Cutting Plane Generation Through Sparse Principal Component Analysis

Santanu S. Dey* Aleksandr M. Kazachkov† Andrea Lodi‡
Gonzalo Muñoz§

February 18, 2021

Abstract

Quadratically-constrained quadratic programs (QCQPs) are optimization models whose remarkable expressiveness have made them a cornerstone of methodological research for nonconvex optimization problems. However, modern methods to solve a general QCQP fail to scale, encountering computational challenges even with just a few hundred variables. Specifically, a semidefinite programming (SDP) relaxation is typically employed, which provides strong dual bounds for QCQPs, but relies on memory-intensive algorithms. An appealing alternative is to replace the SDP with an easier-to-solve linear programming relaxation, while still achieving strong bounds. In this work, we make advances towards achieving this goal by developing a computationally-efficient linear cutting plane algorithm that emulates the SDP-based approximations of nonconvex QCQPs. The cutting planes are required to be *sparse*, in order to ensure a numerically attractive approximation, and *efficiently computable*. We present a novel connection between such sparse cut generation and the sparse principal component analysis problem in statistics, which allows us to achieve these two goals. We show extensive computational results advocating for the use of our approach.

1 Introduction

Nonconvex *quadratically-constrainted quadratic programs* (QCQPs) are highly expressive models with wide applicability. For example, they can represent mixed-integer polynomial optimization over a compact feasible region, which already captures a broad set of real-world problems. At the same time, the

*Georgia Institute of Technology, Atlanta, GA, USA (santanu.dey@isye.gatech.edu).
†University of Florida, Gainesville, FL, USA (akazachkov@ufl.edu).
‡Canada Excellence Research Chair in Data Science for Real-Time Decision-Making. Polytechnique Montréal, Montréal, QC, Canada (andrea.lodi@polymtl.ca).
§Universidad de O'Higgins, Rancagua, Chile (gonzalo.munoz@uoh.cl).

flexible modeling capabilities of QCQPs imply steep computational challenges involved in designing practical general-purpose techniques for these problems.

One successful approach for solving QCQPs is based on *cutting planes*, or *cuts*, in which a more computationally tractable relaxation of the initial QCQP is formulated and then iteratively refined. Although there exist numerous families of cuts for QCQPs (see, for example, [12, 13, 17, 42, 43, 48, 51]), we focus on understanding and improving the performance of sequential *linear* cutting plane methods for solving QCQPs, which have the advantage over more complex cutting surfaces in that linear relaxations can be more easily embedded in intelligent search of the feasible region, such as spatial branch and bound [33].

One crucial aspect that has been largely underexplored for QCQPs is the effect of cut *sparsity*, in terms of the number of nonzero coefficients involved in the cut, on computational performance. Sparsity can be exploited by most modern linear programming solvers to obtain significant speedups; see for example, results and discussion in [1, 8, 25, 26, 47, 55]. While denser cuts are typically stronger, they also significantly increase the time required to solve the resulting relaxations and are associated with *tailing off* effects, in which numerical issues impair convergence.

We investigate this tradeoff between strength of dense cuts and potential speedup using sparse cuts, by developing an understanding of *which cuts* to generate and apply, with the goals of quickly improving the relaxation quality while encouraging favorable computational speed and convergence properties. Prior work has attempted to convert dense cuts into sparse ones [45]; in contrast, we draw a connection to literature from statistics on *sparse principal component analysis* (SPCA), in order to directly generate sparse cuts based on where they can best tighten the current relaxation. Sparse cuts generated in this manner have recently been shown to have promising strength in a computational study by Baltean-Lugojan et al. [5]. We substantially expand on those results through extensive computational experiments on larger-scale instances in which we evaluate sparse cuts, compared to dense ones, in terms of their strength, convergence properties, and effect on solution time.

Concretely, we consider nonconvex QCQPs of the form

$$\begin{aligned} \min_{x \in \mathbb{R}^n} \quad & x^\mathsf{T} Q_0 x + c_0^\mathsf{T} x \\ & x^\mathsf{T} Q_i x + c_i^\mathsf{T} x + d_i \leq 0 \qquad i = 1, \ldots, m. \end{aligned} \tag{QCQP}$$

The main computational difficulties that arise in solving (QCQP) are due to lack of convexity and numerical instability associated with nonlinear expressions. One method to overcome these hurdles that has garnered considerable attention is based on *semidefinite programming* (SDP).

The standard SDP relaxation of (QCQP), due to Shor [50], is obtained by adding new variables X_{ij}, $1 \leq i \leq j \leq n$, to replace and represent the bilinear terms $x_i x_j$ in (QCQP), and then relaxing the nonconvex condition $X = xx^\mathsf{T}$,

yielding

$$\begin{aligned} \min_{x,X} \quad & \langle X, Q_0 \rangle + c_0^\intercal x \\ & \langle X, Q_i \rangle + c_i^\intercal x + d_i \leq 0 \qquad i = 1, \dots, m \\ & X - xx^\intercal \succeq 0. \end{aligned} \tag{SDP}$$

The constraint $X - xx^\intercal \succeq 0$ is typically expressed, by Schur's complement, as

$$\begin{bmatrix} 1 & x^\intercal \\ x & X \end{bmatrix} \succeq 0. \tag{1}$$

The SDP relaxation is then typically further refined by adding the well-known linear McCormick inequalities [40], to obtain better bounds.

It has been shown that (SDP) can provide tight approximations to the optimal value of (QCQP); some notable examples of such strong performance are the maximum cut problem (MaxCut) [30], optimal power flow [37], and box-constrained quadratic programs (BoxQP) [2, 17, 19, 60]. We also refer the reader to [20, 56] and references therein for exactness results regarding the SDP relaxation.

Our goal goes beyond solving the SDP and obtaining a good dual bound: we would like to obtain an *outer approximation* of the original set that can mimic the strength of the SDP relaxation, but that is lightweight and *linear*. This way, it can be embedded in a branching scheme with ease in order to solve the original QCQP instance to provable optimality.

In principle, to approximate the SDP relaxation (SDP), one can exploit the maturity of linear programming (LP) solvers by iteratively refining LP relaxations of (SDP) in a cutting-plane fashion [49]. Consider a first LP relaxation of (SDP), obtained by removing the SDP constraint and adding the McCormick inequalities [40]:[1]

$$\begin{aligned} \min_{x,X} \quad & \langle X, Q_0 \rangle + c_0^\intercal x \\ & \langle X, Q_i \rangle + c_i^\intercal x + d_i \leq 0 \qquad i = 1, \dots, m \\ & \text{McCormick inequalities.} \end{aligned} \tag{LP}$$

Let $(\tilde{x}, \tilde{X})$ denote an optimal solution to this relaxation. Throughout the paper, we will refer to the matrix $M(x, X)$ for a given $(x, X) \in \mathbb{R}^n \times \mathbb{R}^{\binom{n}{2}+n}$, which we define as

$$M(x, X) \coloneqq \begin{bmatrix} 1 & x^\intercal \\ x & X \end{bmatrix}. \tag{M}$$

For notational convenience, we define

$$\tilde{M} \coloneqq M(\tilde{x}, \tilde{X}). \tag{$\tilde{M}$}$$

[1]The explicit form for the McCormick inequalities is not consequential for this paper, aside from them being linear. In fact, our results apply more generally, e.g., if the initial LP relaxation is strengthened with any valid linear inequalities for (QCQP).

If $\tilde{M} \succeq 0$, then (SDP) is solved. Otherwise, we can efficiently find a vector $v \in \mathbb{R}^{n+1}$ such that

$$v^\intercal \tilde{M} v < 0,$$

e.g., an eigenvector of $\tilde{M}$ with negative eigenvalue. Then, the *linear* inequality

$$\left\langle vv^\intercal, \begin{bmatrix} 1 & x^\intercal \\ x & X \end{bmatrix} \right\rangle \geq 0 \tag{2}$$

is valid for (SDP) and cuts off $(\tilde{x}, \tilde{X})$. This procedure can be viewed as a finite approximation of the semi-infinite outer description of the *positive semidefinite* (PSD) cone:

$$M \succeq 0 \iff \langle vv^\intercal, M \rangle \geq 0 \quad \forall v \in \mathbb{R}^{n+1}. \tag{3}$$

This family of cutting planes has been considered before (see [5, 6, 45, 49]), and the main drawback has been repeatedly acknowledged: the vector v will typically be *dense*, which can cause numerical instability in the LP after adding many cuts of the form (2). Fukuda et al. [29] provide one way of avoiding this issue, in the presence of *structured sparsity* of the objective and linear constraints in (SDP). Under such structured sparsity, one can (without any loss in relaxation quality) replace (1), the PSD requirement on the full matrix $M(x, X)$, with the requirement that a set of *small* principal submatrices of $M(x, X)$ is PSD, which implies that it suffices to only add the cuts of type (2) associated with the eigenvectors of those principal submatrices.

We instead follow a different direction, in which we directly enforce a target sparsity, say k, on the vector v. This involves searching for a v such that

$$v^\intercal \tilde{M} v < 0 \quad \text{and} \quad \|v\|_0 := |\text{supp}(v)| \leq k, \tag{4}$$

where $\text{supp}(v)$ denotes the nonzero entries of v and $|\cdot|$ is the cardinality operator. We call a vector $v \in \mathbb{R}^{n+1}$ a *k-sparse-eigencut* if it satisfies (4), and if the k-length vector obtained by only taking the nonzero elements of v is a unit eigenvector of the principal submatrix of $\tilde{M}$ defined on the indices in $\text{supp}(v)$.

Contributions We formulate the problem of finding a k-sparse-eigencut, or determining if none exists, as an optimization problem and show that it is equivalent to Sparse Principal Component Analysis (SPCA). This implies the problem is $\mathcal{NP}$-hard, and from the results of Blekherman et al. [9], we observe that there exist matrices where k-sparse-eigencuts exist, but there are no $(k-1)$-sparse-eigencuts. In spite of these negative worst-case results, the same connection with SPCA allows us to use an empirically-strong heuristic for this problem to efficiently compute one strong k-sparse-eigencut in practice. We then devise a novel scheme to compute multiple k-sparse-eigencuts for a given $\tilde{M}$, and we conduct extensive computational experiments using a cutting plane approach. Our results strongly advocate for the importance of sparsity, and show that a lightweight polyhedral approach can successfully approximate the quality of SDP-based approaches.

