BQO-SEP: A Library of Separation Routines for Boolean Quadratic Optimization

Franklin Djeumou Fomeni∗ Konstantinos Kaparis†
Adam N. Letchford‡
Draft, 2nd July 2016

Abstract
We describe BQO-SEP, an Open Source library of separation routines, written in C, for Boolean quadratic optimization (BQO). As well as containing several exact and heuristic separation routines for BQO itself, it contains separation routines for the closely-related max-cut problem, along with routines for mapping fractional solutions and valid inequalities from BQO to max-cut and vice-versa. Some computational results are presented.

Keywords: Boolean quadratic optimization, max-cut problem, branch-and-cut, polyhedral combinatorics.

1 Introduction
By a Boolean quadratic optimization problem, which we denote by BQO, we mean a problem of the form

$$\min \left\{ x^T Q x : A x \leq b, \ x \in \{0, 1\}^n \right\},$$

where $Q \in \mathbb{Q}^{n \times n}$, $A \in \mathbb{Q}^{m \times n}$ and $b \in \mathbb{Q}^m$. A wide variety of important problems can be formulated as BQOs. This includes, for example, problems in location, scheduling, network design and facility layout [25], problems in statistical physics and VLSI design [3], problems in various branches of mathematics [10], and even problems in sport scheduling [12].

If we omit the constraints $Ax \leq b$, we obtain the unconstrained case, which we call UBQO. It is known [18] that UBQO is equivalent to the max-cut problem, in which one wishes to partition the nodes of an edge-weighted

∗GERAD, Polytechnique Montréal, Canada. E-mail: franklin.djeumou@gerad.ca
†Department of Business Administration, University of Macedonia, Greece. E-mail: k.kaparis@uom.edu.gr
‡Department of Management Science, Lancaster University, United Kingdom. E-mail: A.N.Letchford@lancaster.ac.uk
graph into two subsets, so as to maximise the sum of the weights of the edges that have exactly one end-node in each subset. Since the max-cut problem is \(\mathcal{NP}\)-hard in the strong sense [15], this implies the same for UBQO, and therefore BQO as well. The mapping between UBQO and max-cut is called the covariance map [10].

As present, the most successful approaches to solving BQO (and max-cut) to proven optimality use either linear programming (LP) or semidefinite programming (SDP) relaxations (e.g., [5, 28]). The LP-based algorithms, and to some extent also the SDP-based ones, can benefit from the addition of strong valid linear inequalities, a.k.a. cutting planes. Even for the relatively simple unconstrained (or max-cut) case, many families of valid inequalities have been discovered (see [10] and also Section 2 below).

For a given family of valid inequalities, a separation routine is an algorithm that takes the solution of an LP or SDP relaxation as input, and searches for violated inequalities in the given family (see, e.g., [17]). For the unconstrained case, efficient exact and heuristic separation routines have been discovered for several families of inequalities (again, see Section 2). On the other hand, to our knowledge, there was no freely available library of separation routines until the writing of this paper.

In this paper, then, we present BQO-SEP, an Open Source library of separation routines for BQOs, written in C. It contains exact and/or heuristic separation routines for several families of valid inequalities for UBQO, and several families of valid inequalities for the max-cut problem. The inequalities for UBQO are of course valid for general BQO. It also include routines that use the covariance map to map valid inequalities and/or fractional points from ‘BQO-space’ to ‘max-cut space’ and vice-versa.

The paper is structured as follows. The literature review is in Section 2. In Section 3, we explain how graphs, points and inequalities are represented (and transformed) in our code. In Sections 4 and 5, we describe our separation routines for BQO and max-cut, respectively. Some computational results are given in Section 6. Finally, some concluding remarks are made in Section 7.

