A Separation Heuristic for 2-Partition Inequalities for the Clique Partitioning Problem

Michael M. Sørensen[*]

July 2020

Abstract

We consider the class of 2-partition inequalities for the clique partitioning problem associated with complete graphs. We propose a heuristic separation algorithm for the inequalities and evaluate its usefulness in a cutting-plane algorithm. Our computational experiments fall into two parts. In the first part, we compare the LP objective values obtained by the proposed separator with those obtained by known separators for 2-partition inequalities and those obtained by exact polynomial-time separators for other classes of inequalities. These results demonstrate the importance of the 2-partition inequalities and the new separator. In the second part of the computational experiments, we apply the separator to different kinds of test instances from the literature. These results show that considerable improvements can be obtained in terms of narrowing the gaps between LP objective values and optimal partition values.

Key Words: Computation; Cutting-plane algorithm; Graph partitioning; Multicut; Polyhedral combinatorics; Separation.

1 Introduction

The clique partitioning problem is to determine a partition of an edge-weighted complete graph into node-disjoint complete subgraphs (cliques) in such a way that the sum of the weights of all edges within the cliques is maximal (or minimal). This problem is NP-hard in the strong sense [32], and we work on a polyhedral approach to solve the problem. For this approach to work, we need separation algorithms, or *separators* for short, that can be used in a cutting-plane algorithm to generate strong, valid, preferably facet-defining, inequalities for the associated polytope.

Grötschel and Wakabayashi [14] introduce the 2-partition inequalities and several other classes of valid inequalities and establish the facet-defining

[*]Department of Economics and Business Economics, Aarhus University, Fuglesangs Allé 4, DK-8210 Aarhus V, Denmark. E-mail: mim@econ.au.dk

properties of these inequalities for the clique partitioning polytope. In [13], the same authors present a cutting-plane algorithm for the clique partitioning problem that uses a heuristic separator for the 2-partition inequalities. In subsequent works by other researchers where 2-partition inequalities are used, almost all the applied separators are either identical to or consist of minor modifications to parts of the one in [13].

In this paper, we introduce a new fast separation heuristic for 2-partition inequalities and evaluate its usefulness in a linear programming (LP) based cutting-plane algorithm. We present the results of several computational experiments. The main performance indicator will be the objective values of the LPs, which give upper bounds on the values of integer optimal solutions. In the following, we will refer to these objective values as the *z-values*, and they will also be evaluated in terms of computation times.

In one experiment, we compare the performance of the new separator to those of existing separators in the literature. Another experiment demonstrates the importance of the 2-partition inequalities by comparing the new separator to exact polynomial-time separators for other classes of facet-defining inequalities. The last group of experiments considers the z-value improvements that can be expected in practical computations. We have imbedded the cutting-plane algorithm (CPA) in a prospective branch-and-cut algorithm for the clique partitioning problem, but here we will focus on results that can be obtained from the CPA.

A rough outline of the paper is as follows. Section 2 considers the clique partitioning polytope and some valid and facet-defining inequalities. Previous work on separation is described in Section 3, and Section 4 presents the new proposed separator. Section 5 describes some details of the cutting-plane algorithm, and Section 6 presents the computational experiments. Finally, we make some concluding remarks.

2 The clique partitioning polytope and valid inequalities

Let $K_n = (V_n, E_n)$ be the complete graph with node set $V_n = \{1, \dots, n\}$ and edge set $E_n = \{U \subset V_n : |U| = 2\}$. For each edge $\{i, j\} \in E_n$, let $x_{ij} = 1$ if nodes i and j belong to the same clique, and $x_{ij} = 0$ otherwise. The clique partitioning polytope associated with K_n, denoted by CP_n, is the convex hull of vectors $x \in \{0, 1\}^{E_n}$ that satisfy the *triangle inequalities*:

$$x_{ij} + x_{ik} - x_{jk} \leq 1, \qquad \text{for } \{j, k\} \in E_n,\ i \in V_n \setminus \{j, k\}. \tag{1}$$

The clique partitioning problem is to determine a 0-1 vector, $x \in \mathrm{CP}_n$, that maximises a linear objective function $\sum_{e \in E_n} c_e x_e$, where the coefficients c_e are derived in different ways dependent on the kind of application. We will go into further details later, but at this point we focus on valid and facet-defining inequalities for CP_n.

We use the following notation. For a subset of nodes, $S \subseteq V_n$, $E_n(S)$ is the set of edges in E_n with both end-nodes belonging to S. For two disjoint subsets of nodes, $S, T \subset V_n$, $\delta(S, T)$ denotes the set of edges in E_n with one end-node in S and the other end-node in T. For a subset of edges, $E \subseteq E_n$, $V_n(E)$ is the set of nodes belonging to the ends of the edges in E, and we define $x(E) = \sum_{e \in E} x_e$.

We now present the 2-partition inequalities from [14]:

Proposition 1 *Let $S, T \subset V_n$ be disjoint and non-empty such that $|S| \leq |T|$. The* 2-partition inequality

$$x(\delta(S,T)) - x(E_n(S)) - x(E_n(T)) \leq |S|$$

is valid for CP_n*. It defines a facet of* CP_n *if and only if $|S| < |T|$.*

Figure 1 shows two examples of support graphs of 2-partition inequalities. Solid lines and dashed lines are edges for which the x-variables have coefficients $+1$ and -1, respectively. We note that triangle inequalities (1) are 2-partition inequalities with $|S| = 1$ and $|T| = 2$.

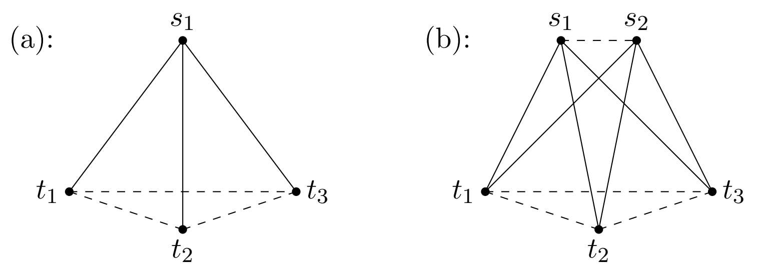

Figure 1: Support graphs of two 2-partition inequalities. (a): $S = \{s_1\}$, $T = \{t_1, t_2, t_3\}$; (b): $S = \{s_1, s_2\}$, $T = \{t_1, t_2, t_3\}$.

The following inequalities from [14] are also of interest here, because we will use them in the computational experiments.

Proposition 2 *Let $C \subset E_n$ be a cycle of odd length at least 5, and let $\bar{C} = \{\{i,k\} \in E_n : \{i,j\}, \{j,k\} \in C\}$ be the set of 2-chords of C. The* 2-chorded odd cycle inequality

$$x(C) - x(\bar{C}) \leq (|C| - 1)/2$$

defines a facet of CP_n*.*

The next inequalities, treated in [9, 10], are associated with odd wheels and bicycle wheels.

Proposition 3 *Let $C \subset E_n$ be a cycle of odd length, let $v \in V_n \setminus V_n(C)$, and let $F = \delta(\{v\}, V_n(C))$. The* odd wheel inequality

$$x(F) - x(C) \leq (|C| - 1)/2 \tag{2}$$

defines a facet of CP_n.

