Noname manuscript No.
(will be inserted by the editor)

Certifiably Optimal Sparse Principal Component Analysis

Lauren Berk · Dimitris Bertsimas

Received: date / Accepted: date

Abstract This paper addresses the sparse principal component analysis (SPCA) problem for covariance matrices in dimension n aiming to find solutions with sparsity k using mixed integer optimization. We propose a tailored branch-and-bound algorithm, Optimal-SPCA, that enables us to solve SPCA to certifiable optimality in seconds for $n = 100$s, $k = 10$s. This same algorithm can be applied to problems with $n = 10{,}000s$ or higher to find high-quality feasible solutions in seconds while taking several hours to prove optimality. We apply our methods to a number of real data sets to demonstrate that our approach scales to the same problem sizes attempted by other methods, while providing superior solutions compared to those methods, explaining a higher portion of variance and permitting complete control over the desired sparsity.
The software that was reviewed as part of this submission has been given the D.O.I. (Digital Object Identifier) **10.5281/zenodo.2027898**.

Keywords: sparse principal component analysis, principal component analysis, mixed integer optimization, sparse eigenvalues

Mathematics Subject Classification (2010) 62H25 Multivariate analysis–Factor analysis and principal components; correspondence analysis · 65F15 Numerical linear algebra–Eigenvalues, eigenvectors · 65K05 Numerical analysis–Mathematical programming methods · 90C06 Mathematical Programming–Large-scale problems · 90C26 Mathematical Programming–Nonconvex programming, global optimization · 90C27 Mathematical Programming–Combinatorial optimization

Lauren Berk, Dimitris Bertsimas
Operations Research Center
Massachusetts Institute of Technology
Cambridge, MA 02139, USA
E-mail: lberk@mit.edu, dbertsim@mit.edu
ORCID: 0000-0002-6617-4447, 0000-0002-1985-1003

1 Introduction

Principal component analysis (PCA) is a statistical technique used to understand the orthogonal directions that account for the majority of the variability in a data set [37]. Given an $m \times n$ data matrix $\mathbf{A}$, with m data points each with n dimensions, we can compute a sample covariance matrix $\mathbf{Q} = \mathbf{A}'\mathbf{A}/(m-1)$. By construction, $\mathbf{Q}$ is symmetric and positive semi-definite. With this sample covariance matrix, or in fact any positive semi-definite matrix $\mathbf{Q}$, the first goal of PCA is to find the first principal component:

$$\max_{\mathbf{x}} \quad \mathbf{x}'\mathbf{Q}\mathbf{x} \quad \text{s.t. } ||\mathbf{x}||_2 = 1. \qquad \text{PCA (1)}$$

The solution to Problem (1) is simply the first (eigenvalue-maximizing) eigenvector of $\mathbf{Q}$, which can be computed in a number of ways, including the simple power method [66].

Subsequent principal components are orthogonal to existing components, and are chosen to maximize the remaining variance explained after subtracting the variance along the prior components. For example, given a first component $\mathbf{x}_1$, we can create an adjusted covariance matrix by projecting:

$$\mathbf{Q}_2 = (I - \mathbf{x}_1\mathbf{x}_1')\mathbf{Q}(I - \mathbf{x}_1\mathbf{x}_1'). \qquad (2)$$

The second principal component is found by maximizing:

$$\max_{\mathbf{x}} \quad \mathbf{x}'\mathbf{Q}_2\mathbf{x} \quad \text{s.t. } ||\mathbf{x}||_2 = 1. \qquad (3)$$

This second component $\mathbf{x}_2$ will explain the optimal amount of remaining variance once $\mathbf{x}_1$ is accounted for. Subsequent components are derived analogously.

PCA and closely-related techniques ("factor analysis" [15], canonical correlation analysis [30, 68]) are widely used in multivariate data analysis to identify a minority of dimensions ("components" or "factors") that can summarize the data. When these factors are interpretable, they can lead to a deeper understanding of the underlying phenomena. Since PCA reduces dimensionality, it can be used for data and image compression [23], as well as identifying separate components in an image or sequence of images [13]. Reducing dimensionality can also be helpful as a preprocessing step in machine learning models; resulting models are better able to avoid overfitting and multicollinearity [22]. Applications are numerous, including medicine [21], horticulture [33], and computer network science [42] among many others.