Throughout the paper, we assume \(n \geq 3\). We also use the following (standard) notation. We let \(K_n\) denote the complete undirected graph with vertex set \(V_n = \{1, \ldots, n\}\) and edge set \(E_n = \{e \subseteq V_n : |e| = 2\}\). Given a set \(S \subseteq V_n\), we let \(E(S)\) denote the set of edges in \(E_n\) that have both end-nodes in \(S\), and \(\delta(S)\) denote the set of edges with exactly one end-node in \(S\). We also let \(x(S)\) denote \(\sum_{i \in S} x_i\). Given two disjoint sets \(S, T \subseteq V_n\), we denote by \(E(S : T)\) the set of edges in \(E_n\) that have one end-node in \(S\) and the other in \(T\). Given any \(F \subseteq E_n\), we let \(y(F)\) denote \(\sum_{e \in F} y_e\), and similarly for \(z(F)\).

We also use the following (again standard) terminology. A set \(C \subseteq E_n\) is a cycle if it induces a connected subgraph of \(K_n\) in which every node has degree 2. The nodes in the subgraph are denoted by \(V(C)\). Given a
cycle $C$, an edge $e \in E_n$ is called a chord of $C$ if $e \subset V(C)$ but $e \notin C$. A chord is called a 2-chord if its end-nodes are two edges apart in the cycle, i.e., if there exist nodes $u,v,w \in V_n$ such that $\{u,v\} \in C$, $\{v,w\} \in C$ and $e = \{u,w\}$. Two disjoint sets $R,S \subset E_n$ and an edge $\{h,h'\} \in E_n \setminus (R \cup S)$ form a bicycle wheel if $R$ is a cycle and $S = \{\{v,h\},\{v,h'\} : v \in V(R)\}$. (The set $R$ is called the rim, the edges in $S$ are called the spokes, and the nodes $h$ and $h'$ are called the hub.)

2 Literature Review

Now we review the literature. Due to space limitations, we only review essential results, and refer the reader to Deza & Laurent [10] for a comprehensive survey of known results up to around 1996.

2.1 The Boolean quadric polytope

The standard 0-1 LP formulation for UBQO, due to Fortet [13], is as follows. For $1 \leq i < j \leq n$, introduce a new binary variable $y_{ij}$, representing the product $x_i x_j$. Then:

$$\begin{align*}
\min & \quad \sum_{i=1}^{n} Q_{ii} x_i + \sum_{1 \leq i < j \leq n} 2Q_{ij} y_{ij} \\
\text{s.t.} & \quad y_{ij} \leq x_i \quad (i \in V_n, j \in V_n \setminus \{j\}) \\
& \quad y_{ij} \geq x_i + x_j - 1 \quad (\{i,j\} \in E_n) \\
& \quad x \in \{0,1\}^n, \quad y \in \{0,1\}^{E_n}.
\end{align*}$$

Padberg [24] calls the convex hull of feasible solutions to this 0-1 LP the Boolean quadric polytope. We will denote it by $BQP_n$.

Padberg [24] showed that the following inequalities define facets of $BQP_n$ under mild conditions:

- The trivial inequalities $y_{ij} \geq 0$, $y_{ij} \leq x_i$, $y_{ij} \leq x_i$, and $y_{ij} \geq x_i + x_j - 1$ for all $\{i,j\} \in E_n$.

- The triangle inequalities

$$\begin{align*}
y_{ij} + y_{ik} + y_{jk} & \geq x_i + x_j + x_k - 1 & (1) \\
y_{ik} + y_{jk} & \leq x_k + y_{ij} & (2)
\end{align*}$$

for all triples $i,j,k \in V_n$.

- The clique inequalities

$$y(E(S)) \geq \alpha x(S) - \binom{\alpha + 1}{2}, \quad (3)$$

for all $S \subseteq V_n$ with $|S| \geq 2$ and all $\alpha \in \{1,\ldots,|S| - 2\}$. 

3
The cut inequalities
\[
y(E(S : T)) \leq x(S) + y(E(S)) + y(E(T)),
\]
for all disjoint \(S, T \subset V_n\) with \(|S| \geq 1\) and \(|T| \geq 2\).

### 2.2 The cut polytope

The max-cut problem on the complete graph \(K_n\), with edge-weights \(w_e\) for all \(e \in E\), can be formulated as the following 0-1 LP:

\[
\begin{align*}
\text{min} & \quad \sum_{e \in E} w_e z_e \\
\text{s.t.} & \quad z_{ij} + z_{ik} + z_{jk} \leq 2 \quad (1 \leq i < j < k \leq n), \\
& \quad z_{ij} - z_{ik} - z_{jk} \leq 0 \quad (\{i, j\} \in E_n; k \in V_n \setminus \{i, j\})
\end{align*}
\]

Here, \(z_e\) takes the value 1 if and only if edge \(e\) lies in the cut.

