ISTITUTO DI ANALISI DEI SISTEMI ED INFORMATICA
"Antonio Ruberti"
CONSIGLIO NAZIONALE DELLE RICERCHE

T. Bonato, M. Jünger, G. Reinelt, G. Rinaldi

LIFTING AND SEPARATION PROCEDURES FOR THE CUT POLYTOPE

R. 14, 2011

Thorsten Bonato – Institut für Informatik, Universität Heidelberg, Germany, thorsten.bonato@informatik.uni-heidelberg.de..

Michael Jünger – Institut für Informatik, Universität zu Köln, Germany, mjuenger@informatik.uni-koeln.de..

Gerhard Reinelt – Institut für Informatik, Universität Heidelberg, Germany, gerhard.reinelt@informatik.uni-heidelberg.de..

Giovanni Rinaldi – Istituto di Analisi dei Sistemi ed Informatica "A. Ruberti" — CNR, Roma, Italy, rinaldi@iasi.cnr.it..

ISSN: 1128–3378

Collana dei Rapporti dell'Istituto di Analisi dei Sistemi ed Informatica "Antonio Ruberti", CNR

viale Manzoni 30, 00185 ROMA, Italy

tel. ++39-06-77161
fax ++39-06-7716461
email: iasi@iasi.cnr.it
URL: http://www.iasi.cnr.it

Abstract

The max-cut problem and the associated cut polytope on complete graphs have been extensively studied over the last 25 years. However, little research has been conducted for the cut polytope on arbitrary graphs. In this study we describe new separation and lifting procedures for the cut polytope on such graphs. These procedures exploit algorithmic and structural results known for the cut polytope on complete graphs to generate valid, and sometimes facet defining, inequalities for the cut polytope on arbitrary graphs in a cutting plane framework. We report computational results on a set of well-established benchmark problems.

Key words: Max-cut problem, cut polytope, separation, branch-and-cut.

1. Introduction

Let $G = (V, E)$ be an undirected graph with vertex set V and edge set E. For each, possibly empty, subset S of V, by $\delta(S)$ we denote the set of edges which have exactly one end in S. We call $\delta(S)$ a *cut* of G with the *shores* S and $V \setminus S$. For a single node s we denote the cut $\delta(\{s\})$ just by $\delta(s)$. For two subsets S and T of V, by $(S : T) := \delta(S) \cap \delta(T)$ we denote the set of edges which have one end in S and the other one in T. Suppose we have edge weights c_e for every edge $e \in E$. Then the *max-cut problem* consists of finding a node set $S \subseteq V$ such that $c(\delta(S)) := \sum_{e \in \delta(S)} c_e$ is as large as possible.

The *cut polytope* $\mathrm{CUT}(G)$ is the convex hull of the characteristic vectors $\chi^{\delta(S)}$ of all the cuts of G and is a subset of the $|E|$-dimensional real space $\mathbb{R}^E$. From a different viewpoint the max-cut problem can also be seen as the problem of finding a vertex of the cut polytope maximizing a given linear function.

The max-cut problem is NP-hard and is one of the most studied combinatorial optimization problems. It is of particular interest because it is the reformulation, in graph theoretical terms, of the well known *unconstrained 0/1 quadratic programming problem*, i.e., the problem of optimizing a quadratic objective function over the set of all 0/1 vectors of fixed dimension.

The max-cut problem has been used in a number of applications such as the optimal design of VLSI circuits or the study of minimum energy configurations of spin glasses (see, e.g., [4]). The latter application is one of the most investigated topics in Statistical Physics. Moreover, it is quite peculiar for an application of Combinatorial Optimization, as in some cases it demands for true optimal rather than suboptimal solutions (even if very good quality could be certified).

A common technique to solve max-cut problems to optimality is *branch-and-cut*. The two key components of a branch-and-cut algorithm are:

(i) a polyhedral *relaxation* of $\mathrm{CUT}(G)$, i.e., a set of linear inequalities describing a polytope P that contains $\mathrm{CUT}(G)$;

(ii) a set of separation procedures that, given a point z outside $\mathrm{CUT}(G)$, find one or more valid inequalities that separate z from P, and thus from $\mathrm{CUT}(G)$.

The effectiveness of the branch-and-cut algorithm strongly depends on how well P approximates $\mathrm{CUT}(G)$ and on how fast the separation procedures are.

An important and widely used polyhedral relaxation of $\mathrm{CUT}(G)$ is:

$$x(F) - x(C \setminus F) \leq |F| - 1 \quad \text{for all cycles } C \text{ of } G \text{ and all } F \subseteq C, |F| \text{ odd}, \tag{1}$$

$$x_e \leq 1 \quad \text{for all } e \in E, \tag{2}$$

$$x_e \geq 0 \quad \text{for all } e \in E, \tag{3}$$

where for an edge set S and corresponding variables, $x(S)$ denotes the sum $\sum_{e \in S} x_e$.

The linear system (1)–(3) describes the so-called *semimetric polytope,* which we denote by $\mathrm{MET}(G)$. The integral points of $\mathrm{MET}(G)$ are exactly the characteristic vectors of the cuts of G. The *cycle inequalities* (1) define facets of both $\mathrm{CUT}(G)$ and $\mathrm{MET}(G)$ if the cycle C is chordless. The *trivial inequalities* (2) and (3) define facets of $\mathrm{CUT}(G)$ and $\mathrm{MET}(G)$ if the edge e does not belong to a *triangle* (cycle of length 3) of G. So, in case of the *complete graph* K_p with p vertices, only triangles induce facets and each triangle generates four distinct inequalities that correspond to the four possible ways of choosing the edge set F.

The polytope $\mathrm{CUT}(K_p)$ has been extensively studied and a number of families of valid or even facet defining inequalities have been characterized (see, e.g., the surveys [12] and [21]). For some of these inequalities separation procedures have been proposed (see, e.g., [5], [6], [11], and [12]). Unfortunately, after the publication of [5], where the study of a description of $\mathrm{CUT}(G)$ by linear inequalities was initiated and where the inequalities (1) were introduced first along with a polynomial time separation algorithm, very little effort was devoted to the study of $\mathrm{CUT}(G)$ on arbitrary graphs. Nevertheless, branch-and-cut algorithms have been applied successfully to many

problems, in particular for the solution of spin glass problems on toroidal graphs. Solutions of instances with size up to 150×150 have been reported in [9] and [10]. For a survey on spin glass computations see [17]. The branch-and-cut algorithms employed were in principle only based on the inequalities (1), for which very effective separation procedures were designed. However, the cycle inequalities are far from being sufficient to solve spin glass instances on more complex graphs. Also some other max-cut instances coming from further real world applications are at present out of the reach of branch-and-cut algorithms.

Very interesting computational results have been obtained after the introduction of an alternative non-polyhedral relaxation of $\mathrm{CUT}(K_p)$. The characteristic vectors of the cuts of K_p are strongly related to certain symmetric positive semidefinite matrices. The set of these matrices is in a one-to-one correspondence with a convex body H_p that, after a suitable affine transformation, contains $\mathrm{CUT}(K_p)$. In addition, linear functions over H_p can be optimized efficiently using, e.g., interior point techniques (see [14]). Using this *semidefinite relaxation*, an approximation algorithm that delivers solutions with a guaranteed value of at least 0.87856 times the optimal value (for non-negative objective functions) could be constructed in [13]. Finally, by strengthening the relaxation H_p with linear inequalities, instances with up to 300 vertices could be solved to optimality [23]. But since semidefinite approaches are working with dense matrices, they cannot profit from the sparsity of the input graph and the number of nodes of a problem is crucial. It therefore seems that 300 nodes are about the limit for this approach. However, for dense graphs algorithms based on semidefinite relaxations at present outperform branch-and-cut algorithms by far. For recent surveys on semidefinite programming see [22] and the book [1].

It is the aim of this paper to show how the efficiency of polyhedral methods for the max-cut problem can be enhanced for sparse graphs. A central question is whether the current knowledge about $\mathrm{CUT}(K_p)$ can be used when solving the max-cut problem on an arbitrary graph G. One possibility is to add the missing edges to the graph G and to assign a zero weight to them. This technique has been successfully used for other combinatorial optimization problems where the sparse structure of the original graph can be exploited to handle the artificially completed graph efficiently. An example is the traveling salesman problem (see [2]), where even if the original graph is not sparse, all computations are carried out on a very sparse subgraph. To do so, it is assumed that the edges of a suitable large subset have no intersection with an optimal solution and thus their corresponding variables are permanently set to zero. A proof that this assumption is correct is eventually provided by the solution algorithm. To the contrary, for the max-cut problem there is no obvious way to exploit the sparsity of the original problem or to use a small working subset of the original edges. Notice that in a dense graph a cut can contain $\Theta(n^2)$ edges. This means that if one uses the above technique of completing the graph with edges of zero weight, the exact solution of the problem on sparse graphs has the same computational difficulties as on complete graphs.

The following simple observations make it clearer why there is such a fundamental difference between the traveling salesman and the max-cut problem, as far as the exploitation of the description of their polyhedra associated with complete graphs is concerned.

Consider G and $G \setminus e$, where $G \setminus e$ is obtained from G by removing edge e. All Hamiltonian cycles in G that do not contain e are also Hamiltonian cycles in $G \setminus e$. Hamiltonian cycles containing e will become infeasible when e is removed. Therefore the traveling salesman polytope $\mathrm{TSP}(G \setminus e)$ can be seen as being obtained simply by intersecting the polytope for G with the hyperplane $\{x : x_e = 0\}$. As a consequence, any valid inequality for $\mathrm{TSP}(G)$ is also valid for $\mathrm{TSP}(G \setminus e)$ and, by combining it with the equation $x_e = 0$, can be turned into an equivalent inequality obtained from the original one by dropping the term in x_e. Thus, a linear description of $\mathrm{TSP}(G \setminus e)$ is readily obtained from a linear description of $\mathrm{TSP}(G)$. In contrast with this, a cut of G reduced to the edges of $G \setminus e$ is a feasible cut of $G \setminus e$ in any case whether it contains e or not. So the relation between the two polytopes is more complicated: $\mathrm{CUT}(G \setminus e)$ is the result of projecting out variable x_e. In principle the linear system of the projection can be obtained via a Fourier-Motzkin procedure. But it would be extremely complex and it is very unlikely that a general criterion can be devised to describe it in a compact way. The implication for polyhedral computations is that valid inequalities for $\mathrm{CUT}(G)$ cannot be trivially used for $\mathrm{CUT}(G \setminus e)$. The only fortunate (and non trivial) case we are

aware was proved in [3]: The projection of the semimetric polytope of a graph G onto $\{x : x_e = 0\}$ is again the semimetric polytope of $G \setminus e$.