Let C be an odd cycle as above, let $u, v \in V_n \setminus V_n(C)$ be distinct nodes, and let $F' = \delta(\{u, v\}, V_n(C))$. The bicycle odd wheel inequality

$$x(F') - x(C) - x_{uv} \leq |C| - 1 \tag{3}$$

defines a facet of CP_n.

Deza *et al.* [10] provide a unifying framework in terms of the so-called clique-web inequalities. These include all of the 2-partition inequalities, odd wheel inequalities, and bicycle odd wheel inequalities. The facet-defining properties of these inequalities for the multicut polytope are also established (see [28] for further details).

The multicut polytope MC_n is the complement of the clique partitioning polytope. Consider vectors $x, y \in \{0, 1\}^{E_n}$ such that $y_e = 1 - x_e$, for all $e \in E_n$. Then

$$x \in \mathrm{CP}_n \quad \Longleftrightarrow \quad y \in \mathrm{MC}_n.$$

This relationship implies that an inequality, which is valid for one of the polytopes, can be transformed to a valid inequality for the other polytope by complementing the variables, and this transformation preserves any facet-defining property of the inequality.

3 Known separators

As mentioned in the introduction, separators are indispensable for a cutting-plane algorithm. For a specific class of valid inequalities for CP_n, the separation problem can be stated as follows: given a point $x^* \in \mathbb{R}^{E_n}$ outside of CP_n, determine an inequality $\sum_{e \in E_n} a_e x_e \leq a_0$ from the class that separates x^* from CP_n, i.e. x^* violates the inequality such that $\sum_{e \in E_n} a_e x^*_e > a_0$. If the separator guarantees to determine a violated inequality from the class, if any exists, the separator is said to be *exact* for this class of inequalities. Otherwise, it is regarded as a heuristic separator, which may fail to return a violated inequality.

Exact separators are preferable from a theoretical point of view, but from a practical point of view, heuristic separators may be very useful, if they are faster than their exact counterparts. NP-complete separation problems especially necessitate the use of one or more heuristic separators. Note that an NP-complete separation problem has an NP-hard companion, where the task usually is to determine a most-violated inequality instead of just any violated one.

The complexity of the separation problem for the 2-partition inequalities has not been established, but [22] proves that it is NP-hard when $|S|$ is fixed. For this reason, all known separators for these inequalities are heuristics, except for a few special cases such as the triangle inequalities.

Below, we first give an account of known exact polynomial-time separators for other classes of facet-defining inequalities for CP_n. This is followed by an outline of known separators for 2-partition inequalities from the literature. Some of these separators will later be used for comparisons with the new proposed separator for 2-partition inequalities.

3.1 Exact polynomial-time separators

Exact separators that run in polynomial time are known for the 2-chorded odd cycle inequalities, the odd wheel inequalities, and the bicycle odd wheel inequalities. These separators use an algorithm for determining a min-weight odd cycle in a graph with edges weighted by values derived from the point x^*. If this graph contains n nodes and m edges, a min-weight odd cycle can be determined by running Dijkstra's shortest-path algorithm n times, and all computations can be done in $O(nm)$ time [11].

Gerards [12] provides a separator for the bicycle odd wheel inequalities (3) with running time $O(n^5)$. Deza *et al.* [10] describe a separator for the odd wheel inequalities (2), which is similar to the one by Gerards, but with running time $O(n^4)$.

Caprara and Fischetti [5] establish that the 2-chorded odd cycle inequalities and odd wheel inequalities can be obtained as $\{0, \frac{1}{2}\}$-cuts. The $\{0, \frac{1}{2}\}$-cut separator, which includes these inequalities, has running time $O(n^5)$. Clearly, the $O(n^5)$ $\{0, \frac{1}{2}\}$-cut separator is not preferable to the $O(n^4)$ separator for the odd wheel inequalities, but we do not know of a faster exact separator for the 2-chorded odd cycle inequalities.

Quite recently, Letchford and Vu [18] considered a separator for large classes of valid inequalities that can be obtained from so-called 'gadgets'. This separator generalises the above $\{0, \frac{1}{2}\}$-cut separator, while its running time is of the same order.

We have implemented the first three of the above separators, and in doing so we have tried to make them as efficient as possible. For example, the algorithm for determining a min-weight odd cycle runs the shortest-path computations in parallel. We also note that each shortest path that is obtained during the odd cycle computations corresponds to a valid inequality, and therefore each of these separators may return several violated inequalities instead of just a single most-violated one.

3.2 Known separators for 2-partition inequalities

This section considers the separators that are described in the literature. As already mentioned, the triangle inequalities (1) are special cases of 2-partition inequalities, and separation for these inequalities can be done in $O(n^3)$ time by enumeration. In the following, we will assume that they are all satisfied by the point $x^* \in \mathbb{R}^{E_n}$ to be separated before any other 2-partition inequalities are generated.

2-partition inequalities have been used as cutting planes in several studies. In [2, 6, 13, 22, 23, 26, 31], fast separation heuristics for 2-partition inequalities with $|S| = 1$ are applied. Except for the one in [26], they are all based on the heuristic described in [13]. Wakabayashi [32] outlines a separation heuristic for the more general case with $|S| \geq 2$, but no examples of its use are available. Sørensen [27] considers a separation heuristic that also allows for $|S| \geq 2$, but we will not go into details here. A heuristic separator for a closely related class of inequalities, the so-called α-inequalities [19] for the edge-weighted clique problem, is used in [29]. Below, we will describe some of these separators in more detail.

The following observation is made in [13, 32]. Suppose that $x^*_{st} \in \{0, 1\}$, and that nodes s, t belong distinctly to each of node sets S, T in a 2-partition inequality violated by x^*. If all triangle inequalities (1) are satisfied, one of these nodes can then be removed from $S \cup T$, and the resulting 'smaller' 2-partition inequality will still be violated by x^*. As will be clear in the following, this fact is used in the construction of node sets S and T.

We first outline the heuristics in Grötschel and Wakabayashi [13, 32]. There are three separators, which we will call GW1, GW2, and GW3 here, and which are meant to be used in a hierarchical manner: when all triangle inequalities are satisfied, GW1 is applied first; GW2 is then applied if GW1 fails; and GW3 is used if GW2 fails.

- GW1 is a fast heuristic separator for 2-partition inequalities with $|S| = 1$. Each node $s \in V_n$ is considered such that $S = \{s\}$, and T is constructed by including nodes u from the sequence $1, 2, \ldots, n$ such that $0 < x^*_{su} < 1$ and $x^*_{tu} = 0$, for all $t \in T$. A second attempt to construct node set T is made by considering the nodes u from the reverse sequence.

- GW2 only differs from GW1 in the way that T is constructed. A node $u \in V_n \setminus T$ is included in T if $x^*_{su} - \sum_{t \in T} x^*_{tu} > 0$.

- GW3 handles the general case where $|S| \geq 2$. It starts out by constructing a list L of 'buckets' of edges with fractional values x^*_e, where each bucket contains edges with similar values, and the buckets are considered in descending order of the values. The first edge $e = \{u, v\}$ in L is used to initialise two node sets $A = \{u\}$ and $B = \{v\}$.