Barahona and Mahjoub \[4\] call the convex hull of feasible solutions to this 0-1 LP the cut polytope. We will denote it by \(\text{CUT}_n\). They show that the following inequalities define facets of \(\text{CUT}_n\):

- The triangle inequalities (5), (6).
- The odd clique inequalities
  \[
  \sum_{i,j \in S} z_{ij} \leq \lfloor \frac{|S|^2}{4} \rfloor,
  \]
  where \(S \subseteq V_n\) is such that \(|S| \geq 3\) and odd.
- The odd bicycle wheel inequalities
  \[
  z(R) + z(S) + z_{hh'} \leq 2|C|,
  \]
  where \(R, S\) and \(\{h, h'\}\) form a bicycle wheel in \(K_n\) with \(|R|\) odd.

Poljak & Turzik \[27\] introduced some more inequalities for \(\text{CUT}_n\), which we call 2-circulant inequalities. These take the form:

\[
z(C) + z(C') \leq \frac{3}{2} (|C| - 1),
\]

where \(C\) is a cycle with \(|C|\) odd, and \(C'\) is the set of 2-chords of \(C\). These inequalities define facets if and only if \(|C| \mod 4 = 1\).

Barahona and Mahjoub also considered the max-cut problem on a sparse graph \(G = (V, E)\). They showed that a valid 0-1 LP formulation can be
obtained with only one z variable per edge in E, provided that the triangle inequalities are replaced with the following odd cycle inequalities:

\[ z(C \setminus F) \geq z(F) - |F| + 1, \tag{10} \]

for all chordless cycles \( C \subseteq E \) and all \( F \subseteq C \) with \(|F| \) odd. A similar idea was used by Padberg [24] to derive a formulation for UBQO instances in which many of the entries of \( Q \) are zero. We let \( \text{CUT}(G) \) denote the cut polytope of a general graph \( G \), and \( \text{BQP}(G) \) denote the polytope obtained from \( \text{BQP}_n \) by projecting out the \( y_{ij} \) variables for \( \{i, j\} \in E_n \setminus E \).

### 2.3 The covariance map and switching

It is known [2, 8, 24] that a point \((x^*, y^*)\) belongs to \( \text{BQP}_n \) if and only if the point \( z^* \) belongs to \( \text{CUT}_{n+1} \), where:

\[
\begin{align*}
z_{i,n+1}^* &= x_i^* \quad (i \in V_n) \\
z_{ij}^* &= x_i^* + x_j^* - 2y_{ij}^* \quad (\{i, j\} \in E_n).
\end{align*}
\]

This map is called the **covariance map**. A consequence of this map is that the inequality \( \alpha^T z \leq \beta \) is valid for \( \text{CUT}_{n+1} \) if and only if the inequality

\[
\sum_{i \in V_n} \left( \sum_{j \in V_{n+1} \setminus \{i\}} \alpha_{ij} \right) x_i - 2 \sum_{e \in E_n} \alpha_e y_e \leq \beta
\]

is valid for \( \text{BQP}_n \).

Another important map, called **switching**, was defined in [4]. Given any \( S \subseteq V_n \), switching leaves \( z_e \) unchanged for all \( e \in E_n \setminus \delta(S) \), but maps \( z_e \) onto \( 1 - z_e \) for all \( e \in \delta(S) \). It is shown in [4] that \( \text{CUT}_n \) is invariant under switching. It follows that, if the inequality \( \lambda^T z \leq \gamma \) is valid for \( \text{CUT}_n \), then the ‘switched’ inequality

\[
\sum_{e \in E_n \setminus \delta(S)} \lambda_e z_e - \sum_{e \in \delta(S)} \lambda_e z_e \leq \gamma - \sum_{e \in \delta(S)} \lambda_e
\]

is also valid, for any \( S \subseteq V_n \).

