Odd-Cycle Separation for Maximum Cut and Binary Quadratic Optimization

Michael Jünger
University of Cologne, Germany
mjuenger@informatik.uni-koeln.de

Sven Mallach
University of Cologne, Germany
mallach@informatik.uni-koeln.de

Abstract

Solving the NP-hard Maximum Cut or Binary Quadratic Optimization Problem to optimality is important in many applications including Physics, Chemistry, Neuroscience, and Circuit Layout. The leading approaches based on linear/semidefinite programming require the separation of so-called odd-cycle inequalities for solving relaxations within their associated branch-and-cut frameworks. In their groundbreaking work, F. Barahona and A.R. Mahjoub have given an informal description of a polynomial-time separation procedure for the odd-cycle inequalities. Since then, the odd-cycle separation problem has broadly been considered solved. However, as we reveal, a straightforward implementation is likely to generate inequalities that are not facet-defining and have further undesired properties. Here, we present a more detailed analysis, along with enhancements to overcome the associated issues efficiently. In a corresponding experimental study, it turns out that these are worthwhile, and may speed up the solution process significantly.

2012 ACM Subject Classification Mathematics of computing → Combinatorial optimization; Mathematics of computing → Linear programming; Mathematics of computing → Integer programming

Keywords and phrases Maximum cut, Binary quadratic optimization, Integer linear programming

Digital Object Identifier 10.4230/LIPIcs.ESA.2019.63

Funding *Michael Jünger*: We gratefully acknowledge the funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 764759 *MINOA Mixed-Integer Nonlinear Optimisation: Algorithms and Applications.*

1 Introduction

There are various applications that require solving the Maximum Cut (henceforth MaxCut) problem to optimality, and this has been achieved in the literature by specialized branch-and-cut algorithms that are based on certain relaxations of MaxCut. A prominent example is the determination of ground states in Ising Spin Glasses [1, 3, 4]. Other applications occur in, e.g., Chemistry, Neuroscience, and Circuit Layout. Also, the generic Binary Quadratic Optimization problem (BQP) has a direct transformation to MaxCut, see, e.g. [1], such that enhanced branch-and-cut algorithms for MaxCut directly lead to enhanced branch-and-cut algorithms for the BQP.

A key element to be carried out repeatedly in the course of such branch-and-cut algorithms is the separation of *odd-cycle inequalities* associated with the *cut polytope* as defined in Section 2. In their groundbreaking work, F. Barahona and A.R. Mahjoub [2] have shown that, under certain conditions, the odd-cycle inequalities induce maximal faces (facets) of the cut polytope – which is desirable for their application within a branch-and-cut algorithm. Moreover, they have given an informal description of a polynomial-time separation

procedure for these inequalities. Since then, this "algorithmic frame" has been used in many computational studies, but we are not aware of any accounts on the details of the respective implementations or experimental evaluations.

As we show in this paper, however, the odd-cycle inequalities derived using a "straightforward" implementation of this frame are frequently *not* facet-inducing – for two reasons that are both inherent to the algorithm. At the same time, it is, to the best of our knowledge, the only polynomial-time one proposed so far. Along with a more detailed analysis of odd-cycle separation, we thus provide extensions to overcome these shortcomings efficiently by extending the original algorithm. Finally, we present an experimental study showing the practical impact of these enhancements using established benchmark instances for MaxCut and the BQP. It turns out that the additional effort invested is typically more than compensated, i.e., has a positive effect on the solution process.

The outline of this paper is as follows: In Section 2, we define the separation problem for odd-cycle inequalities in the context of solving MaxCut by branch-and-cut, present the outline of the polynomial time separation algorithm given in [2], and point out the shortcomings of a naïve implementation. In Section 3, we analyze the problem formally and present various strategies to enhance the separation procedure. These are experimentally evaluated in Section 4.

2 The Maximum Cut Problem and Odd-Cycle Inequalities

Let $G = (V, E)$ be a simple undirected graph, i.e., there are no loops and no parallel edges. For $k > 1$, a *walk* in G is a set of edges $W = \{\{v_0, v_1\}, \{v_1, v_2\}, \ldots, \{v_{k-1}, v_k\}\} \subseteq E$. If, in addition, the vertices $v_0, v_1, \ldots, v_k \in V$ are pairwise different, W is called a *path*. Provided that W is a walk and $\{v_0, v_k\} \in E$, we call $W \cup \{v_0, v_k\}$ a *closed walk*. If P is a path, and $\{v_0, v_k\} \in E$, then $C = P \cup \{v_0, v_k\}$ is called a *(simple) cycle*. We will also refer to a closed walk that is not a simple cycle as a *non-simple cycle*.

For $W \subseteq V$, let $\delta(W) := \{e = \{u, v\} \in E \mid u \in W, v \in V \setminus W\}$. The edge subsets $\delta(W)$ for any $W \subseteq V$ are the *cuts* of G. They correspond to bipartitions of the vertex set V into W and $V \setminus W$. Given weights w_e for $e \in E$, the maximum cut problem, formulated as an integer linear programming problem, reads:

$$\max \sum_{e \in E} w_e x_e \tag{1}$$

$$\sum_{e \in Q} x_e - \sum_{e \in C \setminus Q} x_e \leq |Q| - 1 \qquad \text{for all } Q \subseteq C \subseteq E,\ C \text{ cycle and } |Q| \text{ odd} \tag{2}$$

$$0 \leq x_e \leq 1 \qquad \text{for all } e \in E \tag{3}$$

$$x_e \in \mathbb{Z} \qquad \text{for all } e \in E \tag{4}$$

Any solution $\hat{x}$ of (2)–(4) corresponds to a cut $\hat{F} = \{e \in E \mid \hat{x}_e = 1\}$ and vice versa. The necessary and sufficient condition for this to hold is that $\hat{F}$ intersects with every cycle in G in an even number of edges which is enforced by the *odd-cycle inequalities* (2) while the trivial constraints (3) give lower and upper bounds. Consequently, any optimum solution x^* of (2)–(4) gives rise to an optimum cut $F^* = \{e \in E \mid x^*_e = 1\}$ with maximum total weight. We refer to $\hat{x}$ and x^* as the characteristic vectors of $\hat{F}$ and F^*, respectively. Also, we will call a cycle associated with an odd-cycle inequality "odd cycle" although $|C|$ may be even, and refer to it as a pair (C, Q) if its unique determination matters.