The separator then continues to enlarge A and B in an iterative manner as follows. The next edge $\{u, v\}$ in L with end-nodes disjoint from $A \cup B$ is chosen, and nodes u, v are inserted into A and B with one node in each set. This is done such that the largest left-hand side value is obtained for the 2-partition inequality that results from (S, T) equal to either (A, B) or (B, A).

The next step in a given iteration is to enlarge A and B temporarily. That is, new node sets $\bar{A}$ and $\bar{B}$ are initialised as copies of A and B, respectively, and for each node $v \notin A \cup B$, it is decided if v should be placed into $\bar{A}$ or $\bar{B}$. This choice is made such that the largest gain to the possible violation of the resulting 2-partition inequality is obtained. This temporary enlargement continues until a violated inequality results, or all nodes are used.

The running time of each separator GW1, GW2, and GW3 is bounded by $O(n^3)$. Böcker *et al.* [2] use GW2, Oosten *et al.* [22] use the separator GW1-GW2, and most of the other separators considered in the literature are based on GW1 or GW2. Tcha *et al.* [31] use a randomised version of GW1, while Catanzaro *et al.* [6] and Recalde *et al.* [23] use randomised versions of GW2. We have also tested the last mentioned separator and therefore give a short description of it.

The separator by Recalde *et al.* works as follows. For each $s \in V_n$, let $W = \{u \in V_n \setminus \{s\} : x^*_{su} \notin \mathbb{Z}\}$, let F be an initially empty set of forbidden nodes, and repeat the following 5 times, if possible. Randomly pick two nodes i, j from $W \setminus F$, and let $T = \{i, j\}$. Keep picking nodes u from $W \setminus F$ such that $x^*_{su} - \sum_{t \in T} x^*_{tu} > 0$ and add u to T until no more nodes are found. Then check if the resulting 2-partition inequality with $S = \{s\}$ and T is violated, and let $F = F \cup T$ (even if the inequality is not violated). We will refer to this separator as GW2-R.

Simanchev *et al.* [26] use a local search procedure to obtain 2-partition inequalities with $|S|$ fixed at 1. We have no implementation of this separator and, therefore, will not test it.

Finally, we outline the separator in [29] as it applies to 2-partition inequalities. This separator works on the set of active (binding) 2-partition inequalities of the current LP, including triangle inequalities. Descriptions of these inequalities, in terms of the node sets involved, are available in a separate data structure, which is maintained beside the LP data. The idea is to extend one of the node sets S or T by a single node that is not already contained therein and that results in a 'most-violated' new inequality.

Since the active inequalities are satisfied at equality, it is very easy to identify extensions that result in new violated inequalities. As the number of active 2-partition inequalities can be limited to the number $|E_n|$ of variables or fewer (by using only inequalities with non-basic slack variables), this separator runs in time bounded by $O(n^4)$ and is very fast in practice. We will refer to this separator as Sor-Ex.

4 The new separator

This section starts by giving a summary of a computational experiment to support a preference for small 2-partition inequalities. This is followed by a presentation of the new separator.

4.1 A preference for small 2-partition inequalities

A 2-partition inequality based on $k = |S| + |T|$ nodes contains $k(k-1)/2$ non-zero left-hand side coefficients. In order to keep LPs sparse, we prefer the use of inequalities with small k to inequalities with large k for the same degree of violation. Then, a relevant question is whether inequalities with large k will contribute significantly to improving the z-values obtained from the LPs. In an attempt to give at least a partial answer to this question, we have used a separator that enumerates all 2-partition inequalities with $k \leq p$, where p is a parameter to be given as input to the separator. This has allowed us to compare the z-values, which can be obtained for different values of p. Note, however, that this separator runs in $O(n^p)$ time and becomes prohibitively slow for large p.

We have used this separator with values of p up to 7 in our cutting-plane algorithm and have run it on a few problem instances with different structural properties. There are some instances, where the size of p does not matter much, because the z-value obtained by using the triangle inequalities is already very close to the value of an optimal partition. Conversely, there are also some instances, where substantial improvement on the z-value can be obtained.

Figure 2 illustrates the effect of parameter p on the z-values for three representatives of these instances (their characteristics are given in Section 6.1). Here, $p = 3$ gives the z-values z_3 obtained by using only triangle inequalities, and we index the z_p values from this base to make comparisons easier. The figure shows that considerable improvements on z_3 are obtained already when $p = 4$, and further improvements follow by incrementing p. However, these improvements tend to become smaller as p increases. This confirms our prior expectations and suggests that the use of 2-partition inequalities with large node sets S, T will only lead to marginal improvements, and that most effort should be placed on using the inequalities on smaller node sets.

4.2 The new separator for 2-partition inequalities

Our proposed separator consists of a construction phase and an improvement phase and works in the following way. Each edge $\{a, b\} \in E_n$, for which

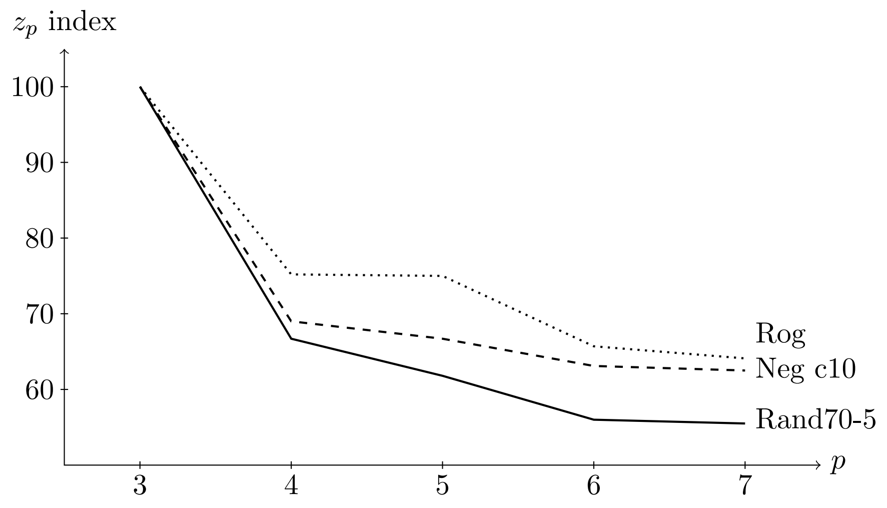

Figure 2: Examples of the effects of sizes $p \geq |S| + |T|$ on z-values.

$0 < x^*_{ab} < 1$, is given as input to the construction phase, which uses the end-nodes of this edge to initialise two node sets $A = \{a\}$ and $B = \{b\}$. These node sets will then be enlarged until they contain a total of p nodes (we use $p = 10$). Subsequently, we extract two subsets $A' \subseteq A$ and $B' \subseteq B$ consisting of the first $\ell \leq p$ nodes, where ℓ is chosen such that A', B' gives a 'best possible' 2-partition inequality, and S will be the smaller set of A', B', and T will be the other one. If the resulting inequality is violated by x^*, it is returned by the separator. Next, the pair A', B' is given as input to the improvement phase, which may modify the node sets in different ways in search of a violated 2-partition inequality. Therefore, two new cuts may be generated for each edge that is used as input.