Padberg [24] introduced an analogous switching operation for \( \text{BQP}_n \). For a given \( S \), one must map:

- \( x_i \) onto \( 1 - x_i \) for all \( i \in S \),
- \( y_{ij} \) onto \( x_j - y_{ij} \) for all \( i \in S \) and \( j \in V_n \setminus S \),
- \( y_{ij} \) onto \( 1 - x_i - x_j + y_{ij} \) for all \( \{i, j\} \subseteq S \).

Again, \( \text{BQP}_n \) is unchanged under this operation.
2.4 Separation

Finally, we review the known separation algorithms. For BQP\(_n\), the separation problem for the triangle inequalities (1), (2) can be solved exactly in \(O(n^3)\) time by mere enumeration. Separation heuristics were presented for the clique inequalities (3), and their switchings, in [31, 32]. An exact separation algorithm was presented in [29] for a family of valid inequalities that arises from an SDP relaxation of UBQO. These inequalities are however never facet-defining.

For CUT\(_n\), many exact separation algorithms are known. Separation for the triangle inequalities (5), (6) can be solved in \(O(n^3)\) time by enumeration. Gerards [16] gave an \(O(n^5)\) time algorithm for the odd bicycle wheel inequalities (8). Poljak & Turzik [27] stated that there exists an exact algorithm for a generalisation of the 2-circulant inequalities (9), but did not give a proof. It is shown in [21, 23] that one can separate in polynomial time over families of valid inequalities that include not only all odd bicycle wheel and 2-circulant inequalities, but also all of their switchings. The algorithm in [23] runs in \(O(n^5)\) time, and the one in [21] is slower. Finally, an exact separation algorithm was presented in [20] for a family of valid inequalities that arise from an SDP relaxation of max-cut. Again, however, these inequalities are never facet-defining.

As for separation heuristics for CUT\(_n\), Helmberg & Rendl [19] present a heuristic for the odd clique inequalities (7) and their switchings. Heuristics for other families of inequalities, not mentioned here, can be found in [9, 14, 19]. See also [22], which shows the equivalence between various separation problems.

Finally, there are a few results known for the cut polytope of a general graph. Barahona & Mahjoub [4] give an exact \(O(n^3)\) time algorithm for the odd cycle inequalities (10). Some fast heuristics for the same inequalities were presented in [2]. Cheng [7] presents an exact routine for a generalisation of the odd bicycle wheel inequalities (8), and Bonato et al. [5] present a heuristic for a generalisation of the (switched) odd clique inequalities.

3 Graphs, Points, Inequalities and Transformations

Before presenting our separation routines, we explain how graphs, points and inequalities are represented in our code. We also present some simple functions that apply the covariance map to points and inequalities.

3.1 Graphs

Some of our functions take an undirected graph as one of their inputs. We represent a graph...
an array of linked lists, one per node?

3.2 Points

Each of our separation routines takes a point as input. We distinguish between four different kinds of points:

- A point \((x^*, y^*) \in [0, 1]^{V_n+E_n}\) that lies outside \(BQP_n\);
- A point \((x^*, y^*) \in [0, 1]^{V_n+E}\) that lies outside \(BQP(G)\);
- A point \(z^* \in [0, 1]^{E_n}\) that lies outside \(CUT_n\);
- A point \(z^* \in [0, 1]^E\) that lies outside \(CUT(G)\).

Points of the first kind are represented by a one-dimensional double-precision array of length \(n\) (for \(x^*\)) and another such array of length \(\binom{n}{2}\) (for \(X^*\)). Points of the second kind are represented in a similar way, except that the array for \(X^*\) is of length \(|E|\).

Points of the third kind are represented by a one-dimensional double-precision array of length \(\binom{n}{2}\). Points of the fourth kind are represented in a similar way, except that the array is of length \(|E|\).