Notation We define $[n] := \{1, \ldots, n\}$. The cardinality of a set I is denoted by $|I|$. We denote the set of n-dimensional vectors with real entries by $\mathbb{R}^n$, and the set of $n \times n$ real-valued matrices by $\mathbb{R}^{n \times n}$. For $v \in \mathbb{R}^n$, $\|v\|_0$ denotes the number of nonzero entries in v. We use $\langle \cdot, \cdot \rangle$ to represent the matrix inner-product. For a matrix $X \in \mathbb{R}^{n \times n}$ and $I \subseteq [n]$, we let X_I be the principal submatrix of X given by the columns and rows indexed by I. Similarly, v_I will be the vector corresponding to the entries of v indexed by I. We denote by $\mathcal{S}^n_+$ the cone of $n \times n$ PSD matrices. For an $n \times n$ matrix X, we also use $X \succeq 0$ to denote $X \in \mathcal{S}^n_+$. We let $\mathcal{S}^{n,k}_+$ denote the cone of $n \times n$ matrices such that every $k \times k$ principal submatrix is PSD; that is,

$$\mathcal{S}^{n,k}_+ := \{X \in \mathbb{R}^{n \times n} \, : \, X_I \in \mathcal{S}^k_+, \, \forall I \subseteq [n], |I| = k\}.$$

2 Literature review

Given the vast literature involving cutting planes for nonconvex QCQPs, here we restrict ourselves to reviewing approaches which rely on the structure

$$X = xx^{\mathsf{T}}, \, x \in [\ell, u], \tag{5}$$

in order to derive inequalities. We refer the reader to [12] for a survey on cutting planes methods for nonlinear optimization.

Historically, the main limitation of using the SDP relaxation (SDP) has been its lack of scalability, but significant progress has been made on this front; see, for example, [18, 62]. One way to alleviate the computational burden of the SDP is leveraging *structured sparsity*. When the bilinear terms are represented as edges of a graph, and this graph is close to being a tree, one can use the framework of Fukuda et al. [29] to avoid imposing (1) on a large matrix, and instead enforce the PSD requirement over small submatrices. We refer the reader to [35, 36, 54] and references therein for other approaches exposing structured sparsity from an SDP perspective.

The most common way to exploit (5) is by adding the McCormick inequalities [40]. These are valid inequalities derived from each equality $X_{i,j} = x_i x_j$ and variable bounds. Using these inequalities can provide strong relaxations in combination with the SDP constraint (1) [2, 16, 17, 19, 60].

More generally, in the presence of (5), one can use valid inequalities for the Boolean Quadric Polytope [44] in order to obtain valid inequalities for (QCQP). Their strong computational performance for BoxQP was shown by Bonami et al. [11]. See also [14, 17, 59] and [13, 23, 27, 46] for cuts that are valid for the convex hull of side constraints together with (5).

An alternative way to generate cuts in the presence of (5) is through the construction of convex sets whose interior *does not* intersect the feasible region. These sets can be used via the intersection cut [4, 52] framework in order to construct valid inequalities. The work in the papers [6, 7, 22, 43] falls in this category. Bienstock et al. [7] introduce the concept of *outer-product-free sets* as convex sets of matrices that do not include any matrix of the form $X = xx^{\top}$

in its interior. Fischetti and Monaci [28] construct *bilinear-free* sets through a bound disjunction and McCormick inequalities.

In contrast to the techniques mentioned above, which would cut through the constraint (1), we derive valid inequalities for the semidefinite *relaxation* of $X = xx^\intercal$ in (5), in which this constraint is replaced with (1). The inequalities we construct are valid for $\{(x, X) : M(x, X) \succeq 0\}$, and additionally are required to be sparse. There are two papers that propose a similar methodology. Qualizza et al. [45] look for k-sparse-eigencuts by iteratively setting to zero the components of a dense eigenvector until reaching the target sparsity. Baltean-Lugojan et al. [5] use a neural network in order to select a promising $k \times k$ submatrix of $M(x, X)$, for $k \leq 6$, to compute a k-sparse-eigencut for the problem. A theoretical analysis on the quality achieved by k-sparse-eigencuts was recently provided in [9, 10]. The authors provide upper and lower bounds on the distance between $\mathcal{S}^n_+$ and $\mathcal{S}^{n,k}_+$.

3 The k-sparse separation problem and SPCA

We seek a k-sparse-eigencut violated by $(\tilde{x}, \tilde{X})$, which can be thought of as an optimal solution to (LP) or a different relaxation, in which $\tilde{M} \not\succeq 0$. This is equivalent to determining if $\tilde{M} \in \mathcal{S}^{n,k}_+$, since it can easily seen that

$$A \in \mathcal{S}^{n,k}_+ \iff v^\intercal A v \geq 0 \quad \forall v \in \mathbb{R}^n, \|v\|_0 \leq k.$$

Finding a k-sparse-eigencut, or determining if none exists, is closely related to the SPCA problem, which we define next.

Definition 1. *Given a matrix $A \in \mathcal{S}^{n+1}_+$ and a sparsity level $k \in \mathbb{N}$, the* k-Sparse Principal Component Analysis (k-SPCA) *problem is defined as*

$$\begin{aligned} \max_{v \in \mathbb{R}^{n+1}} \quad & v^\intercal A v \\ & \|v\|_2 = 1 \\ & \|v\|_0 \leq k. \end{aligned} \tag{6}$$

Its decision version reads: given $A \in \mathcal{S}^{n+1}_+$, $k \in \mathbb{N}$, and $K \in \mathbb{R}_{\geq 0}$, determine if there exists $v \in \mathbb{R}^n$ such that

$$\|v\|_2 = 1, \quad \|v\|_0 \leq k, \quad v^\intercal A v > K. \tag{7}$$

We begin by showing that one can reduce the decision version of the SPCA problem to determining if a k-sparse-eigencut exists. The proof is trivial.

Proposition 1. *Let $A \in \mathcal{S}^{n+1}_+$, $k \in \mathbb{N}$ and $K \in \mathbb{R}$. A vector v satisfies* (7) *if and only if v is a k-sparse-eigencut of $\tilde{A}$ defined as*

$$\tilde{A} := KI - A,$$

where I is the $(n+1) \times (n+1)$ identity.

This immediately implies that finding a k-sparse-eigencut is $\mathcal{NP}$-hard [39]. Although we cannot hope for an efficient algorithm for finding our proposed cuts, SPCA is a well-studied problem and efficient practical heuristics exist [3, 24, 34, 38, 61]. We pursue the connection of our separation problem and SPCA through the problem of finding the *most violated* k-sparse-eigencut:

$$\begin{aligned} \min_{v \in \mathbb{R}^{n+1}} \quad & v^\intercal \tilde{M} v \\ & \|v\|_2 = 1 \\ & \|v\|_0 \leq k. \end{aligned} \tag{8}$$

If the value of (8) is negative, we obtain a valid k-sparse-eigencut, and otherwise, none exists. Whenever the matrix $\tilde{M}$ is negative semidefinite, problem (8) is exactly a k-SPCA problem as (6). In our case, however, we are interested in studying the problem with no assumption over $\tilde{M}$. Lemma 1 shows that, in any case, (8) is equivalent to (6).

Lemma 1. *For every $(\tilde{x}, \tilde{X}) \in \mathbb{R}^n \times \mathbb{R}^{\binom{n}{2}+n}$ and $k \in \mathbb{N}$, there exists a matrix $A \succeq 0$ such that* (8) *is equivalent to solving the k-SPCA problem* (6)*.*

Proof. Let $\lambda^{\max}$ be the largest eigenvalue of $\tilde{M}$. We can equivalently rewrite (8) as

$$\begin{aligned} \min_{v} \quad & v^\intercal (\tilde{M} - \lambda^{\max} I) v + \lambda^{\max} v^\intercal v \\ & \|v\|_2 = 1 \\ & \|v\|_0 \leq k, \end{aligned}$$

with I the $(n+1) \times (n+1)$ identity matrix. Since $\|v\|_2 = 1$, the above is equivalent to

$$\begin{aligned} \lambda^{\max} - \max_{v} \quad & v^\intercal (\lambda^{\max} I - \tilde{M}) v \\ & \|v\|_2 = 1 \\ & \|v\|_0 \leq k. \end{aligned}$$

It easy to see that $\lambda^{\max} I - \tilde{M}$ is positive semidefinite. We define $A = \lambda^{\max} I - \tilde{M}$ and conclude that (8) is equivalent to an SPCA problem. □

Due to the $\mathcal{NP}$-hardness results discussed above, it is unlikely there exists a polynomial-time algorithm to compute a separating k-sparse-eigencut. However, one could still hope to always find a good cut independently of the computing time involved. The following example, constructed by Blekherman et al. [9], shows that this cannot be done in general.

Example 1. *Consider the $n \times n$ matrix $G(a, b)$ defined by as the matrix with b in the diagonal entries and $-a$ in the off-diagonal entries, for some $a, b \geq 0$.*

That is, $G(a,b)$ takes the form

$$G(a,b) = \begin{bmatrix} b & -a & \cdots & -a \\ -a & b & \cdots & -a \\ \vdots & & \ddots & \vdots \\ -a & \cdots & -a & b \end{bmatrix}.$$

The results in [9] show that if a, b are such that

$$(k-1)a \leq b < (n-1)a,$$

then all $k \times k$ submatrices of $G(a,b)$ are PSD, but $G(a,b) \not\succeq 0$. In other words, even if a given $n \times n$ matrix is not PSD, there is no guarantee that an $(n-1)$-sparse-eigencut exists. ■

In Example 1, the matrix $G(a,b)$ is dense, and one may naturally wonder if the same result holds for a sparse matrix, since in this case one might expect it would be easier to find sparse cuts. The next example shows that no such hope can be realized in general.