A disadvantage of PCA is that the identified components typically have non-zero loadings on every variable. While PCA reduces dimensionality in the sense that the first few principal components span a smaller space than the original data, it does not reduce the number of relevant variables. In order to reproduce results or add additional data points to the model, researchers still need to collect all the original variables. For example, PCA is often used in genetics research for reducing the dimensionality of data sets that can contain 10,000 or more variables [58,31,44]. There is a real concern in this application that many of the loadings on the principal components reflect noise in the data, and there is a desire for PCA to help with variable selection in addition to dimensionality reduction. Traditional principal components are also difficult to interpret. By 1967, in a pivotal paper on PCA case studies, Jeffers [35] was already discussing the need for methods that reduce the number of variables involved in components to improve interpretability.

These considerations inspired work to sparsify principal components without compromising their explanatory power by solving the sparse principal component analysis (SPCA) problem. Specifically, a constraint on the ℓ_0 "norm" of $\mathbf{x}$, $\|\mathbf{x}\|_0$ (that is, $|\{i|x_i \neq 0\}|$) is added to Problem

(1), limiting the number of non-zero loadings of $\mathbf{x}$ to some integer k. The SPCA problem is then,

$$\begin{aligned}\max_{\mathbf{x},\mathbf{y}} \quad & \mathbf{x}'\mathbf{Q}\mathbf{x} \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i^2 = 1 \\ & \|\mathbf{x}\|_0 \le k.\end{aligned} \qquad \text{SPCA (4)}$$

Subsequent components in this case are more complicated. It is no longer possible to find components that are mutually orthogonal that also maximize variance explained. The main focus of this paper is solving the first SPCA component problem in Problem (4), but we explore the issue of multiple components further in Section 2.2.

Research into SPCA has been accelerating recently, due to the growing scale of research problems. Generally, increasing the number of samples in a data set does not increase the computational difficulty of the problem, since the size of the covariance matrix $\mathbf{Q}$ is only determined by the number of variables in the data. However, modern applications such as genomic analysis and physics involve high-dimensional data sets which drastically increase the size of $\mathbf{Q}$ and the size of the solution space, pushing the limits of computational feasibility.

Despite decades of research, no existing method can solve Problem (4) to provable optimality in practical time. In this paper, we develop an algorithm, based on principles from mixed integer optimization (MIO), that solves this problem to optimality. In addition to advancing the state of the art for SPCA, the present work also contributes to the growing literature [63] reconsidering difficult problems in statistics and machine learning through the lens of modern optimization techniques. Topics studied include belief propagation [69], support vector machines [71,52], classification and regression trees [8], graph clustering [17], parameter setting [62], subset selection [53], factor analysis [7], k-means clustering [34] and many others. Collectively, these new methods are aspiring to change machine learning from an art of heuristics into a science of optimality.

A set of problem definitions will assist our discussion of the literature. Broadly, formulations of the SPCA problem fall into these categories (with $\gamma < 0$, k a positive integer, and $\mathbf{Q}$ an $n \times n$ positive semi-definite matrix) [50,60]:

- ℓ_1-constrained: $\max \mathbf{x}'\mathbf{Q}\mathbf{x}$ s.t. $\|\mathbf{x}\|_2 = 1, \|\mathbf{x}\|_1 \le k$;
- ℓ_0-constrained: $\max \mathbf{x}'\mathbf{Q}\mathbf{x}$ s.t. $\|\mathbf{x}\|_2 = 1, \|\mathbf{x}\|_0 \le k$;
- ℓ_1-penalized: $\max \mathbf{x}'\mathbf{Q}\mathbf{x} + \gamma\|\mathbf{x}\|_1$ s.t. $\|\mathbf{x}\|_2 \le 1$;
- ℓ_0-penalized: $\max \mathbf{x}'\mathbf{Q}\mathbf{x} + \gamma\|\mathbf{x}\|_0$ s.t. $\|\mathbf{x}\|_2 \le 1$.

In particular, the ℓ_0-constrained formulation is exactly Problem (4). Additionally, there are convex relaxations of these problems, and some papers [60] consider alternative primary objectives to $\mathbf{x}'\mathbf{Q}\mathbf{x}$. Our algorithm is designed to address the ℓ_0-constrained problem.