The *cut polytope* is the convex hull of the feasible solutions to the maximum cut problem associated with $G = (V, E)$:

$$P_{\text{CUT}}(G) = \text{conv}\{x \in \mathbb{R}^E \mid x \text{ satisfies (2)–(4)}\}$$

At the core of any linear programming based branch-and-cut algorithm, a sequence of linear programming relaxations of (1)–(3) is solved by a *cutting plane algorithm*: The first relaxation just consists of (1) and (3). Then it is the task of an *odd-cycle separation algorithm* to decide if the optimum solution $\hat{x}$ of the current relaxation satisfies all odd-cycle constraints (2). If so, the linear programming relaxation (1)–(3) is solved to optimality and its objective function value is used as an upper bound in a branch-and-bound scheme. If not, the separation algorithm must provide at least one odd-cycle inequality that is violated by $\hat{x}$. The produced inequalities are then added to strengthen the current relaxation, this new relaxation is solved to optimality, and the process is iterated until the separation algorithm decides that the solution of the current linear program satisfies all odd-cycle constraints.

In branch-and-cut, it is preferred that the separated inequalities define facets of the polytope associated with the problem, so here, we would prefer odd-cycle inequalities that define facets of the cut polytope $P_{\mathrm{CUT}}(G)$. Barahona and Mahjoub [2] have given a proof that an odd-cycle inequality defines a facet of $P_{\mathrm{CUT}}(G)$ if and only if its associated cycle C is *chordless*, i.e., if there is no edge in $E \setminus C$ that connects two vertices of C.

2.1 Polynomial-Time Separation of Odd-Cycle Inequalities

We describe the polynomial time algorithm of Barahona and Mahjoub [2]. For this purpose, we rewrite the odd-cycle inequalities (2) associated with a graph $G = (V, E)$ as follows:

$$\sum_{e \in Q} (1 - x_e) + \sum_{e \in C \setminus Q} x_e \geq 1 \qquad \text{for all } Q \subseteq C \subseteq E,\ C \text{ cycle and } |Q| \text{ odd} \tag{5}$$

Our task is to solve the separation problem for $\hat{x} \in [0,1]^E$. Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two copies of G, and denote with $u_1 \in V_1$ the representative of $u \in V$ in G_1 and with $u_2 \in V_2$ the one in G_2. Based on these copies, define a new weighted graph $G_S = (V_S, E_S)$ with $V_S = V_1 \cup V_2$, $E_S = E_1 \cup E_2 \cup E_3$. The additional edge set $E_3 \subseteq V_1 \times V_2$ consists of the edges $\{u_1, v_2\}$ and $\{v_1, u_2\}$ for each $\{u, v\} \in E$. The edges $\{u_1, v_1\} \in E_1$ receive weights $\omega(\{u_1, v_1\}) = \hat{x}_{\{u,v\}}$, the edges $\{u_2, v_2\} \in E_2$ also receive weights $\omega(\{u_2, v_2\}) = \hat{x}_{\{u,v\}}$, and the edges $\{u_1, v_2\}, \{v_1, u_2\} \in E_3$ receive "inverted" weights $\omega(\{u_1, v_2\}) = \omega(\{v_1, u_2\}) = 1 - \hat{x}_{\{u,v\}}$. To ease notation, we will frequently write $\omega(F) \coloneqq \sum_{e \in F} \omega(e)$ for any edge set $F \subseteq E_S$ or $F \subseteq E$. Moreover, we will also write $V_S(F)$ or $V(F)$ to denote the vertex set $\{v \in V_S \mid \exists\, \{v, w\} \in F\}$ or $\{v \in V \mid \exists\, \{v, w\} \in F\}$, respectively.

The fundamental property of the construction described is that, for any $u \in V$, any path $P_u \subseteq E_S$ from u_1 to u_2 in G_S corresponds to a closed walk C in the original graph G containing u. Moreover, it inevitably involves an odd number of edges from the set E_3 that, as the associated ω-weights indicate, define the subset $Q \subseteq C$.

Thus, each such path P_u corresponds to an odd closed walk (C, Q) in G that may however have vertex as well as edge repetitions, i.e., C is not necessarily a simple cycle. Fig. 1 shows a small example where two different paths of equal length (w.r.t. ω as well as the total *number* of edges) in G_S lead to either a simple or non-simple cycle in G.

Now, the central idea of the algorithm is to compute, for each $u \in V$, an ω-*shortest* path P_u from u_1 to u_2 in G_S.

If then $\omega(P_u) \geq 1$ for every $u \in V$, then there is no odd closed walk (C, Q) that violates

$$\sum_{e \in Q} (1 - x_e) + \sum_{e \in C \setminus Q} x_e \geq 1 \qquad \text{for all } Q \subseteq C \subseteq E,\ C \text{ closed walk and } |Q| \text{ odd} \tag{6}$$

and, therefore, no simple cycle C that violates (5).

Otherwise, if $\omega(P_u) < 1$ for some $u \in V$, then P_u corresponds to an odd closed walk (C, Q) that violates (6). However, only if (C, Q) is "accidentally" simple, we have found a violated inequality of type (5).