In this paper we want to exhibit a new possibility of how to exploit knowledge about the cut polytope for complete graphs for the solution of max-cut problems in sparse graphs. A brief outline of our approach is the following.

Suppose that we are given a point $z \in \mathbb{R}^E$ that satisfies all the inequalities (1)–(3) but does not belong to $\mathrm{CUT}(G)$ for some arbitrary graph $G = (V, E)$ with n nodes. The central task of a branch-and-cut algorithm now is to find an inequality valid for $\mathrm{CUT}(G)$ (possibly facet defining) that is not satisfied by z. First, by a sequence of operations z is transformed to a point $\overline{z} \in \mathbb{R}^{\overline{E}}$, where $\overline{E}$ is the edge set of a graph $\overline{G}$ defined on p nodes where p is usually much smaller than n. The transformation guarantees that $\overline{z}$ is outside $\mathrm{CUT}(\overline{G})$ but satisfies all inequalities (1)–(3). If $\overline{G} = K_p$, then $\overline{z}$ can be seen as a fractional solution of a cutting plane algorithm applied to a max-cut instance on the complete graph K_p. At this point all the polyhedral machinery available for the max-cut on complete graphs can be used. Thus, some separation procedures for the cut polytope on complete graphs are applied to $\overline{z}$ that (hopefully) generate an inequality $\overline{a}x \geq \overline{a}_0$, valid for $\mathrm{CUT}(K_p)$ and violated by $\overline{z}$. Finally, a sequence of lifting procedures is applied to $\overline{a}x \geq \overline{a}_0$ that transforms it to an inequality $ax \geq a_0$ valid for $\mathrm{CUT}(G)$ and violated by z. This way, we can exploit algorithmic and structural results available for $\mathrm{CUT}(K_p)$ also for solving the max-cut problem on sparse graphs.

The paper is organized as follows. In Sections 2 and 3 we introduce a graph contraction technique that, given a fractional LP solution, contracts all edges with a corresponding integral LP value. Moreover, we describe how to transform cutting planes for the associated contracted LP solution into cutting planes for the original LP solution. In Section 4 we propose a method for the artificial completion of a contracted graph in order to be able to apply results for the max-cut problem on complete graphs. We also explain how, if a cutting plane is found, the respective variables of the artificially added edges can be projected out. Section 5 discusses the separation procedures which we use in our algorithm. Finally, in Section 6 we report about computational results on well-established benchmark problems.

2. Contraction and lifting

Let x be a vector in $\mathbb{R}^E$ and let B be a subset of E. We define the auxiliary vector x^B with

$$x^B_e := \begin{cases} -x_e & \text{if } e \in B, \\ x_e & \text{otherwise.} \end{cases}$$

Consider the mapping $s_B \colon \mathbb{R}^E \to \mathbb{R}^E$ with $s_B(x) := x^B + \chi^B$. We refer to this mapping as *switching along* B. Let A be an arbitrary subset of E and let $A \Delta B := (A \cup B) \setminus (A \cap B)$ denote the *symmetric difference* of A and B. Then, it is easy to verify that $s_B(\chi^A) = \chi^{A \Delta B}$.

Any cut $\delta(S)$ of G is in particular a subset of E. Moreover, the set of cuts of G is closed under taking the symmetric difference. This means that we can use the switching along a cut to transform the characteristic vectors of any two cuts into one another. In other words, the switching mapping along a cut is an automorphism of the cut polytope $\mathrm{CUT}(G)$ as well as of the unit hypercube $[0, 1]^E$ (see, e.g., [12] for further details). In addition, if z is a point (vertex) of $\mathrm{MET}(G)$ then this property is preserved when switching z along a cut of G.

Similar to the switching of vectors, we can define a switching operation on inequalities. Consider an inequality $ax \leq \alpha$ and a cut $\delta(S)$. Let $a\big(\delta(S)\big)$ denote the sum $\sum_{e \in \delta(S)} a_e$. We say that the inequality

$$a^{\delta(S)} x \leq \alpha - a\big(\delta(S)\big)$$

is obtained by *switching* $ax \leq \alpha$ *along* $\delta(S)$. If $ax \leq \alpha$ is valid, or facet defining, for $\mathrm{CUT}(G)$ then this is also true for the switched inequality (see [5]). Moreover, if the inequality switched along $\delta(S)$ is tight at a point z, i.e., it is satisfied at equality by z, then the original inequality is tight at the switched point $s_{\delta(S)}(z)$. As a result, whenever we want to separate a point z from

the cut polytope, we can use $s_{\delta(S)}(z)$ instead. Namely, each inequality separating $s_{\delta(S)}(z)$ will separate the original point z, after having been switched along $\delta(S)$.

Suppose we are given a point z inside $\mathrm{MET}(G)$ but outside $\mathrm{CUT}(G)$ for which we want to find a separating inequality. We define the edge sets $E_0 := \{e \in E : z_e = 0\}$ and $E_1 := \{e \in E : z_e = 1\}$. Since z belongs to $\mathrm{MET}(G)$, there exists a cut $\delta(W)$ such that $E_1 \subseteq \delta(W)$ and $E_0 \cap \delta(W) = \emptyset$.

The switched point $\tilde{z} := s_{\delta(W)}(z)$ is also an element of $\mathrm{MET}(G)$. Moreover, due to the special properties of the cut $\delta(W)$ we have

$$0 \leq \tilde{z}_e < 1 \quad \text{for } e \in E. \tag{4}$$

Therefore, from now on we can assume, without loss of generality, that a point $\tilde{z}$, given as an input to a separation procedure, always satisfies (4). We can now apply the separation procedures to $\tilde{z}$ rather than to z. If $ax \leq \alpha$ is a violated inequality for $\tilde{z}$ then the procedure simply returns the inequality $a^{\delta(W)}x \leq \alpha - a(\delta(W))$ as a result of the separation.

Definition 2.1. Let $G = (V, E)$ be a graph with edge weights $z \in \mathbb{R}^E$ such that $z \in \mathrm{MET}(G)$ and let $e \in E^0 = \{f \in E : z_f = 0\}$. The graph $\overline{G} = (\overline{V}, \overline{E})$ and the vector $\overline{z} \in \mathbb{R}^{\overline{E}}$ are obtained from G and z by *contracting* edge $e = st$ if the following holds:

(*i*) $\overline{V} = V \setminus \{s, t\} \cup \{w\}$ and $\overline{E} = E \setminus \delta(s) \setminus \delta(t) \cup \{wu : u \in S \cup T \cup B\}$, where B is the set of common neighbors of s and t in G, and S and T are the neighbors of s and t, respectively, that are not contained in B.

(*ii*) $\overline{z}_{uv} = z_{uv}$, for $u, v \in V \setminus \{s, t\}$,
$\overline{z}_{uw} = z_{us}$, for $u \in S \cup B$,
$\overline{z}_{uw} = z_{ut}$, for $u \in T$.

Note again, that for nodes $u \in B$ we must have $z_{su} = z_{tu}$. If this is not the case and, say, $z_{su} > z_{tu}$, then the inequality $x_{su} - x_{tu} - x_{ts} \leq 0$ is violated, contradicting the assumption $z \in \mathrm{MET}(G)$. Thus, the contraction operation is well-defined.

Assume now that the edge weights z of a graph G satisfy (4) and that a sequence of applications of the edge contraction operation has been performed on G and z until such an operation is no longer applicable. Call $\overline{G}$ and $\overline{z}$ the resulting contracted graph and the associated edge weight vector, respectively.

Proposition 2.2. *The graph $\overline{G}$ and the vector $\overline{z}$ have the following properties:*

(*i*) *$\overline{z}$ has only fractional components,*

(*ii*) *$\overline{z}$ satisfies* (1)–(3),

(*iii*) *$\overline{z}$ is a vertex of* $\mathrm{MET}(\overline{G})$ *if and only if z is a vertex of* $\mathrm{MET}(G)$,

(*iv*) *every cut of $\overline{G}$ corresponds to a cut of G that is disjoint from E^0.*

Proof. For a proof of claim (iii), see [16]. All the other claims are easy to verify. ∎

Of course, $\overline{G}$ will have at most as many nodes or edges as G. An example of the combined switching and contraction procedure is illustrated in Figure 1 which shows the original graph G in (a), the graph $\tilde{G}$ that results from switching along the cut $\delta\big(\{c, i\}\big)$ in (b), and the contracted graph $\overline{G}$ in (c).

Suppose now that a valid inequality for $\mathrm{CUT}(\overline{G})$ has been found that is violated by $\overline{z}$. We want to lift this inequality to one valid for $\mathrm{CUT}(G)$ that is violated by z. To do so, we must reverse the contraction operation described before. To this end we introduce the following *node-splitting* operation.

Definition 2.3. Given a graph $G = (V, E)$, an inequality $ax \leq \alpha$ in $\mathbb{R}^E$ valid for $\mathrm{CUT}(G)$, a node $w \in V$ and a partition (S, T, B) of the neighbors of w. The graph $G' = (V', E')$ and the inequality $a'x' \leq \alpha$ in $\mathbb{R}^{E'}$ are obtained from G and a, by *splitting* node w into s and t with respect to the partition (S, T, B), if the following holds:

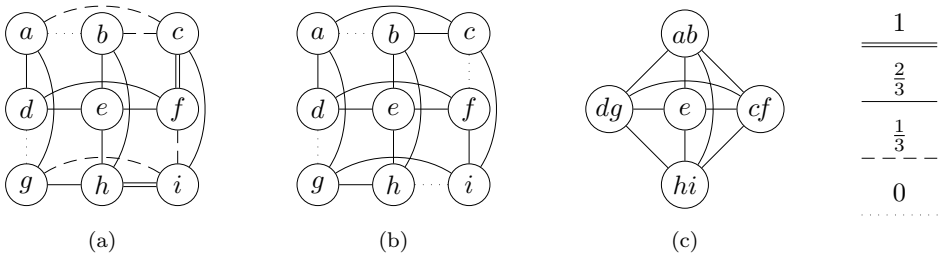


Figure 1: Switching and contraction of a graph.