The construction phase takes inspiration from a heuristic principle developed by Kernighan and Lin [15]. To initialise the enlargement of A and B, we set $\delta_A(i) = x^*_{ai}$ and $\delta_B(i) = x^*_{bi}$ for each node $i \in V_n \setminus (A \cup B)$. Then, in order to determine the k'th node ($k = 3, \ldots, p$) to add to A or B, we calculate, for each node i, the contribution to the violation of the inequality that is obtained by adding i as the k'th node to A or B. Let $\gamma_A(i)$ and $\gamma_B(i)$ be the gains obtained by adding i to A and B, respectively. Then,

$$\gamma_A(i) = \begin{cases} \delta_B(i) - \delta_A(i), & \text{when } |A| \geq |B| \\ \delta_B(i) - \delta_A(i) - 1, & \text{when } |A| < |B|. \end{cases}$$

In the latter case, when $|A| < |B|$, the smallest node set is enlarged, and the right-hand side coefficient must be incremented by 1. $\gamma_B(i)$ is calculated analogously.

The k'th node that is added to A or B is the node $k^* \in V_n \setminus (A \cup B)$

for which $\gamma_A(k^*)$ or $\gamma_B(k^*)$ is maximal. Suppose, for example, that $\gamma_A(u)$ is maximal. We then add u to node set A, and for all nodes i not contained in $A \cup B$ we update $\delta_A(i) := \delta_A(i) + x^*_{ui}$. During this update, we also pick the $k+1$'st node to add to A or B. This process is repeated until A and B contain a total of p nodes.

Let g_k be the gain to the violation that is obtained by extracting the k'th node from A or B, e.g. $g_k = \gamma_A(k^*)$ if the k'th node is extracted from A. We also have $g_1 = -1$ and $g_2 = x^*_{ab}$ from the initialisation of A, B. These gains are additive, for $k = 1, \ldots, p$, and although some of the values g_k are negative, there may be an index k' that attains a positive sum of gains. In this case, we obtain a violated inequality by extracting the first k' nodes from $A \cup B$, giving the subsets $A' \subseteq A$ and $B' \subseteq B$. Accordingly, we consider

$$G_k = \sum_{j=1}^{k} g_j \quad \text{and} \quad D_k = G_k / \sqrt{k(k-1)/2}, \qquad \text{for } k = 1, \ldots, p. \tag{4}$$

G_k in (4) equals the violation of the 2-partition inequality that is associated with the first k nodes in $A \cup B$, and D_k is the Euclidian distance between x^* and the hyperplane that is defined by the resulting inequality. Because we prefer the distance measure to the violation measure, we choose the index $\ell \in \{1, \ldots, p\}$ such that D_ℓ is maximal. Note that we will get $\ell \geq 4$ for any violated inequality, when all triangle inequalities are satisfied by x^*.

The heuristic then enters the improvement phase. This phase attempts to modify the node sets in two steps. First, it is examined if the violation can be increased by removing nodes, one at a time, from A' and B', which is often possible when $|A'| = |B'|$. If this is not sufficient to obtain a violated inequality, the second step attempts to exchange nodes in $A' \cup B'$ with nodes not in $A' \cup B'$; this happens one pair of nodes at a time.

The computational complexity of the construction phase is bounded by $O(n^3)$. For each initial edge, the enlargements of $A \cup B$ can be done in $O(n)$ computations because the size p is fixed and independent of n. Furthermore, the extraction of subsets A', B' can be done in constant time. The complexity of the improvement phase depends on the number of exchanges that takes place. However, each exchange of a pair of nodes takes $O(n)$ time, and computational experience shows that few exchanges are performed.

5 Cutting-plane algorithm

We use a cutting-plane algorithm (CPA), which we have developed over several years. In the present context, it works as follows. The initial LP only contains bounds $0 \leq x_e \leq 1$ on the variables. Subsequently, we call the *triangle optimiser* routine described below to determine an LP solution x^* that satisfies all triangle inequalities (1). Once such an x^* is available, and if it contains fractional values, it will be used as input to one of the above described separators. If any violated cuts are returned by this separator, a selection is made of cuts to add to the LP. The resulting LP is then reoptimised, and the triangle optimiser is invoked again on the new solution. This process continues until no more cuts are obtained or it is deemed futile to continue cut generation due to tailing off of the LP objective values. During this process, inactive cuts are deleted in order to keep the number of LP constraints under control.

The separators use a tolerance limit of 0.01 to determine if an inequality is violated by x^*. That is, an inequality $a^T x \leq a_0$ is only returned as a cut if $a^T x^* - a_0$ exceeds this tolerance. We use the C API of Cplex to handle the LPs, and all LPs are solved by the dual simplex routine CPXdualopt().

5.1 Cut selection

Some of the separators may generate many cuts, and instead of adding all of them to the LP, we make a selection among the most violated ones. This selection takes into consideration the depth of the cuts (distance to x^*) and their parallelism. This is similar to the process described in [1].

We first sort the cuts in non-increasing depths $(a^T x^* - a_0)/\|a\|$ and only keep those with depths greater than a lower tolerance equal to 0.5 times the largest depth and not smaller than 0.002. Next, we want to avoid adding two cuts $a^T x \leq a_0$ and $b^T x \leq b_0$ that are too parallel. That is, we set an upper tolerance limit of 0.5 on the parallelism $|a^T b|/(\|a\|\|b\|)$ of the cuts. The selection of cuts proceeds by selecting the first cut in the sorted list, then traversing the list and selecting only cuts that are not too parallel to any already selected cut, and we restrict the number of selected cuts to $\min(|E_n|, 1500)$. The tolerances used here have been determined after some experiments with different settings.

5.2 Cut deletion

As mentioned above, inactive inequalities are removed from the LPs. Every time the LP has been reoptimised, we check the basis status of each slack variable and count the number of consecutive times the variable has been basic. Whenever this number reaches 5, we consider the inequality to be inactive and subject to removal.

Unfortunately, it sometimes happens that successive LP solutions get trapped for a while in a hyperplane with the same objective value. In this situation, there is a possibility that a deleted constraint might subsequently become violated and added again, and eternal cycling through solutions in this hyperplane would result. For this reason, any deletion of inequalities from the LP only takes place, when the value of the objective has strictly improved since the previous deletions took place.

5.3 Triangle optimiser

The separator for triangle inequalities (1) enumerates all triplets of nodes. However, for each edge $\{j,k\} \in E_n$, it returns only the most violated inequality $x_{ij} + x_{ik} - x_{jk} \leq 1$, if any exists. All inequalities obtained in this way are subjected to the cut selection process, and therefore some of them may not be added to the LP. The LP is then reoptimised, and the process is repeated until all triangle inequalities are satisfied by the current LP solution. Cut deletions also take place during this process.

5.4 Tailing-off termination

During the first rounds of adding cuts, we usually observe considerable improvements of the z-values. However, as the CPA proceeds, and the LP solutions get closer to the convex hull of integer feasible solutions, these improvements of the z-values become gradually smaller. Furthermore, some of the separators are capable of generating violated cuts for a very large number of rounds. Therefore, it may become desirable to terminate the CPA before it has exhausted all possibilities of adding new cuts. This becomes particularly relevant, when the CPA is embedded in a branch-and-cut algorithm, and there is a choice to branch instead of adding more cuts.

