Mathematical Programming Computation (2023) 15:445–470
https://doi.org/10.1007/s12532-023-00236-6

FULL LENGTH PAPER

Faster exact solution of sparse MaxCut and QUBO problems

Daniel Rehfeldt[1] · Thorsten Koch[1,2] · Yuji Shinano[1]

Received: 31 January 2022 / Accepted: 3 February 2023 / Published online: 15 April 2023

Abstract
The maximum-cut problem is one of the fundamental problems in combinatorial optimization. With the advent of quantum computers, both the maximum-cut and the equivalent quadratic unconstrained binary optimization problem have experienced much interest in recent years. This article aims to advance the state of the art in the exact solution of both problems—by using mathematical programming techniques. The main focus lies on sparse problem instances, although also dense ones can be solved. We enhance several algorithmic components such as reduction techniques and cutting-plane separation algorithms, and combine them in an exact branch-and-cut solver. Furthermore, we provide a parallel implementation. The new solver is shown to significantly outperform existing state-of-the-art software for sparse maximum-cut and quadratic unconstrained binary optimization instances. Furthermore, we improve the best known bounds for several instances from the 7th DIMACS Challenge and the QPLIB, and solve some of them (for the first time) to optimality.

Keywords Maximum-cut · Quadratic unconstrained binary optimization · Branch-and-cut

Mathematics Subject Classification 90C27 · 90C10

✉ Daniel Rehfeldt
rehfeldt@zib.de

Thorsten Koch
koch@zib.de

Yuji Shinano
shinano@zib.de

[1] Applied Algorithmic Intelligence Methods departement, Zuse Institute Berlin, Takustr. 7, 14195 Berlin, Germany

[2] TU Berlin, Chair of Software and Algorithms for Discrete Optimization, Str. des 17. Juni 135, 10623 Berlin, Germany

1 Introduction

Given an undirected graph $G = (V, E)$, and edge weights $w : E \to \mathbb{Q}$, the *maximum-cut* (MaxCut) problem is to find a partition (V_1, V_2) of V such that the summed weight of the edges between V_1 and V_2 is maximized. MaxCut is one of the fundamental $\mathcal{NP}$-hard optimization problems [28] and has applications for example in VLSI design [3] and the theory of spin glasses in physics [33].[1] The latter application is particularly interesting, because it requires an exact solution of the MaxCut problem.

A problem that is equivalent to MaxCut is the *quadratic unconstrained binary optimization* (QUBO) problem. Given a matrix $Q \in \mathbb{Q}^{n \times n}$, the corresponding QUBO problem can be formulated as

$$\begin{aligned} \min\ & x^T Qx \\ x \in & \{0, 1\}^n. \end{aligned}$$

Any QUBO instance can be formulated as a MaxCut instance in a graph with $n + 1$ vertices, and any MaxCut instance on a graph (V, E) can be formulated as a QUBO instance with $n = |V| - 1$, see e.g. [4]. The focus of this article is mostly on MaxCut algorithms, but due to the just mentioned equivalence, all results can be (and indeed are) applied to QUBO as well.

The huge recent interest in quantum computing has also put MaxCut and QUBO in the spotlight: Both of them can be heuristically solved by current quantum annealers. However, Jünger et al. [25] demonstrate on a wide range of test-sets that digital computing methods prevail against state-of-the-art quantum annealers.

For digital computers, many heuristics have been proposed both for MaxCut and QUBO. See Dunning et. al. [13] for a recent overview. There have also been various articles on exact solution. See Barahona et al. [4] for an early, Rendl et al. [37] for a more recent, and Jünger et al. [25] for an up-to-date overview. In the last years, more focus has been put on the development of methods that are best suited for *dense* instances, see for example [20, 23, 31] for state-of-the-art methods. However, the maximum number of nodes for MaxCut (or number of variables for QUBO) instances that can be handled by these methods is roughly 300. In contrast, this article aims to advance the state of the art in the practical exact solution of *sparse* MaxCut and QUBO instances. The largest (sparse) instance solved in this article has more than 10,000 nodes.

1.1 Contribution and structure

This article describes the design and implementation of a branch-and-cut based MaxCut and QUBO solver. In particular, we suggest several algorithmic improvements of key components of a branch-and-cut framework.

Section 2 shows how to efficiently solve a well-known linear programming (LP) relaxation for the MaxCut problem by using cutting planes. Among other things, we demonstrate how the separation of maximally violated constraints, which was

[1] As a side note, the 2021 Nobel prize in Physics was awarded for work on spin glasses.

described by many authors as being too slow for practical use, can be realized with quite moderate run times.

Section 3 is concerned with another vital component within branch-and-cut: reduction techniques. We review methods from the literature and propose new ones. The reduction methods can be applied for preprocessing and domain propagation.

Section 4 shows how to integrate the techniques from the previous two sections as well as several additional methods in a branch-and-cut algorithm. Parallelization is also discussed.

Section 5 provides computational results of the newly implemented MaxCut and QUBO solver on a large collection of test-sets from the literature. It is shown that the new solver outperforms the previous state of the art. Furthermore, the best known solutions of several benchmark instances can be improved and one is even solved (for the first time) to optimality.

1.2 Preliminaries and notation

In the remainder of this article, we assume that a MaxCut instance $I_{MC} = (G, w)$ with graph $G = (V, E)$ and edge weights w is given. Graphs are always assumed to be undirected and simple, i.e., without parallel edges or self-loops. Given a graph $G = (V, E)$, we refer to the vertices and edges of any subgraph $G' \subseteq G$ as $V(G')$ and $E(G')$ respectively, An edge between vertices $u, v \in V$ is denoted by $\{u, v\}$. An edge set $C = \{\{v_1, v_2\}, \{v_2, v_3\}, ..., \{v_{k-1}, v_k\}, \{v_1, v_k\}\}$ is called a *cycle*. A cycle C is called *simple* if all its vertices have degree 2 in C. An edge $\{u, w\} \in E \backslash C$ is called a *chord* of C if both u and w are contained in (an edge of) C. If no such $\{u, w\}$ exists, we say that C is *chordless*. Given a graph $G = (V, E)$ and a $U \subseteq V$, we define the induced *edge cut* as $\delta(U) := \{\{u, v\} \in E \mid u \in U, v \in V \setminus U\}$.

Finally, for any function $x : M \mapsto \mathbb{R}$ with M finite, and any $M' \subseteq M$, we define $x(M') := \sum_{i \in M'} x(i)$.

2 Solving the relaxation: efficient separation of odd-cycle cuts

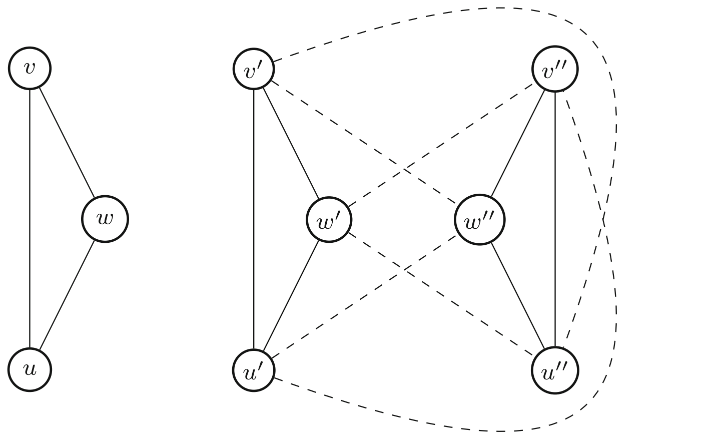
This section is concerned with an integer programming (IP) formulation for MaxCut due to Barahona and Mahjoub [5], given below.

$$\max \quad w^T x \tag{1}$$

$$\text{s.t.} \sum_{e \in F} x(e) - \sum_{e \in C \setminus F} x(e) \leq |F| - 1 \text{ for all cycles } C, F \subseteq C, |F| \text{ odd} \tag{2}$$

$$x(e) \in \{0, 1\} \quad \text{for all } e \in E. \tag{3}$$

The formulation is based on the observation that for any edge cut $\delta(U)$ and any cycle C the number of their common edges, namely $|C \cap \delta(U)|$, is even. This property is enforced by the constraints (2). These constraints are called *cycle inequalities*.



v
w
u
v'
v''
w'
w''
u'
u''
(a) Original graph.
(b) Auxiliary graph, consisting of two copies of the original graph (bold edges), and additional connecting edges (dashed).