This does not affect the correctness of the approach: Even when using (C, Q) "as is" in the cutting plane algorithm, i.e., when possibly adding an inequality of type (6) rather than one of type (5) to the linear program, the final result satisfies all the latter and thus solves the relaxation (1)–(3) of MaxCut. However, while inequalities (6) are valid for $P_{\mathrm{CUT}}(G)$, they are not facet-inducing. And even if a cycle (C, Q) derived from an ω-shortest path in G_S is simple, it is very likely to have chords as we show in Sect. 3.3. So in summary, when using the described approach, the likelihood to generate cutting planes that do *not* correspond to facets of $P_{\mathrm{CUT}}(G)$ is high. Fortunately, we can efficiently construct from P_u a simple odd cycle (C, Q) that violates (5), and even one that is chordless, as we shall see in Sect. 3.2 and 3.3, respectively.

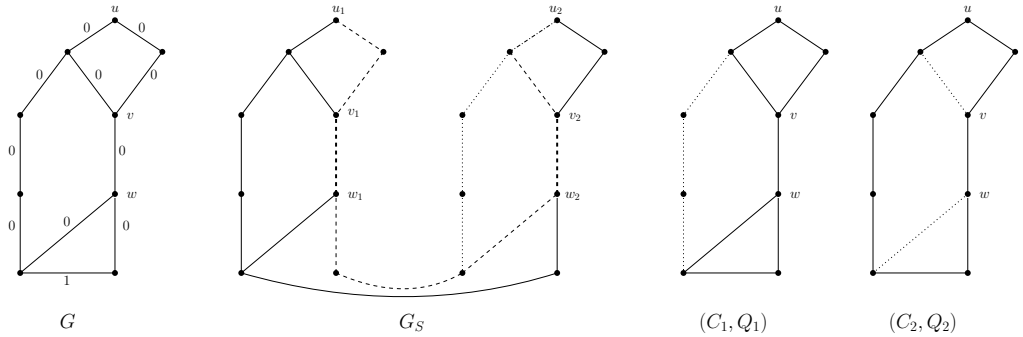

Figure 1 Left: An example graph G whose edges are annotated with an (integral) linear program solution. Middle: The corresponding separation graph G_S (only zero-weight edges are shown). The dashed edges correspond to a shortest u_1-u_2-path that induces a non-simple odd closed walk in G (edge $\{v, w\}$ is used twice, see the left of the two cycles depicted on the right). If the subpath within the right copy of G is replaced by the dotted one, the induced cycle is simple (but not chordless).

3 Non-Simple and Non-Chordless Simple Odd Cycles

To the best of our knowledge, the possible non-simplicity of odd closed walks derived with the common procedure, as well as the possible presence of chords, has attracted almost no further consideration in the literature before. However, as we will see, it matters not only in theory but also in practice.

Besides the preserved correctness of the separation algorithm already discussed, another possible reason why non-simplicity has been ignored may be the existence of an easy proof that any odd closed walk (C, Q) contains at least one simple cycle (C', Q') with $C' \subseteq C$, $Q' \subseteq Q$, and with $|Q'|$ odd:

If (C, Q) is non-simple, then either there is at least one edge $e = \{u, v\} \in E$ that occurs $k > 1$ times on C. Or, each edge $e \in E$ occurs at most once on C, but there is a vertex $v \in V$ whose degree w.r.t. the edges of C is $d_C(v) := |\{w \in V : \{v, w\} \in C\}| > 2$ (cf. Fig. 2). Since we assumed before that no edge is contained twice in C, $d_C(v)$ must then be even[1].

[1] The present case of odd closed walks induced by simple paths P in G_S with weight $\omega(P) < 1$ implies $k \leq 2$ since any combination of an E_3-instance with an E_1- or E_2-instance of the same edge $e \in E$ leads to an ω-weight of at least one. Moreover, the simplicity of P_u implies $d_C(v) \leq 4$ for all $v \in V$.

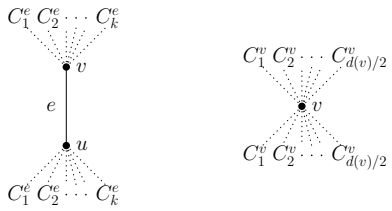

Figure 2 Multiple cycles comprising an edge $\{u, v\} \in E$ or a single vertex $v \in V$.

In the first case, let $C_1^e, \ldots, C_k^e$ be the corresponding closed walks that result from starting a path in v – each time using a different edge that is not e – and continuing the path until v is reached again first using e. In the second case, let $C_1^v, \ldots, C_{\frac{d_C(v)}{2}}^v$ be the respective closed walks that result from starting a path in v using each time a different edge, and ending when v is first reached again.

We refer to the associated subsets of Q as $Q_i^e := C_i^e \cap Q$, $i \in \{1, \ldots, k\}$, and $Q_i^v := C_i^v \cap Q$, $i \in \{1, \ldots, \frac{d_C(v)}{2}\}$, respectively. Since Q is odd, there is at least one $i^* \in \{1, \ldots, k\}$ ($i^* \in \{1, \ldots, \frac{d_C(v)}{2}\}$) such that $|Q_{i^*}^e| \leq |Q|$ ($|Q_{i^*}^v| \leq |Q|$) is odd as well. Now either $(C_{i^*}^e, Q_{i^*}^e)$ ($(C_{i^*}^v, Q_{i^*}^v)$) is simple in which case we are done. Or, we find an edge $f \in C_{i^*}^e$, $f \neq e$, that occurs more than once on $C_{i^*}^e$ (a vertex $w \neq v$ in $C_{i^*}^v$ such that $d_{C_{i^*}^v}(w) > 2$) and have thus reduced the extraction of a simple cycle to a strictly smaller non-simple one. Thus, since any simple cycle has length at least three, the according recursion must terminate after finitely many steps with a cycle that is simple.

It however remains open how to *extract* a (chordless) simple odd closed walk from a non-simple one *efficiently*. This will be addressed in Sect. 3.2 and Sect. 3.3 after clarifying in Sect. 3.1 which kinds of non-simplicity can occur from the method described in Sect. 2.1.

3.1 Cases of Non-Simplicity

The landscape of different cases and their implications concerning the facet-defining property of the associated odd-cycle inequalities is depicted in Fig. 3.