3.3 Inequalities

Similarly, we distinguish between four different kinds of inequalities:

- A valid inequality for \(BQP_n\) of the form \(\alpha^T x + \beta^T y \leq \gamma\), where \(\alpha \in \mathbb{Q}^n\) and \(\beta \in \mathbb{Q}^{E_n}\);
- A valid inequality for \(BQP(G)\) of the same form, except that \(\beta \in \mathbb{Q}^E\);
- A valid inequality for \(CUT_n\) of the form \(\alpha^T z \leq \beta\), where \(\alpha \in \mathbb{Q}^{E_n}\);
- A valid inequality for \(CUT(G)\) of the same form, except that \(\alpha \in \mathbb{Q}^{|E|}\).

The inequalities of a given type are represented in the similar way to the corresponding points. For example, inequalities of the first kind are represented by a double-precision array of length \(n\) (for \(\alpha\)), another such array of length \(\binom{n}{2}\) (for \(\beta\)), and a double-precision variable (for \(\gamma\)). However, our separation routines can output several inequalities rather than just one. Therefore...
3.4 Transforms

For the benefit of the user, we include eight (?) simple routines that use the covariance map to transform points and inequalities from ‘BQO space’ to ‘max cut space’ and vice-versa:

- Function MapPointBoolCut takes as input an unsigned integer $n$ and a point $(x^*, X^*)$ that lies outside $\text{BQP}_n$, and outputs the corresponding point $y^*$ that lies outside $\text{CUT}_{n+1}$. The additional node created by the map is numbered $n + 1$.
- Function MapPointCutBool takes an unsigned integer $n$ and a point $y^*$ that lies outside $\text{CUT}_n$, and outputs the corresponding point $(x^*, X^*)$ that lies outside $\text{BQP}_{n-1}$. The node $n$ is eliminated by the map.
- Function MapIneqBoolCut takes an unsigned integer $n$ and a valid inequality for $\text{BQP}_n$, and outputs the corresponding valid inequality for $\text{CUT}_{n+1}$. The additional node created by the map is numbered $n + 1$.
- Function MapIneqCutBool takes an unsigned integer $n$ and a valid inequality for $\text{CUT}_n$, and outputs the corresponding valid inequality for $\text{BQP}_{n-1}$. The node $n$ is eliminated by the map.
- Function MapPointBoolCutSparse, MapPointCutBoolSparse, MapIneqBoolCutSparse and MapIneqCutBoolSparse are similar, except that they are designed for points that lie outside $\text{BQP}_n$ or $\text{CUT}_n$. They take a graph as an additional input...

Note: perhaps only need two routines, one for points and one for inequalities. Then we would need an additional input: an unsigned integer stating which type of point or inequality is to be transformed. Something like:

```c
bool MapPoint (unsigned n, double * xstar, double * ystar, double * zstar, unsigned pointtype, graph G)
bool MapIneq (unsigned n, double * alpha, double * beta, double * gamma, unsigned ineqtype, graph G)
```

Then, if the pointtype or ineqtype is sparse, graph G can be set to NULL.

4 Routines for the Boolean Quadric Polytope

In this section, we describe our separation routines for $\text{BQP}_n$. 
4.1 Function QuadSepClq

This function searches for violated clique inequalities of the form (??)... We can perhaps use a greedy heuristic, that starts with a triangle inequality of the form (1) and then iteratively inserts nodes into $S$, adjusting $\alpha$ as it goes along. See, e.g., [31, 32].

4.2 Function QuadSepCut

This function searches for violated cut inequalities of the form (4)... We can perhaps use a greedy heuristic, that starts with a triangle inequality of the form (2) and then iteratively inserts nodes into $S$ and $T$.

4.3 Function QuadSepSwClq

This function is similar to function QuadSepClq, except that it searches for violated switched clique inequalities. These take the form:

$$y(E(S)) + y(E(T)) \geq \alpha x(S) + (\alpha + 1)x(T) + y(E(S : T)) + \alpha(\alpha + 1)/2$$

for all disjoint $S, T \subset V_n$ with $|S \cup T| \geq 2$ and for $1 - |T| \leq \alpha \leq |S| - 2$. (We remark that they were rediscovered, apparently independently from Padberg, in [6, 30].)

We can perhaps use a greedy heuristic, that starts with a triangle inequality (of either type) and then iteratively inserts nodes into $S$ and/or $T$, adjusting $\alpha$ as it goes along. Again, see [31, 32]...