Fig. 1 MaxCut graph and corresponding auxiliary graph for cycle cut separation

2.1 Cutting plane separation

Barahona and Mahjoub [5] show that the LP-relaxation of Formulation 1 can be solved in polynomial time. More precisely, they describe how to separate the constraints (2) in polynomial time, as demonstrated in the following. First, rewrite the constraints (2) as

$$\sum_{e \in F} (1 - x(e)) + \sum_{e \in C \setminus F} x(e) \geq 1 \quad \text{for all cycles } C,\ F \subseteq C,\ |F| \text{ odd.} \tag{4}$$

Next, construct a new graph H from the MaxCut graph $G = (V, E)$. This graph H consists of two copies $G' = (V', E')$ and $G'' = (V'', E'')$ of G, connected by the following additional edges. For each $v \in V$ let v' and v'' be the corresponding vertices in G' and G'', respectively. For each edge $\{v, w\} \in E$ let $\{v', w''\}$ and $\{v'', w'\}$ be in H. Finally, for any (LP-relaxation vector) $x \in [0, 1]^E$ define the following edge weights p on H: For each $e = \{v, w\} \in E$, set $p(\{v', w'\}) := p(\{v'', w''\}) := x(e)$ and $p(\{v', w''\}) := p(\{v'', w'\}) := 1 - x(e)$. The construction is exemplified in Fig. 1. Consider, for example, the edge $\{v, w\}$ in Fig. 1a. The weight p of the corresponding (dashed) edges $\{v', w''\}$ and $\{v'', w'\}$ in Fig. 1b is $1 - x(\{v, w\})$. The weight p of the corresponding (bold) edges $\{v', w'\}$ and $\{v'', w''\}$ is $x(\{v, w\})$.

Given an LP-relaxation vector $x \in [0, 1]^E$, we can find violated inequalities (2) as follows. For each $v \in V$ compute a shortest path between v' and v'' in the weighted graph (H, p). By construction of H, such a path contains an odd number of edges which are neither in E' nor E''. Let F be the corresponding set of edges in E; i.e. for each edge $\{v', w''\}$ or $\{v'', w'\}$ that is in the shortest path, let $\{v, w\}$ be in F. Furthermore, the edges of the shortest path correspond to a closed walk C in G. The length of the shortest path in (H, p) is equal to $\sum_{e \in F} (1 - x(e)) + \sum_{e \in C \setminus F} x(e)$. Thus, if for each $v \in V$ the corresponding shortest path between v' and v'' in (H, p) has length at least 1, the vector x is an optimal solution to the LP-relaxation of Formulation 1. Otherwise, we have found at least one violated constraint.

Although shortest paths can be computed in polynomial time, the literature has so far considered the above separation procedure as too time-consuming to be directly used in practical exact MaxCut or QUBO solution. Instead, heuristics are employed and exact cycle separation is only used if no more cuts can be found otherwise, see, e.g., [3, 4, 9, 25, 33]. However, as we will show in the following, the exact separation can actually be realized in a practically quite efficient way.

2.2 Fast computation of maximally-violated constraints

Initially, we observe that it is usually possible to considerably reduce the size of the auxiliary graph H described above. First, all edges e of H with $p(e) = 1$ (or practically, with $p(e)$ being sufficiently close to 1) can be removed. Because all edge weights are non-negative, no such edges can be contained in a path of weight smaller than 1. Second, one can contract edges e with $p(e) = 0$. Both of these operations can be done implicitly while creating the auxiliary graph (e.g., edges with weight 1 are never added). In this way, one can use cache-efficient, static data structures, such as the compressed-sparse-row format, see e.g. [29], for representing the auxiliary graph.

For computing a shortest path, we use a modified version of Dijkstra's algorithm. For any vertex v in the auxiliary graph let $d(v)$ denote the distance of v to the start vertex, as computed by the algorithm. We use the following modifications. First, we stop the execution of the algorithm as soon as we scan a vertex v with $d(v) \geq 1$. Second, as already observed in Jünger and Mallach [26], one does not need to proceed the shortest path computation from any vertex, say v', in the auxiliary graph where the twin vertex, v'', has already been scanned and the following condition holds: $d(v') + d(v'') \geq 1$.

Finally, we use an optimized implementation of Dijkstra's algorithm together with a specialized binary heap. For the latter, we exploit the fact that the values (i.e. vertex indices) of the key, value pairs inserted into the heap are natural numbers bounded by the number of vertices of the auxiliary graph.

2.3 Post-processing

As already mentioned above, the edges of the shortest path computed in the auxiliary graph correspond to a closed walk in G—but not necessarily to a simple cycle. Thus, Jünger and Mallach [26] suggest to extract all simple cycles from such a closed walk and separate the corresponding inequalities. We follow this suggestion (although we note that this modification is performance neutral in our implementation).

Barahona and Mahjoub [5] observe that a cycle inequality is only facet-defining if the corresponding cycle is chordless. If a cycle C has a chord e, one readily obtains two smaller cycles C_1 and C_2 with $C_1 \cup C_2 = C \cup \{e\}$ and $C_1 \cap C_2 = \{e\}$. One verifies that any cycle inequality defined on C can be written as the sum of two cycle inequalities defined on C_1 and C_2, where e is in the odd edges set F of exactly one of the two cycle inequalities. Jünger and Mallach [27] suggest a procedure to extract from any simple cycle C with corresponding violated cycle-inequality a chordless cycle C' whose cycle-inequality is also violated. This procedure runs in $O(|E|)$. However, a disadvantage of this approach is that it finds only one such chordless cycle, which might not be the smallest or most violated one. Additionally, there can be several such chordless cycles. In the following, we suggest a procedure to find several non-overlapping chordless cycles with corresponding violated cycle inequality from a given cycle C with violated cycle inequality.

Consider a simple cycle $C = \{\{v_1, v_2\}, \{v_2, v_3\}, ..., \{v_{k-1}, v_k\}, \{v_1, v_k\}\}$ and let $F \subseteq C$ with $|F|$ odd. Assume there is a vector $x \in [0, 1]^E$ such that the cycle inequality corresponding to C and F is violated, that is:

$$\sum_{e \in F} (1 - x(e)) + \sum_{e \in C \setminus F} x(e) < 1.$$

For each $i = 2, ..., k$ define $P_i := \{\{v_1, v_2\}, \{v_2, v_3\}, ..., \{v_{i-1}, v_i\}\}$ and store the following information.

- $f(i) := |F \cap P_i|$,
- $q(i) := \sum_{e \in F \cap P_i} (1 - x(e)) + \sum_{e \in (C \cap P_i) \setminus F} x(e)$.

This information can be computed in total time $O(|C|)$: Traverse the nodes v_i, $i = 2, 3, .., k$ of C in this order and compute the above two values for i from $i - 1$.

With the above information at hand, traverse for each $i = 2, ..., k$ the incident edges of v_i. Whenever a chord $\{v_i, v_j\}$ with $j < i$ is found, check whether the cycle inequality of one or both of the corresponding cycles is violated. This check can be performed in constant time by using the precomputed information for the indices i and j. For example, if $f(i) - f(j)$ is even, one of the corresponding two cycle inequalities is

$$q(i) - q(j) + 1 - x(\{v_i, v_j\}) \geq 1.$$

If a violated cycle inequality is found, add the corresponding chord together with a flag that indicates which of the two possible cycles is to be used to some (initially empty) queue R. Once the incident edges of all nodes v_i for $i = 2, ..., k$ have been traversed, sort the elements of R according to the size of the corresponding cycles in non-decreasing order. Consider all indices of the original cycle as unmarked. Check the (implicit) cycles in R in non-decreasing order. Let $\{v_i, v_j\}$ with $i < j$ be the corresponding chord. If both i and j are unmarked, mark the indices $i + 1, i + 2, ..., j - 1$. Otherwise, discard the (implicit) cycle. Finally, add all cycle inequalities corresponding to non-discarded cycles to the cut pool. The overall procedure runs in $O(|E| \log(|E|))$. In practice, its run time is completely neglectable.

Finally, we suggest a procedure to obtain additional cycle cuts from the auxiliary graph. This approach is particularly useful for MaxCut instances with few vertices, because the number of generated cycle inequalities separated in each round is limited by the number of vertices of the MaxCut instance (if we ignore additional cycle-inequalities that are possibly found by the above post-processing). The procedure makes use of the symmetry of the auxiliary graph. Assume that we have computed a shortest path between a pair of vertices, say v' and v'', as described above. Recall that

$d(w)$ denotes the distance of any vertex w to the start vertex v'. If there is a twin pair of vertices u', u'' such that none of them are part of the shortest path between v' and v'', and $d(u') + d(u'') < 1$, we can get another violated cycle inequality as follow: First, we take the v'-u' path computed by the algorithm. Second, we consider the v'-u'' path, and transform it to an u''-v'' path (of same length) by exploiting the symmetry of the auxiliary graph. By combining the two paths, we obtain an v'-v'' path of length $d(u') + d(u'')$.