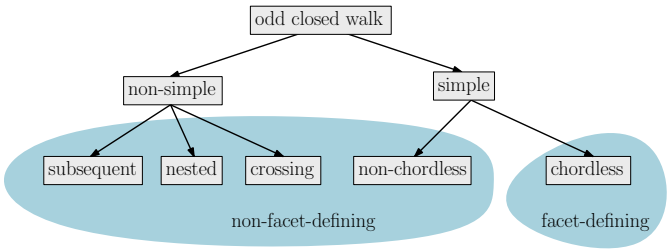

Figure 3 Categorization of odd closed walks.

Let P_u be an ω-shortest path from u_1 to u_2 in G_S, and let (C, Q) be its induced odd closed walk in G. We may assume w.l.o.g. (and enforce in an implementation) that P_u does not enter u_1 and that it does not leave u_2 which implies that $d_C(u) = 2$.

Now if (C, Q) is non-simple, there must be at least one vertex v with $d_C(v) > 2$, i.e., $v \neq u$. The simplicity of P_u then implies that it must traverse v_1 as well as v_2. Denote the corresponding subpath, that itself defines an odd closed walk, with P_v. It is *fully nested* within P_u, i.e., $P_v \subsetneq P_u$. Moreover, since the number of E_3-edges in P_u as well as in P_v is odd, the number of E_3-edges in $P_u \setminus P_v$ must be even. Hence, while its induced edge set in G is still a closed walk, it is not an odd one.

Of course, an odd closed walk (C, Q) induced by an ω-shortest path P_u in G_S may have not only one but several vertices $v \in V$ with $d_C(v) > 2$. If all the associated subpaths of P_u are fully nested, the previous arguments imply that only the innermost one of them induces a simple odd cycle. Otherwise, any two strict subpaths of P_u, say P_v and P_w, may either occur *subsequently*, i.e., $P_v \cap P_w = \emptyset$, or *crossing*, i.e., either w_1 or w_2 occurs in P_v but not both. Fig. 4 displays the respective vertex sequences using the notation v_s and v_e (for "start" and "end") since, for any $v \in V$, $v \neq u$, v_2 may be visited by P_u before v_1 is visited and vice versa.

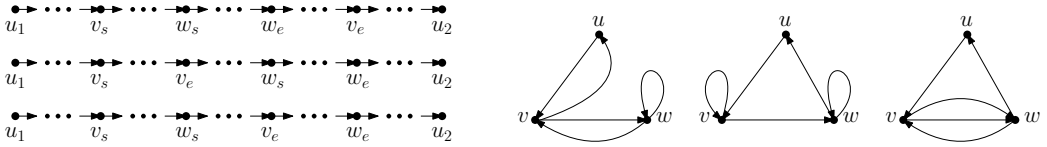

Figure 4 Left: Schematic depiction of (from top to bottom) two fully nested, subsequent, and crossing subpaths P_v and P_w of P_u. Right: The corresponding interpretations in the original graph (from left to right).

In the subsequent case, both P_v and P_w correspond to odd closed walks since they both comprise an odd-cardinality subset of E_3-edges (but neither $P_u \setminus P_v$ nor $P_u \setminus P_w$ does). In the crossing case, P_v and P_w both correspond to odd closed walks as well. However, if the subpaths from v_s to w_s and from v_e to w_e refer to the same edge set in G, then both odd closed walks associated to P_v and P_w are equivalent (and thus lead to the same inequality).

3.2 Handling Non-Simple and Chorded Simple Odd Cycles

There are basically three ways to deal with non-simplicity.

3.2.1 The "Take-As-Is" Strategy

Since the potential non-simplicity of a closed odd walk (C, Q) does not affect the validity of the associated inequality, one may consider to simply *use it as a cutting plane as is*.

3.2.2 The "Throw-Away" Strategy

Another strategy is to *discard* non-simple closed odd walks, i.e., to generate an inequality only if a closed odd walk computed is "accidentally" simple.

However, to still preserve an *exact* separation algorithm, it must then be sure that *at least one* simple cycle is still identified if a violated odd-cycle inequality exists. Moreover, as already discussed in Sect. 2.1, any vertex $u \in V$ may have several ω-shortest paths P_u some of which may induce non-simple cycles, and that may even all have the same number of edges. So "throwing away" any such P_u containing a subpath P_v by "trusting" that a simple cycle will be identified when computing an ω-shortest v_1-v_2-path may fail in general.

Fortunately, the desired guarantee can nevertheless be established if we strive for (ω, ℓ)-shortest paths instead, where $\ell(v) \in \mathbb{Z}$ denotes the number of edges used in a (currently) shortest path to each vertex $v \in V_S$. In this context, a u_1-v-path P in G_S involving edge $\{v, w\} \in E_S$ is (ω, ℓ)-shorter than the best previous one, if either "classically" $\text{dist}(u) + \omega(u, v) < \text{dist}(v)$, or if $\text{dist}(u) + \omega(u, v) = \text{dist}(v)$ and $\ell(u) + 1 < \ell(v)$. With these extensions, the original separation algorithm will identify at least one violated odd-cycle inequality if any exists, as we will now show.

▶ **Theorem 1.** *Let $P_u \subseteq E_S$ be an ω-shortest path from u_1 to u_2 in G_S with weight $\omega(P_u) < 1$ and of edge length $\ell(P_u)$. Then there exists a vertex $w \in V$ such that* each *(ω, ℓ)-shortest path P_w in G_S satisfies $\omega(P_w) \leq \omega(P_u)$, $|P_w| \leq |P_u|$, and whose induced odd closed walk (C, Q) in G is* simple.

