. Node counts, solution time, and number of solved instances for $k = 5$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#nodes	time	#opt	#nodes	time	#opt
Color02 (51):									
0	0	0	0	12 307.5	3251.0	1	59 593.2	3217.3	1
0	0	1	0	7073.5	1389.2	15	16 846.4	1085.7	13
0	0	1	1	6246.7	1395.0	12	19 256.0	1157.5	13
0	1	1	0	6600.9	1298.6	17	14 415.8	951.6	15
0	1	1	1	5428.8	1303.3	13	21 167.7	1173.5	14
1	0	1	0	7090.9	1389.4	15	16 903.2	1089.0	13
1	0	1	1	6217.5	1394.6	12	19 150.0	1156.9	13
1	1	1	0	6634.9	1298.7	17	14 349.0	951.6	15
1	1	1	1	5429.1	1303.8	13	21 126.8	1174.0	14
Random (150):									
0	0	0	0	22 469.3	3600.0	0	108 817.0	3600.0	0
0	0	1	0	8992.0	390.9	138	18 300.4	317.5	101
0	0	1	1	39 885.6	1227.8	66	22 599.6	346.5	104
0	1	1	0	8772.4	387.7	139	16 946.8	310.8	101
0	1	1	1	33 928.0	1048.2	71	15 319.0	267.1	102
1	0	1	0	8852.2	385.4	138	19 695.4	335.4	99
1	0	1	1	41 157.2	1253.1	64	26 690.3	384.4	103
1	1	1	0	8263.0	375.5	139	18 607.5	319.0	100
1	1	1	1	32 798.9	1018.5	74	18 537.9	295.6	102
Steiner-80 (81):									
0	0	0	0	41 221.9	3531.1	1	244 553.3	3523.6	1
0	0	1	0	10 658.9	324.1	78	9873.6	117.4	68
0	0	1	1	51 458.3	772.5	53	9155.9	123.0	63
0	1	1	0	9222.6	279.5	79	7521.0	101.6	66
0	1	1	1	35 771.4	621.9	47	5877.2	92.0	69
1	0	1	0	9235.3	286.8	74	13 105.9	150.7	64
1	0	1	1	46 283.1	727.7	57	9904.6	131.3	62
1	1	1	0	8538.4	267.3	75	8972.6	123.9	62
1	1	1	1	37 354.6	677.6	47	8960.4	118.7	66
Steiner-160 (81):									
0	0	0	0	25 296.7	3600.0	0	69 120.5	3600.0	0
0	0	1	0	21 407.8	1704.4	33	55 550.8	1235.9	34
0	0	1	1	29 560.1	2078.3	19	59 292.0	1339.4	42
0	1	1	0	19 434.8	1567.2	35	37 142.1	815.9	39
0	1	1	1	35 033.8	2199.3	17	48 275.8	1028.4	42
1	0	1	0	20 622.5	1616.9	31	45 424.8	1069.5	37
1	0	1	1	41 125.9	2836.4	18	53 866.7	1240.1	38
1	1	1	0	18 509.4	1544.3	32	26 965.3	660.4	43
1	1	1	1	37 400.1	2599.2	16	27 568.3	697.2	46

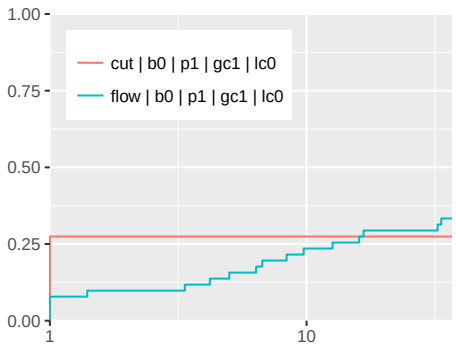

FIGURE 6. Selected combinations for the Color02 test set and $k = 5$.

8.3.2. *The Random Test Set.* For the Random test set, Table 6 shows that the cut formulation can be solved significantly faster than the flow formulation: The former has 26 instances in the class $[0, 100)$ compared to only 2 instances for the latter. Thus, the flow formulation clearly leads to longer running times. In contrast, the

TABLE 5. Instance counts, solution time, and number of solved instances for test set Color02 and $k = 5$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#inst.	time	#opt	#inst.	time	#opt
time range [0, 100):									
0	0	1	0	6	12.4	6	7	4.0	7
0	0	1	1	6	8.3	6	7	3.5	7
0	1	1	0	6	15.4	6	7	4.6	7
0	1	1	1	6	7.3	6	7	4.7	7
1	0	1	0	6	12.4	6	7	4.0	7
1	0	1	1	6	8.3	6	7	3.5	7
1	1	1	0	6	15.4	6	7	4.6	7
1	1	1	1	6	7.3	6	7	4.7	7
time range [100, 1000):									
0	0	1	0	1	390.9	1	0	—	—
0	0	1	1	1	466.6	1	0	—	—
0	1	1	0	1	449.4	1	0	—	—
0	1	1	1	1	560.4	1	0	—	—
1	0	1	0	1	391.4	1	0	—	—
1	0	1	1	1	466.6	1	0	—	—
1	1	1	0	1	449.1	1	0	—	—
1	1	1	1	1	559.7	1	0	—	—
time range [1000, 3600):									
0	0	1	0	37	3537.0	2	36	3600.0	0
0	0	1	1	37	3600.0	0	36	3600.0	0
0	1	1	0	37	3459.1	3	36	3600.0	0
0	1	1	1	37	3504.5	1	36	3600.0	0
1	0	1	0	37	3536.7	2	36	3600.0	0
1	0	1	1	37	3600.0	0	36	3600.0	0
1	1	1	0	37	3459.0	3	36	3600.0	0
1	1	1	1	37	3504.4	1	36	3600.0	0

TABLE 6. Instance counts, solution time, and number of solved instances for test set Random and $k = 5$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#inst.	time	#opt	#inst.	time	#opt
time range [0, 100):									
0	0	1	0	2	23.8	2	26	19.2	26
0	0	1	1	2	27.0	2	26	35.4	26
0	1	1	0	2	16.9	2	26	21.4	26
0	1	1	1	2	11.2	2	26	31.3	26
1	0	1	0	2	21.0	2	26	20.8	26
1	0	1	1	2	24.1	2	26	39.2	26
1	1	1	0	2	14.6	2	26	24.1	26
1	1	1	1	2	9.5	2	26	35.3	26
time range [100, 1000):									
0	0	1	0	9	236.9	9	1	270.5	1
0	0	1	1	9	366.6	9	1	395.4	1
0	1	1	0	9	249.4	9	1	353.1	1
0	1	1	1	9	292.5	9	1	129.1	1
1	0	1	0	9	237.2	9	1	269.3	1
1	0	1	1	9	366.6	9	1	397.6	1
1	1	1	0	9	249.2	9	1	352.3	1
1	1	1	1	9	292.3	9	1	128.7	1
time range [1000, 3600):									
0	0	1	0	28	2451.8	18	27	3401.8	4
0	0	1	1	28	3443.6	1	27	3581.7	2
0	1	1	0	28	2205.5	18	27	3447.5	1
0	1	1	1	28	3593.7	1	27	3600.0	0
1	0	1	0	28	2450.2	18	27	3401.4	4
1	0	1	1	28	3443.6	1	27	3580.6	2
1	1	1	0	28	2207.8	18	27	3445.5	1
1	1	1	1	28	3593.8	1	27	3600.0	0

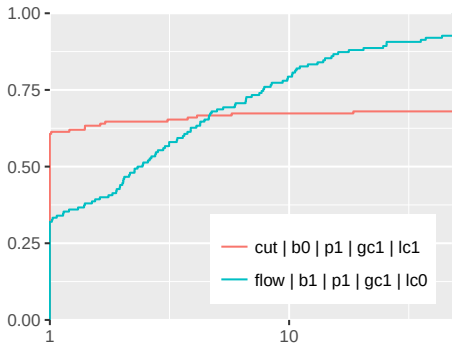

FIGURE 7. Selected combinations for the Random test set and $k = 5$.

class $[1000, 3600)$ of hard instances contains almost the same number of instances (28 vs. 27). Interestingly, in this class, the flow formulation solves significantly more instances if the separation of the local bounded-edge cuts is turned off, which is also the better choice for the hard instances solved with the cut formulation. Moreover, it can be seen that the propagation technique is advantageous for hard instances solved with the flow formulation if the local cuts are deactivated, whereas it leads to larger running times for the cut formulation—both for the case that the local bounded-edge cut is not used.

Regarding the articulation-vertex branching rule, it can be seen that the flow formulation clearly benefits from this rule when applied to the easy instances, whereas the cut formulation is slightly weakened by it. For the other instances, the branching rule has no significant influence on both formulations.

Table 4 reveals that, in the flow formulation, the local bounded-edge cut does not help either. In contrast, it improves the cut formulation if the propagation algorithm is also used.

We again compare the best settings (flow formulation with propagation, with branching rule, and without local cuts, as well as the cut formulation with propagation and local cuts but without the branching rule) in Figure 7. It is apparent that the flow formulation solves much more instances. Taking Table 6 into account, this also becomes clear because the flow formulation is much more successful especially applied to the hard instances. However, if the cut formulation solves an instance, it is typically faster as it was already the case for the Color02 test set.

8.3.3. *The Steiner-80 Test Set.* We now turn to the Steiner-80 test set. Table 7 shows that, again, there exist significantly less easy instances for the flow formulation compared to the cut formulation (1 vs. 26). Regarding the flow formulation, it is noticable that the local cuts and the propagation algorithm are disadvantageous on the medium instances in class $[100, 1000)$ and on the hard instances in class $[1000, 3600)$. The results for the cut formulation show that hard instances can only be solved if both the propagation algorithm and the local bounded-edge cuts are used. Table 4 shows that the articulation-vertex branching rule weakens the cut formulation both in terms of running time and solved instances. The same holds for the flow formulation and the local cuts, whereas the propagation algorithms improves the method, yielding slightly more solved instances.