3 Simplifying the problem: reduction techniques

Reduction techniques are a key ingredient for the exact solution of many $\mathcal{NP}$-hard optimization problems, such as Steiner tree [36] or vertex coloring [34]. For QUBO, several reductions methods have been suggested in the literature. Basic techniques can already be found in Hammer et al. [22]. The perhaps most extensive reduction framework is given in Tavares et. al. [10]. Recently, Glover et al. [18] provided efficient realizations and extensions of the classic *first* and *second order derivative* and *co-derivative* techniques [21]. We have implemented the methods from Glover et al. [18] for this article. However, we do not provide details, but rather concentrate on MaxCut reduction techniques in the following.

For MaxCut, there are several articles that discuss reduction techniques for unweighted MaxCut. Ferizovic et al. [14] provide the practically most powerful collection of such techniques. Lange et al. [32] provide techniques for general (weighted) MaxCut instances. In the following, we will describe some of their methods. Furthermore, we suggest new MaxCut reduction methods. Their practical strength will be demonstrated in Sect. 5.

Initially, we note that any edge with weight 0 can be removed from I_{MC}. Any solution to this reduced version of I_{MC} can be extended to a solution of same weight to the original instance (in linear time). Thus, in the following we assume no edges have weight 0. We also note that for the incidence vector $x \in \{0, 1\}^E$ of any graph cut one obtains a corresponding (but not unique) vertex assignment $y \in \{0, 1\}^V$ that satisfies for all $\{u, v\} \in E$ the relation $y(u) \neq y(v) \iff x(\{u, v\}) = 1$. This correspondence will be used repeatedly in the following.

3.1 Cut-based reduction techniques

The first reduction technique from Lange et al. [32] is based on the following proposition.

Proposition 1 [32] *Let $e \in E$ and $U \subset V$ such that $e \in \delta(U)$. If*

$$|w(e)| \geq \sum_{a \in \delta(U) \setminus \{e\}} |w(a)|,$$

then there is an optimal solution $x \in \{0, 1\}^E$ to I_{MC} with $x(e) = \beta$, where $\beta = 1$ if $w(e) > 0$, and $\beta = 0$ if $w(e) < 0$.

Note that in the case of $x(e) = 0$, one can simply contract e. In the case of $x(e) = 1$, one needs to multiply the weights of the incident edges of one of the endpoints of e by -1 before the contraction.

One way to check for all $e \in E$ whether an $U \subset V$ exists such that the conditions of Proposition 1 are satisfied is by using Gomory-Hu trees. We have only implemented a simpler check that considers for an edge $e = \{v, u\} \in E$ the sets $\{v\}$ and $\{u\}$ as U, as already suggested in Lange et al. [32]. A combined check for all edges can be made in $O(|E|)$. We note that this test corresponds to the first order derivative reduction method (mentioned above) for QUBO. This relation can be readily verified by means of the standard transformations between MaxCut and QUBO.

The next reduction technique from Lange et al. [32] is based on triangles, and is given below.

Proposition 2 [32] *Assume there is a triangle in G with edges $\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_2, v_3\}$. Let $V_1 \subset V$ such that $\{v_1, v_2\}, \{v_1, v_3\} \subset \delta(V_1)$, and $V_2 \subset V$ such that $\{v_1, v_2\}, \{v_2, v_3\} \subset \delta(V_2)$. If*

$$-w(\{v_1, v_3\}) - w(\{v_1, v_2\}) \geq \sum_{e \in \delta(V_1) \setminus \{\{v_1,v_3\},\{v_1,v_2\}\}} |w(e)|$$

and

$$-w(\{v_1, v_2\}) - w(\{v_2, v_3\}) \geq \sum_{e \in \delta(V_2) \setminus \{\{v_1,v_2\},\{v_2,v_3\}\}} |w(e)|,$$

then there is an optimal solution $x \in \{0, 1\}^E$ to I_{MC} with $x(\{v_1, v_2\}) = 0$.

Similarly to the previous proposition, we only implemented tests for the simple cases of $\{v_1\}$, $\{v_2\}$, $\{v_1, v_3\}$, and $\{v_2, v_3\}$ for V_1 and V_2, respectively.

In the following, we propose a new reduction test based on triangles, which complements the above one from Lange et al. [32].

Proposition 3 *Assume there is a triangle in G with edges $\{v_1, v_2\}$, $\{v_1, v_3\}$, $\{v_2, v_3\} \in E$ such that $w(\{v_1, v_2\}) > 0$, $w(\{v_1, v_3\}) > 0$, and $w(\{v_2, v_3\}) < 0$. Let $V_1 \subset V$ such that $\{v_1, v_2\}, \{v_1, v_3\} \in \delta(V_1)$ and let $V_2 \subset V$ such that $\{v_1, v_2\}, \{v_2, v_3\} \in \delta(V_2)$. If*

$$w(\{v_1, v_2\}) + w(\{v_1, v_3\}) \geq \sum_{e \in \delta(V_1) \setminus \{\{v_1,v_2\},\{v_1,v_3\}\}} |w(e)|, \tag{5}$$

and

$$w(\{v_1, v_2\}) - w(\{v_2, v_3\}) \geq \sum_{e \in \delta(V_2) \setminus \{\{v_1,v_2\},\{v_1,v_3\}\}} |w(e)|, \tag{6}$$

then there is an optimal solution $x \in \{0, 1\}^E$ to I_{MC} such that $x(\{v_1, v_2\}) = 1$.

Proof Let $x \in \{0, 1\}^E$ be a feasible solution to I_{MC} with $x(\{v_1, v_2\}) = 0$. We will construct a feasible solution $x' \in \{0, 1\}^E$ with $x(\{v_1, v_2\}) = 1$ such that $w^T x' \geq w^T x$. Thus, there exists at least one optimal solution $x \in \{0, 1\}^E$ with $x(\{v_1, v_2\}) = 1$.

Because $x(\{v_1, v_2\}) = 0$, it needs to hold that either

$$x(\{v_1, v_3\}) = x(\{v_2, v_3\}) = 0 \tag{7}$$

or

$$x(\{v_1, v_3\}) = x(\{v_2, v_3\}) = 1. \tag{8}$$

We just consider the case (7); the second one can be handled in an analogeous way. Let $y \in \{0, 1\}^V$ be a vertex assignment corresponding to x; i.e., for all $\{u, v\} \in E$ it holds that $y(u) \neq y(v) \iff x(\{u, v\}) = 1$. Define a new vertex assignment $y' \in \{0, 1\}^V$ as follows

$$y'(v) := \begin{cases} 1 - y(v) & \text{if } v \in V_1 \\ y(v) & \text{otherwise.} \end{cases}$$

Let $x' \in \{0, 1\}^E$ be the cut corresponding to y'; i.e., for all $\{u, v\} \in E$ it holds $x'(\{u, v\}) = 1$ if $y(u) \neq y(v)$, and $x'(\{u, v\}) = 0$ otherwise. Note that for all $e \in E \setminus \delta(V_1)$ it holds that $x'(e) = x(e)$. For all $e \in \delta(V_1)$ it holds that $x'(e) = 1 - x(e)$. In particular,

$$x'(\{v_1, v_2\}) = x'(\{v_1, v_3\}) = 1, \tag{9}$$

because of $x(\{v_1, v_2\}) = x(\{v_1, v_3\}) = 0$. Thus, we obtain

$$\begin{aligned}
\sum_{e \in E} w(e)x'(e) &= \sum_{e \in E \setminus \delta(V_1)} w(e)x'(e) + \sum_{e \in \delta(V_1)} w(e)x'(e) \\
&= \sum_{e \in E \setminus \delta(V_1)} w(e)x(e) + \sum_{e \in \delta(V_1)} w(e)x'(e) \\
&\overset{(9)}{=} \sum_{e \in E \setminus \delta(V_1)} w(e)x(e) \\
&\quad + \sum_{e \in \delta(V_1) \setminus \{\{v_1, v_2\}, \{v_1, v_3\}\}} w(e)x'(e) + w(\{v_1, v_2\}) + w(\{v_1, v_3\}) \\
&\overset{(5)}{\geq} \sum_{e \in E \setminus \delta(V_1)} w(e)x(e) \\
&\quad + \sum_{e \in \delta(V_1) \setminus \{\{v_1, v_2\}, \{v_1, v_3\}\}} (w(e)x'(e) + |w(e)|) \\
&\geq \sum_{e \in E \setminus \delta(V_1)} w(e)x(e) \\
&\quad + \sum_{e \in \delta(V_1) \setminus \{\{v_1, v_2\}, \{v_1, v_3\}\}} w(e)(1 - x'(e)) \\
&= \sum_{e \in E \setminus \delta(V_1)} w(e)x(e) \\
&\quad + \sum_{e \in \delta(V_1) \setminus \{\{v_1, v_2\}, \{v_1, v_3\}\}} w(e)x(e)
\end{aligned}$$

$$
\begin{aligned}
&= \sum_{e \in E \setminus \delta(V_1)} w(e)x(e) + \sum_{e \in \delta(V_1)} w(e)x(e) \\
&= \sum_{e \in E} w(e)x(e),
\end{aligned}
$$

which concludes the proof.