Example 2. *Consider now a $n \times n$ matrix defined as $G(a,b)$ and let $m \in \mathbb{N}$. Let $N = nm$ and define the block diagonal $N \times N$ matrix $\tilde{G}(a,b)$ whose blocks are given by $G(a,b)$, with sparsity $1/m$.*

Since $\tilde{G}(a,b) \succeq 0$ if and only if $G(a,b) \succeq 0$, it suffices to consider

$$(n-2)a \leq b < (n-1)a$$

in order to obtain that $\tilde{G}(a,b) \not\succeq 0$. In this case, the existence of n-sparse-eigencuts is guaranteed, but there is no $(n-1)$-sparse-eigencut. ■

While the above examples are negative results, they are only informing us that we cannot hope to devise an efficient method that will work in the worst case. In practice, the story might be different. In the following, we analyze the empirical performance of these cuts.

4 Empirical expressiveness of k-sparse-eigencuts

The computational hardness of solving (8) to find a k-sparse-eigencut lies in selecting an optimal *support*, i.e., where the nonzeroes in the solution should be. If the support of size k of an optimal solution of (8) is known, finding the actual optimal solution reduces to computing an eigendecomposition of the corresponding $k \times k$ submatrix of $\tilde{M}$. This support selection is what Baltean-Lugojan et al. [5] tackle via a neural network. Their results suggest that k-sparse-eigencuts can substantially close the gap between the optimal values of (LP) and the SDP relaxation with McCormick inequalities added. However, for our purposes, we desire a more refined analysis of the *expressiveness*, or modeling power, of k-sparse-eigencuts, particularly in comparison to the performance of *dense* eigencuts.

In this section, we perform experiments with low-dimensional instances for which we can exhaustively enumerate all $\binom{n+1}{k}$ possible supports, to compute cuts via the eigendecompositions of all possible $k \times k$ principal submatrices of $\tilde{M}$. Our goal is to answer the following questions:

Question 1. What is the right number of k-sparse-eigencuts to add?

Question 2. What is the appropriate level of sparsity, k, to use?

Question 3. Given a budget on the number of cuts we can add, can we efficiently identify a set of strong k-sparse-eigencuts?

To this end, first, in Section 4.3.1, we evaluate how much a *single* k-sparse-eigencut can improve the dual bound in an optimization problem. Then, in Section 4.3.2, we assess how much the dual bound improves if we add multiple k-sparse-eigencuts, including *all* possible cuts of this type, or only a limited number of cuts. To answer Question 3, we examine metrics by which a good set of eigencuts can be selected, in order to find a few k-sparse-eigencuts that are responsible for most of the bound improvement.

The answer to these questions motivate the choices we make in our main algorithm, Algorithm 1, which is presented in Section 6. Specifically, the enumeration experiments justify how many sparse cuts we should generate at each round, and whether or not we should allow the use of a few dense eigencuts.

4.1 Experimental setup for enumeration experiments

We consider three low-dimensional instances of the BoxQP library [15, 53]: `spar30-060-1`, `spar30-080-1` and `spar030-100-1`; these have the form

$$\min_x \; x^\intercal Q x + c^\intercal x$$
$$x \in [0,1]^n$$

with $Q \not\succeq 0$. For these instances, $n = 30$, and the second number in their names (60, 80, and 100) indicates (roughly) the percentage of nonzero entries in Q. While for succinctness we report on only these three instances, we obtained similar outcomes with other instances from the family.

For each instance, we first solve the LP relaxation (LP) of (SDP). As before, let $(\tilde{x}, \tilde{X})$ be an optimal solution to this LP. Then, we enumerate all possible $\binom{n+1}{k}$ supports for $k = 4$, and compute eigenvectors for all negative eigenvalues of the corresponding $k \times k$ submatrix of $\tilde{M}$. We also compute *dense* cuts, which are just eigenvectors of all negative eigenvalues of $\tilde{M}$. We only add one round of cuts. A *round* of cuts is obtained from a given LP solution, without adding any new cuts to the LP and reoptimizing to get a new solution.

We implemented the enumeration code in C++. We use the Eigen library [31] for linear algebra computations and the LP solver is Gurobi 9.0 [32]. In order to measure relaxation quality, we also solve the SDP relaxation (SDP) using the C++ Fusion API of MOSEK version 9.2 [41].

Table 1: Percent gap closed by k-sparse-eigencuts from a single support, contrasted with requiring a single $k \times k$ principal submatrix to be PSD, and with adding dense cuts.

	1-supp-cuts		1-supp-PSD		
Instance	Max	Avg	Max	Avg	Dense cuts
`spar30-060-1`	3.37	0.25	3.69	0.31	38.77
`spar30-080-1`	4.19	0.57	4.83	0.75	49.31
`spar30-100-1`	4.56	0.95	4.95	1.20	60.94

4.2 Performance measures

We measure strength through the commonly used *gap closed*. Let us denote the initial (LP) optimal value as $\mathrm{LP}_{\mathrm{opt}}$ and the SDP optimal value of (SDP) with McCormick inequalities as $\mathrm{SDP}_{\mathrm{opt}}$, which is an upper bound on the value any set of eigencuts can achieve. For a subsequent LP relaxation LP', obtained by adding additional cuts to the base LP, the gap closed $GC(\mathrm{LP}')$ is

$$GC(\mathrm{LP}') = 100 \times \frac{\mathrm{LP}'_{\mathrm{opt}} - \mathrm{LP}_{\mathrm{opt}}}{\mathrm{SDP}_{\mathrm{opt}} - \mathrm{LP}_{\mathrm{opt}}}.$$

Given an optimal solution $(\tilde{x}, \tilde{X})$ to the initial LP relaxation and an eigencut of the form (2), that is, $\langle vv^{\mathsf{T}}, \tilde{M} \rangle \geq 0$, the *violation* is the nonnegative number $-\langle vv^{\mathsf{T}}, \tilde{M} \rangle$. In all our experiments, $\|v\|_2 = 1$, making the violation of different cutting planes comparable.

While we focus on relaxation strength in this section, we apply more comprehensive quality metrics, including solution time, in the more extensive experiments of Section 6.

4.3 Results

4.3.1 Single support reports

In Table 1, for $k = 4$, we summarize how much gap is closed by a single $k \times k$ principal submatrix through cuts, compared to requiring that submatrix is PSD and to dense cuts. The columns labeled *1-supp-cuts* show the maximum and average gap closed when all k-sparse-eigencuts for a single $k \times k$ support are added to (LP). The columns labeled *1-supp-PSD* show the maximum and average gap closed when a single $k \times k$ support is imposed to be PSD. The column labeled *Dense cuts* shows the gap closed by adding all dense cuts.

From this table, we draw several conclusions. First, we see that convexifying a single $k \times k$ principal submatrix has only limited impact on gap closed. Adding a PSD requirement for one $k \times k$ submatrix never closes more than 5% of the gap, and the average is usually around 1%. The corresponding k-sparse-eigencuts also do not perform well in this case, though the gap closed from a round of cuts is

Table 2: Percent gap closed when multiple supports are used to generate k-sparse-eigencuts, contrasted with requiring the corresponding principal submatrices to be PSD, and with adding dense cuts.

	k-cuts			k-PSD		
Instance	All	Top (gap)	Top (viol)	All	Top (psd)	Dense cuts
`spar30-060-1`	50.51	31.46	18.93	61.31	44.38	38.77
`spar30-080-1`	80.95	63.20	42.03	91.53	84.08	49.31
`spar30-100-1`	87.84	63.29	66.01	95.32	82.51	60.94

not significantly different from the gap closed by imposing a single submatrix to be PSD. Second, the performance of dense cuts is remarkable: a large percentage of the gap is closed with only one round of cuts, which for these instances is 14–16 cuts. Lastly, there is a trend that, overall, cuts are more effective in dense instances. This is somewhat expected: the larger the number of zeros in the objective, the less the objective value will change when a sparse support is convexified. However, the strength of sparse cuts from a single support, as a proportion of the gap closed by dense cuts, is higher in sparser instances.

4.3.2 Multiple support reports

The single-support experiments suggest that, in order to have some impact with k-sparse-eigencuts, adding cuts across many supports simultaneously is necessary. This raises the question of which supports to use, given that, in practice, we have a budget on the number of cuts we can add. We examine this in our next set of experiments, in which we evaluate different support selection criteria. We assign a score to each of the 31,465 supports, then select the top 5% of supports (1,574 supports) having the highest score.

The scores we test for each support are: (1) *gap*, which is the change in objective value when all k-sparse-eigencuts from that support are added; (2) *violation*, which is the maximum violation of any k-sparse-eigencut from that support; and (3) *psd*, which denotes the change in objective value after adding the PSD requirement on that support. We include both gap and violation as scores, because violation is commonly used to measure the importance of a cut, due to it being cheap to compute relative to resolving an LP; hence, we are interested in evaluating whether violation is a good proxy for expected objective improvement in this setting.

Table 2 shows the results. Columns 2–4, with the heading k-cuts, give the gap closed by k-sparse-eigencuts when all possible such cuts are added, and when the top 5% of cuts are added, sorted by "gap" and "violation". Columns 5 and 6, with the heading k-PSD, show the gap closed when requiring $k \times k$ principal submatrices are PSD, both for all such matrices, and for only the top 5% sorted by their "psd" score. Column 7 repeats the gap closed by dense cuts from Table 1.

From Table 2, we observe many interesting phenomena. First, we see that adding all sparse cuts significantly outperforms all dense cuts; however, while only 14–16 dense cuts are added per instance, the number of sparse cuts is in the thousands. On one hand this indicates sparse cuts can replicate the strength of dense cuts, but on the other hand, any speed advantages from resolving the LP with sparser cuts are likely to be negated by that many additional constraints. Second, adding cuts from only the top 5% of supports can achieve 60 to 77% of the gap closed by all supports. This indicates that, although multiple supports need to be considered to produce a strong relaxation, an intelligent choice of a small number of supports can suffice. Last, concerning the violation measure: while, on average, the selection of cuts via violation performs worse than its gap closed counterpart, it can happen, as in `spar30-100-1`, that the top 5% of supports sorted by violation together close more gap than if the top 5% of supports were chosen by their individual objective improvement.

4.4 Summary of enumeration experiments

We conclude this section with the high-level messages imparted by these experiments. First, k-sparse-eigencuts closely emulate, in strength, the addition of the PSD requirement to a support. Second, no single support has a significant effect in terms of gap closed. Therefore, one should find multiple supports that together perform well. Third, the violation of a cut presents a good and efficient heuristic for cut selection in this setting. Finally, dense cuts are remarkably strong, considering there are usually very few of them available.