As before, we compare the best settings (flow formulation with branching rule and propagation, and cut formulation with branching rule and local cuts) in Figure 8. The analysis is the same as for the two other test sets: The flow formulation solves more instances but the cut formulation is faster.

TABLE 7. Instance counts, solution time, and number of solved instances for test set Steiner-80 and $k = 5$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#inst.	time	#opt	#inst.	time	#opt
time range [0, 100):									
0	0	1	0	1	81.4	1	26	17.3	26
0	0	1	1	1	84.4	1	26	18.5	26
0	1	1	0	1	38.3	1	26	12.4	26
0	1	1	1	1	57.7	1	26	12.4	26
1	0	1	0	1	49.4	1	26	16.8	26
1	0	1	1	1	37.2	1	26	11.9	26
1	1	1	0	1	81.5	1	26	10.0	26
1	1	1	1	1	8.5	1	26	11.1	26
time range [100, 1000):									
0	0	1	0	5	254.1	5	0	—	—
0	0	1	1	5	464.0	5	0	—	—
0	1	1	0	5	288.6	5	0	—	—
0	1	1	1	5	379.0	5	0	—	—
1	0	1	0	5	254.2	5	0	—	—
1	0	1	1	5	469.8	5	0	—	—
1	1	1	0	5	288.3	5	0	—	—
1	1	1	1	5	378.6	5	0	—	—
time range [1000, 3600):									
0	0	1	0	7	1774.7	5	5	3600.0	0
0	0	1	1	7	2858.4	2	5	3600.0	0
0	1	1	0	7	2198.4	5	5	3600.0	0
0	1	1	1	7	3064.7	2	5	2412.2	3
1	0	1	0	7	1777.6	5	5	3600.0	0
1	0	1	1	7	2879.4	2	5	3600.0	0
1	1	1	0	7	2187.3	5	5	3600.0	0
1	1	1	1	7	3065.1	2	5	2406.7	3

TABLE 8. Instance counts, solution time, and number of solved instances for test set Steiner-160 and $k = 5$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#inst.	time	#opt	#inst.	time	#opt
time range [0, 100):									
0	0	1	0	0	—	—	0	—	—
0	0	1	1	0	—	—	0	—	—
0	1	1	0	0	—	—	0	—	—
0	1	1	1	0	—	—	0	—	—
1	0	1	0	0	—	—	0	—	—
1	0	1	1	0	—	—	0	—	—
1	1	1	0	0	—	—	0	—	—
1	1	1	1	0	—	—	0	—	—
time range [100, 1000):									
0	0	1	0	1	142.2	1	0	—	—
0	0	1	1	1	292.1	1	0	—	—
0	1	1	0	1	968.1	1	0	—	—
0	1	1	1	1	184.1	1	0	—	—
1	0	1	0	1	681.4	1	0	—	—
1	0	1	1	1	565.5	1	0	—	—
1	1	1	0	1	335.1	1	0	—	—
1	1	1	1	1	651.9	1	0	—	—
time range [1000, 3600):									
0	0	1	0	41	3494.2	1	32	3556.5	1
0	0	1	1	41	3392.7	2	32	3463.3	4
0	1	1	0	41	3505.5	3	32	3211.6	4
0	1	1	1	41	3600.0	0	32	3489.5	4
1	0	1	0	41	3438.0	3	32	3554.9	1
1	0	1	1	41	3437.9	3	32	3477.4	3
1	1	1	0	41	3526.1	3	32	3211.1	4
1	1	1	1	41	3500.9	1	32	3490.3	4

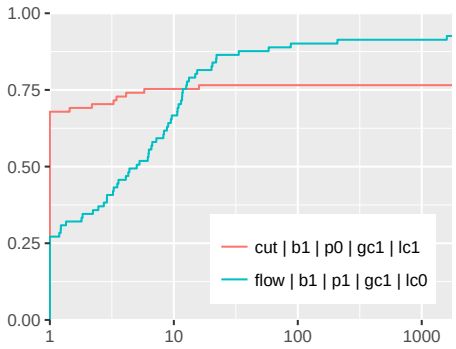

FIGURE 8. Selected combinations for the Steiner-80 test set and $k = 5$.

8.3.4. *The Steiner-160 Test Set.* Table 8 clearly shows that this is the hardest test set: Almost all instances are classified as hard in the class $[1000, 3600)$. For this set of instances, the cut formulation slightly outperforms the flow formulation. Since we can only solve very few instances for this test set, a reliable discussion of the impact of the different techniques is not possible. However, Table 4 shows that the usage of local cuts weakens the flow formulation—both in terms of the number of solved instances and running times. Additionally, the branching rule does not help solving the flow formulation. The propagation rule improves the solution process of the flow formulation on average but might also lead to slightly less solved instances. In contrast, the cut formulation benefits from the articulation-vertex branching rule and, even more clearly, from the propagation algorithm. Using local bounded-edge cuts weakens the cut formulations in terms of solution time. The number of instances, however, increases if the separation of the local cuts is enabled.

Regarding the best settings, the situation is not as clear as for the other test sets. For the flow formulation, the propagation should be used. Turning on the branching rule gives faster running times but leads to 3 less instances solved. For the cut formulation, one should also use the propagation as well as the branching rule but turn off (considering averages) the local cuts. However, local cuts lead to 3 more solved instances for the best combination.

In summary, the cut formulation is significantly faster and can solve, for the first time, more instances.

8.4. **Discussion of the Numerical Results for $k = 10$.** The main overview of all results is given in Table 9. Again, the solution process always benefits from activating the global bounded-edge cut, so we only discuss combinations that use this cut.

8.4.1. *The Color02 Test Set.* Table 10 shows that the number of easy and hard instances is comparable for the flow and cut formulation and that the cut formulation is significantly faster on the easy instances. For the hard instances, we only solve up to 2 instances, so that trends cannot be discussed reliably. In Table 9 one can additionally see that both the flow and the cut formulation are weakened by using the local bounded-edge cuts. Thus, we only discuss the cases further where the local cuts are not used. Here, the propagation algorithm is always beneficial and yields a speed-up of 6.1 % for the flow formulation and even 11.4 % for the cut formulation. As before, the articulation-vertex branching rule makes no difference since almost no articulation vertices exist in the Color02 test set.

In contrast to the case of $k = 5$, the cut formulation is now better both w.r.t. the number of solved instances (5 more than the flow variant) and the running times

TABLE 9. Node counts, solution time, and number of solved instances for $k = 10$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#nodes	time	#opt	#nodes	time	#opt
Color02 (51):									
0	0	0	0	4641.1	3206.3	1	31 916.4	3207.0	1
0	0	1	0	2652.5	1363.5	16	5687.9	788.2	18
0	0	1	1	2568.4	1666.2	11	7762.4	1018.8	18
0	1	1	0	2328.2	1281.9	16	4560.9	698.5	21
0	1	1	1	2195.1	1464.2	13	8084.2	1042.3	15
1	0	1	0	2652.5	1365.2	16	5655.4	788.1	18
1	0	1	1	2522.5	1666.2	11	7733.0	1019.0	18
1	1	1	0	2394.0	1282.2	16	4543.6	698.4	21
1	1	1	1	2117.3	1464.9	13	8127.7	1042.5	15
Random (150):									
0	0	0	0	6058.8	3600.0	0	104 162.1	3600.0	0
0	0	1	0	3916.7	776.0	131	7288.4	202.9	127
0	0	1	1	29 974.6	1551.6	71	13 539.5	331.1	123
0	1	1	0	3952.0	767.4	127	4287.8	145.4	134
0	1	1	1	28 627.2	1543.5	80	9309.9	259.1	131
1	0	1	0	4119.0	800.9	129	7937.9	213.2	129
1	0	1	1	33 028.2	1677.9	67	12 570.2	312.5	127
1	1	1	0	4075.3	788.5	123	4626.0	151.7	135
1	1	1	1	25 527.2	1485.6	81	8333.7	240.0	134
Steiner-80 (81):									
0	0	0	0	17 555.3	3600.0	0	210 020.8	3600.0	0
0	0	1	0	7784.2	627.7	75	2992.5	83.2	72
0	0	1	1	65 726.2	1390.9	49	5875.2	139.5	72
0	1	1	0	6159.6	530.8	77	2504.5	70.6	76
0	1	1	1	39 691.0	921.0	46	2955.3	84.8	75
1	0	1	0	5338.4	494.6	73	3982.9	102.9	71
1	0	1	1	41 644.8	1039.6	52	5536.8	130.7	71
1	1	1	0	5994.7	560.4	73	2699.6	79.3	75
1	1	1	1	30 696.2	796.4	51	3576.3	99.8	74
Steiner-160 (81):									
0	0	0	0	14 495.6	3600.0	0	48 710.2	3600.0	0
0	0	1	0	14 405.5	3322.2	5	14 489.4	1195.7	37
0	0	1	1	6183.3	3594.4	1	9686.0	1611.1	27
0	1	1	0	14 594.8	3276.0	10	14 543.8	1032.9	43
0	1	1	1	6609.0	3472.9	5	6616.8	1072.2	38
1	0	1	0	16 100.7	3302.9	12	15 899.6	1338.1	38
1	0	1	1	4470.6	3575.5	1	11 866.1	1934.8	24
1	1	1	0	14 296.9	2955.1	12	10 691.3	888.6	44
1	1	1	1	5129.1	3379.8	4	9346.1	1414.5	32

(almost twice as fast as the flow variant). Thus, the overall best setting is the cut formulation with activated propagation and global bounded-edge cut.