In order to detect when the tailing off of the z-values becomes prevalent, we use two parameters, a tolerance $\theta \in (0;1]$ and a maximum count η. Let z' be the z-value from the previous round and z be the current value (with $z \leq z'$). We use the condition $z \geq \theta z'$ to determine the potential of tailing off, and if this condition continues to be met in η consecutive rounds, the CPA terminates with z-value z. In the following computational experiments, we will use two different settings of these parameters and state them explicitly.

6 Computational experiments

This section explains the computational experiments and presents the results of using the CPA on several problem instances from different sources. We first present the test instances and their sources. This is followed by computational comparisons of the separators described in Sections 3 and 4. Finally, we present the results of applying the CPA with the use of 2-partition inequalities on the test instances.

6.1 Applications and test instances

The problem instances we consider fall into four categories: three types of applications and random instances. They are described in the following subsections.

6.1.1 Aggregation of binary relations

One type of application is the problem of aggregation of binary relations (ABR) into an equivalence relation as stated, e.g. in [13]. Here, one is given a set of n objects each with m qualitative characteristics. A node is defined for each object, and the similarity of each pair of objects is expressed as an edge weight that is calculated as: 2 times the number of characteristics that are similar among the two objects minus the number of characteristics m. Hence, we seek a clique partition that maximises the total similarities among the objects.

We use the largest instances from [13, 32] (Companies and the UNO instances) together with one small instance (Workers). A new and larger instance (Soup), which has not been presented previously, is also included. This instance originates from a consumer survey with data provided by a former colleague.

6.1.2 Machine cell formation

Another application is that of machine cell formation (MCF) in cellular manufacturing as explained in [22] and the references therein. We are given a set of p parts and q machines and a $p \times q$ matrix A with entries $a_{ij} = 1$, if part i must be processed by machine j, and $a_{ij} = 0$ otherwise. A node is defined for each part and each machine, so that $n = p + q$, and the edge weights of the complete graph K_n are obtained as follows. An edge between two 'part-nodes' or two 'machine-nodes' has weight 0; an edge between a part-node and a machine-node (corresponding to part i and machine j) has weight $+1$, if $a_{ij} = 1$, and weight -1, if $a_{ij} = 0$.

We use MCF instances from the literature, for which the triangle inequalities are insufficient to obtain integer optimal LP solutions. Table 1 gives an overview. The instances Mcc and Rog are obtained from 0-1 matrices associated with applications of marketing techniques and industrial purchasing behaviour, respectively, but the edge weights are obtained as described above.

6.1.3 Cluster editing

The third application is a graph approximation problem also known as cluster editing, e.g. [2, 26] and further references therein. Given a graph $G = (V, E)$ on $|V| = n$ nodes, the cluster editing problem is to determine a set of edge modifications (insertions and deletions) of minimum cardinality, such that the modified graph is a clique partition. Let $P \subseteq E_n$ be the edge set of a clique partition. The number of insertions into P and deletions from E are then $|P \setminus E|$ and $|E| - |P \cap E|$, respectively. In order to minimise the number of edge modifications, the weights of the edges in E_n are $+1$ for the edges in $E_n \setminus E$ and -1 for the edges in E. This problem also comes in a

Name	$p \times q$	Source
Cha	43×16	Chan and Milner [7]
Gro	20×23	Kumar, Kusiak and Vannelli [17]
Ira	19×12	Rogers and Kulkani [24]
Kin	24×14	King [16]
Mcc	24×16	McCormick, Schweitzer and White [20]
Mil	35×25	Miltenburg and Zhang [21]
Rog	30×30	Rogers and Kulkani [24]
Sei	22×11	Seifoddini [25]
Sul	22×11	Sule [30]

Table 1: Some MCF instances from the literature.

weighted version with modification costs associated with all pairs of nodes. The transformation of edge weights to this version is straightforward.

We use cluster editing instances from Böcker *et al.* [2]. The data files are available at `www.uni-jena.de/peace` and contain maximisation versions of the instances (the edge weights have opposite signs to those mentioned above). In [2], some reduction methods are presented, which are used to pre-process problem instances. These methods result in weighted versions of reduced cluster editing instances, and there are almost 3000 such instances in the data set. However, most of the instances are trivial in the sense that integer optimal LP solutions are obtained when all triangle inequalities (1) are satisfied. The number of remaining non-trivial instances is around 240, and we will only consider some of the most interesting of these instances here.

6.1.4 Random instances

The random problem instances come from different sources. We use some of the 'negative' instances (Neg) from [3], which are converted from equicut instances by simply skipping the size restrictions on the subsets of nodes of a partition. The last two digits of an instance name, such as Neg s80, state the percentage of edges of the complete graph with 0 weights. The remaining edges have random integer weights drawn from the interval -10 to 10, not including 0.

We also use the instances Rand100-5 and Rand100-100 on 100 nodes from [4] and [8], respectively. Rand100-5 has edge weights that are random integers in the interval -5 to 5, whereas Rand100-100 has random weights in the interval -100 to 100. From these two instances, we have also obtained eight smaller instances by keeping only the first 50, 60, 70, and 80 nodes of each original instance.

6.2 Comparisons of the separators

In this section, we compare the z-values that can be obtained by the new separator with those obtained by the known separators. The purpose is to examine the potential of the separators by letting them generate new cuts as long as they possibly can. For this reason, the tailing-off termination mechanism is suspended by using parameters $(\theta, \eta) = (1, 1000)$. As a consequence, the CPA sometimes runs for a long time, and it is only practically possible to consider some of the smaller problem instances.

We first make a comparison of the new separator with the known separators for 2-partition inequalities. The results are presented in Table 2, where the first two columns state the instance and number of nodes of the complete graph K_n. The next columns contain the z-values obtained by the CPA, when different separators are applied: z_3 comes from the first call of the triangle optimiser, i.e. the z-value that is obtained when only triangle inequalities are used; GW is the 3-step heuristic GW1–GW3; GW2-R and Sor-Ex are named as in Section 3.2; and New is the new separator proposed in this paper. The last row of the table summarises the total computation times used by the CPA to obtain the stated z-values.

			LP objective values			
Instance	n	z_3	GW	GW2-R	Sor-Ex	New
Mcc	40	56.67	50.16	45.29	43.32	43.00
Cha	59	84.00	75.03	68.02	67.45	67.00
Mil	60	57.33	50.31	46.79	46.11	46.00
Rog	65	97.33	85.63	73.36	67.13	62.26
Neg c10	50	1110.00	841.88	841.51	717.54	687.54
Neg c00	50	1361.50	973.40	1018.04	829.02	792.32
Neg s80	60	495.72	494.76	485.60	474.87	473*
Neg tt80	70	637.24	635.21	625.53	605.16	597.99
Rand50-5	50	819.50	586.74	640.80	513.07	492.09
Rand50-100	50	15,062.50	11,029.10	11,796.60	9613.69	9131.57
Rand60-5	60	1189.50	885.85	920.88	707.45	665.55
Rand60-100	60	21,035.00	15,194.20	15,657.60	12,479.20	11,831.50
Rand70-5	70	1612.50	1218.29	1258.80	938.74	871.74
Rand70-100	70	28,962.50	21,620.60	22,043.30	16,691.10	15,558.80
total time (sec)			+200,000	26,282	2,109	28,568

Table 2: z-values from the known separators for 2-partition inequalities and the new one.