As for the previous triangle test, we only consider the simple cases of $\{v_1\}$, $\{v_2\}$, $\{v_1, v_3\}$, and $\{v_2, v_3\}$ for V_1 and V_2 in our implementation.

Note that Lange et al. [32] furthermore propose a generalization of Proposition 2 to more general connected subgraphs. Also Proposition 3 could be generalized in a similar way. However, since we only implemented reductions tests for the triangle conditions, we do not provide details on this generalization here. We also note that exploiting this more general condition for effective practical reductions is not straight-forward and seems computationally considerably more expensive than the triangle tests.

3.2 Further reduction techniques

In the following, we propose two additional reduction methods, based on different techniques. One uses the reduced-costs of the LP-relaxation of Formulation 1, and one exploits simple symmetries in MaxCut instances.

We start with the latter. If successful, the test based on the following proposition allows one to contract two (possibly non-adjacent) vertices.

Proposition 4 *Assume there are two distinct vertices $u, v \in V$ such that $N(u) \setminus \{v\} = N(v) \setminus \{u\}$. If there exists a non-zero α such that $w(e) = \alpha w(e')$ for all pairs e, e' with $e \in \delta(u) \setminus \{u, v\}$, $e' \in \delta(v) \setminus \{u, v\}$, $e \cap e' \neq \emptyset$, and moreover*

- $\{u, v\} \notin E \ \vee \ w(\{u, v\}) < 0$ *in case of* $\alpha > 0$
- $\{u, v\} \notin E \ \vee \ w(\{u, v\}) > 0$ *in case of* $\alpha < 0$,

then there is an optimal vertex solution $y \in \{0, 1\}^V$ to I_{MC} such that $y(u) = y(v)$ if $\alpha > 0$, and $y(u) = (1 - y(v))$ if $\alpha < 0$.

Proof We consider only the case $\alpha > 0$; the case $\alpha < 0$ can be shown in a similar way. Let $y \in \{0, 1\}^V$ with $y(v) \neq y(u)$. We will construct a $y' \in \{0, 1\}^V$ with $y'(v) = y'(u)$ such that the weight of the induced cut of y' is not lower than the weight of the induced cut of y. In this way, the proof is complete, because we can apply this construction also for any optimal vertex assignment

Let $x \in \{0, 1\}^E$ be the induced cut of y. Assume that

$$
\sum_{e \in \delta(u) \setminus \{u,v\}} w(e)x(e) \geq \alpha \sum_{e \in \delta(v) \setminus \{u,v\}} w(e)x(e). \tag{10}
$$

Otherwise, switch the roles of u and v in the following.

Let $f : \delta(v) \setminus \{\{u, v\}\} \to \delta(u) \setminus \{\{u, v\}\}$ such that $e \cap f(e) \neq \emptyset$ for all $e \in \delta(v) \setminus \{u, v\}$. Note that f is well-defined. Define a new cut $x' \in \{0, 1\}^E$ as follows

$$x'(e) := \begin{cases} x(e) & \text{if } e \in E \setminus \delta(v) \\ x(f(e) & \text{if } e \in \delta(v) \setminus \{\{u, v\}\} \\ 0 & \text{if } e = \{u, v\} \end{cases}$$

Because of (10) and $\{u, v\} \notin E \ \vee \ w(\{u, v\}) < 0$ it holds that $w^T x' \geq w^T x$.

The condition of Proposition 4 can be checked efficiently in practice by using hashing techniques, similar to the ones used for the *parallel rows* test for mixed-integer programs [2].

A well-known reduction method for binary integer programs, which was already used for MaxCut [4], is as follows. Consider a feasible solution $\tilde{x}$ to the LP-relaxation of Formulation 1, with reduced-costs $\tilde{w}$, and with objective value $\tilde{U}$. Further, let L be the weight of a graph cut. If for an $e \in E$ it holds that $\tilde{x}(e) = 0$ and $\tilde{U} - \tilde{w}(e) < L$, one can fix $x(e) := 0$. If for a $e \in E$ it holds that $\tilde{x}(e) = 1$ and $\tilde{U} + \tilde{w}(e) < L$, one can fix $x(e) := 1$. This method can also be used for LP-solutions (obtained during separation) that satisfy only a subset of the cycle inequalities (2). In the following, we will only consider optimal LP-solutions $\tilde{x}$ (possibly for a subset of the cycle inequalities). Since we furthermore consider only LP-solutions obtained by the Simplex algorithm, all non-zero variables have reduced-cost 0.

From incident fixed edges one obtains a (non-unique) partial vertex assignment $y' : V' \to \{0, 1\}$. This assignment can be used to obtain additional fixings, as detailed in the following proposition.

Proposition 5 *Let $\tilde{x}$ be an optimal solution to the LP-relaxation of Formulation 1, with reduced-costs $\tilde{w}$, and objective value $\tilde{U}$. Let L be an upper bound on the weight of a maximum-cut. Let $V' \subset V$ and $y' : V' \to \{0, 1\}$ such that for any optimal vertex assignment $y \in \{0, 1\}^V$ it holds that $y(v) = y'(v)$ for all $v \in V'$. Further, let $u \in V \setminus V'$ and define*

$$\tilde{\Delta}_0 := \sum_{\{u,v\} \in \delta(u) | v \in V', y(v)=0} \tilde{w}(\{u, v\})$$

and

$$\tilde{\Delta}_1 := \sum_{\{u,v\} \in \delta(u) | v \in V', y(v)=1} \tilde{w}(\{u, v\}).$$

For any optimal vertex assignment $y \in \{0, 1\}^V$ the following conditions hold. If $L + \tilde{\Delta}_0 > \tilde{U}$, then $y(u) = 0$. If $L + \tilde{\Delta}_1 > \tilde{U}$, then $y(u) = 1$.

The proposition follows from standard linear programming results. If one of the conditions of the proposition is satisfied, one can fix all edges between u and V'.

4 Solving to optimality: branch-and-cut

This section describes how to incorporate the methods introduced so far together with additional components in an exact branch-and-cut algorithm. This branch-and-cut algorithm has been implemented based on the academic MIP solver SCIP [7]. Besides being a stand-alone MIP solver, SCIP provides a general branch-and-cut framework. Most importantly, we rely on SCIP for organizing the branch-and-bound search, and the cutting plane management. Most native, general-purpose algorithms of SCIP such as primal heuristics, conflict analysis, or generic cutting planes are deactivated by our solver for performance reasons.

4.1 Key components

In the following, we list the main components of the branch-and-cut framework that was implemented for this article.

Presolving For presolving, the reduction methods described in this article are executed iteratively within a loop. This loop is reiterated as long as at least one edge has been contracted during the previous round, and the predefined maximum number of loop passes has not been reached yet.

Domain propagation For domain propagation we use the reduced-cost criteria described in Sect. 3.2. The simple single-edge fixing is done by the generic reduced-costs propagator plug-in of SCIP. For the new implication based method we have implemented an additional propagator.

A classic propagation method, e.g. [4], is as follows: Consider the connected components induced by edges that have been fixed to 0 or 1. All additional edges in these connected components can be readily fixed. However, this technique brought no benefits in our experiments, since the variable values of such edges are implied by the cycle inequalities (2).

Decomposition It is well-known that connected components of the graph underlying a MaxCut instance can be solved separately, see e.g. [25]. More generally, one can solve biconnected components separately (this simple observation does not seem to have been mentioned in the MaxCut literature so far). Since several benchmark instances used in this article contain many very small biconnected components, we solve components with a limited number of vertices by enumeration. In this way, we avoid the overhead associated with creating and solving a new MaxCut instance for each subproblem.