8.4.2. *The Random Test Set.* Table 11 reveals that the cut formulation is again superior: There are 20 easy instances whereas we only have 2 easy instances for the flow formulation. Conversely, the flow formulation has 41 hard instances, whereas there are only 3 hard instances for the cut variant. For the easy instances, it is clear that the local bounded-edge cuts hinder the solution process and for deactivated local cuts, the propagation slows down the solution process. Since all easy instances are solved, this means that using the propagation algorithm is not required. Interestingly, the cut formulation clearly benefits from using the local cuts on the hard instances. Since there are only 3 of them, this is, however, not a reliable trend. In addition, Table 9 shows that, on average, local cuts do not yield better methods. In the same table it can also be seen that propagation is beneficial, especially when applied to the cut formulation. The branching rule does not yield improved results. In summary, the cut formulation is the distinct winner and should be used with the propagation algorithm.

TABLE 10. Instance counts, solution time, and number of solved instances for test set Color02 and $k = 10$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#inst.	time	#opt	#inst.	time	#opt
time range [0, 100):									
0	0	1	0	7	20.2	7	9	4.8	9
0	0	1	1	7	15.5	7	9	5.3	9
0	1	1	0	7	16.7	7	9	6.5	9
0	1	1	1	7	20.0	7	9	7.5	9
1	0	1	0	7	20.2	7	9	4.8	9
1	0	1	1	7	15.5	7	9	5.3	9
1	1	1	0	7	16.7	7	9	6.5	9
1	1	1	1	7	19.8	7	9	7.5	9
time range [100, 1000):									
0	0	1	0	1	346.0	1	0	—	—
0	0	1	1	1	687.5	1	0	—	—
0	1	1	0	1	222.7	1	0	—	—
0	1	1	1	1	131.0	1	0	—	—
1	0	1	0	1	346.3	1	0	—	—
1	0	1	1	1	690.0	1	0	—	—
1	1	1	0	1	223.2	1	0	—	—
1	1	1	1	1	130.8	1	0	—	—
time range [1000, 3600):									
0	0	1	0	35	3483.9	2	32	3595.2	1
0	0	1	1	35	3600.0	0	32	3545.1	1
0	1	1	0	35	3494.4	1	32	3546.2	2
0	1	1	1	35	3600.0	0	32	3586.9	1
1	0	1	0	35	3489.9	2	32	3595.2	1
1	0	1	1	35	3600.0	0	32	3544.8	1
1	1	1	0	35	3494.5	1	32	3539.3	2
1	1	1	1	35	3600.0	0	32	3586.7	1

TABLE 11. Instance counts, solution time, and number of solved instances for test set Random and $k = 10$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#inst.	time	#opt	#inst.	time	#opt
time range [0, 100):									
0	0	1	0	2	15.8	2	20	33.0	20
0	0	1	1	2	31.3	2	20	41.4	20
0	1	1	0	2	29.3	2	20	32.1	20
0	1	1	1	2	25.2	2	20	40.4	20
1	0	1	0	2	14.5	2	20	31.8	20
1	0	1	1	2	21.2	2	20	38.9	20
1	1	1	0	2	14.8	2	20	31.6	20
1	1	1	1	2	37.9	2	20	39.7	20
time range [100, 1000):									
0	0	1	0	13	348.6	13	5	175.0	5
0	0	1	1	13	361.1	13	5	283.1	5
0	1	1	0	13	263.8	13	5	336.0	5
0	1	1	1	13	394.9	13	5	322.6	5
1	0	1	0	13	347.1	13	5	174.6	5
1	0	1	1	13	361.0	13	5	283.0	5
1	1	1	0	13	262.3	13	5	335.4	5
1	1	1	1	13	394.9	13	5	322.5	5
time range [1000, 3600):									
0	0	1	0	41	2320.1	27	3	3226.8	1
0	0	1	1	41	3259.9	6	3	2510.1	2
0	1	1	0	41	2501.3	19	3	3600.0	0
0	1	1	1	41	3239.5	8	3	3426.7	1
1	0	1	0	41	2323.7	27	3	3221.9	1
1	0	1	1	41	3259.7	6	3	2502.4	2
1	1	1	0	41	2501.9	19	3	3600.0	0
1	1	1	1	41	3244.5	8	3	3424.6	1

TABLE 12. Instance counts, solution time, and number of solved instances for test set Steiner-80 and $k = 10$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#inst.	time	#opt	#inst.	time	#opt
time range [0, 100):									
0	0	1	0	0	—	—	22	13.7	22
0	0	1	1	0	—	—	22	15.1	22
0	1	1	0	0	—	—	22	14.2	22
0	1	1	1	0	—	—	22	15.4	22
1	0	1	0	0	—	—	22	11.6	22
1	0	1	1	0	—	—	22	13.6	22
1	1	1	0	0	—	—	22	10.4	22
1	1	1	1	0	—	—	22	10.4	22
time range [100, 1000):									
0	0	1	0	1	151.7	1	1	178.2	1
0	0	1	1	1	110.0	1	1	180.1	1
0	1	1	0	1	239.7	1	1	436.1	1
0	1	1	1	1	246.5	1	1	242.6	1
1	0	1	0	1	151.9	1	1	178.9	1
1	0	1	1	1	110.1	1	1	179.4	1
1	1	1	0	1	240.2	1	1	436.4	1
1	1	1	1	1	263.9	1	1	243.7	1
time range [1000, 3600):									
0	0	1	0	8	1798.2	7	1	3600.0	0
0	0	1	1	8	3058.8	4	1	3600.0	0
0	1	1	0	8	2013.9	5	1	3600.0	0
0	1	1	1	8	2918.8	2	1	3600.0	0
1	0	1	0	8	2102.1	5	1	3600.0	0
1	0	1	1	8	3084.9	3	1	3600.0	0
1	1	1	0	8	2160.8	4	1	3600.0	0
1	1	1	1	8	2918.0	2	1	3600.0	0

TABLE 13. Instance counts, solution time, and number of solved instances for test set Steiner-160 and $k = 10$.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	#inst.	time	#opt	#inst.	time	#opt
time range [0, 100):									
0	0	1	0	0	—	—	1	28.4	1
0	0	1	1	0	—	—	1	28.3	1
0	1	1	0	0	—	—	1	29.1	1
0	1	1	1	0	—	—	1	29.0	1
1	0	1	0	0	—	—	1	18.3	1
1	0	1	1	0	—	—	1	18.5	1
1	1	1	0	0	—	—	1	15.8	1
1	1	1	1	0	—	—	1	16.0	1
time range [100, 1000):									
0	0	1	0	0	—	—	0	—	—
0	0	1	1	0	—	—	0	—	—
0	1	1	0	0	—	—	0	—	—
0	1	1	1	0	—	—	0	—	—
1	0	1	0	0	—	—	0	—	—
1	0	1	1	0	—	—	0	—	—
1	1	1	0	0	—	—	0	—	—
1	1	1	1	0	—	—	0	—	—
time range [1000, 3600):									
0	0	1	0	72	3456.5	3	30	3320.2	4
0	0	1	1	72	3593.7	1	30	3600.0	0
0	1	1	0	72	3477.0	5	30	3388.5	3
0	1	1	1	72	3591.4	1	30	3600.0	0
1	0	1	0	72	3409.2	8	30	3321.9	4
1	0	1	1	72	3600.0	0	30	3600.0	0
1	1	1	0	72	3415.0	7	30	3387.5	3
1	1	1	1	72	3569.9	1	30	3600.0	0

8.4.3. *The Steiner-80 and Steiner-160 Test Sets.* We now turn to the Steiner-80 test set. Again, the cut formulation is obviously easier to solve than the flow formulation; see Table 12 (22 vs. 0 easy instances). Since there is only one medium instance, no trends can be discussed. The flow formulation has 8 hard instances and the cut variant only 1. Table 9 additionally shows that the local cuts are unfavorable, but that the propagation and the branching rule help to improve the solution process on average for the flow formulation. The best combination is thus obtained by activating the propagation algorithm and the branching rule—although the latter is deactivated in the settings which solve the most instances. In the cut formulation only the propagation routine leads to improved results.

At last, we briefly discuss the results for the Steiner-160 instances. Table 9 states that the cut formulation is superior, which is also confirmed by Table 13. The clear winner is the cut formulation with activated articulation-vertex branching rule and activated propagation algorithm.

8.5. **General Observations.** In the following, we summarize the results that we obtained on the different test sets and draw conclusions from the overall results—independent of a specific test set. Based on our preceding analysis, we split our test instances into two classes for which we can observe a different behavior: dense graphs (Color02) and sparse graphs (Random, Steiner-80, Steiner-160).

8.5.1. *General Observations for Dense Graphs.* Independent of the value of k and the problem formulation, we observe that it is almost always preferable to deactivate the separation of local bounded-edge cuts. This behavior is expected because the local bounded-edge cut improves the global cut only if the removal of the nodes that are assigned to a part splits the underlying graph into several connected components; cf. Section 3.3. Consequently, it is unlikely that a local bounded-edge cut can be applied in a dense graph (in comparison to a sparse graph).

To evaluate the impact of the propagation algorithm, we compare in the following the parameterizations with deactivated branching rule and bounded-edge cut separation, as well as activated global bounded-edge cut, which are on average the best settings. For $k = 2$, the improvement caused by the propagation algorithm in the flow formulation is 1.8 %, whereas the cut formulation is improved by 3.6 %. If $k = 5$, the improvements are 6.5 % and 12.4 %, respectively, as well as 6.0 % and 11.4 %, respectively, for $k = 10$. Thus, the improvement for the cut formulation is about twice as large as for the flow formulation. A possible explanation for this behavior is that the cut formulation needs to separate inequalities to ensure connectivity of a solution, whereas connectivity in the flow formulation is modeled by flow constraints that do not need to be separated. For this reason, it is expected that the propagation algorithm is more powerful in the cut formulation than in the flow model. Furthermore, it is reasonable that the improvement is better for increasing values of k, since the propagation algorithm can only find variable reductions if removing one part (almost) disconnects the graph, which is more likely for a larger value of k.