Separators GW and GW2-R perform worse than Sor-Ex in all instances with larger z-values and more time consumption. In fact, the GW separator turns out to result in large time consumption by the CPA in some instances, where it seems to be able to generate cuts 'forever', possibly because relatively few cuts are generated in each round compared to some of the other separators. A particularly discouraging example appears in the instance Neg tt80, where we terminated the CPA after 36 hours. Investigation into this particular case revealed that, on top of the 'never-ending' rounds of cut generation, many cuts were large in terms of the number of nodes involved in the 2-partition inequalities, and the LP reoptimisations proceeded very slowly. For this reason, we set a time limit on the CPA of 5 hours for all remaining computations. This limit was exceeded only when using the GW separator, and the z-values reported for instances Rand60-100, Rand70-5, and Rand70-100 are those obtained on termination after this time limit.

Using the new separator gives better z-values than using Sor-Ex, but also involves considerably higher time consumption by the CPA. We prefer the ability to obtain better z-values over shorter time consumption, because we believe that we can control the time consumption of the CPA by adjusting the tailing-off termination rule in practical computations.

Table 3 shows corresponding results for comparisons between the new separator and the exact polynomial-time separators for the other inequality classes mentioned.

			LP objective values			
Instance	n	z_3	OBW	OW	ZH	New
Mcc	40	56.67	47.88	44.28	44.17	43.00
Cha	59	84.00	71.64	68.96	68.94	67.00
Mil	60	57.33	52.50	46.73	46.62	46.00
Rog	65	97.33	79.25	73.17	73.17	62.26
Neg c10	50	1110.00	819.10	765.76	765.74	687.54
Neg c00	50	1361.50	945.35	907.67	907.67	792.32
Neg s80	60	495.72	486.37	475.12	474.66	473*
Neg tt80	70	637.24	623.22	607.63	607.13	597.99
Rand50-5	50	819.50	572.52	553.57	553.42	492.09
Rand50-100	50	15,062.50	10,653.60	10,174.80	10,171.30	9131.57
Rand60-5	60	1189.50	808.51	793.00	793.00	665.55
Rand60-100	60	21,035.00	14,344.90	14,023.30	14,023.30	11,831.50
Rand70-5	70	1612.50	1079.53	1075.00	1075.00	871.74
Rand70-100	70	28,962.50	19,348.10	19,308.30	19,308.30	15,558.80
total time (sec)			24,717	1,269	9,932	28,555

Table 3: z-values from the exact separators for other inequality classes and the new separator for 2-partition inequalities.

Here we will discuss the performance of these separators with respect to the z-values that are obtained. The odd bicycle wheel (OBW) separator performs the worst, when considered in isolation as is done here. It might prove worthwhile to combine this separator with some of the other ones, e.g. the odd wheel (OW) separator. However, it is beyond the scope of this paper to explore the potential of such combinations.

Comparing the z-values obtained by the OW separator and the $\{0, \frac{1}{2}\}$-

cut separator (ZH), an interesting observation can be made. The latter separator, which generates odd wheel and 2-chorded odd cycle inequalities (among others), only performs slightly better than the dedicated separator for the odd wheel inequalities. This suggests that the 2-chorded odd cycle inequalities are not very useful in practical computations, except perhaps in some special cases, which we have not seen here.

It is evident that the new separator for 2-partition inequalities outperforms the exact separators in all instances, and in many cases considerably better z-values are obtained. This is also the most time-consuming separator to use in the CPA, as shown in the bottom line of the table. However, this is due to more cuts being used in more rounds and with more reoptimisations being performed. Taking into account the tailing-off termination detection of the CPA, this will not be an issue in practical computations.

6.3 Testing the CPA with the new separator

In this section, we present the results that are obtained, when the new separator for 2-partition inequalities is used by the CPA. In contrast to the results presented above, we change the settings of tailing-off termination so that the CPA will stop when adding more cuts seems to be of little practical use. We first outline the experimental setup and then present computational results for the various types of problem instances.

6.3.1 Experimental setup

All computations are performed on a Lenovo Thinkpad T570 computer with Windows 10 operating system. It has an Intel Core i7-7600U CPU running at 2.8 GHz and 16 GB of RAM. The CPA is coded in C++ and compiled with Visual Studio, and we use Cplex version 12.7.1 as the LP solver.

In all experiments, we have imposed a time limit of 5 hours (18,000 seconds) on the CPA. The parameters for tailing-off termination are set to $(\theta, \eta) = (0.999, 4)$ so that the CPA terminates when the z-values do not improve by more than a factor of 0.001 in four consecutive rounds. It seems to be better to branch at this point instead of continuing to add cuts with only little effect. This is an issue that remains to be further examined in future work, when the CPA is embedded in a more advanced branch-and-cut framework.

6.3.2 Results for ABR and MCF instances

Table 4 presents the results for the ABR and MCF instances. In this table and the following ones, LB gives the best known value (lower bound) of a feasible clique partition and, as above, z_3 states the z-value obtained by the triangle optimiser. z_{LP} is the z-value of the solution returned by the CPA (missing values mean that the new separator has not been applied after the triangle optimiser). Throughout, we use asterisks (*) to indicate the value of an optimal clique partition and integer optimal LP solution. '% gap' is the percentage of the gap between z_3 and LB, which is closed by z_{LP}. '# cut' shows the total number of 2-partition inequalities (excluding triangle inequalities) added to LPs by the CPA, and '# rnd' is the number of rounds of cut generation involved. 'Time' is the total time used by the CPA in seconds, rounded to the nearest tenth.

Instance	n	LB	z_3	z_{LP}	% gap	# cut	# rnd	Time
Workers	34	964*	964*	–	–	0	0	0.0
Companies	137	81802*	81802*	–	–	0	0	0.1
Uno 1a	158	12197*	12197*	–	–	0	0	0.1
Uno 2a	158	72820*	72820*	–	–	0	0	0.2
Uno 3a	158	73068*	73068*	–	–	0	0	0.0
Soup	209	4622*	4706.00	4625.50	99.4	376	9	5.1
Ira	31	38*	48.67	38.07	99.3	869	19	0.7
Sul	31	46*	48.00	46.00	100	78	7	0.1
Sei	33	54*	55.67	54*	100	61	3	0.1
Kin	38	41*	41.00	41*	–	10	2	0.0
Mcc	40	43*	56.67	43.26	98.9	2015	25	2.3
Gro	43	53*	75.33	53.02	99.9	3368	27	4.3
Cha	59	67*	84.00	68.09	93.6	5188	25	13.6
Mil	60	46*	57.33	46.60	94.7	6105	26	14.3
Rog	65	60*	97.33	63.97	89.4	17281	51	101.1

Table 4: Results for some instances of aggregation of binary relations and machine cell formation.

The ABR instances are shown in the upper part of Table 4. In the first five instances, we obtain integer optimal LP solutions already after using the triangle optimiser, and it is not necessary to apply the new separator. These results are, therefore, only interesting in order to demonstrate the efficiency of the triangle optimiser and the CPA to filter through a large number of inequalities to select the important ones. As is evident, this can be done in small fractions of a second. Considering the Soup instance, the CPA adds 376 2-partition inequalities as cuts during 9 rounds. This closes a very large part of the gap between z_3 and the optimal partition value of 4622, but the LP solution obtained by the CPA is not integer. (We use branch-and-cut to obtain optimal integer solutions, but we do not explain details here.)