(*i*) $V' = V \setminus w \cup \{s,t\}$ and $E' = E \setminus \delta(w) \cup (s : S \cup B) \cup (t : T \cup B) \cup \{st\}$;

(*ii*) the components of a' are given by

$$\begin{aligned} a'_{st} &= -\sum_{v \in T} |a_{wv}|, \\ a'_{tv} &= 0 \qquad \text{for all } v \in B, \\ a'_{tv} &= a_{wv} \quad \text{for all } v \in T, \\ a'_{sv} &= a_{wv} \quad \text{for all } v \in S \cup B, \\ a'_{uv} &= a_{uv} \quad \text{for all } uv \in E' \setminus \big(\delta(s) \cup \delta(t)\big), \end{aligned}$$

where we assume, without loss of generality, that $\sum_{v \in T} |a_{wv}| \leq \sum_{v \in S} |a_{wv}|$.

Theorem 2.4. *If $ax \leq \alpha$ is valid for* CUT(G) *and if G' and $a'x' \leq \alpha$ are obtained from G and $ax \leq \alpha$ by splitting node w into s and t, with respect to the partition (S,T,B), then the inequality $a'x' \leq \alpha$ is valid for* CUT(G').

Proof. Clearly, we have $a'\big(\delta(W)\big) = a\big(\delta(W)\big)$ for any cut $\delta(W)$ which does not contain edge st. So, assume that st is part of $\delta(W)$. Without loss of generality, let $s \in W$. Suppose that $a'\big(\delta(W)\big) > \alpha$. We define the extended node set $W' = W \cup \{t\}$. For the corresponding induced cut $\delta(W')$

$$\begin{aligned} a'\big(\delta(W')\big) &= a'\big(\delta(W)\big) - a'_{st} + \sum_{v \in T \setminus W} a'_{tv} - \sum_{v \in T \cap W} a'_{tv} \\ &\geq a'\big(\delta(W)\big) - a'_{st} - \sum_{v \in T \setminus W} |a'_{tv}| - \sum_{v \in T \cap W} |a'_{tv}| \\ &\geq a'\big(\delta(W)\big) > \alpha. \end{aligned}$$

But $\delta(W')$ does not contain st, therefore we get a contradiction. ∎

An operation similar to the one of Definition 2.3 has been described in [5], and it is also called *node splitting*. The coefficients of the inequality resulting from node splitting in [5] are the same as in Definition 2.3, except for the coefficient of edge st which is given as $a'_{st} = -\sum_{v \in T} a_{wv}$. The two operations, although very similar, have been introduced to accomplish quite different tasks.

The node splitting of [5] is a tool to generate new facet defining inequalities of CUT(G+e), where G+$e := (V, E \cup \{e\})$ and e is the edge st, starting from a known facet defining inequality of CUT(G). In other words, it performs what is also called the *one variable lifting* of an inequality while preserving the property of being facet defining. The validity of the produced inequality depends of the choice of the partition (S,T,B). Under some conditions for this partition, it is proved in [5] that the generated inequality is not only valid, but also facet defining as claimed in the following theorem restated using the conventions of Definition 2.3.

Theorem 2.5 (Theorem 2.6 (a) in [5]) *Let $G = (V, E)$ be a graph and $ax \leq \alpha$ be an inequality facet defining for* CUT(G)*. Let G' and a' be obtained from G and a by* splitting *node $w \in V$ with respect to a partition (S, T, B) of the neighbors of w in G. Then the inequality $a'x' \leq \alpha$ is facet defining for* CUT(G') *if $B = \emptyset$ and w belongs to a set $W \subseteq V$ such that $a\chi^{\delta(W)} = \alpha$ and T is a nonempty subset of $\{u \in V \setminus w : uw \in E \text{ and } a_{uw} > 0\} \cap W$.*

For the sake of completeness we have to say that in the original Theorem 2.6 in [5] the graph G' can also contain any edge tu with $a'_{tu} = 0$ for $u \in V$, provided that $a_{zu} = 0$ for each edge zu incident in u. In other words, node u does not belong to the *support graph* of vector a, which is the subgraph of G whose edges have a corresponding nonzero component in a.

In conclusion, Theorem 2.5 is a tool to prove that some inequalities are facet defining for CUT(G). For example, in [5] it is used to prove that the general cycle inequalities (1) have this property, using the fact that the triangle inequalities are facet defining for CUT(K_3).

Quite differently, the purpose of the node splitting of Definition 2.3 is to produce an inequality valid for CUT(G') no matter how the sets of the partition (S, T, B) are chosen. As we will show in the following, edge contraction and node splitting are reverse operations. We use node splitting to reconstruct the graph modified by edge contraction. The sets of the partition contain the necessary information for doing so.

One may wonder whether there are conditions for the inequality $a'x' \leq \alpha$ of Definition 2.3 to be facet defining. While Theorem 2.5 can only seldomly serve this purpose, we can use the following generalization instead that applies also to the case when B is not empty.

Theorem 2.6. *Let $G = (V, E)$ be a graph and $ax \leq \alpha$ be a facet defining inequality for* CUT(G)*. Let G' and a' be obtained from G and a by* splitting *node $w \in V$ with respect to a partition (S, T, B) of the neighbors of w in G. Then the inequality $a'x' \leq \alpha$ is facet defining for* CUT(G') *if w belongs to a set $W \subseteq V$ such that $a\chi^{\delta(W)} = \alpha$ and:*

(i) $a_{wv} \geq 0$ *for all* $v \in T \cap W$,

(ii) $a_{wv} \leq 0$ *for all* $v \in T \setminus W$,

(iii) $a(v : W) = a(v : V \setminus W)$ *for all* $v \in B$.

Proof. Assume $f'x' \leq f_0$ is facet defining for CUT(G') such that $\{x' \in \text{CUT}(G') : f'x' = f_0\} \supseteq \{x' \in \text{CUT}(G') : a'x' = \alpha\}$. Proving the theorem amounts to showing that there exists a scalar $\lambda > 0$ such that $f' = \lambda a'$.

Based on the tight cuts of $ax \leq \alpha$ which correspond to tight cuts of $a'x' \leq \alpha$, each of them induced by a node set Y with $\{s, t\} \subseteq Y$ or $\{s, t\} \subseteq V' \setminus Y$, we can already assume that there exists a $\lambda > 0$ such that $f'_e = \lambda a'_e$, for all $e \in E' \setminus (\{\{s, t\} : B) \cup \{st\})$, and that for all $u \in B$ we have $f'_{us} + f'_{ut} = \lambda a'_{us}$, because each of these cuts either contains both edges us and ut or none of them.

By (i) and (ii), the cut $\delta(W')$, where $W' = W \setminus \{z\} \cup \{s\}$, is tight for $a'x' \leq \alpha$ because

$$\begin{aligned} a'(\delta(W')) &= a(\delta(W)) + a'_{st} + a(z : T \cap W) - a(z : T \setminus W) \\ &= a(\delta(W)) + a'_{st} + \sum_{v \in T} |a_{zv}| \\ &= a(\delta(W)) = \alpha. \end{aligned}$$

Let $u \in B \cap W$. Then (iii) implies that also the cut $\delta(W \setminus \{u\})$ is tight for $ax \leq \alpha$ and, because $a'_{ut} = 0$, also $\delta(W' \setminus \{u\})$ is tight for $a'x' \leq \alpha$. Thus, we obtain

$$\begin{aligned} 0 &= f'(\delta(W')) - f'(\delta(W' \setminus \{u\})) \\ &= f'_{ut} + f'(u : V' \setminus W') - f'(u : W') = f'_{ut}, \end{aligned}$$

since

$$\begin{aligned} f'(u : V' \setminus W) - f'(u : W') &= \lambda(a'(u : V' \setminus W') - a'(u : W')) \\ &= \lambda(a(u : V \setminus W) - a(u : W)) = 0. \end{aligned}$$

If $u \in B \setminus W$, then we obtain $f'_{ut} = 0$ in an analogous way. Therefore, for all $u \in B$ we have $f'_{us} = \lambda a'_{us}$ and $f'_{ut} = 0 = \lambda a'_{ut}$. Finally,

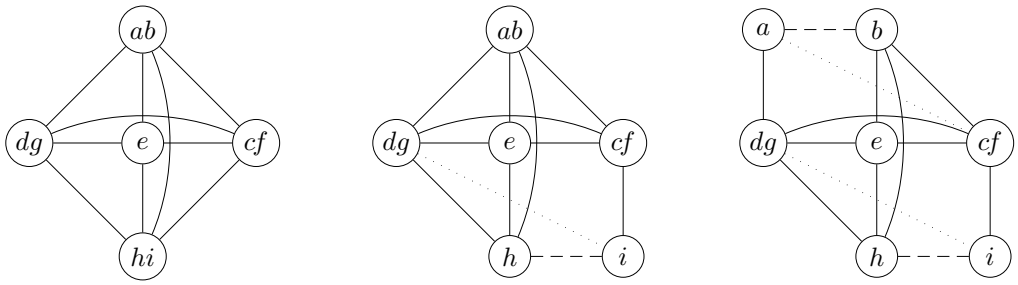
$$\begin{aligned} 0 &= f'(\delta(W')) - f'(\delta(W' \cup \{t\})) \\ &= f'_{st} + f'(t : W' \setminus \{s\}) - f'(t : V' \setminus W') = f'_{st} - \lambda a'_{st} \end{aligned}$$

and we have shown that $f' = \lambda a'$. ∎

Assume now that the inequality $a'x' \leq \alpha$ is derived from an inequality violated by a point $\overline{z}$ which is obtained by contraction of a point z. We wonder if also $a'x' \leq \alpha$ is violated by z. The following theorem, whose proof comes directly from the definitions 2.1 and 2.3, provides an answer.