The impact of the branching rule cannot be evaluated on our test set of dense instances, because the corresponding graphs contain almost no articulation vertices. Hence, it remains to discuss the impact of the problem formulation. Based on our findings above, we use the setting with activated propagation routine and global bounded-edge cut as well as disabled local bounded-edge cut and branching rule for our comparison. We observe that the cut formulation is on average 10.9 % faster than the flow formulation for $k = 2$. For $k = 5$ and $k = 10$, respectively, the speed-up even increases to 26.7 % and 45.5 %, respectively. Thus, in terms of running time, the cut formulation clearly dominates the flow formulation. In terms

of solved instances, however, the flow formulation is preferable since it performs better on the harder instances.

8.5.2. *General Observations for Sparse Graphs.* In contrast to dense graphs, there is a qualitative difference if the novel techniques are used in the flow or the cut formulation. For $k = 2$, the propagation algorithm has only a minor impact on the running time for small graphs (Random, Steiner-80). The larger instances (Steiner-160), however, clearly benefit from the propagation routine in the cut formulation, whereas the routine has a negative effect in the flow formulation. One explanation for this might be that the flow constraints enforce connectivity in sparse graphs much better than in dense graphs if k is small. In contrast to this, the cut formulation needs to separate inequalities to ensure connectivity, which increases the size of the LP relaxations. By propagating connectivity, however, fewer inequalities need to be added, keeping the LP relaxations small. If the number of parts k is increased, both the cut and flow formulation benefit on average from the propagation algorithm.

Similarly, using the local bounded-edge cut in combination with the global version has almost always a negative effect on the running time in the flow formulation. In the cut formulation, however, the situation is more complicated. For smaller values of k, the separation of the local cut should be activated. Although this might lead to larger running times on some instances, this increase is relatively small in comparison to the speed-up we gain on larger instances (e.g., Steiner-160 for $k = 2$). For $k = 10$, however, the separation routine has in general no positive impact.

Concerning the branching rule, we cannot observe a clear trend whether it is beneficial to activate it or not. By analyzing results on single instances, it seems that we can benefit by using this rule only if a graph does not contain too many articulation vertices. At first glance, this might be surprising. However, if a graph contains many articulation vertices, it is very likely that the default branching rule of SCIP frequently branches on articulation vertices on its own. Thus, it is not necessary to guide the branching process to use articulation vertices and we can benefit from the SCIP rules that take other parameters of a fractional solution into account to find a good branching decision. This conclusion is also supported by the results on hard instances reported in Tables 7, 8, 12, and 13: Since these instances are hard, we need many more nodes in the branch-and-bound tree and thus it is likely to use an articulation vertex in a branching decision. Consequently, there is almost no difference in the running time if the branching rule is enabled or not.

Finally, regarding the average running times, the cut formulation is almost always faster than the flow formulation. However, and analogously to the dense instances, the flow formulation is often able to solve more instances.

8.5.3. *Conclusions for Good Default Settings.* Based on the analysis of Section 8.5.1 and 8.5.2, we have added some switches to our code to enable or disable the novel techniques if it seems to be favorable:

- The separation of local bounded-edge cuts is disabled in the flow formulation. Moreover, we deactivate the separation routine in the cut formulation if either $k > 5$ or if the density of the graph exceeds 15 %.
- The articulation-vertex branching rule is deactivated in both formulations if the number of articulation vertices is larger than 25 % of all vertices in the graph.
- The propagation routine and global bounded-edge cut is enabled in both formulations.

Using these parameterized settings, we have conducted new experiments on the four test sets described above. Table 14 summarizes these experiments.

TABLE 14. Node counts, solution time, and number of solved instances for parameterized settings. We report on the reduction of nodes and time as well as on the improvement on solved instances in comparision to the default setting in brackets.

	flow formulation			cut formulation		
number of parts k	#nodes	time	#opt	#nodes	time	#opt
Color02 (51):						
2	9965.5 (-28%)	919.6 (-30%)	19 (+6)	15 581.0 (-19%)	839.5 (-19%)	19 (+4)
5	6660.6 (-46%)	1300.4 (-60%)	17 (+16)	16 469.0 (-72%)	1008.5 (-69%)	14 (+13)
10	2382.1 (-49%)	1280.7 (-60%)	16 (+15)	4539.0 (-86%)	700.0 (-78%)	21 (+20)
Random (150):						
2	6453.9 (-41%)	212.1 (-40%)	141 (+18)	8110.3 (-32%)	158.3 (-29%)	144 (+19)
5	8255.5 (-63%)	377.0 (-90%)	139 (+139)	20 591.5 (-81%)	324.2 (-91%)	102 (+102)
10	4067.4 (-33%)	790.7 (-78%)	123 (+123)	4624.5 (-96%)	151.9 (-96%)	135 (+135)
Steiner-80 (81):						
2	6538.8 (-88%)	90.9 (-87%)	76 (+35)	5494.5 (-94%)	52.7 (-90%)	76 (+35)
5	8012.9 (-81%)	255.9 (-93%)	76 (+75)	7709.1 (-97%)	103.1 (-97%)	66 (+65)
10	6045.0 (-66%)	563.2 (-84%)	73 (+73)	2650.1 (-99%)	78.5 (-98%)	75 (+75)
Steiner-160 (81):						
2	38 915.5 (-27%)	1473.4 (-58%)	26 (+25)	54 498.6 (-51%)	1336.6 (-62%)	26 (+25)
5	18 414.4 (-27%)	1545.2 (-57%)	32 (+32)	28 658.2 (-59%)	696.6 (-81%)	46 (+46)
10	14 254.4 (-2%)	2958.1 (-18%)	12 (+12)	10 675.5 (-78%)	891.0 (-75%)	44 (+44)

For the flow formulation, we can see that the parameterized settings perform very well in comparison to the standard settings described in the previous sections. Besides two outliers (Steiner-160 for $k = 2$ and Steiner-80 for $k = 10$), the average running time of the parameterized settings is comparable to or only slightly worse than the best of the standard settings. In particular, the parameterized setting achieves for all test sets in the case of $k = 5$ the best average running times and improves the best setting for Steiner-80 by 4.2 %.

In the cut formulation, the best results of the parameterized settings are achieved for $k = 10$, where it realizes the best results on Color02 and Steiner-160 as well as almost the best absolute running times for Random and Steiner-80. For smaller values of k, the running times of the parameterized settings are between 5.5 % and 9.1 % worse than the best setting on Color02 and Steiner-160. On the other test sets, the absolute running times are rather small, and thus, the deviation from the best settings is larger in percentage although the absolute deviation is in most cases quite small.

In summary, the parameterized settings reliably produce good results for the flow formulation over all test sets. In the cut formulation, the performance of the parameterized settings is on average worse than the best of the standard settings. The best results, however, are often achieved by an outlier and the best setting differs between the different test sets. If we take this into account and again compare the running times, the parameterized settings perform also well in the cut formulation over all test sets. Thus, the parameterization provides a resonable mechanism to efficiently solve C-Max-k-Cut problems. Finally, the parameterized settings significantly outperform the state-of-the-art techniques.

8.6. **An Application to Gas and Power Networks.** In this section we finally focus on solving C-Max-k-Cut problems on realistic gas and power networks. Typically, these instances are large in terms of the number of nodes, very sparse, and contain a significant number of articulation vertices. Since these parameters have already been taken into account in finding a parameterized setting in the previous section, there is hope that these settings also allow to efficiently solve C-Max-k-Cut for gas and power instances.

The gas instances (Gas) used in our experiments have been extracted from gas networks of the publicly available GasLib library [45]. This test set consists of 5 instances for which the number of nodes varies between 24 and 582. The density of the corresponding graphs lies between 0.4 % and 9.1 %. Between 45.9 % and 64.2 % of the vertices are articulation vertices. An overview of these numbers can be found in Table 20 in the appendix.

The test set of power instances (Power) consists of all power flow networks extracted from the matpower [39] software tool that contain at least 10 and at most 10 000 vertices. These graphs are also very sparse and, besides two outliers, the density is smaller than 9.5 %. Furthermore, all instances contain articulation vertices. Table 21 in the appendix contains a detailed overview of these characteristics for every instance.

In our experiments, we compare the performance of the parameterized setting against the standard settings that we have used in Sections 8.2–8.4. Besides the measures used in the preceding sections, Table 15 also gives the arithmetic mean of primal-dual gap (column "gap") of unsolved instances within a test set in order to be able to evaluate the quality of the best solutions found.

For the gas instances, we observe that the parameterized setting is the best setting in the cut formulation for $k \in \{2, 5\}$, both in terms of running time and average gap. For $k = 10$, the parameterization also performs very well leading to almost the smallest average running times and gaps. In the flow formulation, the

TABLE 15. Solution time, number of solved instances, and gaps for different settings.