We have two further remarks to the above results for the ABR instances. 1) In [13], triangle inequalities are not sufficient to obtain an optimal integer LP solution of the instance Workers, and the separator GW1 is applied to achieve such a solution. We use a lower violation tolerance than the one used in [13], and this is sufficient to obtain an integer LP solution from the triangle inequalities. 2) The z_3-value for the instance Uno 3a is not identical to the one reported in [13, 32]. We have used the data set presented in [32], and according to Martin Grötschel (personal communication around 1999–2000), some of the data was lost and might not have been recovered correctly.

The MCF instances appear in the lower part of Table 4. Except for two instances, the CPA returns fractional LP solutions, but the associated z_{LP}-values are quite close to the values of the optimal clique partitions. Again, we see that the use of 2-partition inequalities closes major parts of the gaps between z_3 and LB. In some instances, several thousand cuts are added, and the time consumed by the CPA is significant. This is the price we pay in order to close large parts of the gaps. Considering the instance Rog, the new separator is used during 51 rounds, and the CPA uses more than 100 seconds. In order to give an impression of the time consumption of the new separator, we note that it accounts for less than 2 seconds out of the total time.

6.3.3 Results for cluster editing instances

We have only become aware of the close relationship between the clique partitioning and cluster editing problems, as pointed out in [26], rather late in the process of this work. Therefore, instances of cluster editing were not considered in the first analyses, which we have conducted for this paper, and we believe that inclusion of such instances would not result in different conclusions. We present computational results for a representative and most interesting subset of these instances.

There are two sets of data from [2], and all instances considered here belong to set 1. Instance names are constructed in order to enable identification of the original data in the data set. Vn/ki/fpℓ is the name of a reduced instance, which now contains n nodes; the reduction has been accomplished by using the value i of parameter k; and the data file has position ℓ in the folder that contains instances with n nodes and parameter $k = i$.

Table 5 presents the computational results. In this table, we have organised instances into three groups. The first group contains instances for which the CPA returns fractional solutions; the second group contains instances where the CPA obtains integer optimal solutions; and the third group contains the largest instances.

The first group of instances shows results that are similar to some of those from MCF instances. In several instances, many cuts are used resulting in large parts of the gaps being closed, and the time consumed by the CPA is significant. Instances in the second group are easier than those in the first group. In general, fewer cuts are added during fewer rounds, integer LP solutions are obtained, and the CPA tends to consume less time. The large instances in the third group are relatively easy. The triangle optimiser

Instance	n	LB	z_3	z_{LP}	% gap	# cut	# rnd	Time
V49/k7.5/fp2	49	124*	262.50	133.23	93.3	11008	41	51.9
V49/k8/fp2	49	159*	297.50	167.06	94.2	11441	34	72.2
V49/k8/fp10	49	147*	290.50	158.48	92.0	14168	40	79.4
V49/k8.5/fp6	49	189*	324.00	196.80	94.2	8758	26	71.0
V60/k7.5/fp8	60	146*	333.00	155.76	94.8	15577	54	144.7
V70/k9/fp4	70	624*	646.00	624.33	98.5	611	4	10.0
V80/k9/fp8	80	515*	655.00	515.54	99.6	1085	7	21.2
V90/k10/fp6	90	769*	861.50	769.00	100	922	3	24.6
V49/k7.5/fp10	49	224*	326.50	224*	100	2413	7	18.1
V60/k7.5/fp7	60	241*	368.00	241*	100	1877	8	15.8
V70/k8.5/fp8	70	532*	589.00	532*	100	528	1	6.0
V80/k9/fp5	80	492*	630.50	492*	100	715	2	12.6
V90/k10/fp8	90	682*	835.00	682*	100	857	1	18.0
V100/k10/fp6	100	780*	935.00	780*	100	1252	3	33.7
V150/k10/fp9	150	1306*	1489.50	1306*	100	1721	1	31.7
V200/k10/fp2	200	2659*	2663.00	2659.00	100	498	3	14.5
V340/k8/fp9	340	11984*	11985.00	11984.00	100	8	2	6.2
V350/k6/fp5	350	10838*	10839.00	10838*	100	4	2	2.6
V550/k8/fp4	550	38147*	38147.50	38147*	100	1	1	23.3
V600/k9/fp4	600	50330*	50330.50	50330.30	40.0	1	2	30.9
V950/k10/fp1	950	124612*	124612.00	124612*	–	1	1	150.4

Table 5: Results for some instances of cluster editing.

obtains LP solutions whose objective values are the same as or very close to the values of the optimal partitions. This tends to leave little work for the other parts of the CPA, and almost all z-values obtained are identical to the optimal partition values. Nevertheless, the CPA only returns integer LP solutions in three of the cases.

The time used by the new separator is very small in all instances. Most time is spent in the first five instances, which also have the largest number of rounds of cut generation. In none of these instances does the total time consumption of the new separator exceed 1.5 seconds. Considering the largest instance on $n = 950$ nodes, almost all time is consumed by the triangle optimiser. The time spent by the new separator to identify a single cut is negligible (it is less than 0.01 second). This is because the LP solution contains mostly integer values, and this leaves little work for the separation heuristic.

6.3.4 Results for random instances

The computational results for the random instances are shown in Table 6. The Neg instances appear in the upper part of the table. Although large parts of the gaps are closed in some of the instances, this is not so distinct as we have seen in other instances, and there are some instances where the CPA does not have much effect beyond the triangle optimiser. This is especially true with regard to instances Neg c80 and Neg o80, where the tailing off condition is triggered from the very first round. Time consumption is also significant or even excessive in the larger instances. In the instance Neg o80, it takes the CPA more than one hour to complete only four rounds.

Instance	n	LB	z_3	z_{LP}	% gap	# cut	# rnd	Time
Neg c80	50	317*	327.92	327.57	3.2	342	4	4.0
Neg c70	50	452*	480.26	453.43	94.9	8921	25	43.1
Neg c60	50	463*	548.84	486.67	72.4	14333	35	70.1
Neg c50	50	549*	671.87	572.71	80.7	11089	32	47.4
Neg c40	50	577*	742.00	604.45	83.4	12801	37	55.3
Neg c30	50	582*	822.00	596.85	93.8	13733	44	68.8
Neg c20	50	604*	953.00	651.99	86.2	13946	39	70.8
Neg c10	50	649*	1110.00	705.18	87.8	15980	47	88.1
Neg c00	50	752*	1361.50	813.39	89.9	15211	42	84.9
Neg s80	60	473*	495.72	492.33	14.9	2084	7	33.2
Neg tt80	70	590	637.24	619.91	36.7	22271	27	750.2
Neg o80	80	718	795.13	793.45	2.2	5645	4	3738.4
Rand50-5	50	479*	819.50	503.41	92.8	13467	35	112.1
Rand50-100	50	8832*	15062.50	9389.75	91.0	16573	40	126.2
Rand60-5	60	613	1189.50	689.89	86.7	29192	43	627.6
Rand60-100	60	10954	21035.00	12281.90	86.8	27975	46	497.3
Rand70-5	70	783	1612.50	917.15	83.8	44823	42	2257.7
Rand70-100	70	14194	28962.50	16403.60	85.0	43541	43	1877.1
Rand80-5	80	974	2127.50	1170.81	82.9	61670	41	6199.4
Rand80-100	80	17299	37773.00	20566.60	84.0	73112	50	7006.0
Rand100-5	100	1407	3345.50	1857.60	76.8	36667	19	18794.5
Rand100-100	100	24103	59888.00	32522.00	76.9	38793	20	18461.9

Table 6: Results for some random instances.