We do not directly consider interactions among supports: that is, we score individual k-length subsets of $[n+1]$, whereas it may be better to score, for example, pairs of supports, or otherwise incorporating combinatorial effects.

Since the experiments in this section required enumeration of all $\binom{n+1}{k} \approx \mathcal{O}((n+1)^k)$ possible sparse supports, the technique is not practical for even medium-size instances or moderate values of k. The same drawback is observed by Baltean-Lugojan et al. [5]; their neural-network-based approach can estimate if a *given* support will have a considerable effect (in terms of the cut generated), but they still need to enumerate supports for their evaluation. Our next goal is to effectively choose strong sets of sparse supports *on the fly*.

5 Computation of multiple sparse cuts

We develop an algorithm to compute multiple k-sparse-eigencuts for a given matrix $\tilde{M}$, using an oracle that can compute a single k-sparse-eigencut. We compare the performance of the cutting planes produced by this algorithm with the results of the previous low-dimensional enumeration experiments.

5.1 Iterative k-sparse-eigencut computation

Our strategy to compute multiple sparse cuts starts with an oracle that efficiently returns a single cut with sparsity level k. Instead of using that cut directly, we take its support, say I, and we add all the k-sparse-eigencuts from $\tilde{M}_I$.[2] Lemma 2 formally states that such cuts exist from $\tilde{M}_I$.

Lemma 2. *Let $\tilde{M} \in \mathbb{R}^{(n+1)\times(n+1)}$ be a symmetric matrix, and suppose $w \in \mathbb{R}^{n+1}$ is such that $w^\intercal \tilde{M} w < 0$. Let $I := \mathrm{supp}(w)$. The number of k-sparse-eigencuts of $\tilde{M}_I$ is between 1 and k.*

Proof. The number of k-sparse-eigencuts from $\tilde{M}_I$ depends on its number of negative eigenvalues. The result follows because the matrix $\tilde{M}_I$ has at most k distinct unit eigenvectors, and at least one of those is associated with a negative eigenvalue. This second fact follows from the existence of a sparse cut using w, since $0 > w^\intercal \tilde{M} w = w_I^\intercal \tilde{M}_I w_I$, so that $\tilde{M}_I$ is not positive semidefinite. □

Based on the results in Section 4, we know that limiting ourselves to one support may lead to a weak set of cuts. The next question is how to use $\tilde{M}$ to generate more cuts, having exhausted the ones from the initial support I. A natural first idea is to target a set of indices from $[n+1]$ that is orthogonal to I, to diversify the areas of $\tilde{M}$ being convexified. We implemented a version of this proposal, and on small- to medium-size BoxQP instances, there were encouraging results. However, this approach is unable to adequately capitalize on another observation from Section 4, that a sufficiently large number of sparse cuts can outperform dense cuts. For example, for a moderate sparsity level such as $k = n/4$, the orthogonal scheme can only generate up to 4 supports and n cuts (by Lemma 2), while we desire more cuts from each iteration.

With that motivation, we present Algorithm 1, our strategy to compute multiple sparse cuts from $\tilde{M}$, without solving another LP. The oracle referenced within Algorithm 1 solves (8). It can be helpful for intuition to keep in mind the case that $k = n+1$, for which Algorithm 1 simply returns all eigenvectors with a negative eigenvalue in an eigendecomposition of the matrix.

We now establish finiteness of this algorithm.

Lemma 3. *Algorithm 1 terminates in a finite number of iterations.*

Proof. At each start of the while loop, the matrix M_I^i has signature[3] (p, q) with $q \geq 1$ (otherwise, the algorithm would have terminated), and M_I^{i+1} has signature $(p, q-1)$. Additionally, if an arbitrary $k \times k$ principal submatrix M_S^i, with $S \subseteq [n+1]$, $|S| = k$, has signature (p, q), then M_S^{i+1} has signature (p', q') with $q' \leq q$. This is because M_S^{i+1} is obtained from adding a PSD matrix $(-\lambda_i w_i w_i^\intercal)$ to M_S^i.

Thus, at each step, the number of negative eigenvalues of *every* $k \times k$ submatrix does not increase, and decreases strictly for at least one such submatrix.

[2] Adding all k-sparse-eigencuts, as opposed to just one per support, is shown to be effective in the computational results of Qualizza et al. [45] with their *Minor PSD cuts*.

[3] The signature of a matrix is (p, q) if it has p positive and q negative eigenvalues.

Algorithm 1: SparseRound($\tilde{M}, k$): one round of k-sparse-eigencuts

Input : A matrix $\tilde{M} \in \mathbb{R}^{(n+1)\times(n+1)}$ with $\tilde{M} \not\succeq 0$, and a sparsity level $k \in \mathbb{N}$.
Parameters: MaxNumSupports: maximum number of considered supports.
Oracle: oracle for solving (8).
Output : A sequence of k-sparse-eigencuts $\{\tilde{w}_j\}_{j=1}^{p}$ such that
$\|\tilde{w}_j\|_2 = 1, \|\tilde{w}_j\|_0 \le k$, and $\tilde{w}_j^\top \tilde{M} \tilde{w}_j < 0$, for $j \in [p]$.

Initialize: $p \leftarrow 0$, $i \leftarrow 1$, $M^1 \leftarrow \tilde{M}$, and $w \leftarrow \text{Oracle}(M^1)$;
while $w^\top M^i w < 0$ *and* $i <$ MaxNumSupports **do**
  $I \leftarrow \text{supp}(w)$;
  Let $\lambda_i^{\min}$ and q_i denote the most negative eigenvalue, and associated unit eigenvector, of M_I^i;
  Let $\tilde{w}_i$ denote q_i lifted to $\mathbb{R}^{n+1}$ by setting all components not in I to 0;
  $M^{i+1} \leftarrow M^i - \lambda_i^{\min} \tilde{w}_i \tilde{w}_i^\top$;
  $i \leftarrow i+1$ and $p \leftarrow p+1$;
  $w \leftarrow \text{Oracle}(M^i)$;
end
return $\{\tilde{w}_j\}_{j=1}^{p}$;

Since there are at most $\binom{n+1}{k}$ principal $k \times k$ submatrices, the algorithm finishes in at most $k\binom{n+1}{k}$ steps. □

The following lemma shows that all k-sparse-eigencuts generated by Algorithm 1 are valid inequalities being violated by $\tilde{M}$. Additionally, it shows precisely which matrices M^i are being cut by each generated k-sparse-eigencut.

Lemma 4. *The sequence of vectors $\{\tilde{w}_j\}_{j=1}^{p}$ generated by Algorithm 1 satisfies*

1. $\tilde{w}_i^\top M^i \tilde{w}_i < 0$ *for every* $i \in [p]$,
2. $\tilde{w}_i^\top \tilde{M} \tilde{w}_i < 0$ *for every* $i \in [p]$, *and*
3. $\tilde{w}_i^\top M^{i+1} \tilde{w}_i = 0$ *and* $\tilde{w}_j^\top M^{i+1} \tilde{w}_j > 0$, *for* $1 \le j \le i-1 \le p$.

Proof. Part 1 follows by assumption of the existence of a cut returned by Oracle. Part 2 follows by noting that

$$\tilde{M} = M^i + \sum_{j=1}^{i-1} \lambda_j^{\min} \tilde{w}_j \tilde{w}_j^\top.$$

Therefore,

$$\tilde{w}_i^\top \tilde{M} \tilde{w}_i = \tilde{w}_i^\top M^i \tilde{w}_i + \sum_{j=1}^{i-1} \lambda_j^{\min} (\tilde{w}_i^\top \tilde{w}_j)^2 < 0,$$

where the last inequality follows from Part 1 and the fact that $\lambda_j^{\min} < 0$. The first statement in Part 3 follows since $\|\tilde{w}_i\|_2 = 1$ and it is the eigenvector associated to $\lambda_i^{\min}$, thus

$$\tilde{w}_i^\top M^{i+1} \tilde{w}_i = \tilde{w}_i^\top M^i \tilde{w}_i - \lambda_i^{\min} = 0.$$

Table 3: Summary of gap closed by multiple k-sparse-eigencuts selected by Algorithm 1, contrasted with same number of cuts selected via enumeration and sorting and imposing PSD-ness.

	k-cuts			k-PSD
Instance	Algorithm 1	Top (gap)	Top (violation)	Top (psd)
`spar30-060-1`	5.48	9.06	6.09	10.01
`spar30-080-1`	14.98	16.83	14.75	25.60
`spar30-100-1`	22.62	25.48	23.22	30.52

For the second statement in Part 3, we proceed via induction over $i - j \geq 1$. For $j = i - 1$,

$$\tilde{w}_{i-1}^\intercal M^{i+1} \tilde{w}_{i-1} = \tilde{w}_{i-1}^\intercal M^i \tilde{w}_{i-1} - \lambda_i^{\min} (\tilde{w}_i^\intercal \tilde{w}_{i-1})^2 = -\lambda_i^{\min} (\tilde{w}_i^\intercal \tilde{w}_{i-1})^2 > 0.$$

Lastly, for general $i - j$,

$$\tilde{w}_j^\intercal M^{i+1} \tilde{w}_j = \tilde{w}_j^\intercal M^i \tilde{w}_j - \lambda_i^{\min} (\tilde{w}_i^\intercal \tilde{w}_j)^2 > 0,$$

where the inequality follows by the inductive step. □

5.2 Enumeration experiments revisited

To get a sense of the quality of the cuts produced by Algorithm 1, we compare them to the other selection procedures evaluated in Section 4.3.2.

Our implementation of ORACLE used in Algorithm 1 is based on the *Truncated Power Method* (TPower) by Yuan and Zhang [61], which is an efficient practical heuristic, with some theoretical guarantees, to generate high-quality solutions for the SPCA problem. Specifically, we run TPower after appropriately modifying the current solution $\tilde{M}$ as per our discussion in Lemma 1.

The results of the experiments are shown in Table 3. For these small examples, Algorithm 1 does not generate as many cuts as scores we considered in the earlier enumeration experiments. For this reason, we restrict the number of supports considered for the "top" selection in Table 3 to be the same number of supports used by Algorithm 1.