setting				flow formulation			cut formulation		
branch.	prop.	glo. cut	loc. cut	time	#opt	gap	time	#opt	gap
Gas (5), $k = 2$:									
0	0	0	0	53.4	4	19.7493	41.7	4	12.9639
0	0	1	0	44.8	4	1.9000	35.2	4	2.2222
0	0	1	1	57.3	4	3.1429	35.6	4	2.2222
0	1	1	0	38.7	4	1.9000	28.3	5	0.0000
0	1	1	1	47.4	4	1.9000	18.4	5	0.0000
1	0	1	0	41.6	4	1.9000	32.3	4	1.9000
1	0	1	1	46.8	4	2.6250	33.7	4	1.9000
1	1	1	0	37.0	4	1.9000	28.2	4	1.9000
1	1	1	1	35.7	4	0.9000	28.0	4	1.7000
parameterized				38.7	4	1.9000	18.4	5	0.0000
Gas (5), $k = 5$:									
0	0	0	0	1122.2	2	26.2453	747.2	2	29.7037
0	0	1	0	156.1	3	1.0083	118.8	3	0.7772
0	0	1	1	168.4	3	0.7105	110.4	3	0.6750
0	1	1	0	187.8	3	0.5639	122.8	3	0.7772
0	1	1	1	123.0	3	0.7105	96.3	3	0.5639
1	0	1	0	180.4	3	0.7417	128.7	3	0.8534
1	0	1	1	141.9	3	0.6912	141.0	3	0.5921
1	1	1	0	193.9	3	0.7417	161.3	3	0.8534
1	1	1	1	165.4	3	0.7105	149.7	3	0.6750
parameterized				187.8	3	0.5639	96.3	3	0.5639
Gas (5), $k = 10$:									
0	0	0	0	3516.9	1	12.0504	2492.0	1	11.9268
0	0	1	0	127.2	3	0.4735	101.4	3	0.2778
0	0	1	1	115.5	3	0.4735	101.4	3	0.2566
0	1	1	0	130.3	3	0.4963	97.5	3	0.3758
0	1	1	1	117.9	3	0.4524	97.9	3	0.3634
1	0	1	0	109.5	3	0.3006	95.7	3	0.3634
1	0	1	1	104.0	3	0.3326	95.6	3	0.3634
1	1	1	0	106.8	3	0.4963	95.7	3	0.3226
1	1	1	1	131.1	3	0.3554	95.6	3	0.3634
parameterized				130.3	3	0.4963	97.5	3	0.3758
Power (32), $k = 2$:									
0	0	0	0	860.1	9	16.4981	790.5	9	15.6085
0	0	1	0	804.4	9	2.7553	837.6	9	2.4743
0	0	1	1	865.2	9	5.1486	755.9	9	3.6348
0	1	1	0	826.2	9	5.5105	807.0	9	9.4938
0	1	1	1	808.2	9	7.5266	745.3	9	11.6592
1	0	1	0	817.4	9	2.9089	833.5	9	2.4110
1	0	1	1	827.5	9	4.4413	754.3	9	2.6793
1	1	1	0	814.6	9	2.3519	797.2	9	2.1478
1	1	1	1	823.6	9	4.4391	749.8	9	2.6443
parameterized				815.0	9	4.9937	747.7	9	9.1254
Power (32), $k = 5$:									
0	0	0	0	3130.5	1	47.1513	2644.2	7	35.9140
0	0	1	0	1109.2	9	12.3866	910.6	9	7.5265
0	0	1	1	1025.3	9	12.3519	986.5	8	6.7955
0	1	1	0	1104.9	9	12.3500	753.4	9	7.5544
0	1	1	1	1015.5	8	11.8535	773.6	10	7.9966
1	0	1	0	1191.9	9	12.3555	1044.9	10	7.5600
1	0	1	1	935.4	8	11.9130	749.5	9	6.9409
1	1	1	0	848.2	9	12.3522	1208.6	8	6.9253
1	1	1	1	1028.2	8	11.9222	850.2	9	6.8653
parameterized				880.8	9	12.3573	874.0	9	7.0728
Power (32), $k = 10$:									
0	0	0	0	3080.5	1	26.5208	3032.3	1	18.1924
0	0	1	0	924.2	8	6.4205	733.2	9	4.4223
0	0	1	1	1031.9	8	6.7192	717.5	9	4.3890
0	1	1	0	786.8	10	7.0099	751.2	9	4.3980
0	1	1	1	908.2	8	6.9106	634.7	10	4.5802
1	0	1	0	844.0	9	6.6589	815.0	9	4.3949
1	0	1	1	903.0	8	6.7380	730.4	10	4.6221
1	1	1	0	828.8	10	6.9165	653.2	10	4.5938
1	1	1	1	854.1	9	7.0290	650.3	10	4.6165
parameterized				844.2	10	6.9594	655.7	10	4.6130

parameterization is also one of the best settings for $k = 2$. For larger k, however, the quality of the parameterized setting deteriorates and the running time is much slower than in the best setting. Moreover, we observe that the cut formulation clearly dominates the flow formulation in terms of running times. Thus, using the parameterized setting on the cut formulation reliably leads to the best performance in solving C-Max-k-Cut on gas instances.

On the power instances, the parameterized setting is only slightly slower than the best setting (at most 3.8 %) in both the cut and flow formulation. The only exceptions are the flow formulation for $k = 10$ and the cut formulation for $k = 5$, where the parameterization is by about 7.3 % and 16.6 % slower. Thus, again, our parameterization performs very well on average. Since the cut formulation leads on average to faster running times than the flow formulation, we conclude, as for the gas instances, that using the parameterized setting on the cut formulation is a good choice to solve C-Max-k-Cut problems on power instances.

9. Conclusion

In this paper we studied tailored branch-and-cut techniques for the connected Max-k-Cut problem. We reviewed existing mixed-integer programming techniques from the literature and showed in an extensive numerical study that these techniques do not yield an effective branch-and-cut algorithm for a large variety of test sets. Thus, we also developed novel techniques, which are shown to yield a much more successful method for solving hard instances. Finally, we showed that we can determine general-purpose combinations of the large set of techniques that yield very good results on the studied test sets and that can also solve large-scale and realistic instances of gas and power transport networks.

Acknowledgments

The last author thanks the DFG for their support within project A05 and B08 in SFB/TRR 154. This research has been performed as part of the Energie Campus Nürnberg and has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 764759. The third author thanks the DFG for their support within project A2 in SFB 666. Moreover, we would like to thank Frauke Liers for many fruitful discussions on the topic of this paper and Tristan Gally for useful hints concerning implementation issues. Finally, we thank Marc E. Pfetsch for helpful discussions and his agreement to use parts of the code developed in [28] as the basis for our code.

References

[1] T. Achterberg. "Constraint integer programming." PhD thesis. TU Berlin, 2007.

[2] T. Achterberg, T. Koch, and A. Martin. "Branching rules revisited." In: *Operations Research Letters* 33.1 (2005), pp. 42–54. DOI: 10.1016/j.orl.2004.04.002.

[3] M. Ambrosius, V. Grimm, T. Kleinert, F. Liers, M. Schmidt, and G. Zöttl. *Endogenous Price Zones and Investment Incentives in Electricity Markets: An Application of Multilevel Optimization with Graph Partitioning*. Tech. rep. Oct. 2018.

[4] M. F. Anjos, B. Ghaddar, L. Hupp, F. Liers, and A. Wiegele. "Solving k-Way Graph Partitioning Problems to Optimality: The Impact of Semidefinite Relaxations and the Bundle Method." In: *Facets of Combinatorial Optimization: Festschrift for Martin Grötschel*. Ed. by M. Jünger and G. Reinelt. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 355–386. DOI: 10.1007/978-3-642-38189-8_15.

[5] F. Barahona and A. R. Mahjoub. "On the cut polytope." In: *Mathematical Programming* 36.2 (1986), pp. 157–173. DOI: 10.1007/BF02592023.

[6] E. R. Barnes, A. Vannelli, and J. Q. Walker. "A New Heuristic for Partitioning the Nodes of a Graph." In: *SIAM Journal on Discrete Mathematics* 1.3 (1988), pp. 299–305. DOI: 10.1137/0401030.

[7] *Boost C++ Libraries*. http://www.boost.org. Accessed 2018-04-12.

[8] R. Carvajal, M. Constantino, M. Goycoolea, J. P. Vielma, and A. Weintraub. "Imposing Connectivity Constraints in Forest Planning." In: *Operations Research* 61.4 (2013), pp. 824–836. DOI: 10.1287/opre.2013.1183.

[9] B. Chaourar. "A linear time algorithm for a variant of the max cut problem in series parallel graphs." In: *Advances in Operations Research* 2017 (2017). DOI: 10.1155/2017/1267108.

[10] S. Chopra, B. Filipecki, K. Lee, M. Ryu, S. Shim, and M. Van Vyve. "An extended formulation of the convex recoloring problem on a tree." In: *Mathematical Programming* 165.2 (2017), pp. 529–548. DOI: 10.1007/s10107-016-1094-3.

[11] S. Chopra and M. R. Rao. "The partition problem." In: *Mathematical Programming* 59.1 (1993), pp. 87–115. DOI: 10.1007/BF01581239.

[12] *Color02 - computational symposium: Graph coloring and its generalizations. Available at*. http://mat.gsia.cmu.edu/COLOR02. 2002.

[13] *CPLEX Optimizer*. https://www.ibm.com/analytics/data-science/prescriptive-analytics/cplex-optimizer. Accessed 2018-04-12.

[14] B. Dilkina and C. P. Gomes. "Solving Connected Subgraph Problems in Wildlife Conservation." In: *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems: 7th International Conference, CPAIOR 2010, Bologna, Italy, June 14-18, 2010. Proceedings*. Ed. by A. Lodi, M. Milano, and P. Toth. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 102–116. DOI: 10.1007/978-3-642-13520-0_14.

[15] E. D. Dolan and J. J. Moré. "Benchmarking Optimization Software with Performance Profiles." In: *Mathematical Programming* 91 (2 2002), pp. 201–213. DOI: 10.1007/s101070100263.

[16] M. Fischetti, M. Leitner, I. Ljubić, M. Luipersbeck, M. Monaci, M. Resch, D. Salvagnin, and M. Sinnl. "Thinning out Steiner trees: a node-based model for uniform edge costs." In: *Mathematical Programming Computation* (2016), pp. 1–27. DOI: 10.1007/s12532-016-0111-0.

[17] A. Fügenschuh and M. Fügenschuh. "Integer linear programming models for topology optimization in sheet metal design." In: *Mathematical Methods of Operations Research* 68.2 (2008), pp. 313–331. DOI: 10.1007/s00186-008-0223-z.

[18] A. Gleixner, L. Eifler, T. Gally, G. Gamrath, P. Gemander, R. L. Gottwald, G. Hendel, C. Hojny, T. Koch, M. Miltenberger, B. Müller, M. E. Pfetsch, C. Puchert, D. Rehfeldt, F. Schlösser, F. Serrano, Y. Shinano, J. M. Viernickel, S. Vigerske, D. Weninger, J. T. Witt, and J. Witzig. *The SCIP Optimization Suite 5.0*. eng. Tech. rep. 17-61. Takustr.7, 14195 Berlin: ZIB, 2017.