1.1 Review of Literature

Formulations and relaxations After Jeffers [35] identified the importance of interpretable and sparse components, early work on sparse principal components focused on rotating the basis of the first few principal components within the subspace they generated, to reduce the ℓ_0 norm of each vector [59][36]. The varimax criterion, developed in 1957 by Kaiser [40], is the most commonly used approach to rotation in both PCA and factor analysis. Today, it is common practice to rotate principal components in the course of data analysis in many fields.

Another early thread of work on SPCA was inspired by the development of LASSO [64] for variable selection and shrinkage. LASSO can be used as a method for solving the sparse regression problem by adding an ℓ_1 penalty to the objective function, which forms the unconstrained, continuous, convex problem [28]. Jolliffe et al. [38] developed SCoTLASS (Simplified Component Technique - LASSO) from these ideas. SCoTLASS introduced the ℓ_1-penalized formulation of SPCA and proposed using gradient ascent techniques to solve it. Later, additional computation methods were developed for SCoTLASS, including [67].

Zou et al. [73] proposed a convex relaxation to the ℓ_1-penalized formulation, using the elastic net (incorporating a ℓ_2 penalty on $\mathbf{x}$ in addition to the ℓ_1 penalty). Leng and Wang [46] also built upon SCoTLASS by replacing the LASSO penalty with an “adaptive” penalty vector that varies by index. A different approach to sparse PCA components was taken by Bair et al. [3], who took into account the correlation of the variables with some outcome when selecting a support. This supervised approach solves a different problem than SPCA, but is motivated by the same objective.

The paper by d’Aspremont et al. [20] developed a convex relaxation of the ℓ_0-constrained formulation called Direct SPCA (DSPCA) using semi-definite optimization. While often providing solutions superior to earlier methods, DSPCA struggled to solve problems on the same scale as other methods, hitting a limit around $n = 100$. Since the sparsity constraint is relaxed in this formulation, solutions can have sparsity k' greater than the target sparsity k. In this case, the solutions are infeasible and super-optimal for the SPCA problem with the target sparsity k, and are feasible (but likely sub-optimal) for the SPCA problem with the target sparsity k'. In a follow-up paper, d’Aspremont et al. [19] developed a greedy approach to solve this formulation and a polynomial time algorithm for providing a certificate of optimality (proving optimality for the semi-definite relaxation, not the ℓ_0-constrained problem). Additionally, Amini and Wainwright [1] explored semi-definite relaxations of the SPCA problem in higher dimensions.

First order methods Recently, a large number of gradient ascent and local search methods have been proposed for solving SPCA problems.

Journée et al. [39] proposed a method called GPower for solving ℓ_0-penalized and ℓ_1-penalized problems that involved combining the power method (that is, identifying the leading eigenvector of a matrix by repeated iterations of $\mathbf{x}_{i+1} = \mathbf{Q}\mathbf{x}_i$ until convergence) for computing the first eigenvector of a matrix with thresholding and renormalizing steps. Hein and Bühler [29] derived an approach based on the inverse power method with similar thresholding. Yuan and Zhang [70] and Luss and Teboulle [50] proposed methods including one we will use in this paper, which reduces sparsity of the solution to a fixed k at each step instead of truncating values that fall below a fixed threshold. Ma et al. [51] expanded this work from individual vectors to sparse subspaces. More recently, Chan et al. [16] proposed a simple, closed form solution that can be applied to very large scale problems with bounded approximation error, and Beck and Vaisbourd [4] developed new, stronger optimality conditions and designed algorithms to find points that satisfy those conditions.

These methods are extremely fast, and are accompanied by results showing convergence to stationary points, but all are susceptible to becoming trapped at local optima. These results are usually compared, then, on their computational tractability, and the variance they explain in numerical experiments. None of these methods provide a certificate of optimality, but are very useful as warm starts for exact methods.

Preprocessing and parallelism Some recent papers on high-dimensional SPCA have incorporated preprocessing stages into their algorithms to improve speed and tractability by reducing the dimensionality of the input data. The key idea is removing variables from the problem (eliminating rows and columns from the $\mathbf{Q}$ matrix) if it can be proven that they cannot occur in an optimal sparse solution to the problem. Zhang and Ghaoui [72] showed that for the ℓ_1-penalized formulation with penalty coefficient γ, any variable can be removed if its variance falls below γ (that is, if $Q_{ii} < \gamma$). Lee et al. [45] developed a similar approach, removing variables whose variance falls below a calculated threshold. More specialized preprocessing exists for individual algorithms [56]. Unfortunately, no such guarantee exists for the ℓ_0-constrained problem in which we are interested. We will use ideas related to these methods, of eliminating possible indices by considering the variance of individual variables, when we develop our branch-and-bound approach.