Theorem 2.7. *Let $G = (V, E)$ be a graph with weights $z \in \mathbb{R}^E$ on its edges such that $z \in \mathrm{MET}(G)$ and let $e \in E$ such that $z_e = 0$. Let $\overline{G} = (\overline{V}, \overline{E})$ be the graph and $\overline{z} \in \mathbb{R}^{\overline{E}}$ be the vector obtained from G and z by* contracting *edge $e = st$. Let the node w and the sets S, T, $B \subseteq V$ be as in Definition 2.1. Let $\overline{a}\,\overline{x} \leq \alpha$ be an inequality valid for $\mathrm{CUT}(\overline{G})$ such that $\overline{a}\,\overline{z} = \alpha + \eta$, with $\eta > 0$. Then the inequality $ax \leq \alpha$ in $\mathbb{R}^E$, obtained from $\overline{G}$ and $\overline{a}$ by* splitting *node w into s and t, with respect to the partition (S, T, B) is valid for G and is such that $az = \alpha + \eta$.*

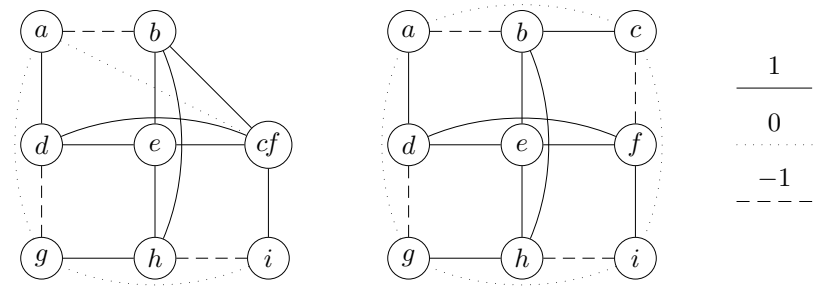
An example of a series of node-splitting operations is shown in Figure 2. In Figure 2(a) the



(a) $w = hi, B = \{dg\}$, $S = \{ab, e\}, T = \{cf\}$, $W = \{cf, hi\}$.

(b) $w = ab, B = \{cf\}$, $S = \{e, h\}, T = \{dg\}$, $W = \{ab, dg\}$.

(c) $w = dg, B = \{a\}$, $S = \{e, cf\}, T = \{h, i\}$, $W = \{b, dg, h, i\}$.



(d) $w = cf, B = \{i\}$, $S = \{d, e\}, T = \{a, b\}$, $W = \{a, b, cf, h\}$.

(e)

Figure 2: Sequential node-splitting.

weighted support graph of a clique inequality is shown that is violated by the fractional point

defined in Figure 1(c). It is assumed that $\{cf, dg, ab, hi\}$ is the order of the edges on which the contraction has been applied to obtain the weighted graph of Figure 1(c), starting from the one of Figure 1(b). The inequality obtained at the end of the sequence of node-splitting operations is shown in Figure 2(e); it violates the fractional point shown in Figure 1(b). Moreover, it is facet defining as one can verify by applying Theorem 2.6 at each execution of the node-splitting operation with the set W defined for each of the figures 2(a)-2(d). A facet defining inequality that violates the original fractional point shown in Figure 1(a) is now easily derived by switching along the cut $\delta(c, i)$.

3. Induced subgraphs and zero lifting

The scheme described in the previous section makes it possible to translate the separation procedure for CUT(G), where G is a graph with n nodes, to the the separation procedure for CUT($\overline{G}$), where $\overline{G}$ is a graph with $\overline{n} < n$ nodes. Such a scheme has the following nice features:

(i) if $z \notin$ CUT(G), then the sequence of contractions of Definition 2.1 produces a point $\overline{z} \notin$ CUT($\overline{G}$);

(ii) if an inequality $\overline{a}\overline{x} \geq \overline{a}_0$, valid for CUT($\overline{G}$) and violated by $\overline{z}$ is identified by a separation algorithm, then it is always possible to lift it to an inequality $ax \geq a_0$ valid for CUT(G) and violated by z;

(iii) the amounts of violation of $\overline{a}\overline{x} \geq \overline{a}_0$ and of $ax \geq a_0$, evaluated in $\overline{z}$ and in z, respectively, coincide.

A possible disadvantage of this scheme is that the size of the contracted graph $\overline{G}$ cannot be kept under control and only depends on the integral components pattern of vector z. Therefore, $\overline{G}$ can be too large for some separation procedures to be applied, like, e.g., the one based on target cuts (see next section).

Another scheme that overcomes this difficulty is the one based on a very simple observation. Let $\overline{G} = (\overline{V}, \overline{E})$ be the subgraph of $G = (V, E)$ induced by $\overline{V} \subset V$, and $z_{\overline{E}}$ be the restriction of z to the components indexed by the edges in $\overline{E}$. If K is a cut of G, then $K \cap \overline{E}$ is a cut of $\overline{G}$. Consequently, if $\overline{z} \notin$ CUT($\overline{G}$), then $z \notin$ CUT(G). Unfortunately, the reverse implication does not always hold true, as we always assume that $z \in$ MET(G) and hence that $\overline{z} \in$ MET($\overline{G}$) for every induced subgraph $\overline{G}$ of G. Take, for example, the case when $\overline{G}$ is planar (or even simply not contractible to K_5). Then CUT($\overline{G}$) = MET($\overline{G}$) and so $\overline{z} \in$ CUT($\overline{G}$).

Once a suitable induced subgraph $\overline{G}$ has been identified and an inequality $\overline{a}\overline{x} \geq \overline{a}_0$ valid for CUT($\overline{G}$) and violated by $\overline{z}$ has been found, it is very easy to lift $\overline{a}\overline{x} \geq \overline{a}_0$ to a valid inequality of CUT(G) violated by z. It is sufficient to apply the so called *zero lifting*, i.e., to assign a zero coefficient to all components of z corresponding to the elements of $E \setminus \overline{E}$ and the value of the coefficient of the corresponding component of $\overline{z}$ to the others, while keeping the same right hand side. Moreover, the amount of violation of the two inequalities, evaluated at the corresponding fractional points, is the same.

This scheme shares the same features mentioned at the beginning of this section, except the first one, with the scheme of the previous section. But it has the advantage that the node size of the graph $\overline{G}$ can be fixed to a sufficiently small number.

The question whether an inequality obtained by zero lifting preserves the property of being facet defining has been investigated in [8], where a sufficient condition is provided. If the graph G has a node u adjacent to any other node and if the induced subgraph $\overline{G}$ contains u, then such a sufficient condition is always satisfied and the zero lifting of any facet defining inequality for CUT($\overline{G}$) defines also a facet of CUT(G).

4. Extension and projection

Suppose we are given a contracted graph $G = (V, E)$ with n nodes. Let z be a point inside $\mathrm{MET}(G)$ but outside $\mathrm{CUT}(G)$ which is the situation if G was obtained by contracting a larger graph with respect to some LP solution.

If we want to apply separation routines for complete graphs, we first have to extend z by introducing artificial values for possibly missing edges. Moreover, the extended point should be still an element of $\mathrm{MET}(K_n)$ but not of $\mathrm{CUT}(K_n)$. We first describe the extension for a single edge.

Let $e = uv$ be a *non-edge* of G, i.e., u and v are nodes in V which are not adjacent in G. Let $G{+}e = (V, E \cup \{e\})$ denote the extended graph. The following three problems have to be addressed:

(*i*) extend z to a point $(z, z_e) \in \mathbb{R}^{E \cup \{e\}}$ inside $\mathrm{MET}(G{+}e)$,

(*ii*) find an inequality $bx \le \beta$ with $b \in \mathbb{R}^{E \cup \{e\}}$ that separates (z, z_e) from $\mathrm{CUT}(G{+}e)$,

(*iii*) transform $bx \le \beta$ into an inequality $ax \le \alpha$ with $a \in \mathbb{R}^E$ that separates z from $\mathrm{CUT}(G)$.

The possible extension values z_e are in the set $\{z_e : (z, z_e) \in \mathrm{MET}(G{+}e)\}$. This set is nonempty because z is in $\mathrm{MET}(G)$ which is a projection of $\mathrm{MET}(G{+}e)$ as noted above. In fact, the above set is the interval $[l_e, u_e]$ with the bounds $l_e := \max\{0, L_e\}$ and $u_e := \min\{U_e, 1\}$, where

$$L_e := \max\{\, z(F) - z(P \setminus F) - |F| + 1 : P \text{ is a } (u,v)\text{-path of } G,\ F \subseteq P,\ |F| \text{ odd}\,\}, \tag{5}$$

$$U_e := \min\{\, -z(F) + z(P \setminus F) + |F| : P \text{ is a } (u,v)\text{-path of } G,\ F \subseteq P,\ |F| \text{ even}\,\}. \tag{6}$$

Notice that the additional restriction to the interval $[0, 1]$ is indeed necessary. This is because a point (z, z_e) with $z_e \in [L_e, U_e]$ is only guaranteed to satisfy all cycle inequalities of the extended graph. However, the linear description of the semimetric polytope also comprises the trivial inequalities. Consider for example the graph of Figure 3. For the point $z = (0.5, 0.5, 0.5)$ we obtain the bounds $L_e = -0.5$ and $U_e = 1.5$. Thus, extending z with, e.g., L_e or U_e would result in a point outside $\mathrm{MET}(G{+}e)$.

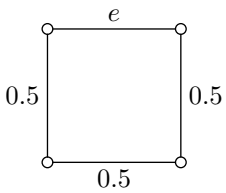


Figure 3: A graph for which $L_e < 0$ and $U_e > 1$.