[19] M. Goemans and D. P. Williamson. "Approximation algorithms for MAX-3-CUT and other problems via complex semidefinite programming." In: *Journal*

of Computer and System Sciences 68.2 (2004), pp. 442–470. DOI: 10.1016/j.jcss.2003.07.012.

[20] O. Goldschmidt and D. S. Hochbaum. "Polynomial algorithm for the k-cut problem." In: *[Proceedings 1988] 29th Annual Symposium on Foundations of Computer Science*. 1988, pp. 444–451. DOI: 10.1109/SFCS.1988.21960.

[21] V. Grimm, T. Kleinert, F. Liers, M. Schmidt, and G. Zöttl. "Optimal price zones of electricity markets: a mixed-integer multilevel model and global solution approaches." In: *Optimization Methods and Software* (2017). Online first. DOI: 10.1080/10556788.2017.1401069. Pre-published.

[22] V. Grimm, A. Martin, M. Schmidt, M. Weibelzahl, and G. Zöttl. "Transmission and Generation Investment in Electricity Markets: The Effects of Market Splitting and Network Fee Regimes." In: *European Journal of Operational Research* 254.2 (2016), pp. 493–509. DOI: 10.1016/j.ejor.2016.03.044.

[23] D. J. Haglin and S. M. Venkatesan. "Approximation and intractability results for the maximum cut problem and its variants." In: *IEEE Transactions on Computers* 40.1 (1991), pp. 110–113. DOI: 10.1109/12.67327.

[24] M. T. Hajiaghayi, G. Kortsarz, R. MacDavid, M. Purohit, and K. Sarpatwar. "Approximation Algorithms for Connected Maximum Cut and Related Problems." In: *Algorithms - ESA 2015: 23rd Annual European Symposium, Patras, Greece, September 14-16, 2015, Proceedings*. Ed. by N. Bansal and I. Finocchi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 693–704. DOI: 10.1007/978-3-662-48350-3_58.

[25] C. Hojny and M. E. Pfetsch. "Polytopes associated with symmetry handling." In: *Mathematical Programming* (2018). DOI: 10.1007/s10107-018-1239-7.

[26] J. Hopcroft and R. Tarjan. "Algorithm 447: Efficient Algorithms for Graph Manipulation." In: *Commun. ACM* 16.6 (1973), pp. 372–378. DOI: 10.1145/362248.362272.

[27] W. R. Inc. *Mathematica, Version 11.0*. Champaign, IL, 2016.

[28] T. Januschowski and M. E. Pfetsch. "Branch-Cut-and-Propagate for the Maximum k-Colorable Subgraph Problem with Symmetry." In: *CPAIOR*. Ed. by T. Achterberg and J. C. Beck. Vol. 6697. Lecture Notes in Computer Science. Springer, 2011, pp. 99–116. DOI: 10.1007/978-3-642-21311-3_11.

[29] M. Jünger, T. M. Liebling, D. Naddef, G. L. Nemhauser, W. R. Pulleyblank, G. Reinelt, G. Rinaldi, and L. A. Wolsey. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-art*. Springer Science & Business Media, 2009. DOI: 10.1007/978-3-540-68279-0.

[30] M. Jünger, G. Reinelt, and W. R. Pulleyblank. "On partitioning the edges of graphs into connected subgraphs." In: *Journal of Graph Theory* 9.4 (1985), pp. 539–549. DOI: 10.1002/jgt.3190090416.

[31] V. Kaibel, M. Peinhardt, and M. E. Pfetsch. "Orbitopal fixing." In: *Discrete Optimization* 8.4 (2011), pp. 595–610. DOI: 10.1016/j.disopt.2011.07.001.

[32] V. Kaibel and M. E. Pfetsch. "Packing and partitioning orbitopes." In: *Mathematical Programming* 114.1 (2008), pp. 1–36. DOI: 10.1007/s10107-006-0081-5.

[33] R. M. Karp. "Reducibility Among Combinatorial Problems." In: *Complexity of Computer Computations*. Ed. by R. E. Miller and J. W. Thatcher. The IBM Research Symposia Series. Plenum Press, New York, 1972, pp. 85–103.

[34] T. Kleinert and M. Schmidt. "Global Optimization of Multilevel Electricity Market Models Including Network Design and Graph Partitioning." In: *Discrete Optimization* (2019). DOI: 10.1016/j.disopt.2019.02.002. In press.

[35] T. Koch, A. Martin, and S. Voß. "SteinLib: An Updated Library on Steiner Tree Problems in Graphs." In: *Steiner Trees in Industry*. Ed. by X. Z. Cheng and

D.-Z. Du. Boston, MA: Springer US, 2001, pp. 285–325. DOI: 10.1007/978-1-4613-0255-1_9.

[36] J. Lee, V. Nagarajan, and X. Shen. "Max-cut under graph constraints." In: *Integer Programming and Combinatorial Optimization*. Ed. by Q. Louveaux and M. Skutella. Vol. 18. Lecture Notes in Computer Science. Springer. 2016, pp. 50–62. DOI: 10.1007/978-3-319-33461-5_5.

[37] F. Liers, A. Martin, and S. Pape. "Binary Steiner trees: Structural results and an exact solution approach." In: *Discrete Optimization* 21 (2016), pp. 85–117. DOI: 10.1016/j.disopt.2016.05.006.

[38] J. T. Linderoth and M. W. P. Savelsbergh. "A Computational Study of Search Strategies for Mixed Integer Programming." In: *INFORMS Journal on Computing* 11.2 (1999), pp. 173–187. DOI: 10.1287/ijoc.11.2.173.

[39] *MATPOWER: a MATLAB power system simulation package*. http://www.pserc.cornell.edu/matpower/. Accessed: 2017-12-20.

[40] S. T. McCormick. "Making Sparse Matrices Sparser: Computational Results." In: *Mathematical Programming* 49 (1990), pp. 91–111.

[41] B. D. McKay and A. Piperno. "Practical graph isomorphism, {II}." In: *Journal of Symbolic Computation* 60 (2014), pp. 94–112. DOI: http://dx.doi.org/10.1016/j.jsc.2013.09.003.

[42] A. Newman. "Complex Semidefinite Programming and Max-k-Cut." In: *1st Symposium on Simplicity in Algorithms (SOSA 2018)*. Ed. by R. Seidel. Vol. 61. OpenAccess Series in Informatics (OASIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018, 13:1–13:11. DOI: 10.4230/OASIcs.SOSA.2018.13.

[43] D. Rehfeldt and T. Koch. *Combining NP-Hard Reduction Techniques and Strong Heuristics in an Exact Algorithm for the Maximum-Weight Connected Subgraph Problem*. Tech. rep. 17-45. Takustr. 7, 14195 Berlin: ZIB, 2017.

[44] F. Ricca and B. Simeone. "Local search algorithms for political districting." In: *European Journal of Operational Research* 189.3 (2008), pp. 1409–1426. DOI: 10.1016/j.ejor.2006.08.065.

[45] M. Schmidt, D. Aßmann, R. Burlacu, J. Humpola, I. Joormann, N. Kanelakis, T. Koch, D. Oucherif, M. E. Pfetsch, L. Schewe, R. Schwarz, and M. Sirvent. "GasLib—A Library of Gas Network Instances." In: *Data* 2.4 (2017). DOI: 10.3390/data2040040.

[46] S. Soltan, M. Yannakakis, and G. Zussman. "Doubly Balanced Connected Graph Partitioning." In: *Proceedings of the Twenty-Eighth Annual ACM-SIAM Symposium on Discrete Algorithms*. 2016, pp. 1939–1950. DOI: 10.1137/1.9781611974782.126.

[47] *SteinLib*. http://steinlib.zib.de/testset.php. 2001.

[48] S. Takeshi. "A Model of Contiguity for Spatial Unit Allocation." In: *Geographical Analysis* 37.1 (2004), pp. 2–16. DOI: 10.1111/j.1538-4632.2005.00605.x.

[49] S. Vicente, V. Kolmogorov, and C. Rother. "Graph cut based image segmentation with connectivity priors." In: *Computer vision and pattern recognition, 2008. CVPR 2008. IEEE conference on*. IEEE. 2008, pp. 1–8. DOI: 10.1109/CVPR.2008.4587440.

[50] Y. Wang, A. Buchanan, and S. Butenko. "On imposing connectivity constraints in integer programs." In: *Mathematical Programming* 166.1 (2017), pp. 241–271. DOI: 10.1007/s10107-017-1117-8.

[51] J. C. Williams, C. S. ReVelle, and S. A. Levin. "Spatial attributes and reserve design models: A review." In: *Environmental Modeling & Assessment* 10.3 (2005), pp. 163–181. DOI: 10.1007/s10666-005-9007-5.