We see that Algorithm 1 performs quite well. It is always within 1% of the gap closed by the supports selected by "violation", and less that 4% away from the gap-closed-based selection. Unlike the other scores, Algorithm 1 does not require a complete enumeration of the supports, and it does not require us to solve an LP or SDP: it achieves our goal of generating supports dynamically.

At this point, we have motivated the use of k-sparse-eigencuts, and we have shown a practical algorithm that can generate many cuts on the fly, using the connection of the separation problem with SPCA. Additionally, we have seen in Section 4 that dense cuts are usually quite strong, and therefore it is sensible

to also consider them when building a high-quality approximation. One needs to be careful though, since adding too many dense cuts can impair the efficient solvability of the LPs, which was the motivation of this work in the first place. In what follows, we show a computational study which includes a way of balancing these forces.

6 Computational experiments

We present a computational study of the tradeoffs between strength and efficiency in eigenvector-based cuts that approximate the semidefinite constraint (1). To do this, we compare the performance of k-sparse-eigencuts and dense cuts in a pure cutting plane procedure.

6.1 Implementation details

All of our algorithms are implemented in C++. As in the experiments in Section 4, we use the Eigen library [31], version 3.3.7, for linear algebra computations and the LP solver is Gurobi 9.0 [32]. We use the C++ Fusion API of MOSEK 9.2 [41] to solve SDP relaxations.

The k-sparse-eigencut oracle used in Algorithm 1 is the same described in Section 5.2: we execute the Truncated Power Method (TPower) by Yuan and Zhang [61] after modifying the current iterate's $\tilde{M}$ as in Lemma 1.

All experiments were performed single-threaded in shared computing environments. Most of the reported results used nodes from the Béluga cluster of Calcul Québec, in which each machine has two Intel Xeon Gold 6148 Skylake CPUs clocked at 2.4 GHz, and a variable amount of memory depending on the node. Some experiments with larger instances were run on a shared server with 512 GB of memory and thirty-two Intel Xeon Gold 6142 CPUs clocked at 2.6 GHz.

6.1.1 Cutting plane algorithm

We use a straightforward implementation of a cutting plane algorithm. We first solve (LP), the standard LP relaxation of (SDP) with added McCormick inequalities for bilinear terms, to obtain a solution $(\tilde{x}, \tilde{X})$. If the corresponding $\tilde{M}$ is not PSD, we use Algorithm 1 to find linear inequalities of the form (2) that are violated by $\tilde{M}$. We then resolve the LP updated with the new cuts, and we repeat the above process until we reach a terminating condition. We next specify which specific cuts are added in each iteration, how we obtain the LP optimal solution $\tilde{M}$, and what are the parameters and numerical tolerances used for our procedure.

Cutting plane families We consider two cut families: *k-sparse-eigencuts*, which were extensively described throughout the paper, and *dense* cuts, which

are the eigenvectors corresponding to the negative eigenvalues of $\tilde{M}$. For dense cuts, we add one for every distinct negative eigenvalue.

The results of the enumeration experiments from Section 4 showed that even a few dense cuts are able to close a large amount of gap. On the other hand, sparse cuts are likely to yield faster LPs. Thus, with dense cuts, one may expect to have a fast improvement in the bound, followed by a tailing off as each iteration takes longer, as compared to sparse cuts for which more iterations might be possible, but each iteration may close less gap. We explore how these two phenomena interact with the following three algorithm types:

- `DENSE`: At each iteration, add all cuts obtained from the eigenvectors associated to negative eigenvalues of the matrix $\tilde{M}$.
- `SPARSE`: At each iteration, generate k-sparse-eigencuts through Algorithm 1; a more detailed description is given in Algorithm 2.
- `HYBRID`: This strategy begins with `DENSE`, switching to `SPARSE` when the LP solve time is at least the value of a parameter HybridSwitchingTime.

The motivation for `HYBRID` is to mix dense and sparse cuts, by focusing on rapid improvement in gap closed in the beginning through dense cuts, and then switching to sparse cuts to moderate the growth in the LP solution time.

Algorithm 2: `SPARSE`(LP, k): k-sparse-eigencuts

Input : Initial LP relaxation of (QCQP), e.g., (LP), and sparsity level $k \in \mathbb{N}$.
Parameters: NumCutsPerIter: maximum number of cuts per iteration;
TerminatingConditions: check when execution should terminate.
Output : A sequence of k-sparse-eigencuts $\{\tilde{w}_j\}_{j=0}^{p}$ such that $\|\tilde{w}_j\|_2 = 1, \|\tilde{w}_j\|_0 \leq k$ for all $j \in [p]$.

Initialize: $\text{LP}_1 \leftarrow \text{LP}$, $p \leftarrow 0$, $t \leftarrow 1$;
while TerminatingConditions *have not been met* **do**
  Let $\tilde{M}$ be an optimal solution to LP_t;
  Let $\{\tilde{w}_j^t\}_{j=0}^{p_t}$ be the output of SparseRound($\tilde{M}, k$), i.e., Algorithm 1, sorted in order of decreasing violation with respect to $\tilde{M}$;
  $p'_t \leftarrow \min\{p_t, \text{NumCutsPerIter}\}$;
  $\text{LP}_{t+1} \leftarrow \text{LP}_t$ with the addition of the first p'_t cuts of form (2) from $\{\tilde{w}_j^t\}_{j=0}^{p_t}$;
  $\tilde{w}_{p+j} \leftarrow \tilde{w}_j^t$ for all $j \in [p'_t]$;
  $p \leftarrow p + p'_t$;
  $t \leftarrow t + 1$;
end
return $\{\tilde{w}_j\}_{j=0}^{p}$;

Solution to cut In an LP solver, one typically has three basic choices of optimization methods: *Simplex*, *Barrier*, and *Barrier with Crossover*. The first and third methods generate an extreme-point solution to the LP, while the second only guarantees a solution in the optimal face (which may or may not be a vertex). We conducted extensive preliminary experiments to determine which strategy to choose, and concluded that *Barrier* always performs better

for both sparse and dense cuts. It is not surprising that the quality of the cuts is better in this case, as cutting off a point in the interior of the optimal face can remove a larger portion of said face. However, within a cutting plane algorithm, a sequence of LPs can often be solved substantially faster with *Simplex*, using its warm start capabilities, than with *Barrier*. Yet in our early experiments, the faster iterations afforded by *Simplex* led to significantly worse, and ultimately slower, bound improvements than with *Barrier*.

Cut management As discussed in Section 4, at each iteration of the cutting plane algorithm, it is desirable to add a large number of k-sparse-eigencuts to ensure sufficient progress. As accumulating an excessive number of inequalities can slow down the LP solution time, especially when many of them become permanently inactive after a few iterations, we implemented a simple cut management policy that is used in all our experiments: a cut is removed from the pool if it is inactive for more than one iteration.

6.1.2 Parameters

Sparsity We set a target sparsity level of $k = \lfloor 0.25(n+1) \rfloor$. This implies that all of the cuts that we add have fewer than 7% nonzero entries in the matrix space, $M(x, X)$.

Limits on cuts and supports We set NumCutsPerIter in Algorithm 2 to $5n$, and MaxNumSupports in Algorithm 1 to 100. We set HybridSwitchingTime, the time limit for generating dense cuts in the `HYBRID` algorithm, to min{10 seconds, 100 times the initial LP solve time}.

TPower initialization A significant detail in our implementation is the initialization of TPower. The TPower method incorporates a truncation step (to achieve sparsity) in the classical power method to compute the largest eigenvalue of a PSD matrix. Since this algorithm is a heuristic for SPCA, the initialization plays a significant role in the performance. We found that initializing TPower with the eigenvector associated to the smallest eigenvalue generally leads to the best (and most consistent) behavior.

Tolerances We use a variety of tolerances throughout our procedure to ensure numerical stability. We say that the cut with respect to a vector v is violated by $\tilde{M}$ if $v\tilde{M}v^\intercal < -\epsilon$, where we set $\epsilon = 10^{-7}$. An eigenvalue is considered negative if it is less than -10^{-6}. If the coefficient of a cut is less than 10^{-9}, we treat it as zero. To measure the progress of the algorithm, at each iteration, we compare the objective value, say z_{new}, to the previous iteration's objective value, say z_{old}; if $|z_{\text{new}} - z_{\text{old}}|/(|z_{\text{old}}| + \epsilon) < 10^{-5}$, we say that the objective has *stalled* for that iteration. For cut management, we track how often each cut is satisfied at equality; for this, we permit a tolerance of 10^{-3}.

Terminating conditions We terminate the cutting plane algorithm (whether it is `SPARSE`, `DENSE`, or `HYBRID`) when one of the following conditions is met: (1) the time limit of 1 hour is reached, (2) no more cuts are added, (3) the objective did not sufficiently improve over the last 100 iterations.

6.2 Instances

We test three families of nonconvex quadratic problems, including several large instances with $n \geq 200$, i.e., with over 40,000 variables in the matrix space. In our representation of the problem, all variables (x, X) are present even if, for example, there is sparsity in (QCQP); solving the SDP for these problems becomes extremely memory intensive, requiring 35 GB or more for the larger instances, whereas the memory needed for the cutting plane algorithm remains relatively manageable. In addition to memory, the SDP also can take a significant amount of time; starting from $n = 200$, most SDP solution times are at least 1200 seconds, with some taking more than an hour. This suggests an advantage to the cutting plane approach in resource-constrained environments.

BoxQP We consider the BoxQP family of QPs [15, 19, 53]. These are problems with nonconvex quadratic objective, and with box constraints $x \in [0, 1]^n$, that is, problems of the form

$$\begin{aligned} \min_x \ & x^\top Q x + c^\top x \\ & x \in [0, 1]^n. \end{aligned}$$

The experiments of Section 4 considered three of these instances. For BoxQP instances, it is well known that the SDP approach provides strong dual bounds to the nonconvex problem; we refer the reader to [19] for a semidefinite programming approach to this class of instances and [21] for a completely positive approach to QPs. Recent papers [11, 58] have considered solving BoxQP with integer linear programming. We opted for not comparing with these approaches since, as we mentioned at the beginning, our goal is to mimic the effect of the semidefinite inequality (1). These alternative integer linear programming approaches cut through the SDP cone and derive valid inequalities for $X = xx^\top$ directly. This makes our approach rather complementary.