We call a cycle or a trivial inequality a *lower inequality for* e if it is tight at (z, l_e). More precisely, the trivial inequality $-x_e \le 0$ is a lower inequality for e if $L_e \le 0$ and thus $l_e = 0$. Otherwise, a lower inequality is given by the cycle inequality defining the value L in (5). Analogously, we call an inequality that is tight at (z, u_e) an *upper inequality for* e. The trivial inequality $x_e \le 1$ is an upper inequality for e if $U_e \ge 1$, i.e., if $u_e = 1$. Otherwise, the cycle inequality defining U in (6) can serve as an upper inequality. We always assume the cycle inequalities to be given in the form $x(F) - x(C \setminus F) \le |F| - 1$. Therefore, the coefficient of the artificial variable x_e is -1 in a lower inequality and $+1$ in an upper inequality, respectively.

If the bounds l_e and u_e coincide, i.e., if the extension value z_e is uniquely defined, we say that e is *rigid*. It is easy to prove the following characterization of rigid edges:

Proposition 4.1. *A non-edge e is rigid if and only if at least one of the following two conditions holds:*

(*i*) *e is a chord of a* tight *cycle, i.e., a cycle that supports a cycle inequality that is tight at z;*

(*ii*) *the end nodes of e are connected by a path whose edges correspond to integral components of z.*

Notice that for every $l_e \leq z_e \leq u_e$ the extended point (z, z_e) is not contained in CUT(G+e) because z lies outside CUT(G).

Suppose now that we have computed an inequality $ax + a_e x_e \leq \alpha$ separating (z, z_e) from CUT(G+e) and let $az + a_e z_e = \alpha + \eta$.

If a_e is zero, then truncating the left hand side (a, a_e) to a results in the inequality $ax \leq \alpha$, valid for CUT(G) and violated by z by the same amount η.

Now assume that the coefficient a_e is positive. We can derive a valid inequality for CUT(G) by projecting out the variable x_e. To this end we need a valid inequality $bx - b_e x_e \leq \beta$ for CUT(G+e) such that b_e is a positive number. Let $bz - b_e z_e = \beta + \theta$. The projected inequality is

$$(a + \frac{a_e}{b_e} b)x \leq \alpha + \frac{a_e}{b_e}\beta \tag{7}$$

and its value at z is

$$(a + \frac{a_e}{b_e} b)x = \alpha + \eta + \frac{a_e}{b_e}(\beta + \theta).$$

Because we want (7) to be violated by z, then θ must be as big as possible.

Instead of taking an arbitrary valid inequality $bx - b_e x_e \leq \beta$ for CUT(G+e), we restrict ourselves to cycle or to trivial inequalities. We need a cycle or a trivial inequality with a negative coefficient for edge e and for which θ is as big as possible. In particular, we take the trivial inequality $z_e \geq 0$ if $L_e < 0$ or the inequality identified to compute L_e in (5), otherwise. Now we have $\theta = l_e - z_e$ and $b_e = -1$. Setting $\eta = \eta_0 + a_e(u_e - z_e)$, we get that the projected inequality is violated by

$$\eta - a_e\theta = \eta_0 - a_e(u_e - l_e).$$

Consequently, the violation of the projected inequality does not depend on which value we choose for z_e in the interval $[l_e, u_e]$.

The case when a_e is negative can be treated analogously and gives the same result.

If e is rigid, the violation of the inequality in G+e is preserved. Otherwise, one can choose any value in the interval $[l_e, u_e]$ for z_e. In any case there will be a decrease in the amount of violation, unless $a_e = 0$.

So, if the support of a separating hyperplane only contains original and rigid edges, then there will be no loss in violation.

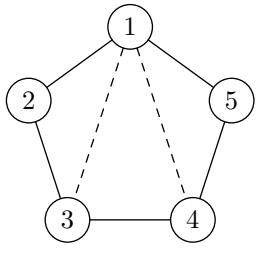


Figure 4: Example for dependency of interval bounds

We observe that the intervals $[l_e, u_e]$ cannot be computed independently, because a choice of a value for z_e may affect the bounds of the interval for all the other non-edges. Figure 4 illustrates this fact. Consider the 5-node cycle and assume that the variables corresponding to the edges of the cycle have all value $\frac{2}{3}$. The interval for both edges from 1 to 3 and 4 is $[0, \frac{2}{3}]$ if the computation is done separately with respect to the cycle. However, if we set $z_{14} = 0$, then edge 13 becomes rigid at value $\frac{2}{3}$.

We discuss an example that demonstrates the potential of our new approach to separation. Figure 5 shows a fractional solution of a (5×5) grid problem with exterior field discussed in [4].

For clarity, node 0 representing this field is not shown. Edges connecting a grid node to node 0 appear as short lines pointing downward to the right. The grid is assumed to be embedded on a torus. So the edges leaving the bottom nodes in a column are the connection between bottom and top nodes of the same column. Analogously, if the first and the last nodes of the same row are connected, then the respective edge is emanating from the last node of a row to the right.

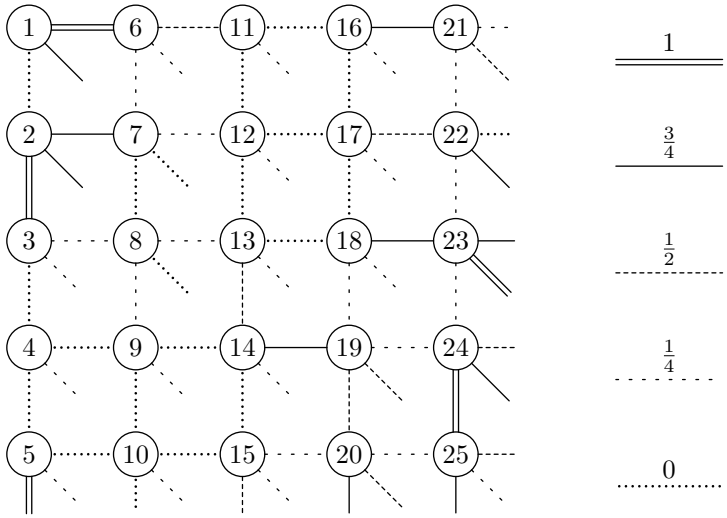


Figure 5: The fractional solution for a spin glass problem of [4].

The authors of [4] write that they were not able to produce a separating inequality for this solution. If we apply our separation approach then we obtain, as a shrunk graph, the left graph of Figure 6. Addition of the three missing edges yields the graph on the right hand side.

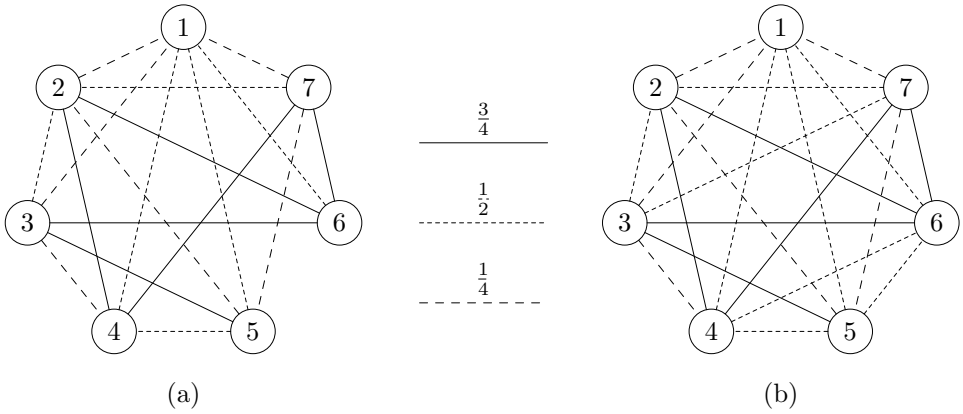


Figure 6: Shrunk and extended graph

It is easy to verify that in this graph there is a violated clique inequality on the nodes $\{1, 2, 3, 6, 7\}$ (and there are some further clique inequalities on five nodes). Moreover, by switching along the cut defined by the nodes 1 and 5, one can see that there is a violated clique inequality on all seven nodes. After lifting and switching we obtain the two inequalities shown in Figure 7. In this figure solid edges correspond to positive and dashed edges to negative coefficients. The absolute values of the coefficients are given explicitly only if they are not equal to 1.

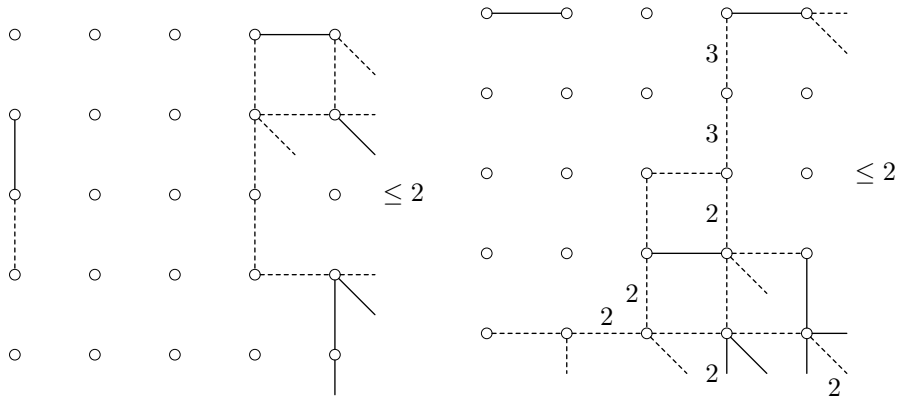


Figure 7: Inequalities violated by the fractional solution of [4].

5. Separation

We now elaborate on two separation procedures that can be used in conjunction with the concepts introduced in the sections 3 and 4.

First, we consider a class of inequalities known as *target cuts* [7]. The concept of target cuts per se is problem-independent. The respective inequalities do not conform to the so-called template paradigm (see [2], Section 5.6); this means that the inequalities in this class do not share a common support structure. A key feature of the target cut approach is that facet defining inequalities are generated. Below, we focus on the separation of target cuts for the cut polytope. For more general information, we refer to [7].