Proof. If the odd closed walk induced by P_u is itself simple, the statement follows by choosing $P_w = P_u$. So suppose this is not the case. Then there exists a strict subpath P_v of P_u. Now either P_v is simple, in which case we may choose $P_w = P_v$. Otherwise, we recurse on the path P_v that clearly satisfies $|P_v| \leq |P_u| - 2$. Since the number of edges of the paths considered thus decreases strictly monotonically, and since any shortest path in G_S has a positive length, this process will eventually terminate at some simple path P_w, $w \in V_S$, as otherwise, there would exist a path of the same ω-weight which is shorter w.r.t. ℓ – a contradiction. ◀

3.2.3 The "Extraction" Strategy

A final strategy to deal with non-simple odd closed walks is to extract the simple odd cycles contained in it. This is motivated by several advantages, e.g.:

- If a shortest inner subpath $P_v \subseteq P_u$ corresponding to a simple cycle in G is found, the shortest path computation w.r.t. v may be omitted (accepting a possible loss of a different inequality corresponding to another shortest v_1-v_2-path).
- A single pass of P_u might allow to extract *several* simple cycles from one non-simple one.
- Information found during the extraction might be used to avoid the addition of duplicate inequalities (cf. the case of crossing subpaths at the end of Sect. 3.1).

Moreover, as expressed in the following observation, it is possible to implement simple cycle extractions requiring only a number of operations that is proportional to the length of the closed walk (which can be no more than $2|V|$). So since a *single pass* of the closed walk is inevitably required to even generate an inequality from it or find out whether it is simple, the respective enhancement does not increase the asymptotic running time, and does not even require the (ω, ℓ)-extension from Sect. 3.2.2 (which is nevertheless worthwhile) in order to provide a safe *exact* separation procedure.

▶ **Observation 2.** *Let P_u be an ω-shortest path from u_1 to u_2 in G_S with weight $\omega(P_u) < 1$, and whose induced odd closed walk (C, Q) in G is* not simple. *Traverse P_u starting from u_1 (or, walking backwards, from u_2) and let $v \in W$ be the* first *vertex such that v_1 and v_2 were* both *visited on this traversal. Then the cycle in G corresponding to the path P_v is simple.*

An inequality derived based on Observation 2 (i.e., on a first *fully nested* subpath P_v of P_u that does not contain any further fully nested subpaths itself) neither needs to be the only one that can be extracted from P_u (cf. Sect. 3.1), nor needs to be maximally violated among these.

Fortunately, subsequent and crossing simple cycles may as well be easily recognized algorithmically by only maintaining the positions of the start and end vertices of the *last* subpath identified as to correspond to a simple cycle. Suppose that P_v was the last such subpath on a traversal of P_u, i.e., the positions of v_s and v_e are known. Assume further that the traversal then arrives at some vertex, say w_e, whose other copy, w_s, has already been visited. If $w_s \prec v_s$, then P_v is fully nested within P_w, and thus P_w need not be considered. Otherwise, either $w_s \prec v_e$ in which case P_v and P_w cross, or $v_e \prec w_s$ in which case P_w is subsequent to P_v (cf. Fig. 4). In any case, we can safely update the two maintained positions to those of w_s and w_e. We will then classify later paths correctly since a later path crossing P_v must inevitably either also cross P_w or contain P_w fully nested.

Additionally, a constant-time check to eliminate some unwanted duplicate inequalities in the case of crossing paths P_v and P_w is right at hand from the final statement in Sect. 3.1. If v_e is the immediate predecessor of w_e on P_u, then the same is likely true (sure if computing (ω, ℓ)-shortest paths) for v_s and w_s. This means that $\{v, w\}$ is a chord (being part of the odd closed walk w.r.t. P_u), and the odd-cycle inequalities according to P_v and P_w are equivalent.

3.2.4 Further Engineering of the Extraction Strategy

A further opportunity to improve the practical performance of the separation procedure is to exploit the symmetry of the graph G_S: Suppose that, while computing a shortest u_1-u_2-path in G_S, we process a vertex v_2 (v_1) such that the distances $\mathrm{dist}(v_1)$ and $\mathrm{dist}(v_2)$ are both known. If $\mathrm{dist}(v_1) + \mathrm{dist}(v_2) \geq 1$, then v_2 (v_1) cannot lie on any u_1-u_2-path that induces a violated odd-cycle inequality and thus need not be considered for the update of other distances.

3.3 Chorded Simple Odd Cycles

Finally, as we now know how to extract simple odd cycles, it is natural to strive for chordless ones, i.e. facet-defining inequalities. However, we will show in this subsection, why we will not always find these when searching for *maximally violated* inequalities (as we do when computing ω- or (ω, ℓ)-shortest paths).

For this purpose, let (C, Q) be a simple cycle, and suppose that, for some $\hat{x} \in [0, 1]^E$, the associated odd-cycle inequality is violated, i.e. $\sum_{e \in Q}(1 - \hat{x}_e) + \sum_{e \in C \setminus Q} \hat{x}_e < 1$.

Suppose further that C has a chord c, i.e., there exist vertices $u, v \in V$ such that $c = \{u, v\} \in E \setminus C$. Partitioning C into the corresponding two $\{u, v\}$-paths P_1 and P_2 partitions Q into Q_1 and Q_2, and inevitably renders either $|Q_1|$ odd and $|Q_2|$ even or vice versa (cf. Fig. 5).

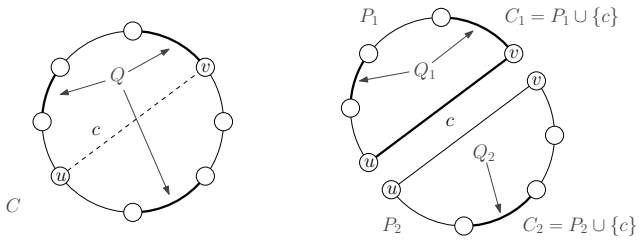

Figure 5 Left: A cycle C with an odd-cardinality edge subset Q (thicker), assumed to have a chord $c = \{u, v\}$. Right: Splitting C into two chordless odd cycles C_1 and C_2, declaring c once as "odd" and once as "even".