Efforts to speed up SPCA methods also include work on parallel computing. Richtárik et al. [60] considered all four main formulations we have discussed, along with several more, and proposed solution methods using alternating maximization. A key contribution of Richtárik et al. was the incorporation of parallel computing methods for matrix-vector products. This work increased the maximum size of tractable problems for earlier methods developed by Richtárik et al.. While we will not be using parallelized code in our work, research into parallel branch-and-bound research has been active since the 1980s, and these ideas may be beneficial for future work [41,47].

Mixed integer optimization and SPCA Traditionally, computational problems have been deemed intractable if polynomial-time algorithms are unavailable or of high degree. The theory of computational complexity developed in the 1970s characterized problems as NP-hard to indicate that they are very unlikely (unless P=NP) to have polynomial time algorithms to solve them. NP-hardness became synonymous with intractability.

Since the early 1990s, MIO solvers, including CPLEX [32] and Gurobi [26], have seen massive gains in speed. In the period 1990–2016, MIO solvers became about 1,250,000 times faster [12,55, 25]. Coupled with hardware enhancements over the same period [65], a MIO problem in 2016 can be solved 2 trillion times faster than in 1990. These speedups encourage us to rethink whether NP-hardness should in fact be synonymous with intractability.

There have been some attempts to apply exact MIO methods to SPCA. Moghaddam et al. [54] devised a branch-and-bound method by bounding eigenvalues of submatrices of the covariance matrix—an idea that we build upon in this paper. However, the bounds in the paper were too weak to provide the method with tractability, and the paper admitted the method takes over an hour to run a case with $n = 40, k = 20$, whereas we will demonstrate that our algorithm can run for n in the 1,000s. More recently, Carrizosa and Guerrero [14] proposed a novel MIO formulation of SPCA that involves nonlinearities, but since solutions could only be identified using a local search method, optimality could not be guaranteed. Asteris et al. [2] used a different combinatorial approach, bipartite matchings, to find solutions to a multi-component version of SPCA with disjoint supports. This paper will explore additional methods based on MIO principles.

The work in this paper is also inspired by the application of mixed integer methods to other problems in statistics. In recent years, these techniques have been applied to classification [11] [8], regression [9], support vector machines [57] and subset selection [10] among others. Many more examples are presented in a survey by Bennett and Parrado-Hernández [5] of optimization methods applied to machine learning problems.

1.2 Motivation for ℓ_0-Constrained SPCA Over Other Formulations

In our review of the literature, we encountered an array of proposed solutions to the sparse principal component analysis problem. Some provided algorithms for solving ℓ_1-based formulations and convex relaxations. Other algorithms approached the ℓ_0-constrained formulation, but only provided local search methods without a guarantee of optimality. Some recent work has expanded the size of the problems that can be attempted using these methods, but has not improved the optimality of the resulting solutions. The exact methods that have been proposed have either been too slow to be practical, or too difficult to be solved globally, again requiring local methods. No approach thus far has provided a provably optimal solution to the ℓ_0-constrained SPCA problem, to which we now turn our attention.

Our work focuses on the ℓ_0-constrained SPCA formulation in Problem (4). We believe this formulation is the most important of the four presented formulations to solve. The parameter k is more interpretable than γ and makes it easier to generate solutions of a desired sparsity. In addition, if a problem can be solved for all k, the ℓ_0-penalized problem is automatically solved as a result: the objective $\max \mathbf{x}'\mathbf{Q}\mathbf{x} + \gamma\|\mathbf{x}\|_0$ could be computed at each sparsity level k, and the minimum selected out of finitely many values. On the other hand, at least in practice, the correspondence does not function in reverse. There may be values of k that correspond to no values of γ, and so performing a binary search on the value of γ cannot turn an ℓ_0-penalized algorithm into a ℓ_0-constrained algorithm.

The ℓ_1 variants of the problem are computationally simpler than the ℓ_0 variants, but at a high cost. ℓ_1 penalties and constraints do not just promote sparsity—they simultaneously penalize ("shrink") the solution's loadings. While this can add some robustness to the model [6], it sacrifices the variance explained, even after post-processing by computing a PCA solution on the final support. It will be clear from the computational experiments that ℓ_1-based solutions will consistently explain less variance than ℓ_0-based solutions.