Considering the Rand instances, time consumption becomes even more excessive as the sizes of the instances increase. For both instances on $n = 100$ nodes, the CPA terminates after expiration of the time limit of 5 hours. The percentages of gaps closed are significant, but again not so large as in the previous instances. Furthermore, these percentages seem to deteriorate as instance sizes increase.

These results demonstrate that random problem instances are generally very challenging for our cutting-plane approach. There does not seem to be much hope of solving instances on more than 50 or 60 nodes with reasonable computational effort.

7 Concluding remarks

This paper introduces a new and fast separation heuristic for 2-partition inequalities for the clique partitioning problem. The new separator is implemented in a cutting-plane algorithm, and we use this algorithm to perform several computational experiments. We show that the z-values obtained by the use of the new separator dominate those that can be obtained by using known separators from the literature, including exact polynomial-time separators for other classes of inequalities.

The computational experiments also evaluate the ability to close the gaps between the LP z-values and the values of optimal clique partitions, and we see that very large parts of the gaps can be closed in most cases. Also, very often, the computational effort required to accomplish this seems to be reasonable. We believe that the ability to close large parts of the gaps will be well worth the effort in a branch-and-cut setting.

Future work will involve further development of a branch-and-cut algorithm that uses the new separator. This algorithm will also be able to handle more general versions of the clique partitioning problem, in particular node-capacitated graph partitioning problems, in which there are restrictions on the number of nodes in the subgraphs of feasible partitions, such as the problem studied in [27].

References

[1] G. Andreello, A. Caprara and M. Fischetti, *Embedding* $\{0, \frac{1}{2}\}$*-cuts in a branch-and-cut framework: A computational study*, INFORMS J. Comput. **19** (2007), 229–238.

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

[3] L. Brunetta, M. Conforti and G. Rinaldi, *A branch-and-cut algorithm for the equicut problem*, Math. Program. **78** (1997), 243–263.

[4] M.J. Brusco and H.-F. Köhn, *Clustering qualitative data based on binary equivalence relations: Neighborhood search heuristics for the clique partitioning problem*, Psychometrika **74** (2009), 685–703.

[5] A. Caprara and M. Fischetti, $\{0, \frac{1}{2}\}$*-Chvátal-Gomory cuts*, Math. Program. **74** (1996), 221–235.

[6] D. Catanzaro, E. Gourdin, M. Labbé and F.A. Özsoy, *A branch-and-cut algorithm for the partitioning-hub location-routing problem*, Comp. Oper. Res. **38** (2011), 539–549.

[7] H.M. Chan and D.A. Milner, *Direct clustering algorithm for group formation in cellular manufacture*, J. Manuf. Syst. **1** (1982), 65–74.

[8] I. Charon and O. Hudry, *Noising methods for a clique partitioning problem*, Discrete Appl. Math. **154** (2006), 754–769.

[9] S. Chopra and M.R. Rao, *The partition problem*, Math. Program. **59** (1993), 87–115.

[10] M. Deza, M. Grötschel and M. Laurent, *Clique-web facets for multicut polytopes*, Math. Oper. Res. **17** (1992), 981–1000.

[11] M.L. Fredman and R.E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms*, J. Ass. Comput. Mach. **34** (1987), 596–615.

[12] A.M.H. Gerards, *Testing the odd bicycle wheel inequalities for the bipartite subgraph polytope*, Math. Oper. Res. **10** (1985), 359–360.

[13] M. Grötschel and Y. Wakabayashi, *A cutting plane algorithm for a clustering problem*, Math. Program. **45** (1989), 59–96.

[14] M. Grötschel and Y. Wakabayashi, *Facets of the clique partitioning polytope*, Math. Program. **47** (1990), 367–387.

[15] B.W. Kernighan and S. Lin, *An efficient heuristic procedure for partitioning graphs*, Bell Syst. Tech. J. **49** (1970), 291–307.

[16] J.R. King, *Machine-component group formation in group technology*, Omega **8** (1980), 193–199.

[17] K.R. Kumar, A. Kusiak and A. Vanelli, *Grouping of parts and components in flexible manufacturing systems*, Eur. J. Oper. Res. **24** (1986), 387–397.

[18] A.N. Letchford and A. Vu, *Facets from gadgets*, Available online in Math. Program. (2019)

[19] E.M. Macambira and C.C. de Souza, *The edge-weighted clique problem: Valid inequalities, facets and polyhedral computations*, Eur. J. Oper. Res. **123** (2000), 346–371.

[20] W.T. McCormick, P.J. Schweitzer and T.W. White, *Problem decomposition and data reorganization by a clustering technique*, Oper. Res. **20** (1972), 993–1009.

[21] J. Miltenburg and W. Zhang, *A comparative evaluation of nine well-known algorithms for solving the cell formation problem in group technology*, J. Oper. Man. **10** (1991), 44–72.

[22] M. Oosten, J.H.G.C. Rutten and F.C.R. Spieksma, *The clique partitioning problem: Facets and patching facets*, Networks **38** (2001), 209–226.

[23] D. Recalde, D. Severín, R. Torres and P. Vaca, *An exact approach for the balanced k-way partitioning problem with weight constraints and its application to sports team realignment*, J. Comb. Optim. **36** (2018), 916–936.

[24] D.F. Rogers and S.S. Kulkarni, *Optimal bivariate clustering and a genetic algorithm with an application in cellular manufacturing*, Eur. J. Oper. Res. **160** (2005), 423–444.

[25] H. Seifoddini, *Comparison between single linkage and average linkage clustering techniques in forming machine cells*, Comput. Ind. Eng. **15** (1988), 210–216.

[26] R.Yu. Simanchev, I.V. Urazova and Yu.A. Kochetov, *The branch and cut method for the clique partitioning problem*, J. Appl. Indust. Math. **13** (2019), 539–556.

[27] M.M. Sørensen, A polyhedral approach to graph partitioning, Ph.D. thesis, Aarhus School of Business, 1995. Available at `www.ResearchGate.net (DOI: 10.13140/2.1.2451.1365)` or from the author.

[28] M.M. Sørensen, *A note on clique-web facets for multicut polytopes*, Math. Oper. Res. **27** (2002), 740–742.

[29] M.M. Sørensen, *New facets and a branch-and-cut algorithm for the weighted clique problem*, Eur. J. Oper. Res. **154** (2004), 57–70.

[30] D.R. Sule, *Machine capacity planning in group technology*, Int. J. Prod. Res. **29** (1991), 1909–1922.

[31] D.-W. Tcha, T.-J. Choi and Y.-S. Myung, *Location-area partition in a cellular radio network*, J. Oper. Res. Soc. **48** (1997), 1076–1081.

[32] Y. Wakabayashi, Aggregation of binary relations: Algorithmic and polyhedral investigations, Doctoral thesis, Universität Augsburg, 1986.