Assume w.l.o.g. that $|Q_1|$ is even. The cycles $C_1 = P_1 \cup \{c\}$, with c "marked as odd", and $C_2 = P_2 \cup \{c\}$, with c "marked as even", correspond to the two odd-cycle inequalities

$$\sum_{e \in Q_1} (1 - x_e) + (1 - x_c) + \sum_{e \in P_1 \setminus Q_1} x_e \geq 1 \tag{7}$$

$$\sum_{e \in Q_2} (1 - x_e) + x_c + \sum_{e \in P_2 \setminus Q_2} x_e \geq 1 \tag{8}$$

whose sum gives the original one above assumed to be violated. This implies that at least one of (7) and (8) needs to be violated as well.

Let $\gamma_1 \coloneqq 1 - \sum_{e \in Q_1}(1 - \hat{x}_e) - (1 - \hat{x}_c) - \sum_{e \in P_1 \setminus Q_1} \hat{x}_e$ and $\gamma_2 \coloneqq 1 - \sum_{e \in Q_2}(1 - \hat{x}_e) - \hat{x}_c - \sum_{e \in P_2 \setminus Q_2} \hat{x}_e$ denote the violations of (7) and (8), respectively. Then $\gamma \coloneqq \gamma_1 + \gamma_2$ is the violation of the original inequality. Therefore, one of (7) or (8) can only be found if the other is not violated at all.

It remains to clarify whether we can *efficiently* extract chordless cycles from a simple but not chordless one. This turns out to be the case when restricting to *one* chordless cycle whose associated inequality is violated. In this case, we may use any chord to (conceptually) split the initial cycle (C, Q) into two cycles, and proceed with these in the same fashion if necessary. This can be implemented in a way such that no chord is used for a split more than once, and the adjacency list of each vertex $v \in V(C)$ is traversed at most once (by continuing at the list position of a possible previous visit of the vertex, cf. Fig. 6). Assuming the presence of the respective data structures, and the use of bucket sort to order the adjacency lists appropriately, an asymptotic running time of $\mathcal{O}(|E|)$ can be achieved.

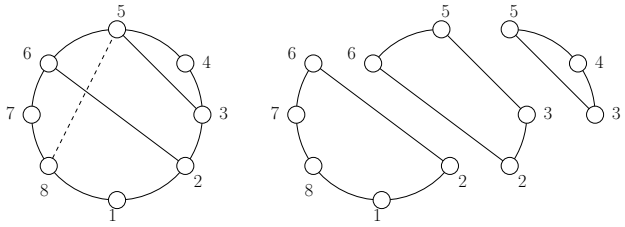

Figure 6 Decomposition of a simple cycle with chords to retrieve *one* chordless cycle. After numbering the vertices consecutively and sorting adjacency lists descendingly w.r.t. this numbering, the decomposition can be carried out in a greedy fashion. If, e.g., the inequality associated to the first chordless cycle depicted on the right turns out not to be violated, it can be neglected by backtracking to vertex 2 and continuing with the next edge of its adjacency list. Even if further subdecompositions take place, the process will never consider any edge more than twice. In particular, no enumerative consideration of chords used as split edges is required – e.g., the chordless cycles related to the chord $\{5, 8\}$ crossing the cycles considered before can be neglected.

4 Experiments

Our experimental study shall particularly address the following two questions:

- Does the "quality" of odd-cycle cutting planes matter, i.e., to what extent is the process of solving the linear programming relaxations of maximum cut and binary quadratic optimization problems affected by the exclusive separation of odd-cycle inequalities that correspond to simple or even chordless simple cycles?
- Is it worthwhile to invest time for the extraction of (chordless) simple cycles (and to save some shortest path computations) rather than to simply employ or discard non-simple cycles?

To this end, we implemented an odd-cycle separation algorithm supporting the "take-as-is"-strategy, the "throw-away"-strategy, the extraction of simple cycles, an improved extraction exploiting the symmetry of the separation graph as described in Sect. 3.2.4, and finally the extraction of one chordless cycle from each simple one. In each of these variants, (ω, ℓ)-shortest paths are computed for each $v \in V$ in general. The three mentioned extraction variants however omit the shortest path computation w.r.t. a vertex $v \in V$ if a simple cycle based on a shortest subpath P_v has been identified before.

Based on this implementation, we solved the linear programming relaxations of several instances from the "binary quadratic and max cut library" [5] by iteratively calling the linear program solver Gurobi[2] in version 8, and our separation algorithm. In case of the MaxCut instances, the initial linear program defined only lower and upper bounds of the variables. In case of BQP instances, we started with all inequalities that correspond to the respective standard linearization. Moreover, after solving a linear program, previously added inequalities not being satisfied at equality, were removed.

For presentation, we selected those instances where *at least one cut was required to solve the relaxation* and the *total time needed* for this was *at most one hour* and *at least half a second*.

4.1 Odd Cycle Separation Quality

The first two experiments relate to the first question mentioned in the beginning of this section. As a first quality indicator, we consider *the total number of inequalities added* while solving the relaxation and when using the different strategies. Since however, a smaller number of cuts might result in an increase of the required number of linear programs to solve, and there is a natural decrease in the number of generated cuts *per iteration* of the extraction and "take-as-is"-strategies compared to the "throw-away"-strategy, we complement this evaluation with another one that considers *the total relaxation solution time*.

In both experiments, the "take-as-is"-strategy serves as the basis (100% level in the figures), and the symmetry-aware version of the extraction strategy is omitted from the presentation, as it generates the same inequalities as the one without symmetry-awareness. Moreover, to reduce side-effects, duplicate inequalities found during any of the variants are not counted and eliminated before passing the cuts to the linear program.