Suppose we are given a point x^* outside $\mathrm{CUT}(G)$. To derive a separating target cut for this point, we basically have to solve a linear program which comprises one row for each vertex of the polytope. For technical reasons (see [7]) we also need an interior point q of $\mathrm{CUT}(G)$. This could be, e.g., the barycenter of its vertices. The existence of q is guaranteed since the cut polytope is full-dimensional. Now, the target cut LP to be solved looks as follows:

$$\max\{a(x^* - q) : a(x - q) \leq 1, \text{ for all vertices } x \text{ of } \mathrm{CUT}(G)\}. \tag{8}$$

If the above LP has an optimum value greater than 1 then the corresponding optimum solution a^* yields the desired separating target cut $a^*x \leq 1$ for x^*.

Of course, enumerating all the vertices of the cut polytope and solving the resulting LP is only viable for small to moderately sized graphs. This restriction can be partly overcome by means of so-called *delayed row generation* which starts with a partial target cut LP and then adds rows as required in the course of the optimization. To do so, one needs an oracle for maximizing a linear function over the cut polytope. This technique is referred to as delayed column generation in [7], because the description of the method is based on the dual of (8). The authors state that the number of rows generated in the course of the above procedure is generally much smaller than the total number of vertices of the polytope. However, whether or not delayed row generation will improve the performance ultimately depends on the efficiency of the oracle.

Since target cuts neither require the graph to be complete nor the existence of specific support structures, they can be separated directly on a contracted LP solution. To obtain reasonable running times, though, target cuts are best applied to small induced subgraphs like the ones described in Section 3. The resulting separating inequalities can then be zero-lifted.

For the computational tests described in the next section we found that the most effective stategy was not to use delayed row generation and to keep the size of induced subgraphs below 12 nodes.

The second separation procedure generates clique inequalities. For a clique (W, F) of order p,

where $p \geq 3$, the associated *clique-* or K_p*-inequality* is:

$$x(F) \leq \left\lceil \frac{p}{2} \right\rceil \left\lfloor \frac{p}{2} \right\rfloor .$$

It defines a facet of the cut polytope if and only if p is odd. Any clique inequality remains valid even if we only consider a subset of its edge set F. It is clear that one has to identify dense subgraphs in order to have a chance for finding a violated clique inequality. Therefore, this class of inequalities is particularly interesting in the context of the extension of LP solutions described in Section 4.

A simple heuristic for the clique separation works as follows. We successively consider each node in the graph, possibly in random order, as initial node v to build a clique of given order p. First, we generate the list of all neighbors of v. If there are less than $p-1$ neighbors, we start over with the next initial node since we cannot obtain a clique with the desired size. Otherwise, suppose there are more than $p-1$ neighbors, in which case we have to choose the clique's nodes subject to maximizing the left hand side value of the associated clique inequality. Initially, we choose the first $p-1$ nodes from the list of neighbors. We then calculate for each neighbor the aggregate LP value of all the incident edges which lead to clique nodes. Furthermore, we mark the clique node with minimum aggregate LP value respectively the non-clique node with maximum such value. Now, we check whether we can increase the left hand side value of the clique inequality by replacing the former node with the latter one. If so, we perform this swap, recalculate the aggregate LP values, and mark the new minimum and maximum neighbor, respectively. This procedure is repeated until the first designated swap occurs that would actually decrease the left hand side value of the clique inequality. At this point, we generate the associated inequality for the current clique and start over with the next initial node v.

The heuristic just described only considers clique inequalities with all positive left hand side coefficients, while the switching of these inequality along any cut of the graph produces clique inequalities with positive and negative left hand side coefficients that would be useful to separate. However, the restriction to positive coefficient in the separation algorithm can be overcome by switching the LP solution x^* along a suitable cut prior to the separation and then switching the obtained separating inequalities accordingly at the end. Appropriate cuts for this purpose can be obtained, e.g., from the best known feasible solution or by maximizing the weight function $0.5 - x^*_e$ for all $e \in E$. Finally, to obtain reasonable running times, we can restrict the above heuristic to search for violated cliques of order 5 and 7 only.

6. Computational results

6.1. Test instances

As a basis for our computational experiments we took the following classes of well-established test instances for both the max-cut problem and unconstrained quadratic 0/1 optimization.

Ising spin glass problems.. We considered 2- and 3-dimensional Ising spin glass problems with nearest neighbor interactions and periodic boundary conditions without external magnetic field. The respective underlying interaction graphs are toroidal grids, which are square grids with additional edges that join the outermost nodes of each row and column, respectively. Similarly, a 3-dimensional toroidal grid is a cubic grid with additional edges such that each horizontal and vertical layer, respectively, is a 2-dimensional toroidal grid.

We generated toroidal grids with either uniformly distributed ± 1 weights or Gaussian distributed integral weights. For the ± 1 weighted instances we initialized the first and the second half of the edge weights with -1 and $+1$, respectively. Then, we computed a random permutation of the edge weights. We considered toroidal $(k \times k)$ grids with $k = 30, 35, \ldots, 85$ and 3-dimensional toroidal grids with $k = 5, \ldots, 8$, generating 10 random instances for each grid size.

For the instances with Gaussian distributed weights we initialized each edge weight with the value of a standard normal random variable generated using the *Marsaglia polar method* [20]. The

initial values were multiplied by 10^5 and then rounded to the closest integer. Finally, a random permutation was applied as well. We considered toroidal $(k \times k)$ grids with $k = 30, 35, \ldots, 185$ and 3-dimensional toroidal grids with $k = 5, \ldots, 11$, again generating 10 random instances per grid size.

BiqMac library.. We also tested our algorithm on instances from the *BiqMac Library* [25]. We considered the three classes of quadratic 0/1 optimization problems, namely the *Beasley instances*, the *Billionnet and Elloumi instances*, and the *Glover, Kochenberger, and Alidaee instances*, as well as several max-cut problem instances whose underlying graphs were generated using the graph generator *rudy* [24]. For the details on sizes, optimum values, lower/upper bounds and origin of these instances, we refer to [25].

Frequency assignment instances.. These instances were generated from real-world data on radio frequency interferences between major Italian cities in the context of a frequency assignment problem. The instances have been made available by C. Mannino [19]. Table 1 gives further details. Using a custom generator, we also created a set of 91 additional instances with characteristics

Name	#Nodes	#Edges	Density	Range of weights	Opt
`man_k48`	48	1 128	1.00	[13146, 841699]	252 518 838
`man_k487a`	487	1 435	0.01	[101, 33631]	1 110 926
`man_k487b`	487	5 391	0.05	[5, 176030]	3 655 475
`man_k487c`	487	8 511	0.07	[5, 203785]	8 640 860

Table 1: Properties of frequency assignment instances.

similar to those of the original frequency assignment instances.

6.2. Computational setup and algorithmic strategies

We implemented the algorithm in C++ and embedded it in the branch-and-cut framework ABACUS [15]. The computational experiments were carried out on an Intel Xeon E5450 processor with 2.5 GHz clock rate, 2×6 MB shared L2-Cache and 8 GB RAM. The operating system was Debian GNU/Linux 4.0. We used ABACUS 2.3 in conjunction with the LP solver CPLEX 8.1.

In addition to the shrink separation, our branch-and-cut solver comprised four separation procedures for cycle inequalities, namely GEN3CYC (3-cycle enumeration), GEN4CYC (4-cycle enumeration), SHOC (spanning tree heuristic), and OC (exact cycle separation). The latter three are identical to the procedures of the same name described in [4].

As a primal heuristic we used a combined rounding and improvement approach. It starts with a spanning tree rounding heuristic to obtain a feasible solution from the current LP solution and then applies a Kernighan-Lin type exchange. To avoid spending too much time, the heuristic passes through alternating active and idle phases. In an active phase, the heuristic is called for every LP solution until the fifth consecutive failure to improve the best known value. The following idle phase ignores a certain number of LP solutions before switching back to an active phase. Initially, the number of ignored LP solutions is set to 100 but doubles each time the preceding active phase failed at each of its improvement attempts. After the first successful such attempt, the idle length is reset.

We used a best-first search strategy for the subproblem selection. Strong branching was done preferring to branch on variables with current value close to 0.5 and large objective function coefficient in absolute value. The CPU time per instance was limited to 10 hours. Tailing-off was deactivated, i.e., we did not enforce early branching but solved the LPs until no further cutting planes could be found. For test instances with an integral objective function we terminated the optimization as soon as the duality gap was below 1 (and not below 2 as it could be done for the ± 1 weighted 2-dimensional toroidal grids).

The contraction of Definition 2.1 can also be applied to a point $z \in \mathbb{R}^R$ for which we do not know if $z \in \mathrm{MET}(G)$ is true or not. If during the contraction of edge st we find a node $u \in B$ for which $z_{su} \neq z_{tu}$, then the nodes s, t, and u identify a violated triangle inequality that can be now lifted to a violated cycle inequality by a sequence of node splittings. If, on the other hand, during all the edge contractions this situation never happens, we can apply an exact cycle separation procedure to the resulting point $\overline{z}$. This procedure is typically quite time consuming, thus reducing the size of the graph on which it has to be applied may reduce the overall separation time considerably.

The following four separation scenarios were tested:

(i) CYC: exclusively uses OC. For toroidal grid graphs, however, it is only invoked in case a preceding call of GEN4CYC failed.

(ii) CON: uses the graph contraction as a heuristic to detect violated cycle inequalities as explained above. If none are found, we perform an exact cycle separation on the contracted LP solution.

(iii) CLQ: is initially identical to CON. Yet, if the contracted LP solution does not violate any cycle inequalities then we extend it and separate 5- and 7-cliques, respectively.

(iv) TC: differs from CLQ in that it separates target cuts (see [7]) on the contracted LP solution instead of clique inequalities on the extended one. For the frequency assignment instances and the toroidal grid graphs the target cut separation uses 300 subgraphs with 10 nodes of the contracted graph. For the remaining instances it uses 600 subgraphs with 8 nodes. Each subgraph is initialized with a random node and then expanded by adding nodes in a greedy manner with respect to the fractionality of the edges that join a candidate node with the already chosen ones. The candidate to be added is the one maximizing the sum of the values $(\frac{1}{2} - |\overline{z}_e - \frac{1}{2}|)$ over all its joining edges e. For the largest frequency assignment instances, 11-node subgraphs were also tried, but the results were inferior to those with subgraphs of size 10.