Primal heuristics Primal heuristics are an important component of practical branch-and-bound algorithms: First, to find an optimal solution (verified by the dual-bound), and second to find strong primal bounds that allow the algorithm to cut off many branch-and-bound nodes. For computing an initial primal solution, we have implemented the MaxCut heuristic by Burer et. al. [11]. We further use the Kernighan-Lin algorithm [30] to improve any (intermediary) solution found by the algorithm of Burer et. al. Additionally, we use this combined algorithm as a local search heuristic whenever a new best primal solution has been found during the branch-and-bound search. In this case, we initiate the heuristic with this new best solution (which can be done by translating the solution into the two-dimensional angle vectors required by the heuristic).

We also implemented the spanning-tree heuristic from Barahona et al. [4], which uses a given (not necessarily optimal) LP-solution to find graph cuts. We execute this heuristic after each separation round.

Separation In each separation round, we initially try to find violated cycle inequalities on triangles of the underlying graph (by enumerating some triangles). Moreover, we use a heuristic strategy based on a minimum spanning tree computation to obtain additional odd-cycle cuts, see [4] for details. Next, we use shortest-path computations to find (locally) maximally violated cuts, as described in Sects. 2.2 and 2.3. Among the speed-up techniques, we have not (yet) implemented the contraction of edges, since the separation routine is already quite fast and other implementations seemed more promising. Finally, we also use the *odd-clique-cuts* introduced in [5]. For separating these cuts, we initially enumerate a (bounded) number of cliques of size 5, 7, and 9 at the beginning of the solving process. In each separation round, we check whether some of the corresponding cuts are violated, and if so, add them to the LP. Overall, we bound the number of cuts that can be added per separation round.

Branching We simply branch on the edge variables and use the *pseudo-costs* branching strategy of SCIP, see Gamrath [16] for more details. Initial experiments showed that the default branching strategy of SCIP, *reliable pseudo-costs* branching, spends too much time on strong-branching to be competitive.

4.2 Parallelization

For parallelizing our solver, we use the *Ubiquity Generator Framework* (UG) [38], a software package to parallelize branch-and-bound based solvers—for both shared- and distributed-memory environments. UG implements a *Supervisor-Worker load coordination scheme* [35]. Importantly, Supervisor functions make decisions about the load balancing without actually storing the data associated with the branch-and-bound (B&B) search tree.

A major problem of parallelizing the B&B search lies in the simple fact that parallel resources can only be used efficiently once the number of open B&B nodes is sufficiently large. Thus, we employ so-called *racing ramp-up* [35]: Initially, each thread (or process) starts the solving process of the given problem instance, but each with different (customized) parameters and random seeds. Additionally, we reserve some threads to exclusively run primal heuristics. During the racing, information such as improved primal solutions or global variable fixings is being exchanged among the threads. We terminate the racing once a predefined number of open B&B nodes has been created by one thread, or the problem has been solved. Once the racing has been terminated and the problem is still unsolved, the open nodes are distributed among the threads and the actual parallel solving phase starts.

Possible further paralleizations would be of the cutting plane generation or the solution of the linear programs. However, as demonstrated in Sect. 5.1, the cutting plane generation only requires a small portion of the entire solving time, so parallizing it would have little overall impact. The parallelization of the linear program (re-)optimizations is possible, but since a Simplex algorithm is used (which is notoriously hard to parallelize), using more than one thread also has little impact.

5 Computational results

This section provides computational results on a large collection of MaxCut and QUBO instances from the literature. We look at the impact of individual components, and furthermore compare the new solver with the state of the art for the exact solution of MaxCut and QUBO instances. An overview of the test-sets used in the following is given in Table 1. The second column gives the number of instances per test-sets. The third and fourth columns give the range of nodes and edges in the case of MaxCut, or the range of variables and non-zero coefficients in the case of QUBO.

Only a few exact MaxCut or QUBO solvers are publicly available, and some, such as BiqMac [37], only as web services. Still, the state-of-the-art solvers BiqBin [20], BiqCrunch [31] and MADAM [23] are freely available. However, we have observed that all of these solvers are outperformed on most instances listed in Table 1 by the recent release 9.5 of the state-of-the-art commercial solver Gurobi [19]. Gurobi solves mixed-integer quadratic programs, which are a superclass of QUBO. In fact, the standard benchmark library for quadratic programs, QPLIB [15], contains various QUBO instances. Compared to the previous release 9.1, Gurobi 9.5 has hugely improved on QUBO (and thereby also MaxCut) instances. For example, while Gurobi 9.1 could not solve any of the *IsingChain* instances from Table 1 in one hour (with one thread), Gurobi 9.5 solves all of them in less than one minute. Thus, in the following, we will use Gurobi 9.5 as a reference for our new solver. We will also provide results from the literature, but the comparison with Gurobi 9.5 allows us to obtain results in the same computational environment. Very recently, an article describing a new solver, called McSparse, specialized to sparse MaxCut and QUBO instances was published [12]. The computational experiments in [12] demonstrate that McSparse outperforms previous MaxCut and QUBO solvers on sparse instances. Like BiqMac [37] and BiqBin [20], this solver is only available via a web interface. However, we will still provide some comparison with our solver in the following by using the results published in [12].

The computational experiments were performed on a cluster of Intel Xeon Gold 5122 CPUs with 3.60 GHz, and 96 GB RAM per compute node. We ran only one job per compute node at a time, to avoid a distortion of the run time measures—originating for example from shared (L3) cache. For our solver, we use the commerical CPLEX 12.10 [24] as LP-solver, although our solver also allows for the use of the non-commercial (but slower) SoPlex [7] instead. For Gurobi we set the parameter *MipGap* to 0. Otherwise, we would obtain suboptimal solutions even for many instances with integer weights.

Table 1 Details of MaxCut (upper part) and QUBO (lower part) test-sets used in this article

Name	#	|V|	|E|	Description
DIMACS	4	512–3375	1536–10125	Instances introduced at the 7th DIMACS Challenge
Mannino	4	48–487	1128–8511	Instances from a radio frequency assignment problem introduced in [9]
$PM1s_{100}$	10	100	495	Instances generated with the *rudy* framework from the BiqMac Lib [40]
$W01_{100}$	10	100	495	Instances generated with the *rudy* framework from the BiqMac Lib [40]
K64-chimera	80	2049	8064	Instances on D-Wave Chimera graphs introduced in [25]
Kernel	14	33–2888	91–2981	Instances from various sources collected by [14]
IsingChain	30	100–300	4950–44850	Instances from an application in statistical physics introduced in [33]
Torus	18	100–343	200–1029	2D and 3D torus instances from an application in statistical physics, introduced in [33]
Name	#	*n*	*nnz*	Description
QPLIB	22	120–1225	602–34876	QUBO instances from the standard benchmark software for quadratic programs, see [15]
BQP100	10	100	471–528	Randomly generated instances introduced in [6]

Table 1 continued

Name	#	n	nnz	Description
BQP250	10	250	3039–3208	Randomly generated instances introduced in [6]
BE120.3	10	120	2176–2253	Randomly generated instances introduced in [8]
BE250	10	250	3268–3388	Randomly generated instances introduced in [8]
GKA$_{a\text{-}d}$	35	20–125	204–7788	Randomly generated instances with different densities introduced in [17]

Table 2 Average times spent in separation and (re-) optimization of the LP for MaxCut and QUBO test-sets

Name	#	Sepa-time [%]	LP-time [%]
BE120.3	10	2.5	79.3
BE250	10	2.8	84.3
BQP100	10	3.0	32.0
BQP250	10	2.6	86.2
GKA$_{a\text{-}d}$	35	6.7	49.1
IsingChain	30	0.0	0.0
K64-chimera	80	10.1	58.3
Kernel	14	1.3	4.8
PM1s$_{100}$	10	13.1	78.3
QPLIB	22	18.1	65.5
Torus	18	16.7	15.5
W01$_{100}$	10	12.0	59.3

5.1 Individual components

This section takes a look at individual algorithmic components introduced in Sects. 2 and 3.

First, we show the run time required for our improved separation of cycle inequalities. Table 2 reports per test-set the average (arithmetic mean) percentual time required for the separation procedure (column four), as well as for solving the LP-relaxations (column five). Recall that the latter is done by CPLEX 12.10, one of the leading commercial LP-solvers. For more than half of the test-sets the average time required for the separation is less than 10 %. Also for the remaining test-sets it is always less than 20 %. Notably, this time also includes adding the cuts (including the triangle inequalities) to the cut pool, which requires additional computations. The time could be further reduced by contracting 0-weight edges in the auxiliary graph, as described in Sect. 2. Notably, both the separation time and LP-solution time are very small for the *IsingChain* and *Kernel* instances. This behavior is due to the fact that many of these instances are already solved during presolving, as detailed in the following,

Next, we demonstrate the strength of the reduction methods implemented for this article. Only results for the MaxCut test-sets are reported. We show the impact of the MaxCut reduction techniques from [32] described in Sect. 3 as well as the QUBO reduction techniques from [18]—by using the standard problem transformations between QUBO and MaxCut. We refer to the combination of these two as *base preprocessing*. Additionally, the methods described in Propositions 3 and 4 are referred to as *new techniques*. Note that Proposition 5 cannot be applied, because no reduced-costs are available.