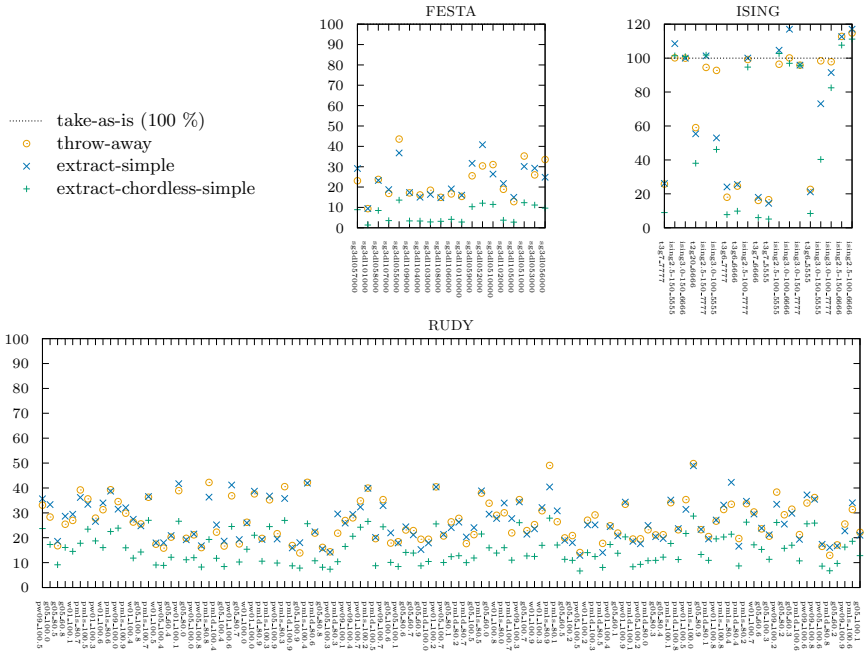

Figure 7 Relative total number of inequalities added (MaxCut instances).

[2] http://www.gurobi.com/

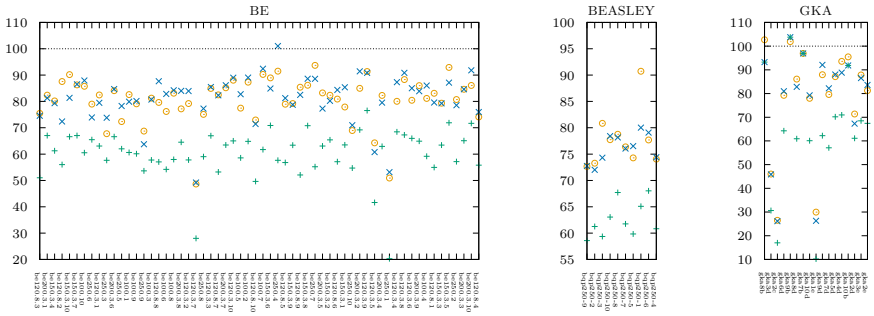

Figure 8 Relative total number of inequalities added (BQP instances).

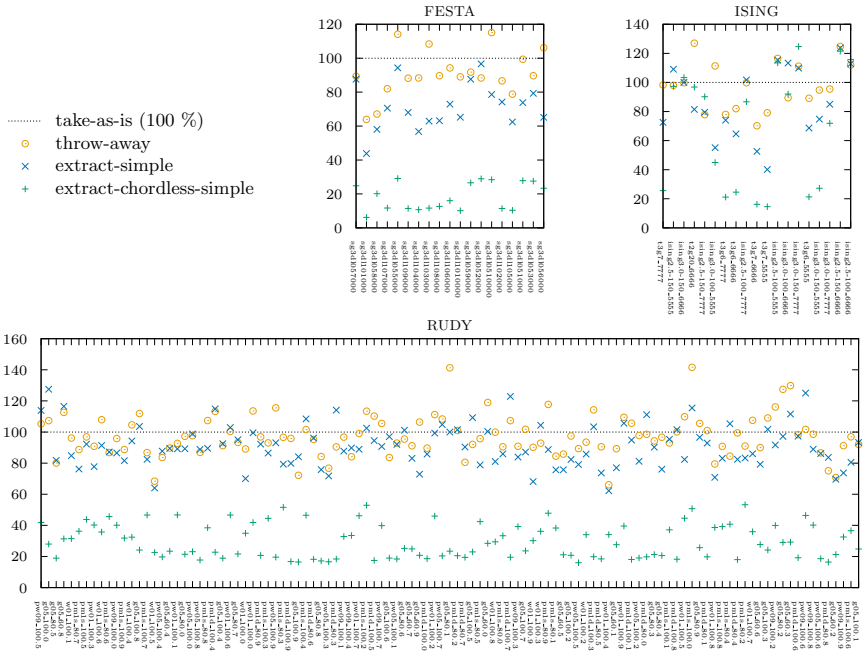

Figure 9 Relative total relaxation solution time (MaxCut instances).

Figs. 7 and 8 show a clear trend of decrease of the total number of inequalities added when restricting to simple or even only chordless simple cycles. Indeed the quality of a cut appears to have a significant impact, as the extraction of the facet-defining chordless simple cycles clearly performs best on average across the instances considered. Moreover, in Figs. 9 and 10, one can see that, in most of the cases, the additional effort to extract simple or one chordless simple cycle does not lead to an increase but a noticeable decrease of the total relaxation solution time. This means in particular that the advantage of requiring fewer cuts is a true one, i.e., not negatively compensated by a significant increase of the number of linear programs required to solve the relaxation.

4.2 Odd Cycle Separation Time

The total relaxation solution time considered in the last experiment is affected by several aspects. In particular, the different runs per instance did not correspond to the same series of linear programs, as different cuts lead to different solutions. To address the second

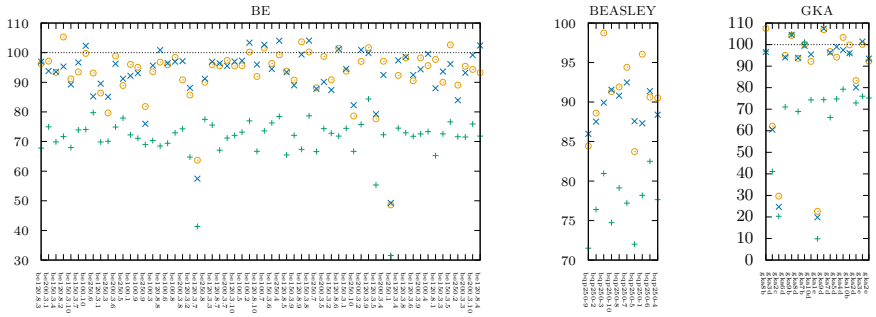

Figure 10 Relative total relaxation solution time (BQP instances).