Appendix A. Detailed Information about the Instances

Table 16. Statistics for the instances of the Color02 test set.

instance	#vertices	#edges	density	#art. vertices	#triangles
1-FullIns_3	30	100	0.2299	0	22
1-FullIns_4	93	593	0.1386	0	119
1-FullIns_5	282	3247	0.082	0	570
1-Insertions_4	67	232	0.1049	0	0
1-Insertions_5	202	1227	0.0604	0	0
2-FullIns_3	52	201	0.1516	0	40
2-FullIns_4	212	1621	0.0725	0	216
2-Insertions_3	37	72	0.1081	0	0
2-Insertions_4	149	541	0.0491	0	0
3-FullIns_3	80	346	0.1095	0	73
3-Insertions_3	56	110	0.0714	0	0
3-Insertions_4	281	1046	0.0266	0	0
4-FullIns_3	114	541	0.084	0	126
4-Insertions_3	79	156	0.0506	0	0
5-FullIns_3	154	792	0.0672	0	204
anna	138	493	0.0522	11	942
david	87	406	0.1085	1	957
DSJC125.1	125	736	0.095	0	278
DSJC125.5	125	3891	0.5021	0	40 259
DSJC125.9	125	6961	0.8982	0	230 206
DSJC250.1	250	3218	0.1034	0	2942
DSJC250.5	250	15 668	0.5034	0	327 913
DSJC250.9	250	27 897	0.8963	0	1 852 358
games120	120	638	0.0894	0	899
miles1000	128	3216	0.3957	0	46 353
miles1500	128	5198	0.6395	0	130 306
miles500	128	1170	0.1439	0	6071
miles750	128	2113	0.26	0	19 623
mug100_1	100	166	0.0335	0	43
mug100_25	100	166	0.0335	0	43
mug88_1	88	146	0.0381	0	35
mug88_25	88	146	0.0381	0	37
myciel3	11	20	0.3636	0	0
myciel4	23	71	0.2806	0	0
myciel5	47	236	0.2183	0	0
myciel6	95	755	0.1691	0	0
myciel7	191	2360	0.1301	0	0
queen10_10	100	1470	0.297	0	5260
queen11_11	121	1980	0.2727	0	7700
queen12_12	144	2596	0.2521	0	10 884
queen13_13	169	3328	0.2344	0	14 976
queen14_14	196	4186	0.219	0	20 104
queen15_15	225	5180	0.2056	0	26 460
queen16_16	256	6320	0.1936	0	34 192
queen5_5	25	160	0.5333	0	320
queen6_6	36	290	0.4603	0	672
queen7_7	49	476	0.4048	0	1260
queen8_12	96	1368	0.3	0	4868
queen8_8	64	728	0.3611	0	2152
queen9_9	81	1056	0.3259	0	3456
wap01a	2368	110 871	0.0396	0	1 731 982

Table 17. Statistics for the instances of the Random test set.

instance	#vertices	#edges	density	#art. vertices	#triangles
n100p05c10	100	263	0.0531	4	23
n100p05c1	100	251	0.0507	3	25
n100p05c2	100	268	0.0541	3	28
n100p05c3	100	233	0.0471	5	18
n100p05c4	100	242	0.0489	5	18
n100p05c5	100	245	0.0495	1	17
n100p05c6	100	224	0.0453	7	17
n100p05c7	100	257	0.0519	4	20
n100p05c8	100	238	0.0481	2	12
n100p05c9	100	266	0.0537	0	18
n100p10c10	100	497	0.1004	0	161

continue next page

instance	#vertices	#edges	density	#art. vertices	#triangles
n100p10c1	100	477	0.0964	0	147
n100p10c2	100	488	0.0986	0	136
n100p10c3	100	514	0.1038	0	178
n100p10c4	100	530	0.1071	0	195
n100p10c5	100	501	0.1012	0	150
n100p10c6	100	483	0.0976	0	149
n100p10c7	100	497	0.1004	0	170
n100p10c8	100	508	0.1026	0	174
n100p10c9	100	522	0.1055	0	170
n100p15c10	100	691	0.1396	0	424
n100p15c1	100	747	0.1509	0	532
n100p15c2	100	751	0.1517	0	537
n100p15c3	100	717	0.1448	0	514
n100p15c4	100	794	0.1604	0	634
n100p15c5	100	758	0.1531	0	563
n100p15c6	100	745	0.1505	0	556
n100p15c7	100	730	0.1475	0	520
n100p15c8	100	708	0.143	0	517
n100p15c9	100	772	0.156	0	601
n100p20c10	100	1016	0.2053	0	1438
n100p20c1	100	988	0.1996	0	1255
n100p20c2	100	992	0.2004	0	1288
n100p20c3	100	998	0.2016	0	1262
n100p20c4	100	965	0.1949	0	1183
n100p20c5	100	981	0.1982	0	1298
n100p20c6	100	967	0.1954	0	1175
n100p20c7	100	985	0.199	0	1280
n100p20c8	100	1010	0.204	0	1349
n100p20c9	100	958	0.1935	0	1163
n50p10c5	50	142	0.1159	0	36
n60p05c4	60	107	0.0605	8	2
n60p20c10	60	374	0.2113	0	264
n60p20c1	60	336	0.1898	0	212
n60p20c2	60	393	0.222	0	363
n60p20c3	60	344	0.1944	0	250
n60p20c4	60	350	0.1977	0	273
n60p20c5	60	342	0.1932	0	221
n60p20c6	60	373	0.2107	0	326
n60p20c7	60	388	0.2192	0	372
n60p20c8	60	362	0.2045	0	277
n60p20c9	60	351	0.1983	0	265
n70p05c4	70	136	0.0563	8	10
n70p10c10	70	253	0.1048	0	65
n70p10c1	70	250	0.1035	0	61
n70p10c2	70	232	0.0961	1	55
n70p10c3	70	235	0.0973	0	37
n70p10c4	70	254	0.1052	0	55
n70p10c5	70	245	0.1014	0	61
n70p10c6	70	255	0.1056	0	51
n70p10c8	70	245	0.1014	0	79
n70p10c9	70	253	0.1048	0	51
n70p15c10	70	351	0.1453	0	165
n70p15c1	70	345	0.1429	0	162
n70p15c2	70	373	0.1545	0	223
n70p15c3	70	345	0.1429	0	159
n70p15c4	70	380	0.1573	0	212
n70p15c5	70	353	0.1462	0	169
n70p15c6	70	365	0.1511	0	179
n70p15c7	70	348	0.1441	0	162
n70p15c8	70	355	0.147	0	160
n70p15c9	70	365	0.1511	0	178
n70p20c10	70	467	0.1934	0	393
n70p20c1	70	501	0.2075	0	481
n70p20c2	70	482	0.1996	0	435
n70p20c3	70	457	0.1892	0	360
n70p20c4	70	471	0.195	0	414
n70p20c5	70	464	0.1921	0	371

continue next page

instance	#vertices	#edges	density	#art. vertices	#triangles
n70p20c6	70	495	0.205	0	447
n70p20c7	70	475	0.1967	0	401
n70p20c8	70	475	0.1967	0	436
n70p20c9	70	461	0.1909	0	368
n80p05c10	80	157	0.0497	11	9
n80p05c3	80	170	0.0538	2	12
n80p05c4	80	166	0.0525	5	10
n80p10c10	80	293	0.0927	0	66
n80p10c1	80	301	0.0953	0	61
n80p10c2	80	298	0.0943	0	87
n80p10c3	80	309	0.0978	0	75
n80p10c4	80	327	0.1035	0	87
n80p10c5	80	302	0.0956	0	79
n80p10c6	80	314	0.0994	1	83
n80p10c7	80	313	0.0991	0	69
n80p10c8	80	300	0.0949	0	64
n80p10c9	80	318	0.1006	0	86
n80p15c10	80	470	0.1487	0	250
n80p15c1	80	471	0.1491	0	270
n80p15c2	80	461	0.1459	0	264
n80p15c3	80	455	0.144	0	247
n80p15c4	80	476	0.1506	0	301
n80p15c5	80	486	0.1538	0	292
n80p15c6	80	470	0.1487	0	248
n80p15c7	80	466	0.1475	0	257
n80p15c8	80	433	0.137	0	197
n80p15c9	80	497	0.1573	0	301
n80p20c10	80	613	0.194	0	582
n80p20c1	80	654	0.207	0	730
n80p20c2	80	630	0.1994	0	652
n80p20c3	80	638	0.2019	0	659
n80p20c4	80	653	0.2066	0	714
n80p20c5	80	647	0.2047	0	739
n80p20c6	80	653	0.2066	0	751
n80p20c7	80	631	0.1997	0	629
n80p20c8	80	638	0.2019	0	661
n80p20c9	80	590	0.1867	0	512
n90p05c10	90	201	0.0502	6	10
n90p05c2	90	198	0.0494	4	12
n90p05c4	90	202	0.0504	5	11
n90p05c8	90	215	0.0537	8	18
n90p05c9	90	199	0.0497	4	17
n90p10c10	90	406	0.1014	0	144
n90p10c1	90	419	0.1046	0	126
n90p10c2	90	416	0.1039	0	119
n90p10c3	90	420	0.1049	0	113
n90p10c4	90	393	0.0981	0	114
n90p10c5	90	360	0.0899	0	60
n90p10c6	90	387	0.0966	0	86
n90p10c7	90	393	0.0981	0	111
n90p10c8	90	400	0.0999	0	144
n90p10c9	90	350	0.0874	0	98
n90p15c10	90	623	0.1556	0	456
n90p15c1	90	598	0.1493	0	390
n90p15c2	90	584	0.1458	0	346
n90p15c3	90	598	0.1493	0	424
n90p15c4	90	585	0.1461	0	342
n90p15c5	90	608	0.1518	0	380
n90p15c6	90	599	0.1496	0	377
n90p15c7	90	586	0.1463	0	344
n90p15c8	90	615	0.1536	0	393
n90p15c9	90	595	0.1486	0	412
n90p20c10	90	842	0.2102	0	1085
n90p20c1	90	804	0.2007	0	966
n90p20c2	90	799	0.1995	0	922
n90p20c3	90	794	0.1983	0	906
n90p20c4	90	796	0.1988	0	935

continue next page

instance	#vertices	#edges	density	#art. vertices	#triangles
n90p20c5	90	851	0.2125	0	1155
n90p20c6	90	799	0.1995	0	879
n90p20c7	90	796	0.1988	0	922
n90p20c8	90	830	0.2072	0	1024
n90p20c9	90	799	0.1995	0	965

TABLE 18. Statistics for the instances of the Steiner-80 test set.