Table 3 shows in the first column the name of the test-set, followed by its number of instances. The next columns show the percentual average number of nodes and edges of the instances after the preprocessing without (column three and four), and with (columns five and six) the new methods. The last two columns report the percentual

Table 3 Average remaining size of MaxCut instances after preprocessing

Test-set	#	Base preprocessing		+new techniques		Relative change	
		|V| [%]	|E| [%]	|V| [%]	|E| [%]	|V| [%]	|E| [%]
IsingChain	30	6.1	0.8	1.1	<0.05	**−82.0**	**<−93.8**
K64-chimera	80	3.1	4.6	3.1	4.6	0.0	0.0
Kernel	14	24.1	30.1	16.4	20.6	**−32.0**	**−31.6**
Mannino	4	64.1	69.3	63.2	68.7	**−1.4**	**−0.9**
Torus	18	80.6	87.5	78.5	85.2	**−2.6**	**−2.6**
W01$_{100}$	10	99.1	94.8	99.1	94.8	0.0	0.0
DIMACS	4	97.0	98.9	96.9	98.9	**−0.1**	0.0
PM1s$_{100}$	10	99.7	99.9	99.7	99.9	0.0	0.0

Bold numbers signify a favorable impact of the new algorithms

relative change between the previous results. The run time is not reported, because it is on all instances below 0.05 s.

The new reduction techniques have an impact on five of the eight test-sets. The strongest reductions occur on *Kernel* and *IsingChain*. We remark that the symmetry-based reductions from Proposition 4 have a very small impact, and only allow for contracting a few dozen additional edges on *Kernel*. We also note that while the *IsingChain* instances are already drastically reduced by the base preprocessing, the new methods still have an important impact, as they reduce the number of edges of several instances from more than a thousand to less than 300. The *IsingChain* instances were already completely solved by reduction techniques in Tavares [39], by using maximum-flow based methods. However the run time was up to three orders of magnitudes larger than in our case. The machine used by Tavares had a Pentium 4 CPU at 3.60 GHz, thus being significantly slower than the machines used for this article. Still, also when taking the different computing environments into account, the run time difference is huge.

5.2 Exact solution

This section compares our new solver with state-of-the-art exact solvers with respect to the mean time, the maximum time, and the number of solved instances. For the mean time we use the shifted geometric mean [1] with a shift of 1 second. In this section, we use only single-thread mode.

First, we provide a comparison with Gurobi 9.5. Table 4 provides the results for a time-limit of one hour. The second column shows the number of instances in the test-set. Columns three gives the number of instances solved by Gurobi, column four the number of instances solved by our solver. Column five and six show the mean time taken by Gurobi and our solver. The next column gives the relative speedup of our solver. The last three columns provide the same information for the maximum run time, Speedups that signify an improved performance of the new solver are marked in bold.

It can be seen that our solver consistently outperforms GUROBI 9.5—both with respect to mean and maximum time. Also, it solves on each test-set at least as many instances as GUROBI. The only test-set where GUROBI performs better is *BQP100*, which, however, can be solved by both solvers in far less than a second.

On the other test-sets, the mean time of the new solver is better, often by large factors (up to 60.07). On the instance sets that can both be fully solved, the maximum time taken by the new solver is in most cases also much smaller. On five of the test-sets, the new solver can solve more instances to optimality than GUROBI 9.5.

Next, we compare our solver with the very recent QUBO and MaxCut solver MCSPARSE, specialized on sparse instances. In Table 5 we provide an instance-wise comparison of our solver and MCSPARSE. We provide the number of branch-and-bound nodes (columns three and four) and the run times (columns five and six) of MCSPARSE and our solver per problem instance. We use the 14 instances that were selected in the article by Charfreitag et al. [12] as being representative of their test-bed. The first seven instances are MaxCut and the last seven QUBO problems. Charfreitag et al. [12] only use one thread per run. Their results were obtained on a system with AMD EPYC 7543P CPUs at 2.8 GHz, and with 256 GB RAM. CPU benchmarks [2] consider this system to be faster than the one used in this article, already in single-thread mode. Furthermore, MCSPARSE is embedded into GUROBI (version 9.1), which is widely regarded as the fastest commercial MIP-solver, whereas our solver is based on the non-commercial SCIP, although we also use a commercial LP-solver.

As to the number of branch-and-bound nodes, the pictures is somewhat mixed—with MCSPARSE requiring fewer nodes on three, and more nodes on four instances. Notably, MCSPARSE also includes a specialized branching strategy, while we use a simple generic one. This feature might explain the smaller number of nodes on some instances. As to the run time, five instances can be solved in less than a second by both solvers (with the new solver being slightly faster). On the remaining nine instances, the new solver is always faster—for all but one instances by a factor of more than 3. On one instance (mannino_k487b), it is even by a factor of more than 40 faster.

Finally, we also provide a few remarks concerning dense instances, although these are not the focus of this article. It is often reported, see e.g. [37], that LP-based approaches using odd-cycle cuts do not work well on dense instances. However, dense instances from the literature are typically randomly generated. We have indeed observed that our solver is not at all competitive with semidefinite-based solvers for randomly generated dense instances with more than around 80 vertices (although these solver typically cannot handle instances with more than 250 vertices). However, the picture can be different on real-world instances. We exemplarily demonstrate this behavior on two test-sets from the literature. First, we selected the test-set PW05 [40], which consists of nine randomly generated instances with 100 vertices and 50 % density each. Second, we selected the IsingChain test-set from Table 1, which consists of instances with up to 50 % density. We compare our solver with the state-of-the-art semidefinite solver BIQBIN. On the (randomly generated) PW05 instances, BIQBIN performs vastly better, solving all instances with a mean time of 64 s. In contrast, our solver cannot solve any of them within one hour, and the primal-dual gaps are

[2] https://www.cpubenchmark.net/singleThread.html#server-thread.

Table 4 Comparison of GUROBI 9.5 (*Grb*) and new solver (*new*)

Test-set		# solved		Mean time (sh. geo. mean)			Maximum time		
	#	Grb	New	Grb [s]	New [s]	Speedup	Grb [s]	New [s]	Speedup
PM1s$_{100}$	10	10	10	192.3	20.9	**9.20**	303.3	48.4	**6.27**
W01$_{100}$	10	10	10	44.1	3.1	**14.23**	97.1	21.5	**4.52**
Kernel	14	14	14	0.7	0.1	**7.00**	14.3	1.1	**13.00**
IsingChain	30	30	30	1.3	<0.05	>**26.00**	41.0	<0.05	>**820.00**
Torus	18	18	18	3.8	0.4	**9.50**	628.0	7.6	**82.63**
K64-chimera	80	80	80	90.1	1.5	**60.07**	195.4	6.0	**32.57**
QPLIB	22	8	**13**	687.4	165.5	**4.15**	3600	3600	1.00
BQP100	10	10	10	0.1	0.1	1.00	0.2	0.3	0.67
BQP250	10	0	**7**	3600	610.6	**5.90**	3600	3600	1.00
BE120.3	10	9	**10**	265.6	50.1	**5.30**	3600	525.1	> **6.86**
BE250	10	0	**8**	3600	571.8	**6.30**	3600	3600	1.00
GKA$_{a\text{-}d}$	35	29	**31**	6.5	6.1	**1.07**	3600	3600	1.00

Bold numbers signify a favorable impact of the new algorithms

Table 5 Comparison of McSparse (*MS*) and our solver (*new*) on seven MaxCut and seven QUBO instances (considered to be representative [12])

Name	|V|	|E|	# B&B nodes		Run time	
			MS	New	MS [s]	New [s]
pm1s_100.3	100	495	341	741	48.2	48.0
pw01_100.0	100	495	171	179	20.0	8.5
mannino_k487b	487	5391	1	15	167.3	4.3
bio-diseasome	516	1188	1	1	9.5	0.6
ca-netscience	379	914	1	1	0.1	0.0
g000981	110	188	1	1	0.0	0.0
imgseg_138032	12,736	23,664	1	1	30.5	3.9
Name	n	nnz	MS	New	$MS\,[s]$	$New\,[s]$
bqp250-3	250	3092	25	17	414.1	84.1
gka2c	50	813	1	1	0.5	0.3
gka4d	100	2010	129	9	219.6	43.7
gka5c	80	721	1	1	0.1	0.1
gka7a	30	241	1	1	0.0	0.0
be120.3.5	120	2248	111	15	257.7	46.6
be250.3	250	3277	107	47	841.0	150.7

up to 4.4%. It should be mentioned that our solver does not perform any better on most randomly-generated instances with 100 or more vertices from the literature. The (real-world) IsingChain, instances, on the other hand, are all solved within 0.1 s by our solver. In contrast, the mean time of BiqBin is 144 s on these instances and four of the instances cannot be solved within one hour. These four include instances with density of around 50 %.