4.4 OTHER OPTIONAL FUNCTIONS

We could have

- QuadSepTri: for triangle inequalities
- QuadSepOddCycHeur: heuristic for odd cycle inequalities
- QuadSepOddCycExact: exact for odd cycle inequalities

These functions would however just be “wrappers”. We would first call MapPointBoolCut, then call the cut version of the separation routine, and finally call MapIneqCutBool.

5 Routines for the Cut Polytope

In this section, we describe our separation routines for the cut polytope.
5.1 Function CutSepTri

The inputs to the function are an unsigned integer \( n \), a double-precision parameter \( 0 \leq \epsilon < 1 \), and a fractional point \( y^* \) that lies outside CUT\( _n \). The point is represented by a double-precision array of length \( \binom{n}{2} \).

This function searches for triangle inequalities (5), (6) that are violated by more than \( \epsilon \). When \( \epsilon = 0 \), the algorithm is exact. For larger values of \( \epsilon \), it is a heuristic. As \( \epsilon \) increases, the algorithm becomes faster, but at the risk of losing some violated inequalities.

Our separation routine is an enhanced version of the one described in [14]. It is described in Algorithm 1. Like the algorithm in [14], it runs in \( O(n^3) \) time and is guaranteed to generate no more than \( O(n^2) \) violated inequalities in a single call. On the other hand, our algorithm is faster, and it has the nice property that it never generates duplicate inequalities.

5.2 Function CutSepOddClq

This function searches for violated odd clique inequalities. We take a “seed” clique, which could be input by the user, and grow it by iteratively adding any pair of nodes that leads to a decrease in the slack. If the slack drops below zero, we break and output the violated inequality. (This is to keep the inequalities as sparse as possible.)

5.3 Function CutSepSwOddClq

This function is similar to CutSepOddClq, except that it searches for violated switched odd clique inequalities. These inequalities take the form:

\[
\sum_{i,j \in S} z_{ij} + \sum_{i,j \in T} z_{ij} - \sum_{i \in S} \sum_{j \in T} z_{ij} \leq \left\lfloor \frac{(|S| - |T|)^2}{4} \right\rfloor,
\]

where \( S \) and \( T \) are two disjoint vertex sets with \(|S| + |T| \geq 3 \) and odd.

When adding a pair of nodes, all four combinations are checked (both added to \( S \), both added to \( T \), or one added to \( S \) and the other to \( T \))...

5.4 Function CutSepOBW

CutSepOBW: heuristic for odd bicycle wheel inequalities

Gerards [16] gave an exact separation algorithm. It starts by observing that the inequality can be written as:

\[
\frac{1}{2} \sum_{i=1}^{c} (\text{slack of two triangle inequalities}) \geq z_{hh'}.
\]

It then considers each edge as a candidate for the “hub”. For each candidate, one checks if there is an odd cycle with weight less than \( z^*(\text{hub}) \). On a complete graph, the running time is \( O(n^5) \).
Algorithm 1: Separation algorithm for triangle inequalities.

\textbf{Input}: A point $x^* \in \mathbb{R}^{|E|}$ to be separated. \hfill \\
\textbf{Output}: FAILURE or a collection of violated triangle inequalities.