For this family, we consider all instances available at `https://github.com/sburer/BoxQP_instances`, and we additionally generate our own larger instances, for $n = 200$ and $n = 250$, using the same instance-generation code available in the link. In total, we conducted our experiments on 111 BoxQP instances, with $n \in [20, 250]$ and density $d \in [0.2, 1]$.[4]

Biq The Biq instances are similar to the BoxQP instances: they only have a nonconvex quadratic objective function, although in this case the quadratic

[4]We define the *density* of a QCQP as the proportion of nonzero entries in $Q_0 = Q$.

is homogeneous.[5] The other difference is that the variables are restricted to be binary. In order to make these instances suitable for our setting, we reformulate each constraint $x_i \in \{0, 1\}$ as

$$x_i(1 - x_i) = 0.$$

Therefore, the instances have the form

$$\begin{aligned} \min_x \ & x^\intercal Q x \\ & x_i(1 - x_i) = 0, \quad i \in [n] \\ & x \in [0, 1]^n, \end{aligned}$$

which is a QCQP. The BIQ instances we consider are all the available instances on the Biq Mac library [57] with n up to 250. We do not consider the larger instances in the library due to memory limitations. We also exclude 14 instances for which none of the methods close any gap (e.g., if the initial solution is PSD).[6] In addition, we remove one instance (`bqp50-9`) in which the objective value of the initial LP is within 10^{-6} of the SDP optimal value. This leaves us with 135 instances from this family, with $n \in [20, 250]$ and $d \in [0.1, 1]$.

MAXCUT We take the MAXCUT instances available in the Biq Mac library. It is well known that a MAXCUT instance over a graph $G = (V, E)$ with weights w_{ij}, $\{i, j\} \in E$ can be formulated as

$$\begin{aligned} \min_x \ & \frac{1}{2} x^\intercal Q x \\ & x_i^2 = 1, \quad i \in [n] \\ & x \in [-1, 1]^n, \end{aligned}$$

where $Q_{ij} = w_{ij}$ for all $i \neq j$. These are also nonconvex QCQPs.

For this class of instances, it is known that the SDP relaxation provides a strong provable approximation of the problem [30]. We test all but 27 of the available instances in the Biq Mac library, pruning any with $n > 250$; this leaves us with 151 instances, where $n \in [60, 225]$ and $d \in [0.1, 0.99]$.

6.3 Computational results

We use two comparison metrics. First, we compare the gap closed by all three methods, to assess relaxation *strength*. Second, we report the solution time of the last LP that is solved, to evaluate how *lightweight* are the linear relaxations we are obtaining. Our most extensive analysis is for the BOXQP instances in Section 6.3.1; we curtail our discussion of the remaining families as the results lead to conceptually similar conclusions.

[5] A *homogeneous* polynomial has the same degree for all nonzero terms.

[6] These are `bqp50-1` through `bqp50-8`, `gka1a`, `gka2a`, `gka3a`, `gka6c`, `gka7c`, and `gka8a`.

Table 4: Results on 111 BoxQP instances for `SPARSE`, `DENSE`, and `HYBRID`. Results are averages over instances grouped by size, under a time limit of 1 hour.

		Gap closed (%)			Last LP time (s)		
Instance group	#	`SPARSE`	`DENSE`	`HYBRID`	`SPARSE`	`DENSE`	`HYBRID`
$n \in [20, 30]$	18	98.50	100.00	100.00	0.10	2.93	2.93
$n \in [40, 50]$	33	98.83	99.90	99.89	0.64	10.73	7.34
$n \in [60, 80]$	21	98.45	96.24	98.17	6.49	28.27	11.69
$n \in [90, 125]$	27	94.62	90.68	95.48	48.09	106.54	49.08
$n \in [200, 250]$	12	75.16	84.70	83.92	520.24	764.30	506.98

6.3.1 BoxQP

In Table 4, we summarize the performance of the `SPARSE`, `DENSE`, and `HYBRID` algorithms. We group instances by size, and the second column states the number of instances in each group. The next three columns give the percent gap closed, and the final three columns provide the time to solve the last LP.

The gap closed by the three methods is similar for smaller instances, and for $n \leq 125$ the relaxations obtained by all the methods are quite strong. The limit HybridSwitchingTime is never reached for the instances with $n < 40$, i.e., only dense cuts are used, and `DENSE` and `HYBRID` have identical performance, while `SPARSE` closes less gap, though with a corresponding final solving time for the LP of 0.1 seconds, compared to nearly 3 seconds for the other two methods.

For the instances with $n \in [40, 50]$, we see that `HYBRID` closes a little less of the gap than `DENSE`, but with a 30% improvement in the LP solution time. Upon a closer observation of this group, we observed that `HYBRID` encounters the limit HybridSwitchingTime for 16 of the 33 instances, but it only adds sparse cuts for 4 of the instances. In the other 12 cases, `HYBRID` (via Algorithm 1) finds no sparse cuts to add, terminating with fewer iterations, and a slightly lower gap closed, than `DENSE`, but with a correspondingly lighter LP.

For the next two groups, $n \in [60, 80] \cup [90, 125]$, we see that sparse cuts, even in isolation, outperform dense ones, whereas the gap closed by `HYBRID` is comparable to that closed by `SPARSE`. As n grows, the importance of including dense cuts increases, and eventually `HYBRID` dominates both `SPARSE` and `DENSE`. Curiously, this holds even when accounting for the last LP time, showing that it is possible to have the best of both worlds, i.e., both strength and speed, especially by combining dense and sparse inequalities. This is more emphatic in the $n \in [90, 125]$ set: `HYBRID` closes nearly 5% more of the gap than `DENSE` with an LP that solves over twice as quickly.

In the final and largest set of instances, in terms of gap closed, sparse cuts alone now lag severely behind either of the procedures involving dense cuts, and the LP time for `HYBRID` is the best of the three algorithms. We do observe a degradation in the gap closed by `HYBRID` relative to `DENSE`, which we investigate

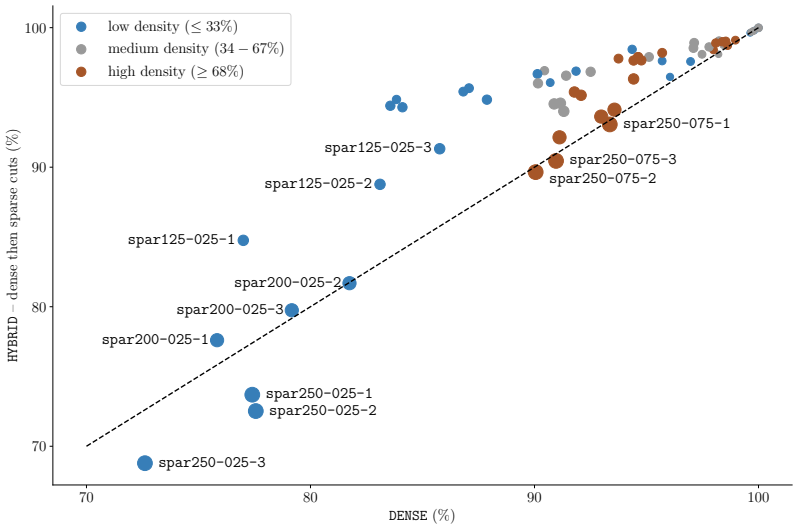

Figure 1: Gap closed after 1 hour by `HYBRID` and `DENSE` algorithms on BoxQP instances, where the diagonal dashed line is a reference for equal performance. Most instances lie above this line, indicating that incorporating sparse cuts can close a substantially larger gap than a procedure with dense cuts alone. The size of the markers is proportional to n, and both methods tend to close less gap on larger instances within the time limit. Similarly, both methods perform worse on sparser instances, though `HYBRID` tends to have stronger performance relative to `DENSE` when an instance is sparser.

further below.

In Figure 1, we focus on `DENSE` and `HYBRID`, and show a more detailed comparison of the gap closed by each method. The dashed diagonal line indicates parity between the two methods; instances above the line are ones in which `HYBRID` performs better, and the opposite for instances below the line. Additionally, in this figure, we classify instances according to their density.

The plot reinforces the message of Table 4, that a method combining sparse and dense cuts has significantly stronger performance than one based on dense cuts alone. We can further see that the improvement of `HYBRID` relative to `DENSE` gets more pronounced on sparser instances, except for the ones in the lower left corner, which we elaborate on next. There are seven instances in which at least one of the two methods closes less than 82% of the gap: `spar125-025-1`, `spar200-025-*`, and `spar250-025-*`. We focus on the three lying below the diagonal line, `spar250-025-1` through `spar250-025-3`, for which `HYBRID` closes around 4% less gap than `DENSE`. Unlike the situation for the rest of the family, `HYBRID` actually performs *fewer* iterations on these three instances. In these three cases, the reason is that an order of magnitude more sparse than dense cuts are produced, and ultimately the LP solution time for `HYBRID` tends to be

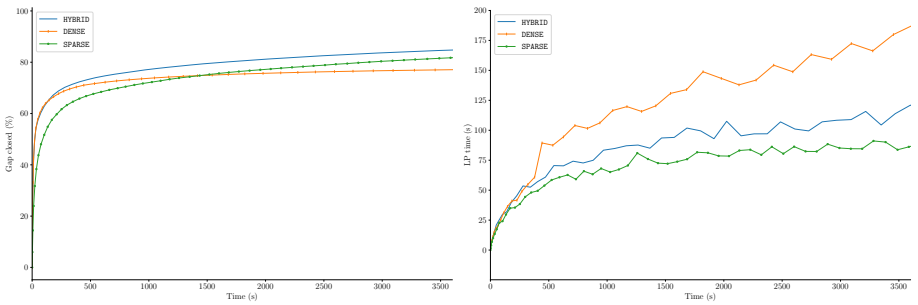

Figure 2: Progress of gap closed (left) and LP time (right) for BoxQP instance `spar125-025-1` for the three methods during the first hour.

slower than for `DENSE`, again in stark contrast to the rest of the family.

Our takeaway is that, although our heuristic parameter choices can be better tuned for larger BoxQP instances, overall `HYBRID` appears to be the better method on average, while `SPARSE` might be preferred for instances with $n \leq 80$, especially if LP solution time is weighed more heavily than gap closed, which can well be the case for a mixed-integer QCQP context.