5.3 Parallelization

Although parallelization is not the main topic of this article, we still provide some corresponding results in the following. To give insights into the strengths and weaknesses of our racing-based parallelization, we provide instance-wise results. We use the test-sets *Mannino* and *DIMACS*, which both contain instances that cannot be solved within one hour by Gurobi and our new solver in single-thread mode. The sizes of the instances are given in Table 6.

Table 7 provides results of Gurobi and the new solver on the *DIMACS* and *Mannino* instances. Both solvers are run once with one thread and once with eight threads. As before, a time-limit of one hour is used. The table provides the percentual primal-dual gap, as well as the run time. The results reveal for both solvers a performance degradation on easy instances with increased number of threads. Most notably on *mannino_k487b*, where Gurobi takes almost 10 times longer with eight threads. On the other hand, the new solver shows a strong speedup on two hard instances that cannot

Table 6 Details on *DIMACS* (left) and *Mannino* (right) instances

Name	| V |	|E|	Name	|V|	|E|
torusg3-8	512	1536	mannino_k487a.mc	487	1435
toruspm3-8-50	512	1536	mannino_k487b.mc	487	5391
torusg3-15	3375	10125	mannino_k487c.mc	487	8511
toruspm3-15-50	3375	10125	mannino_k48.mc	48	1128

be solved in one hour singke-threaded, namely *toruspm3-8-50* and *mannino_k487c*. On the latter, one even observes a super-linear speedup. This speedup can be at least partly attributed to the exclusive use of primal heuristics on one thread during racing, which finds an optimal solution quickly in both cases. On the other hand, in single-thread mode the best primal solution is sub-optimal even at the time-limit.

Finally, Table 8 provides results for several previously unsolved MaxCut and QUBO benchmark instances from the *QPLIB* and the 7th DIMACS Challenge. We also report the previous best known solution values (*previous primal*) from the literature, which were taken from the *QPLIB* and the *MQLib* [13]. For the *QPLIB* instances we report the results from the one hour single-thread run in Sect. 5.2. However, for the DIMACS instances, *torusg3-15* and *toruspm3-15-50*, we performed additional runs. Note that the DIMACS instances were originally intended to be solved with negated weights. However, it seems that most publications, e.g., [13], do not perform this transformation. Thus, we also use the unmodified instances, to allow for better comparison. However, we additionally report the solution values of the transformed instances, these transformed instances are marked by a $\star$. We used a machine with 88 cores of Intel Xeon E7-8880 v4 CPUs @ 2.20GHz. We ran the two instances (non-exclusively) for at most 3 days while using 80 threads. Both *torusg3-15* and *torusg3-15** could be solved to optimality in this way, but *toruspm3-15-50* and *toruspm3-15-50** still remained with a primal-dual gap of 1.8 percent each.

Finally, we note that using more than 8 threads (on the above machine with 88 cores) does not provide additional speed-ups on most instances, neither for our solver nor Gurobi. This behaviour can be put down to the fact that we parallelize mostly the branch-and-bound search, and the number of open branch-and-bound nodes is usually quite small on most instances.

6 Conclusion and outlook

This article has demonstrated how to design a state-of-the-art solver for sparse QUBO and MaxCut instances, by enhancing and combining key algorithmic ingredients such as presolving and cutting-plane generation. The newly implemented solver outperforms both the leading commercial and non-commercial competitors on a wide range of test-sets from the literature. Moreover, the best known solutions to several instances could be improved.

Table 7 Results of GUROBI 9.5 (*Grb*) and the new solver (*new*), with one (*-T1*) and eight (*-T8*) threads each

Name	Primal-dual gap [%]				Run time [s]			
	Grb-T1	Grb-T8	New-T1	New-T8	Grb-T1	Grb-T8	New-T1	New-T8
torusg3-8	0.0	0.0	0.0	0.0	1494.2	1178.5	8.5	9.3
toruspm3-8-50	1.8	1.8	0.5	0.0	>3600	>3600	>3600	1415.8
torusg3-15	6.8	3.4	1.3	0.4	>3600	>3600	>3600	>3600
toruspm3-15-50	9.5	12.2	2.3	2.3	>3600	>3600	>3600	>3600
mannino_k487a	0.0	0.0	0.0	0.0	3.5	10.7	1.1	1.3
mannino_k487b	0.0	0.0	0.0	0.0	9.2	80.5	4.3	2.8
mannino_k487c	0.1	0.0	0.1	0.0	>3600	3176.7	>3600	398.2
mannino_k48	0.0	0.0	0.0	0.0	0.1	0.4	2.7	3.8

Table 8 Improved solutions for MaxCut (first four) and QUBO (last two) benchmark instances

Name	gap [%]	New primal	Previous primal
torusg3-15	**opt**	**286626481**	282534518
torusg3-15*	**opt**	**292031950**	–
toruspm3-15-50	1.8	3010	2968
toruspm3-15-50*	1.8	3008	–
QPLIB_3693	1.0	−1154	−1148
QPLIB_3850	1.1	−1198	−1192

Bold numbers signify that an optimal solution was found

For QUBO and MaxCut instances with not more than 10 % density, the computational results obtained for this article strongly suggest the use of our new solver. However, for instances with 20 % or more density, using semidefinite programming based solvers such as BiqBin [20], BiqCrunch [31] and MADAM [23] seems usually far more promising. However, one should keep in mind that most dense instances from the literature are randomly generated. For instances from real-world applications the picture can be somewhat different, especially when presolving is effective (which is not included in any of the mentioned solvers). A prominent example are the dense Ising chain instances discussed in this article, which can all be solved in less than 0.1 seconds by our solver, but some of which cannot be solved even in one hour by a state-of-the-art semidefinite programming based solver.

There are various promising routes for further improvement. Examples would be a new branching strategy, or the implementation of additional separation methods [9]. In this way, a considerable further speedup of the new solver might be achieved.

Funding Open Access funding enabled and organized by Projekt DEAL. The work for this article has been conducted in the Research Campus MODAL 646 funded by the German Federal Ministry of Education and Research (BMBF) (fund numbers 05M14ZAM, 647 05M20ZBM).

Data availibility The data used for the computational experiments in this article is available on the websites: https://biqmac.aau.at/biqmaclib.html. http://bqp.cs.uni-bonn.de/library/html/index.html. https://qplib.zib.de.

Code availability The full code was made available to the reviewers

Conflict of interest The authors declare no conflict of interest.