1. Sort the edges with $y^*_e > (2 + \epsilon)/3$ in non-increasing order of $y^*$ value and store in a list $L$; \hfill \\
2. Label all edges in $L$ as “unmarked”; \hfill \\
3. for each edge $\{i, j\}$ in $L$ do \hfill \\
4. Mark $\{i, j\}$; \hfill \\
5. Among all nodes $k \in V \setminus \{i, j\}$ such that $\{i, k\}$ and $\{j, k\}$ are unmarked, let $h$ be the node that maximises $x^*_{ih} + x^*_{jh}$; \hfill \\
6. if $x^*_{ij} + x^*_{ih} + x^*_{jh} > 2 + \epsilon$ then \hfill \\
7. \quad Return the violated inequality $x_{ij} + x_{ik} + x_{jk} \leq 2$; \hfill \\
8. end \hfill \\
9. end \hfill \\
10. Sort the edges with $y^*_e < (1 - \epsilon)/2$ in non-decreasing order of $y^*$ value and store in a list $L'$; \hfill \\
11. Label all edges in $L'$ as “unmarked”; \hfill \\
12. for each edge $\{i, j\}$ in $L'$ do \hfill \\
13. Mark $\{i, j\}$; \hfill \\
14. Among all nodes $k \in V \setminus \{i, j\}$ such that $\{i, k\}$ and $\{j, k\}$ are unmarked, compute $\delta_h = |x^*_{ih} - x^*_{jh}|$; \hfill \\
15. Let $h$ be the node that maximises $\delta_h$; \hfill \\
16. if $-x^*_{ij} + x^*_{ih} - x^*_{jh} > \epsilon$ then \hfill \\
17. \quad Return the violated inequality $-x_{ij} + x_{ih} - x_{jh} \leq 0$; \hfill \\
18. end \hfill \\
19. if $-x^*_{ij} - x^*_{ih} + x^*_{jh} > \epsilon$ then \hfill \\
20. \quad Return the violated inequality $-x_{ij} - x_{ih} + x_{jh} \leq 0$; \hfill \\
21. end \hfill \\
22. end
One can run Gerards’ algorithm on general graph too. There are m candidates for the hub. For a given hub, the auxiliary graph contains $O(n)$ nodes and $O(m)$ edges. The time taken per odd cycle computation is $O((m+n \log n))$. So the total running time is $O(mn(m+n \log n))$.

This is unacceptably large. There are two easy ways to speed it up. First, the maximum possible violation is $z^*_{hh'}$. So, if one wishes to find inequalities violated by more than $\epsilon$, one need only consider edges with $z^*_e > \epsilon$ as hub candidates. A more sophisticated argument shows that one need not consider edges with $z^*_e$ close to 1 as hubs either. Also, before each odd cycle computation, we can drop all edges with weight $\geq z^*_e - \epsilon$.

5.5 Function CutSepSwOBW

CutSepSwOBW: heuristic for switched odd bicycle wheel inequalities

It seems likely that we just have to switch some of the triangle inequalities mentioned above. Probably, this means adding “odd” and “even” edges to the auxiliary graphs...

5.6 Function CutSepCirc

CutSepCirc: heuristic for $(p,2)$-circulant inequalities

We try to reconstruct Poljak & Turzik’s idea. Write the inequality as:

$$\sum_{i=1}^{c} (\text{slack of weakened triangle inequality}) \geq 3.$$

Construct a digraph with one node for each ordered pair of nodes. For all ordered triples $i, j, k$, insert a directed arc from node $(i, j)$ to node $(j, k)$, and give it a weight equal to the slack mentioned. Find a minimum weight directed cycle in this graph. If its weight is less than 3, we have found a violated inequality.

The auxiliary digraph has $O(n^2)$ nodes and $O(n^3)$ arcs. So the algorithm runs in $O(n^5)$ time. This is the same as the time for odd bicycle wheels, and it is also the same as the one in [23].

For a sparse graph, the auxiliary graph has $O(m)$ nodes and $O(mn)$ arcs. The time taken is then $O(m^2 n)$.

To speed it up, note that we only need to include an arc if the slack is less than $3 - 2\epsilon$.

5.7 Function CutSepSwCirc

CutSepSwCirc: heuristic for switched $(p,2)$-circulant inequalities.

We probably need to switch the weakened triangle inequalities...
5.8 Function CutSepOddCycHeur

This function is for the sparse version of max-cut. It is an enhanced version of the spanning-tree heuristic separation for violated odd cycle inequalities described in [2]. The inputs to the function are an unsigned integer \( n \) and a fractional point \( x^* \) that lies outside \( \text{CUT}_n \). The point is represented by a double-precision array of length \( \binom{n}{2} \). The function outputs all the violated odd-cycle inequalities and runs in \( O(n^2) \) time. We define an edge \( e \in E_n \) as ‘odd’ if \( x^*_e > \frac{1}{2} \). The full description of the routine is given in Algorithm 2.

**Explain the enhancement.**

A similar enhancement was used in [26], in a separation heuristic for a family of valid inequalities for the TSP.