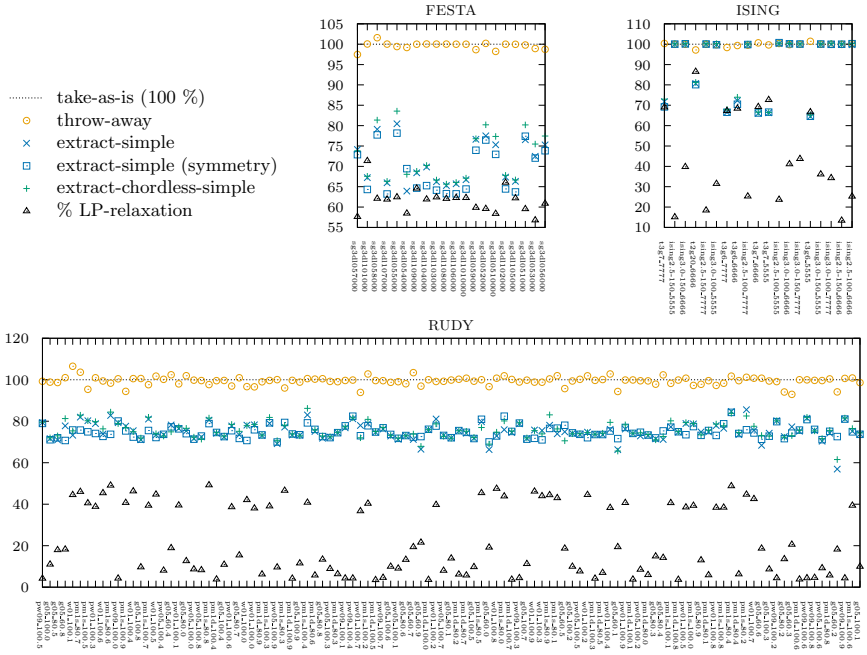

Figure 11 Relative accumulated separation times (MaxCut instances).

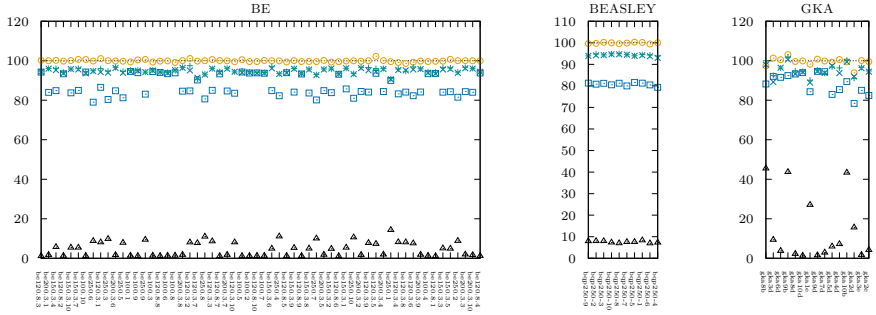

Figure 12 Relative accumulated separation times (BQP instances).

question mentioned at the beginning of this section, i.e., in order to show that simple and even chordless simple cycle extraction truly come at negligible cost – we now create equal preconditions for all separation strategies as follows: After solving a linear program, the separator is called four times, once with each of the different strategies. After that, *one* cutting plane set (the inequalities generated in the "extract-simple"-run) is added to the linear program and the procedure is iterated.

In this experiment, we thus consider the *accumulated time spent in the separation procedure* over all linear programs solved in order to solve the respective relaxation, with the presentation restricted to those instances where this was *at least half a second*. As another difference to the previous experiment, here, duplicate inequalities are not avoided in general. In case of the extraction strategies, they are however avoided in the way described in Sect. 3.2.3.

The results are shown in Figs. 11 and 12, as well as the percentage of time spent in separation w.r.t. the total time to solve the relaxation in the fastest of the four cases (denoted "% LP-relaxation"). As before, the "take-as-is"-strategy (100% level) serves as the basis for comparisons. Across almost all the instances considered, the extraction of simple cycles results in a quicker separation compared to the "take-as-is"- and the "throw-away"-strategy. Not surprisingly, the latter two perform very similarly, as both require a single pass of each identified odd closed walk. Exploiting symmetries usually gives another measurable speedup. This may of course as well be applied to the chordless extraction variant which has anyway not been implemented as efficiently as indicated in Sect. 3.3.

References

1 Francisco Barahona, Michael Jünger, and Gerhard Reinelt. Experiments in quadratic 0–1 programming. *Mathematical Programming*, 44(1):127–137, May 1989. doi:10.1007/BF01587084.

2 Francisco Barahona and Ali Ridha Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2):157–173, June 1986. doi:10.1007/BF02592023.

3 Caterina De Simone, Martin Diehl, Michael Jünger, Petra Mutzel, Gerhard Reinelt, and Giovanni Rinaldi. Exact ground states of Ising spin glasses: New experimental results with a branch-and-cut algorithm. *Journal of Statistical Physics*, 80(1-2):487–496, 1995.

4 Caterina De Simone, Martin Diehl, Michael Jünger, Petra Mutzel, Gerhard Reinelt, and Giovanni Rinaldi. Exact ground states of two-dimensional $\pm J$ Ising spin glasses. *Journal of Statistical Physics*, 84(5):1363–1371, September 1996. doi:10.1007/BF02174135.

5 Franz Rendl, Giovanni Rinaldi, and Angelika Wiegele. A Branch and Bound Algorithm for Max-Cut Based on Combining Semidefinite and Polyhedral Relaxations. In Matteo Fischetti and David P. Williamson, editors, *Integer Programming and Combinatorial Optimization*, pages 295–309, Berlin, Heidelberg, 2007. Springer.