References

1. Achterberg, T.: Constraint integer programming. In: Ph.D. Thesis, Technische Universität Berlin (2007)

2. Achterberg, T., Bixby, R.E., Gu, Z., Rothberg, E., Weninger, D.: Presolve reductions in mixed integer programming. INFORMS J. Comput. **32**(2), 473–506 (2020). https://doi.org/10.1287/ijoc.2018.0857
3. Barahona, F., Grötschel, M., Jünger, M., Reinelt, G.: An application of combinatorial optimization to statistical physics and circuit layout design. Oper. Res. **36**(3), 493–513 (1988). https://doi.org/10.1287/opre.36.3.493
4. Barahona, F., Jünger, M., Reinelt, G.: Experiments in quadratic 0–1 programming. Math. Program. **44**(1–3), 127–137 (1989). https://doi.org/10.1007/BF01587084
5. Barahona, F., Mahjoub, A.R.: On the cut polytope. Math. Program. **36**(2), 157–173 (1986). https://doi.org/10.1007/BF02592023
6. Beasley, J.: Heuristic algorithms for the unconstrained binary quadratic programming problem. Tech. Rep. (1998)
7. Bestuzheva, K., Besançon, M., Chen, W.K., Chmiela, A., Donkiewicz, T., van Doornmalen, J., Eifler, L., Gaul, O., Gamrath, G., Gleixner, A., Gottwald, L., Graczyk, C., Halbig, K., Hoen, A., Hojny, C., van der Hulst, R., Koch, T., Lübbecke, M., Maher, S.J., Matter, F., Mühmer, E., Müller, B., Pfetsch, M.E., Rehfeldt, D., Schlein, S., Schlösser, F., Serrano, F., Shinano, Y., Sofranac, B., Turner, M., Vigerske, S., Wegscheider, F., Wellner, P., Weninger, D., Witzig, J.: The SCIP Optimization Suite 8.0. ZIB-Report 21-41, Zuse Institute Berlin (2021). http://nbn-resolving.de/urn:nbn:de:0297-zib-85309
8. Billionnet, A., Elloumi, S.: Using a mixed integer quadratic programming solver for the unconstrained quadratic 0–1 problem. Math. Program. **109**(1), 55–68 (2007). https://doi.org/10.1007/s10107-005-0637-9
9. Bonato, T., Jünger, M., Reinelt, G., Rinaldi, G.: Lifting and separation procedures for the cut polytope. Math. Program. **146**(1–2), 351–378 (2014). https://doi.org/10.1007/s10107-013-0688-2
10. Boros, E., Hammer, P.L., Tavares, G.: Preprocessing of unconstrained quadratic binary optimization. Tech. Rep. (2006)
11. Burer, S., Monteiro, R.D.C., Zhang, Y.: Rank-two relaxation heuristics for MAX-CUT and other binary quadratic programs. SIAM J. Optim. **12**(2), 503–521 (2002). https://doi.org/10.1137/S1052623400382467
12. Charfreitag, J., Jünger, M., Mallach, S., Mutzel, P.: McSparse: Exact solutions of sparse maximum cut and sparse unconstrained binary quadratic optimization problems. In: Phillips, C.A., Speckmann, B. eds.) Proceedings of the Symposium on Algorithm Engineering and Experiments (ALENEX 2022), online first (2022)
13. Dunning, I., Gupta, S., Silberholz, J.: What works best when? A systematic evaluation of heuristics for max-cut and QUBO. INFORMS J. Comput. **30**(3), 608–624 (2018). https://doi.org/10.1287/ijoc.2017.0798
14. Ferizovic, D., Hespe, D., Lamm, S., Mnich, M., Schulz, C., Strash, D.: Engineering Kernelization for Maximum Cut. In: Blelloch, G.E., Finocchi, I. (eds.) Proceedings of the Symposium on Algorithm Engineering and Experiments, ALENEX 2020, Salt Lake City, UT, USA, January 5-6, 2020, pp. 27–41. SIAM (2020). doi:https://doi.org/10.1137/1.9781611976007.3
15. Furini, F., Traversi, E., Belotti, P., Frangioni, A., Gleixner, A.M., Gould, N., Liberti, L., Lodi, A., Misener, R., Mittelmann, H.D., Sahinidis, N.V., Vigerske, S., Wiegele, A.: QPLIB: a library of quadratic programming instances. Math. Program. Comput. **11**(2), 237–265 (2019). https://doi.org/10.1007/s12532-018-0147-4
16. Gamrath, G.: Enhanced predictions and structure exploitation in branch-and-bound. Technische Universitaet Berlin (Germany) (2020)
17. Glover, F., Kochenberger, G.A., Alidaee, B.: Adaptive memory tabu search for binary quadratic programs. Manag. Sci. **44**(3), 336–345 (1998)
18. Glover, F.W., Lewis, M.W., Kochenberger, G.A.: Logical and inequality implications for reducing the size and difficulty of quadratic unconstrained binary optimization problems. Eur. J. Oper. Res. **265**(3), 829–842 (2018). https://doi.org/10.1016/j.ejor.2017.08.025
19. Gurobi Optimization, LLC: Gurobi Optimizer Reference Manual (2021). https://www.gurobi.com
20. Gusmeroli, N., Hrga, T., Lužar, B., Povh, J., Siebenhofer, M., Wiegele, A.: Biqbin: A parallel branch-and-bound solver for binary quadratic problems with linear constraints. ACM Trans. Math. Softw. (2022). https://doi.org/10.1145/3514039
21. Hammer, P.L., Hansen, P., Simeone, B.: Roof duality, complementation and persistency in quadratic 0–1 optimization. Math. Program. **28**(2), 121–155 (1984). https://doi.org/10.1007/BF02612354
22. Hammer, P.L., Rudeanu, S.: Boolean Methods in Operations Research and Related Areas. Springer (1968)

23. Hrga, T., Povh, J.: MADAM: a parallel exact solver for max-cut based on semidefinite programming and ADMM. Comput. Optim. Appl. **80**(2), 347–375 (2021). https://doi.org/10.1007/s10589-021-00310-6
24. IBM: CPLEX (2020). https://www.ibm.com/analytics/cplex-optimizer
25. Jünger, M., Lobe, E., Mutzel, P., Reinelt, G., Rendl, F., Rinaldi, G., Stollenwerk, T.: Quantum annealing versus digital computing: an experimental comparison. ACM J. Exp. Algorithmics (2021). https://doi.org/10.1145/3459606
26. Jünger, M., Mallach, S.: Odd-cycle separation for maximum cut and binary quadratic optimization. In: Bender, M.A., Svensson, O., Herman, G. (eds.) 27th Annual European Symposium on Algorithms, ESA 2019, September 9-11, 2019, Munich/Garching, Germany, LIPIcs, vol. 144, pp. 63:1–63:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). doi:https://doi.org/10.4230/LIPIcs.ESA.2019.63
27. Jünger, M., Mallach, S.: Exact facetial odd-cycle separation for maximum cut and binary quadratic optimization. INFORMS J. Comput. **33**(4), 1419–1430 (2021). https://doi.org/10.1287/ijoc.2020.1008
28. Karp, R.: Reducibility among combinatorial problems. In: R. Miller, J. Thatcher (eds.) Complexity of Computer Computations, pp. 85–103. Plenum Press (1972). doi:https://doi.org/10.1007/978-1-4684-2001-2_9
29. Kepner, J., Gilbert, J.: Graph algorithms in the language of linear algebra. SIAM (2011). https://doi.org/10.1137/1.9780898719918
30. Kernighan, B.W., Lin, S.: An efficient heuristic procedure for partitioning graphs. Bell Syst. Tech. J. **49**(2), 291–307 (1970). https://doi.org/10.1002/j.1538-7305.1970.tb01770.x
31. Krislock, N., Malick, J., Roupin, F.: Biqcrunch: a semidefinite branch-and-bound method for solving binary quadratic problems. ACM Trans. Math. Softw. (2017). https://doi.org/10.1145/3005345
32. Lange, J., Andres, B., Swoboda, P.: Combinatorial Persistency Criteria for Multicut and Max-Cut. In: IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019, pp. 6093–6102. Computer Vision Foundation / IEEE (2019). doi:https://doi.org/10.1109/CVPR.2019.00625
33. Liers, F.: Contributions to determining exact ground-states of ising spin-glasses and to their physics. In: Ph.D. Thesis, University of Cologne (2004)
34. Lin, J., Cai, S., Luo, C., Su, K.: A reduction based method for coloring very large graphs. In: Sierra, C. (ed.) Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017, pp. 517–523. ijcai.org (2017). https://doi.org/10.24963/ijcai.2017/73
35. Ralphs, T., Shinano, Y., Berthold, T., Koch, T.: Parallel Solvers for Mixed Integer Linear Optimization, pp. 283–336. Springer International Publishing, Cham (2018). https://doi.org/10.1007/978-3-319-63516-3_8
36. Rehfeldt, D., Koch, T.: Implications, Conflicts, and Reductions for Steiner Trees. In: Singh, M., Williamson, D.P. (eds.) Integer Programming and Combinatorial Optimization - 22nd International Conference, IPCO 2021, Atlanta, GA, USA, May 19-21, 2021, Proceedings, Lecture Notes in Computer Science, vol. 12707, pp. 473–487. Springer (2021). https://doi.org/10.1007/978-3-030-73879-2_33
37. Rendl, F., Rinaldi, G., Wiegele, A.: Solving max-cut to optimality by intersecting semidefinite and polyhedral relaxations. Math. Program. **121**(2), 307–335 (2010). https://doi.org/10.1007/s10107-008-0235-8
38. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T., Winkler, M.: Solving open MIP instances with parascip on supercomputers using up to 80, 000 cores. In: 2016 IEEE International Parallel and Distributed Processing Symposium, IPDPS 2016, Chicago, IL, USA, May 23-27, 2016, pp. 770–779. IEEE Computer Society (2016). doi:10.1109/IPDPS.2016.56
39. Tavares, G.: New algorithms for quadratic unconstrained binary optimization (qubo) with applications in engineering and social sciences. In: Ph.D. Thesis, Rutgers, the State University of New Jersey-New Brunswick (2008)
40. Wiegele, A.: BiqMac Library: A collection of Max-Cut and quadratic 0-1 programming instances of medium size. Tech. Rep. (2007)