instance	#vertices	#edges	density	#art. vertices	#triangles
i080-001	80	120	0.038	21	8
i080-002	80	120	0.038	16	4
i080-003	80	120	0.038	12	4
i080-004	80	120	0.038	16	2
i080-005	80	120	0.038	15	2
i080-011	80	350	0.1108	0	123
i080-012	80	350	0.1108	0	122
i080-013	80	350	0.1108	0	111
i080-014	80	350	0.1108	0	118
i080-015	80	350	0.1108	0	112
i080-021	80	3160	1.0	0	82 160
i080-031	80	160	0.0506	5	10
i080-032	80	160	0.0506	8	12
i080-033	80	160	0.0506	5	8
i080-034	80	160	0.0506	4	13
i080-035	80	160	0.0506	5	11
i080-041	80	632	0.2	0	684
i080-042	80	632	0.2	0	642
i080-043	80	632	0.2	0	648
i080-044	80	632	0.2	0	637
i080-045	80	632	0.2	0	653
i080-101	80	120	0.038	13	4
i080-102	80	120	0.038	12	5
i080-103	80	120	0.038	12	4
i080-104	80	120	0.038	16	2
i080-105	80	120	0.038	13	5
i080-111	80	350	0.1108	0	110
i080-112	80	350	0.1108	0	107
i080-113	80	350	0.1108	0	122
i080-114	80	350	0.1108	0	109
i080-115	80	350	0.1108	0	117
i080-131	80	160	0.0506	3	12
i080-132	80	160	0.0506	6	6
i080-133	80	160	0.0506	2	6
i080-134	80	160	0.0506	5	5
i080-135	80	160	0.0506	5	10
i080-141	80	632	0.2	0	629
i080-142	80	632	0.2	0	651
i080-143	80	632	0.2	0	706
i080-144	80	632	0.2	0	644
i080-145	80	632	0.2	0	641
i080-201	80	120	0.038	16	0
i080-202	80	120	0.038	17	0
i080-203	80	120	0.038	15	6
i080-204	80	120	0.038	18	1
i080-205	80	120	0.038	13	1
i080-211	80	350	0.1108	0	100
i080-212	80	350	0.1108	0	111
i080-213	80	350	0.1108	0	101
i080-214	80	350	0.1108	0	104
i080-215	80	350	0.1108	0	99
i080-231	80	160	0.0506	3	11
i080-232	80	160	0.0506	3	10
i080-233	80	160	0.0506	6	11
i080-234	80	160	0.0506	4	17
i080-235	80	160	0.0506	6	11
i080-241	80	632	0.2	0	647
i080-242	80	632	0.2	0	596

continue next page

instance	#vertices	#edges	density	#art. vertices	#triangles
i080-243	80	632	0.2	0	658
i080-244	80	632	0.2	0	616
i080-245	80	632	0.2	0	663
i080-301	80	120	0.038	14	3
i080-302	80	120	0.038	12	2
i080-303	80	120	0.038	17	4
i080-304	80	120	0.038	12	3
i080-305	80	120	0.038	16	4
i080-311	80	350	0.1108	0	103
i080-312	80	350	0.1108	0	109
i080-313	80	350	0.1108	0	107
i080-314	80	350	0.1108	0	95
i080-315	80	350	0.1108	0	113
i080-331	80	160	0.0506	4	6
i080-332	80	160	0.0506	7	8
i080-333	80	160	0.0506	7	12
i080-334	80	160	0.0506	5	12
i080-335	80	160	0.0506	3	7
i080-341	80	632	0.2	0	641
i080-342	80	632	0.2	0	676
i080-343	80	632	0.2	0	656
i080-344	80	632	0.2	0	652
i080-345	80	632	0.2	0	665

TABLE 19. Statistics for the instances of the Steiner-160 test set.

instance	#vertices	#edges	density	#art. vertices	#triangles
i160-001	160	240	0.0189	28	4
i160-002	160	240	0.0189	26	0
i160-003	160	240	0.0189	34	2
i160-004	160	240	0.0189	26	3
i160-005	160	240	0.0189	26	7
i160-011	160	812	0.0638	0	174
i160-012	160	812	0.0638	0	183
i160-013	160	812	0.0638	0	148
i160-014	160	812	0.0638	0	180
i160-015	160	812	0.0638	0	156
i160-021	160	12 720	1.0	0	669 920
i160-031	160	320	0.0252	10	11
i160-032	160	320	0.0252	12	5
i160-033	160	320	0.0252	10	4
i160-034	160	320	0.0252	11	14
i160-035	160	320	0.0252	7	4
i160-041	160	2544	0.2	0	5130
i160-042	160	2544	0.2	0	5481
i160-043	160	2544	0.2	0	5347
i160-044	160	2544	0.2	0	5404
i160-045	160	2544	0.2	0	5331
i160-101	160	240	0.0189	31	5
i160-102	160	240	0.0189	30	7
i160-103	160	240	0.0189	35	3
i160-104	160	240	0.0189	29	5
i160-105	160	240	0.0189	27	3
i160-111	160	812	0.0638	0	181
i160-112	160	812	0.0638	0	160
i160-113	160	812	0.0638	0	177
i160-114	160	812	0.0638	0	180
i160-115	160	812	0.0638	0	155
i160-131	160	320	0.0252	11	12
i160-132	160	320	0.0252	12	9
i160-133	160	320	0.0252	13	9
i160-134	160	320	0.0252	15	10
i160-135	160	320	0.0252	10	6
i160-141	160	2544	0.2	0	5377
i160-142	160	2544	0.2	0	5424
i160-143	160	2544	0.2	0	5291
i160-144	160	2544	0.2	0	5420
i160-145	160	2544	0.2	0	5406
i160-201	160	240	0.0189	29	0

continue next page

instance	#vertices	#edges	density	#art. vertices	#triangles
i160-202	160	240	0.0189	28	5
i160-203	160	240	0.0189	30	0
i160-204	160	240	0.0189	30	3
i160-205	160	240	0.0189	29	6
i160-211	160	812	0.0638	0	159
i160-212	160	812	0.0638	0	167
i160-213	160	812	0.0638	0	178
i160-214	160	812	0.0638	0	170
i160-215	160	812	0.0638	0	151
i160-231	160	320	0.0252	12	12
i160-232	160	320	0.0252	6	6
i160-233	160	320	0.0252	6	7
i160-234	160	320	0.0252	12	12
i160-235	160	320	0.0252	8	14
i160-241	160	2544	0.2	0	5290
i160-242	160	2544	0.2	0	5233
i160-243	160	2544	0.2	0	5397
i160-244	160	2544	0.2	0	5405
i160-245	160	2544	0.2	0	5271
i160-301	160	240	0.0189	26	4
i160-302	160	240	0.0189	26	2
i160-303	160	240	0.0189	31	2
i160-304	160	240	0.0189	28	4
i160-305	160	240	0.0189	29	1
i160-311	160	812	0.0638	0	178
i160-312	160	812	0.0638	0	171
i160-313	160	812	0.0638	0	162
i160-314	160	812	0.0638	1	174
i160-315	160	812	0.0638	0	158
i160-331	160	320	0.0252	10	14
i160-332	160	320	0.0252	11	8
i160-333	160	320	0.0252	12	6
i160-334	160	320	0.0252	15	6
i160-335	160	320	0.0252	15	9
i160-341	160	2544	0.2	0	5240
i160-342	160	2544	0.2	0	5343
i160-343	160	2544	0.2	0	5422
i160-344	160	2544	0.2	0	5311
i160-345	160	2544	0.2	0	5265

TABLE 20. Statistics for the instances of the Gas testset.

instance	#vertices	#edges	density	#art. vertices
GasLib_24	24	25	0.0906	15
GasLib_134	134	133	0.0149	86
GasLib_135	135	157	0.0174	41
GasLib_40	40	45	0.0577	19
GasLib_582-v2	582	609	0.0036	267

TABLE 21. Statistics for the instances of the Power test set.

instance	#vertices	#edges	density	#art. vertices
case118.power	118	179	0.0259	9
case1354pegase.power	1354	1710	0.0019	382
case145.power	145	422	0.0404	12
case14.power	14	20	0.2198	1
case1888rte.power	1888	2308	0.0013	640
case1951rte.power	1951	2375	0.0012	626
case2383wp.power	2383	2886	0.001	528
case24_ieee_rts.power	24	34	0.1232	1
case2736sp.power	2736	3495	0.0009	308
case2737sop.power	2737	3497	0.0009	308
case2746wop.power	2746	3505	0.0009	320
case2746wp.power	2746	3505	0.0009	319
case2848rte.power	2848	3442	0.0008	946
case2868rte.power	2868	3471	0.0008	916
case2869pegase.power	2869	3968	0.001	556
case300.power	300	409	0.0091	68
case3012wp.power	3012	3566	0.0008	610
case30.power	30	41	0.0943	4

continue next page

instance	#vertices	#edges	density	#art. vertices
case30pwl.power	30	41	0.0943	4
case30Q.power	30	41	0.0943	4
case3120sp.power	3120	3684	0.0008	626
case33bw.power	33	37	0.0701	1
case39.power	39	46	0.0621	11
case57.power	57	78	0.0489	1
case6468rte.power	6468	8065	0.0004	1819
case6470rte.power	6470	8066	0.0004	1810
case6495rte.power	6495	8084	0.0004	1814
case6515rte.power	6515	8104	0.0004	1822
case89pegase.power	89	206	0.0526	12
case9241pegase.power	9241	14207	0.0003	1414
case_ieee30.power	30	41	0.0943	4
case_illinois200.power	200	245	0.0123	33

[1]Christopher Hojny, Hendrik Lüthen, Technische Universität Darmstadt, Discrete Optimization, Dolivostr. 15, 64293 Darmstadt, Germany; [2]Imke Joormann, Technische Universität Braunschweig, Institute for Mathematical Optimization, Universitätsplatz 2, 38106 Braunschweig, Germany; [3]Martin Schmidt, Trier University, Department of Mathematics, Universitätsring 15, 54296 Trier, Germany

E-mail address: hojny@mathematik.tu-darmstadt.de, i.joormann@tu-braunschweig.de, luethen@mathematik.tu-darmstadt.de, martin.schmidt@uni-trier.de