We conclude this section with two plots detailing the progress of the three algorithms over the course of the hour. While this will be for only one instance, `spar125-025-1`, it exemplifies the relative behavior of the approaches that we observed during most of the experiments.

In Figure 2, the left panel shows the progress in gap closed throughout the hour, while the right panel shows the corresponding sequence of LP solution times. We see that `DENSE` suffers from tailing off, with a nearly flat curve after the first ten minutes of computation. Meanwhile, `SPARSE` eventually surpasses the gap closed of `DENSE`, though its rate of change is initially much slower. On the other hand, `HYBRID`, by design, captures the initial steep improvement achieved by dense cuts, after which it makes steady progress with the more efficient, but less aggressive, sparse cuts. Thus, if a shorter solution time is enforced, `HYBRID` would continue to dominate `DENSE`, whereas the same cannot be said for `SPARSE` compared to `DENSE`.

The right panel underscores the effect of the density of the cuts. The solution times of `DENSE` are rapidly increasing, while for `SPARSE`, the LPs never take more than 100 seconds to solve. `HYBRID` shows the expected middle-ground: the rapid initial gap closed comes at the expense of a nonnegligible initial increase of the LP solution times. However, switching to sparse cuts keeps the LP under control, while still being able to improve the gap closed. In fact, to test this further, we performed an extended test for this instance with a time limit of 1 day instead of 1 hour, at the end of which the LP solution times for `SPARSE`, `HYBRID`, and `DENSE` were 129.7, 127.1, and 375.8 seconds, corresponding to a gap closed of 96.4, 96.3, and 88.1 percent, respectively. Hence, `DENSE` creates a sequence of increasingly heavy LPs with stagnating bound improvement, whereas `SPARSE`

Table 5: Results on 135 Biq instances for SPARSE, DENSE, and HYBRID. Results are averages over instances grouped by size, under a time limit of 1 hour.

		Gap closed (%)			Last LP time (s)		
Instance group	#	SPARSE	DENSE	HYBRID	SPARSE	DENSE	HYBRID
$n \in [20, 90]$	18	98.70	99.47	99.81	1.33	15.14	7.26
$n = 100$	31	94.92	82.53	95.00	31.33	91.35	34.82
$n \in [120, 150]$	41	90.18	89.35	92.61	125.87	262.56	132.43
$n \in [200, 250]$	45	54.72	65.72	64.06	479.61	830.75	519.96

and HYBRID do not substantially slow down relative to the statistics after one hour and show sustained progress in the objective value.

6.3.2 Biq

In Table 5, we summarize the performance of SPARSE, DENSE, and HYBRID for the 135 Biq instances. The structure of the table is the same as Table 4.

We have similar conclusions as for the BoxQP instances. For the smallest set of Biq instances, all three methods yield strong relaxations, with SPARSE closing about 1% less gap than HYBRID, but having an 80% reduction in solving time, while HYBRID already comes with a 50% faster LP than DENSE, on average. In the next group, with $n = 100$, we see that HYBRID and SPARSE dominate DENSE, both closing around 12.5% more of the gap on average while requiring only 34–38% of the time to solve the final LP compared to DENSE. This is a marked difference in gap closed, showing a substantially larger benefit to using sparse cuts in moderate-sized Biq instances, compared to the more modest advantages we observed in the BoxQP family. Just as for the BoxQP instances, the relative performance, in terms of gap closed, of DENSE starts to improve again for larger n, while accordingly the quality of SPARSE deteriorates, and HYBRID retains its status as a happy compromise of the two.

6.3.3 MaxCut

Analogously to the other families, Table 6 summarizes our results for the 151 MaxCut instances. The high-level trends remain the same as with the other two families, but there are a few notable differences for the largest group of instances. The gap closed is considerably less than for the other two families, with all methods closing only 5–6% of the gap. The reason is that the LP relaxation quickly becomes very heavy, and fewer than 15 iterations are able to be performed within the time limit.

In Figure 3, we take as a case study the MaxCut instance pm1s_100.1, and we show the detailed evolution of all three methods over the course of the one hour time limit, with respect to gap closed (left panel) and LP solution time (right panel). For the time comparison in the right panel, the relative ordering

Table 6: Results on 151 MaxCut instances for `SPARSE`, `DENSE`, and `HYBRID`. Results are averages over instances grouped by size, under a time limit of 1 hour.

		Gap closed (%)			Last LP time (s)		
Instance group	#	`SPARSE`	`DENSE`	`HYBRID`	`SPARSE`	`DENSE`	`HYBRID`
$n = 60$	10	97.45	98.73	98.86	3.20	13.69	10.98
$n = 80$	30	93.61	93.43	96.65	18.59	47.48	24.07
$n = 100$	99	79.36	77.44	82.66	60.09	107.76	86.74
$n \in [150, 225]$	12	6.00	5.13	5.85	717.56	775.20	704.32

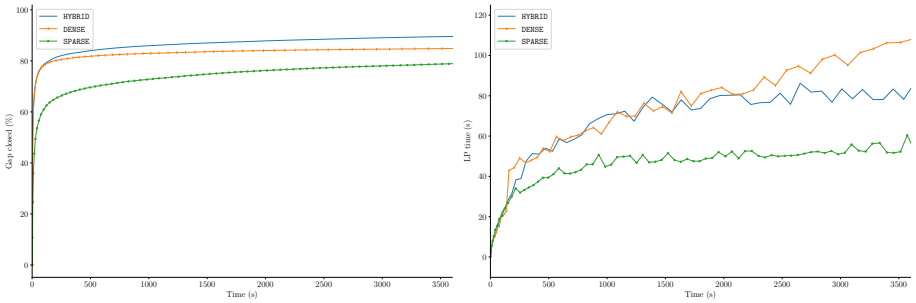

Figure 3: Progress of gap closed (left) and LP time (right) for MaxCut instance `pm1s_100.1` for the three methods over one hour.

of the algorithms is the same as for the BoxQP instance plotted in Figure 2, with `SPARSE` requiring the least amount of time throughout, followed by `HYBRID`, and then `DENSE`. As we did for `spar125-025-1`, here too we used an extended time limit of 1 day to observe the longer-run behavior: at the end of the day (not plotted), the LP solution was computed by for `SPARSE`, `DENSE`, and `HYBRID` in 58.5, 193.0, and 73.5 seconds, respectively.

However, the story for gap closed is different: while `HYBRID` continues to do well, picking up on the early momentum afforded by dense cuts and then continually increasing its relative advantage with respect to `DENSE`, the `SPARSE` algorithm seems to have much slower convergence than for the BoxQP setting. Indeed, after one day of computation time, `SPARSE` still trails `DENSE` in gap closed, with 92.6% of the gap closed by `SPARSE`, compared to 92.9% gap closed by `DENSE`. In the meantime, `HYBRID` improves from the one-hour mark gap closed of 89.6% to a final gap closed of 97.0%.

7 Concluding remarks

The availability of strong, sparse outer approximations of an SDP constraint is practically important, such as when solving QCQPs with LP-based spatial branch and bound. In this case, the need for such relaxations comes from the large number of LPs that need to be solved within that process and the target of quickly obtaining good bounds. A related, but distinct, motivation comes from convergence: without sparsity, a cutting plane approach may stall, either due to slower iterations or due to numerical issues.

This paper introduces a viable method to apply sparse eigenvector-based cutting planes to capture the strength of an SDP constraint in a linear programming context. While these families of cuts have been considered before, their empirical performance has been limited on one hand by the high density of eigenvector-based inequalities, causing linear programs that are too slow for practical purposes, and on the other hand by the relative weakness of sparsified versions of the dense cuts.

We first empirically justify, through our enumeration experiments, that sparse cuts tend to indeed be weak in isolation, while we observe that dense cuts are surprisingly strong when compared to all available sparse cuts. With this in mind, and via our novel connection between k-sparse-eigencut generation and the classical Sparse PCA problem, we develop `HYBRID`, an efficient separation routine for producing a multitude of strong and sparse eigenvector-based cuts. This method combines the initial strength of dense cuts and the steady progress that can be obtained when generating multiple sparse cuts using Algorithm 1. Our computational results indicate that we can successfully produce strong, lightweight linear relaxations; for QCQP instances with at most 100 variables and across three different benchmark families, within an hour of computation, we tend to close more than 90% of the gap between the initial LP relaxation and the SDP relaxation, representing a significant improvement with respect to either dense or sparse cuts alone. Further, our experiments indicate that a driving force for this relative improvement is that LPs with sparse cuts solve much faster: for example, in the last iteration, the LP with dense cuts can be two to three times slower, on average, than one with sparse cuts. Moreover, even with a much longer time limit, dense cuts often never attain the same gap closed as when sparse cuts are employed.

Future work involves further improving scalability. Right now, our handling of the variable matrix is straightforward, and all entries in X are created and stored. Avoiding this is a crucial step into scaling our approach into even larger instances and to be able to properly embed it in spatial branch and bound.

Acknowledgments

This research was enabled in part by support provided by Calcul Québec (calculquebec.ca) and Compute Canada (computecanda.ca). GM would like to thank the Institute for Data Valorization (IVADO) for their support through the Postdoctoral Fellow program, and to the Government of Chile for their financial support through the FONDECYT grant number 11190515.

References

[1] E. Amaldi, S. Coniglio, and S. Gualandi, *Coordinated cutting plane generation via multi-objective separation*, Math. Program., 143 (2014), pp. 87–110.

[2] K. M. Anstreicher, *Semidefinite programming versus the reformulation-linearization technique for nonconvex quadratically constrained quadratic programming*, J. Global. Optim., 43 (2009), pp. 471–484.

[3] M. Asteris, D. S. Papailiopoulos, and G. N. Karystinos, *Sparse principal component of a rank-deficient matrix*, in 2011 IEEE International Symposium on Information Theory Proceedings, IEEE, 2011, pp. 673–677.

[4] E. Balas, *Intersection cuts—a new type of cutting planes for integer programming*, Oper. Res., 19 (1971), pp. 19–39.

[5] R. Baltean-Lugojan, P. Bonami, R. Misener, and A. Tramontani, *Scoring positive semidefinite cutting planes for quadratic optimization via trained neural networks*. Working paper., 2019, `http://www.optimization-online.org/DB_HTML/2018/11/6943.html`.