5.9 Function CutSepOddCycExact

This function also searches for odd cycle inequalities that are violated by \( x^* \). However it is an exact separation algorithm rather than a heuristic one which runs in \( O(n^3) \) time, see [2]. It starts with the construction of a graph \( H = \left( V'_n \cup V''_n, E'_n \cup E''_n \cup E'''_n \right) \) from the original graph \( G = (V_n, E_n) \), where \( V'_n = \{ i' : i' \text{ is a copy of } i \in V_n \} \), \( V''_n = \{ i'' : i'' \text{ is a copy of } i \in V_n \} \), \( E'_n = \{ \{ i', j' \} : i', j' \in V'_n \} \), \( E''_n = \{ \{ i'', j'' \} : i'', j'' \in V''_n \} \) and \( E'''_n = \{ \{ i', j'' \}, \{ i'', j' \} : i', j' \in V'_n, i'', j'' \in V''_n \} \). Then the core of the algorithm can be described as in Algorithm 3.

6 Computational Experiments

**se:experiments** Perhaps give results obtained by applying the routines to some standard 0-1 QP and/or max-cut instances.

7 Concluding Remarks

**se:end** It would be good to extend the library in several ways. First, it would be good to add routines for psd, rpsd and gap inequalities. Second, it would be nice to consider the Boolean quadric and cut cones... and maybe the bipartite subgraph polytope [1]...

Mention the RLT and \((s,t)\) inequalities here [11]?

Mention also the Quadratic knapsack polytope?

References

Algorithm 2: Separation heuristic for odd cycle inequalities.

Input: A point $x^* \in \mathbb{R}^{(2)}$ to be separated.
Output: Failure or a collection of violated odd cycle inequalities.

1. Assign the weight $\min\{x^* e, 1 - x^* e\}$ to each edge $e \in E_n$;
2. Compute a minimum-weight spanning tree $T$ of $G = (E_n, V_n)$ (using Prim’s algorithm);
3. for each node $i$ do
   4. Label all nodes ‘even’;
   5. Traverse the tree in a breadth-first manner, starting at $i$;
   6. Label nodes ‘odd’ or ‘even’ according to the number of ‘odd’ edges in the path from $i$ to the given node;
   7. Record the total weight of the path in the tree from $i$ to each node;
   8. Record the maximum weight encountered in the path from $i$ to each node;
9. for each non-tree edge incident on node $i$ do
   10. Set $C$ to be its fundamental cycle;
   11. Set $F = \{e \in C : e$ is ‘odd’ $\}$;
   12. if $F$ is ‘even’ then
      13. Let $e_m$ be the edge in $C$ with the highest weight;
      14. if $e_m \in F$ then
         15. Remove $e_m$ from $F$;
      16. end
   17. else
      18. Add $e_m$ to $F$;
   19. end
20. end
21. Check the resulting odd-cycle inequality for violation;
22. end
23. end
Algorithm 3: Exact separation algorithm for odd cycle inequalities.

\textbf{Input}: A point $x^* \in \mathbb{R}^{(\mathbb{Z})}$ to be separated.

\textbf{Output}: FAILURE or a collection of violated odd cycle inequalities.

1. Assign weight $x^*_{i,j}$ to each of the edges $\{i', j\}' \in E'_n$ and $\{i'', j''\}' \in E''_n$.
2. Assign weight $1 - x^*_{i,j}$ to each of the edges $\{i', j''\}'$ and $\{i'', j'\}' \in E'''_n$.
3. For each $i \in V_n$ do
   4. Compute the shortest path (with respect to the weights) in $H$
      from $i'$ to $i''$ as $\text{Shortestpath}(i', i'')$;
   5. If the ‘length’ of $\text{Shortestpath}(i', i'')$ is less than 1 then
      6. Construct the cycle $C \subseteq E_n$ from $\text{Shortestpath}(i', i'')$;
         Construct $F$ as $\{\{i, j\} \in E_n : \{i', j''\}', \{i'', j'\}' \in E'''_n$ belong to $\text{Shortestpath}(i', i'')\}$;
      7. Return the violated odd cycle inequality $x(F) - x(C \setminus F) \leq |F| - 1$;
   8. end
9. end