We allowed each separation procedure to add up to 600 cutting planes per separation phase to a preliminary constraint pool. Yet, only the best 300 inequalities were added to the current LP where the cutting planes were ranked in decreasing order according to the angle of their gradient with the objective function gradient.

6.3. Results of experiments

We start with two preliminary observations. Firstly, the scenarios CLQ and TC performed similar to CON for all 2-dimensional grids: regardless of the scenario, only cycle inequalities were separated in the course of the optimization. Secondly, CON ran out of memory for every $G_{0.5}$ graph of order 60 and 80. This is because the cycle inequalities give very poor bounds for these instances and thus the solver is forced to branch frequently. Technically, this would also have happened in the CYC scenario. Yet, CYC is generally slower than CON and thus the time limit was reached before the algorithm could run out of memory. CLQ and TC, on the other hand, were both able to solve these instances to optimality.

We split the comparison of the performance of the separation scenarios into two parts. The first one covers those classes in which we could solve (almost) all instances to optimality. In this case we use the average CPU time reduction with respect to the CYC scenario to measure the performance of the remaining ones. For the other classes we use the gap closure with respect to CYC as measure.

6.3.1. CPU time reduction.

Table 2 lists the results on CPU time reduction. The rows refer to the classes of test instances. The columns are organized as follows:

- 'Class' specifies the class of test instances. We considered Beasley instances of size $n = 50, 100, 250$ (`beasley`), Billionnet and Elloumi instances of size $n = 120, 250$ (`be`), Glover, Kochenberger, and Alidaee instances with the settings a, b, c, and d (`gka`), two- and three-dimensional toroidal grids with uniformly distributed ± 1 weights (`tpm`) and with Gaussian distributed integral weights (`tg`), unweighted graphs with edge probability $\frac{1}{2}$ and order $n = 60$ (`g05_60`), ± 1-weighted graphs of order $n = 80, 100$ and density $d = 0.1$ (`pm1s`), graphs of order $n = 100$ and density $d = 0.1$ with weights chosen from the ranges $[-10, 10]$ (`w01`) and $[0, 10]$ (`pw01`), and finally the original frequency assignment instances (`man`) as well as the artificially generated ones (`pman`).
- '#Files' lists the total number of instances in each class.
- '#Limit' gives the number of instances per class that neither scenario could solve to optimality within ten hours.
- '#Wins' for each separation scenario lists the number of instances per class for which this scenario took the least time to solve them to optimality. The instances counted by '#Limit' were excluded from this evaluation. The line total may exceed the number of remaining instances since different scenarios can have almost identical CPU times. The maximum value over all scenarios is typeset bold for each class.
- '#Add. rejects' for each of the scenarios CON, CLQ, and TC gives the number of instances per class that were excluded from the evaluation of the respective average CPU time reduction in addition to those counted by '#Limit'. We rejected an instance if CYC or the respective scenario exceeded the time limit. Nonzero entries consist of two values. The first one is the total number of additionally rejected instances. The second value, given in parentheses, specifies how many of these rejections were caused by a time limit violation of the respective scenario itself. The difference between first and second value gives the number of instances that could be solved to optimality by the respective scenario while CYC failed. The nonzero entries with maximum such difference are typeset bold for each class.
- 'Avg. CPU time red.' for each of the scenarios CON, CLQ, and TC lists its respective average CPU time reduction compared to CYC. The entries specify the reduction averaged over all instances in a given class except for those counted by '#Limit' and by the respective entry of '#Add. rejects'. The maximum average reduction over all scenarios is typeset bold for each class.

Since CYC exceeded the time limit for all `g05_60` instances, we used the ten hour time limit as lower bound on the computation time. Thus, the average reduction values for this class are to be understood as lower bounds as well. We see that CON clearly dominates with average time reductions ranging from 54% to 97% for most of the classes and an average reduction over all classes except `g05_60` of almost 55%. Particularly noteworthy are the results for the 2-dimensional Ising spin glass problems, even more so since CYC uses the linear time GEN4CYC heuristic. It is remarkable that the more general shrink separation is able to outperform this specialized heuristic.

CLQ and TC, on the other hand, give mixed results and are sometimes even outperformed by CYC. CLQ, despite partly good time reductions of up to 97%, is on average approximately 50% slower than CYC. The results for TC are even worse at first glance, with an average time increase of 2083%. Yet, this is due to the extremely bad performance on the `gka.b` instances. If we omit this class in the evaluation, TC reaches a solid average time reduction of 35% for the remaining classes.

6.3.2. Gap closure.

Table 3 lists the results on the gap closure. Again, the rows refer to the different instance classes. The columns 'Class' and '#Files' are defined as before. The remaining columns are organized as follows:

- '#Wins' for each scenario specifies the number of instances per class for which it gave the smallest relative gap after ten hours of computation. The line total may exceed the number

			#Wins				#Add. rejects			Avg. CPU time red. [%]		
Class	#Files	#Limit	CYC	CON	CLQ	TC	CON	CLQ	TC	CON	CLQ	TC
beasley50	10	0	0	6	**8**	**8**	0	0	0	92	**97**	97
beasley100	10	0	0	**6**	**6**	4	0	0	0	**97**	97	96
beasley250	10	5	0	**5**	0	0	**2(0)**	3(3)	**2(0)**	**59**	9	41
be120.3	10	0	0	**9**	1	0	**1(0)**	1(1)	1(1)	**57**	−85	16
be250	10	6	0	**4**	0	0	0	1(1)	0	**54**	−23	47
gka.a	8	0	0	4	**5**	3	0	0	0	93	**94**	91
gka.b	10	0	0	**10**	0	0	0	0	4(4)	**58**	19	−40 215
gka.c	7	0	0	2	3	**5**	0	0	0	95	95	**96**
gka.d	10	4	0	**6**	1	1	**1(0)**	1(1)	1(1)	**67**	−1	41
tpm.2d	120	8	0	**112**	—[b]	—[b]	**18(0)**	—[b]	—[b]	**96**	—[b]	—[b]
tg.2d	320	0	0	**320**	—[b]	—[b]	**1(0)**	—[b]	—[b]	**90**	—[b]	—[b]
tpm.3d	40	5	5	**25**	4	7	**1(0)**	5(5)	1(1)	**23**	−159	−56
tg.3d	70	7	2	**39**	17	20	4(2)	5(5)	**3(0)**	**44**	−40	26
g05_60[c]	10	0	0	—[a]	**10**	0	—[a]	0	3(3)	—[a]	**≥ 87**	≥ 62
pm1s	20	0	**14**	6	0	0	0	0	0	**−16**	−309	−62
w01	10	0	**8**	1	0	1	0	0	0	**−37**	−458	−73
pw01	10	0	**8**	2	0	0	0	0	0	**−24**	−685	−102
man	4	0	0	**3**	1	0	**2(0)**	**2(0)**	**2(0)**	**71**	70	61
pman	57	0	7	23	**27**	22	9(7)	**9(0)**	10(3)	67	**68**	64

[a] The CON scenario ran out of memory for every instance.
[b] Equivalent to CON scenario.
[c] The CYC scenario exceeded the ten hour time limit on all instances. Instead of omitting the entire class, we used the limit as a lower bound on the computation time of CYC.

Table 2: Statistics on the CPU times of different separation scenarios.

of instances since different scenarios can result in the same relative gap. The maximum value over all scenarios is typeset bold for each class.

- 'Avg. gap cl.' for each of the scenarios CON, CLQ, and TC, respectively, lists its average gap closure with respect to the CYC scenario after ten hours of computation. The maximum average closure over all scenarios is typeset bold for each class.

CON delivered the smallest gaps in most cases; yet, its average gap closure over all classes is only about 13%. CLQ and TC reach an average gap closure of 20% and 18%, respectively, with peak values of 89% and 81%. Still, all three scenarios only work well for about one third of the tested classes.

6.3.3. Frequency assignment instances.

As mentioned before, the man instances are based on real-world data. The underlying graphs are fairly sparse with densities ranging from 1% to 7%. The only exception is man_k48; yet, this instance is easy to solve and is hence omitted.

We performed additional experiments for this interesting set of instances using a variation CYC′ of the CYC scenario without CPU time limit. Also, to accelerate the optimization, it activates the tailing-off control which enforces a branching step if the minimal change of the objective function value between the solution of one hundred successive linear programming relaxations in the subproblem optimization has been below 0.0015%.

Still, even CYC′ could not solve the largest instance man_k487c to optimality; it eventually ran out of memory due to the steadily growing branch-and-cut tree. For the remaining two instances, Table 4 shows that CON was the dominant scenario with an average time reduction of almost 99% compared to CYC′. CLQ and TC performed slightly worse with average reductions of 97% and 86%, respectively, yet still being far superior to CYC′.

Table 5 lists the sizes of the branch-and-cut trees at the end of each optimization run. CON reduced the tree size by 26% on average. Still, the scenario needed to branch, indicating that cycle inequalities alone are not sufficient to solve these instances at the root node. Here, the

		#Wins				Avg. gap cl. [%]		
Class	#Files	CYC	CON	CLQ	TC	CON	CLQ	TC
beasley500	10	0	**8**	2	0	**68**	68	68
be100	10	0	**10**	0	0	**10**	4	−18
be120.8	10	0	**10**	0	0	**6**	0	−8
be150.3	10	0	**9**	0	1	**29**	−23	13
be150.8	10	2	**4**	3	1	**4**	4	4
be200.3	10	**7**	3	0	0	**1**	−2	1
be200.8	10	**6**	0	3	1	−17	**−5**	−20
gka.e	5	**3**	2	1	2	**11**	4	10
gka.f	5	**3**	1	1	0	**−4**	−5	−5
g05_80	10	0	—[a]	**10**	0	—[a]	**89**	81
g05_100	10	0	1	**7**	2	26	59	**65**
pm1d	20	0	**19**	1	0	**19**	11	7
w05	10	0	**9**	0	1	**43**	33	41
w09	10	0	**10**	0	0	**19**	13	8
pw05	10	0	0	**10**	0	12	**69**	56
pw09	10	0	2	**8**	0	6	**8**	2
pman	34	5	0	**25**	4	-12	**22**	7

[a] The CON scenario ran out of memory for every instance.