[6] D. Bienstock, C. Chen, and G. Muñoz, *Outer-product-free sets for polynomial optimization and oracle-based cuts*, Math. Program., (2020), pp. 1–44.

[7] D. Bienstock, C. Chen, and G. Muñoz, *Intersection cuts for polynomial optimization*, in Integer Programming and Combinatorial Optimization, Springer International Publishing, 2019, pp. 72–87.

[8] R. E. Bixby, *Solving real-world linear programs: A decade and more of progress*, Oper. Res., 50 (2002), pp. 3–15.

[9] G. Blekherman, S. S. Dey, M. Molinaro, and S. Sun, *Sparse PSD approximation of the PSD cone*, arXiv preprint arXiv:2002.02988, (2020). Working paper.

[10] G. Blekherman, S. S. Dey, K. Shu, and S. Sun, *Hyperbolic relaxation of k-locally positive semidefinite matrices*, arXiv preprint arXiv:2012.04031, (2020), `https://arxiv.org/abs/2012.04031`. Working paper.

[11] P. Bonami, O. Günlük, and J. Linderoth, *Globally solving nonconvex quadratic programming problems with box constraints via integer programming methods*, Math. Program. Comput., 10 (2018), pp. 333–382.

[12] P. Bonami, J. Linderoth, and A. Lodi, *Disjunctive cuts for mixed integer nonlinear programming problems*, Progress in Combinatorial Optimization, (2011), pp. 521–541.

[13] P. Bonami, A. Lodi, J. Schweiger, and A. Tramontani, *Solving quadratic programming by cutting planes*, SIAM J. Optim., 29 (2019), pp. 1076–1105.

[14] E. Boros, Y. Crama, and P. L. Hammer, *Chvátal cuts and odd cycle inequalities in quadratic 0–1 optimization*, SIAM J. Discrete Math., 5 (1992), pp. 163–177.

[15] S. Burer, *Optimizing a polyhedral-semidefinite relaxation of completely positive programs*, Math. Program. Comput., 2 (2010), pp. 1–19.

[16] S. Burer, *A gentle, geometric introduction to copositive optimization*, Math. Program., 151 (2015), pp. 89–116.

[17] S. Burer and A. N. Letchford, *On nonconvex quadratic programming with box constraints*, SIAM J. Optim., 20 (2009), pp. 1073–1089.

[18] S. Burer and R. D. Monteiro, *A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization*, Math. Program., 95 (2003), pp. 329–357.

[19] S. Burer and D. Vandenbussche, *Globally solving box-constrained nonconvex quadratic programs with semidefinite-based finite branch-and-bound*, Comput. Optim. Appl., 43 (2009), pp. 181–195.

[20] S. Burer and Y. Ye, *Exact semidefinite formulations for a class of (random and non-random) nonconvex quadratic programs*, Math. Program., (2018), pp. 1–17.

[21] J. Chen and S. Burer, *Globally solving nonconvex quadratic programming problems via completely positive programming*, Math. Program. Comput., 4 (2012), pp. 33–52.

[22] A. Chmiela, G. Muñoz, and F. Serrano, *On the implementation and strengthening of intersection cuts for QCQPs*, Tech. Report 20–29, ZIB, Takustr. 7, 14195 Berlin, 2020.

[23] S. S. Dey, B. Kocuk, and A. Santana, *Convexifications of rank-one-based substructures in QCQPs and applications to the pooling problem*, J. Global. Optim., (2019), pp. 1–46.

[24] S. S. Dey, R. Mazumder, and G. Wang, *A convex integer programming approach for optimal sparse PCA*, arXiv preprint arXiv:1810.09062, (2018), `https://arxiv.org/abs/1810.09062`. Working paper.

[25] S. S. Dey and M. Molinaro, *Theoretical challenges towards cutting-plane selection*, Math. Program., 170 (2018), pp. 237–266.

[26] S. S. Dey, M. Molinaro, and Q. Wang, *Approximating polyhedra with sparse inequalities*, Math. Program., 154 (2015), pp. 329–352.

[27] S. S. Dey, A. Santana, and Y. Wang, *New SOCP relaxation and branching rule for bipartite bilinear programs*, Optim. Eng., 20 (2019), pp. 307–336.

[28] M. Fischetti and M. Monaci, *A branch-and-cut algorithm for mixed-integer bilinear programming*, Eur. J. Oper. Res, (2019).

[29] M. Fukuda, M. Kojima, K. Murota, and K. Nakata, *Exploiting sparsity in semidefinite programming via matrix completion I: General framework*, SIAM J. Optimiz., 11 (2001), pp. 647–674.

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

[31] G. Guennebaud, B. Jacob, et al., *Eigen v3*. http://eigen.tuxfamily.org, 2010.

[32] Gurobi Optimization, LLC, *Gurobi optimizer reference manual*, 2020, `http://www.gurobi.com`.

[33] R. Horst and H. Tuy, *Global Optimization*, Springer Berlin Heidelberg, 1996.

[34] M. Journée, Y. Nesterov, P. Richtárik, and R. Sepulchre, *Generalized power method for sparse principal component analysis.*, J. Mach. Learn. Res., 11 (2010).

[35] J. B. Lasserre, *Convergent SDP-relaxations in polynomial optimization with sparsity*, SIAM J. Optimiz., 17 (2006), pp. 822–843.

[36] M. Laurent, *Sum of squares, moment matrices and optimization over polynomials*, in Emerging Applications of Algebraic Geometry, vol. 149 of IMA Vol. Math. Appl., Springer, New York, 2009, pp. 157–270.

[37] J. Lavaei and S. H. Low, *Zero duality gap in optimal power flow problem*, IEEE T. Power Syst., 27 (2011), pp. 92–107.

[38] L. Mackey, *Deflation methods for sparse PCA*, in Advances in Neural Information Processing Systems, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, eds., vol. 21, Curran Associates, Inc., 2009, pp. 1017–1024.

[39] M. Magdon-Ismail, *NP-hardness and inapproximability of sparse PCA*, Inform. Process. Lett., 126 (2017), pp. 35–38.

[40] G. P. McCormick, *Computability of global solutions to factorable nonconvex programs: Part I – Convex underestimating problems*, Math. Program., 10 (1976), pp. 147–175.

[41] MOSEK ApS, *The MOSEK optimization toolbox for C++ manual. Version 9.2.21*, 2019, `https://docs.mosek.com/9.2/cxxfusion/index.html`.

[42] B. Müller, F. Serrano, and A. M. Gleixner, *Using two-dimensional projections for stronger separation and propagation of bilinear terms*, SIAM J. Optim., 30 (2020), pp. 1339–1365.

[43] G. Muñoz and F. Serrano, *Maximal quadratic-free sets*, in International Conference on Integer Programming and Combinatorial Optimization, Springer, 2020, pp. 307–321.

[44] M. Padberg, *The boolean quadric polytope: some characteristics, facets and relatives*, Math. Program., 45 (1989), pp. 139–172.

[45] A. Qualizza, P. Belotti, and F. Margot, *Linear programming relaxations of quadratically constrained quadratic programs*, in Mixed Integer Nonlinear Programming, vol. 154 of IMA Vol. Math. Appl., Springer, New York, 2012, pp. 407–426.

[46] H. Rahman and A. Mahajan, *Facets of a mixed-integer bilinear covering set with bounds on variables*, J. Global. Optim., 74 (2019), pp. 417–442.

[47] J. K. Reid, *A sparsity-exploiting variant of the Bartels-Golub decomposition for linear programming bases*, Math. Program., 24 (1982), pp. 55–69.

[48] A. Santana and S. S. Dey, *The convex hull of a quadratic constraint over a polytope*, SIAM J. Optim., 30 (2020), pp. 2983–2997.

[49] H. D. Sherali and B. M. P. Fraticelli, *Enhancing RLT relaxations via a new class of semidefinite cuts*, J. Global. Optim., 22 (2002), pp. 233–261.

[50] N. Z. Shor, *Quadratic optimization problems*, Sov. J. Comput. Syst. S.+, 25 (1987), pp. 1–11.

[51] M. Tawarmalani, J.-P. P. Richard, and K. Chung, *Strong valid inequalities for orthogonal disjunctions and bilinear covering sets*, Math. Program., 124 (2010), pp. 481–512.

[52] H. Tuy, *Concave programming with linear constraints*, in Doklady Akademii Nauk, vol. 159, Russian Academy of Sciences, 1964, pp. 32–35.

[53] D. Vandenbussche and G. Nemhauser, *A branch-and-cut algorithm for nonconvex quadratic programs with box constraints*, Math. Program., 102 (2005), pp. 559–575.

[54] M. J. Wainwright and M. I. Jordan, *Treewidth-based conditions for exactness of the Sherali-Adams and Lasserre relaxations*, Tech. Report 671, University of California, September 2004.

[55] M. Walter, *Sparsity of lift-and-project cutting planes*, in Operations Research Proceedings 2012, Springer, 2014, pp. 9–14.

[56] A. L. Wang and F. Kılınç-Karzan, *On the tightness of SDP relaxations of QCQPs*, arXiv preprint arXiv:1911.09195, (2019), https://arxiv.org/abs/1911.09195. Working paper.

[57] A. Wiegele, *Biq Mac Library*, 2007, http://biqmac.uni-klu.ac.at/biqmaclib.html. Accessed August 14, 2020.

[58] W. Xia, J. C. Vera, and L. F. Zuluaga, *Globally solving nonconvex quadratic programs via linear integer programming techniques*, INFORMS J. Comput., (2019).

[59] Y. Yajima and T. Fujie, *A polyhedral approach for nonconvex quadratic programming problems with box constraints*, J. Global. Optim., 13 (1998), pp. 151–170.

[60] Y. Ye, *Approximating quadratic programming with bound constraints*, Math. Program., 84 (1997), pp. 219–226.

[61] X.-T. Yuan and T. Zhang, *Truncated power method for sparse eigenvalue problems*, J. Mach. Learn. Res., 14 (2013), pp. 899–925.

[62] A. Yurtsever, J. A. Tropp, O. Fercoq, M. Udell, and V. Cevher, *Scalable semidefinite programming*, arXiv preprint arXiv:1912.02949, (2019), https://arxiv.org/abs/1912.02949. Working paper.