Table 3: Statistics on the gap closure for different separation scenarios.

	CPU time [sec]			
Instance	CYC′	CON	CLQ	TC
man_k487a	209	**6**	11	58
man_k487b	38 167	**42**	82	243
man_k487c	—[a]	—[a]	**28 316**	31 133

[a] Out of memory error.

Table 4: CPU times for frequency assignment instances.

remaining two scenarios are clearly preferable. CLQ reduced the tree size by 74% on average while TC was even able to solve both instances at the root node.

	#Nodes in B&C-tree			
Instance	CYC′	CON	CLQ	TC
man_k487a	31	39	11	**1**
man_k487b	41	9	7	**1**
man_k487c	—[a]	—[a]	15	**9**

[a] Out of memory error.

Table 5: Number of nodes in the branch-and-cut tree for frequency assignment instances.

On the largest instance man_k487c, both CYC′ and CON ran out of memory. CLQ and TC, on the other hand, were able to solve the instance to optimality within 7 hours 52 minutes and 8 hours 39 minutes, respectively.

In conclusion, the CON scenario, though capable of accelerating the optimization significantly, may not be sufficient to solve certain problems to optimality. The more difficult problems can be dealt with by separating additional clique inequalities or target cuts. If a slight increase in CPU time is not an issue, one can even apply the target cut separation to the easier problem instances, which allows to solve them at the root node instead of resorting to branching.

6.3.4. Effect of graph shrinking.

Finally, we present some statistics on the sizes of the contracted graphs. In the course of the cutting plane procedure, we consecutively numbered those LP solutions for which the shrink separation

never performed an edge contraction leading to a violated 3-cycle inequality, i.e., in which the graph contraction was not prematurely aborted. Figure 8 shows the sizes of the contracted graphs for each of the toroidal (100×100) grids with Gaussian distributed weights as well as for the instance `man_k487b`. The horizontal axis is labeled with the above LP solution numbering and the vertical axis gives the size of the corresponding shrunk graphs. Figure 8(a) displays the results, in standard scaling, for the toroidal (100×100) grids; here, we superimposed the data of the ten random instances (see Section 6.1). Figure 8(b) shows the same results in logarithmic scaling. Finally, Figure 8(c) provides the respective data, in standard scaling, for the instance `man_k487b`.

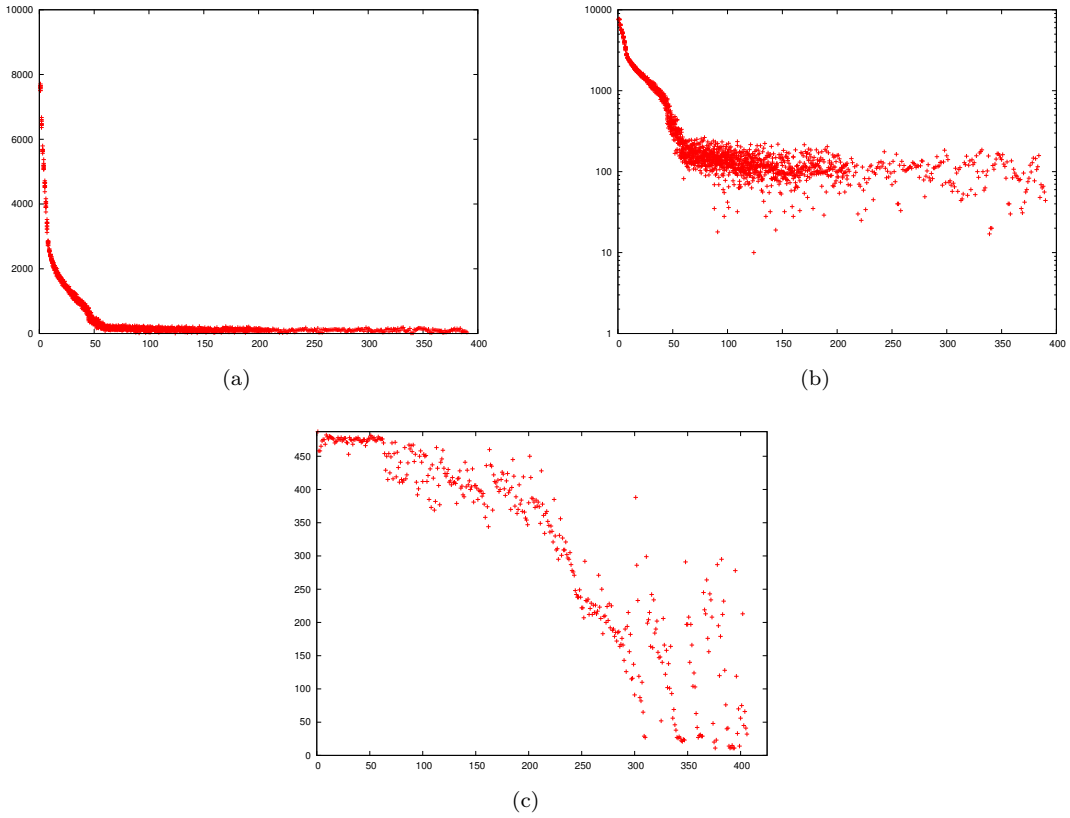


Figure 8: Sizes of contracted graphs.

7. Conclusion

In this paper we have developed a new separation approach for the the max-cut problem in general graphs. The idea of shrinking graphs associated with fractional LP solutions has proved to be successful in many cases. Clearly, the success heavily depends on the max-cut instance. As a byproduct the shrinking procedure provides a very fast separation heuristic for cycle inequalities which together with an exact separation algorithm constitutes the ideal approach to finding violated cycle inequalities.

Acknowledgment

We would like to thank Carlo Mannino for providing the interesting max-cut instances arising in the frequency assignment context.

References

[1] M. Anjos and J. Lasserre (eds.). Handbook on Semidefinite, Conic and Polynomial Optimization. Springer (2012)

[2] D.L. Applegate, R.E. Bixby, V. Chvátal, and W.J. Cook. The Traveling Salesman Problem: A Computational Study. Princeton University Press (2006)

[3] F. Barahona. On cuts and matchings in planar graphs. Mathematical Programming **60**, 53–68 (1993)

[4] F. Barahona, M. Grötschel, M. Jünger, and G. Reinelt. An application of combinatorial optimization to statistical physics and circuit layout design. Operations Research **36**, 493–513 (1988)

[5] F. Barahona and A.R. Mahjoub. On the cut polytope. Mathematical Programming **36**, 157–173 (1986)

[6] E. Boros and P.L. Hammer. Cut-polytopes, boolean quadric polytopes and nonnegative quadratic pseudo-boolean functions. Mathematics of Operations Research **18**, 245–253 (1993)

[7] C. Buchheim, F. Liers, and M. Oswald. Local cuts revisited. Operations Research Letters **36**, 430–433 (2008)

[8] C. De Simone. Lifting facets of the cut polytope. Operations Research Letters **9**, 341–344 (1990)

[9] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch and cut algorithm. Journal of Statistical Physics **80**, 487–496 (1995)

[10] C. De Simone, M. Diehl, M. Jünger, P. Mutzel, G. Reinelt, and G. Rinaldi. Exact ground states of two-dimensional $\pm J$ Ising spin glasses. Journal of Statistical Physics **84**, 1363–1371 (1996)

[11] C. De Simone and G. Rinaldi. A cutting plane algorithm for the max-cut problem. Optimization Methods and Software **3**, 195–214 (1994)

[12] M.M. Deza and M. Laurent. Geometry of Cuts and Metrics. Algorithms and Combinatorics, vol. 15. Springer-Verlag, Berlin (1997)

[13] M.X. Goemans and D.P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. Journal of the ACM **42**, 1115–1145 (1995)

[14] C. Helmberg, F. Rendl, R.J. Vanderbei, and H. Wolkowicz. An interior-point method for semidefinite programming. SIAM Journal on Optimization **6**, 342–361 (1996)

[15] M. Jünger and S. Thienel. The ABACUS system for branch-and-cut-and-price algorithms in integer programming and combinatorial optimization. Software: Practice and Experience **30**, 1325–1352 (2000)

[16] M. Laurent and S. Poljak. One-third-integrality in the max-cut problem. Mathematical Programming **71**, 29–50 (1995)

[17] F. Liers, M. Jünger, G. Reinelt, and G. Rinaldi. Computing exact ground states of hard Ising spin glass problems by branch-and-cut. In: A. Hartmann and H. Rieger (eds.) New Optimization Algorithms in Physics, pp. 47–70. Wiley-VCH (2004).

[18] O.L. Mangasarian. Nonlinear Programming. Classics in Applied Mathematics, vol. 10. SIAM (1994)

[19] C. Mannino. Personal communication (2011)

[20] G. Marsaglia and T.A. Bray. A convenient method for generating normal variables. SIAM Review **6**, 260–264 (1964)

[21] S. Poljak and Z. Tuza. Maximum cuts and large bipartite subgraphs. In: W. Cook et al. (eds.) Combinatorial Optimization. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 20, pp. 181–244 (1995)

[22] F. Rendl. Semidefinite relaxations for integer programming. In: M. Jünger et al. (eds.): 50 Years of Integer Programming 1958-2008: The Early Years and State-of-the-Art Surveys, pp. 687–726. Springer (2010)

[23] F. Rendl, G. Rinaldi, and A. Wiegele. Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. Mathematical Programming **121**, 307–335 (2010)

[24] G. Rinaldi. Rudy: A graph generator. www-user.tu-chemnitz.de/~helmberg/rudy.tar.gz (1998)

[25] A. Wiegele. BiqMac Library. biqmac.uni-klu.ac.at/biqmaclib.html (2007)