1.3 Our Approach

The approach taken in this paper is to develop a tailored branch-and-bound algorithm to address the ℓ_0-constrained SPCA problem. This algorithm leverages fast first order methods to find feasible solutions, and derives upper and lower bounds from linear algebra principles.

In numerical experiments involving real and generated data sets, we demonstrate that our approach:

1. Finds near-optimal (often optimal) solutions in seconds (for $n = 10,000s, k = 10s$).
2. Gives certifiably optimal solutions in seconds (for $n = 100s, k = 10s$).
3. Provides superior solutions compared to existing methods, by explaining a higher portion of variance and permitting complete control over the desired sparsity.

1.4 Structure

The structure of the paper is as follows. In Section 2, we develop at a high level a branch-and-bound approach to the ℓ_0-constrained SPCA problem and prove the correctness of the algorithm for finding optimal solutions. In Section 3, we derive four upper bounds and three lower bounds for the SPCA problem with partially-determined support using ideas from linear algebra. In Section 4, we discuss parameters that affect the algorithm's performance, strategies for improving computational tractability, and experimental evidence for making principled parameter choices. In Section 5, we perform a variety of computational tests on synthetic and real data sets to assess the algorithmic and statistical performance of our approach, and compare our results to those of other methods from the literature, demonstrating our claim that we are able to provide solutions that explain greater variance than existing methods. In Section 6, we include some concluding remarks.

1.5 Notation

We will refer throughout the paper to a more explicit formulation of the ℓ_0-constrained SPCA problem we will call SPCA-MIO:

$$\begin{aligned} \max_{\mathbf{x},\mathbf{y}} \quad & \mathbf{x}'\mathbf{Q}\mathbf{x} \\ \text{s.t.} \quad & \sum_{i=1}^{n} x_i^2 = 1 \\ & -y_i \le x_i \le y_i \quad i = 1, \dots, n \\ & \sum_{i=1}^{n} y_i = k \\ & \mathbf{x} \in [-1,1]^n \\ & \mathbf{y} \in \{0,1\}^n. \end{aligned} \qquad \text{SPCA-MIO} \quad (5)$$

Here $\mathbf{y}$ is the support vector that captures which components of $\mathbf{x}$ are non-zero. $\mathbf{Q}$ is an $n \times n$ positive semi-definite covariance matrix. We restrict $\sum_{i=1}^{n} y_i = k$ exactly instead of using an inequality, since this does not restrict the feasibility of the $\mathbf{x}$ values we are interested in(since any solution with $\|\mathbf{x}\|_0 \le k$ corresponds to at least one support $\mathbf{y}$ with $\|\mathbf{y}\|_0 = k$), and helpfully limits the feasible space of solutions from the perspective of algorithm design. That is, fixing $\sum_{i=1}^{n} y_i = k$ allows us to say that when $n - k$ components of $\mathbf{y}$ are fixed at 0, the remaining k must be fixed at 1.

We use the notation $\|\cdot\|_p$ to denote the ℓ_p norm of $\mathbf{x}$. That is, $\|\mathbf{x}\|_1 = \sum_i |x_i|$, $\|\mathbf{x}\|_2 = \sqrt{\sum_i x_i^2}$, and $\|\mathbf{x}\|_0 = |\{i : x_i \neq 0\}|$, the ℓ_0 "pseudo-norm". We use $|\cdot|$ to refer to the absolute values of a number or vector, and the set cardinality when applied to a set. In general, we reserve capital boldface letters like $\mathbf{Q}$ for matrices, boldface lowercase letters like $\mathbf{x}$ for vectors, non-bold lower case letters like k for scalars, and script capital letters like $\mathcal{N}$ for sets.

2 A Branch-and-Bound Algorithm for SPCA

Since the SPCA-MIO is non-convex, it cannot be solved by any but the most flexible, non-convex mixed integer nonlinear solvers, such as Baron, SCIP, and Couenne. A direct attempt was made to use an existing solver (Couenne) to solve the problem, but Couenne was unable to solve even the smallest problems n=13, k=10 within an hour. We report a full demonstration in Section 5.5, concluding that a general branch-and-bound approach is not tractable for solving the SPCA-MIO model on the scale of problems addressed by other existing SPCA algorithms.

In this section, we will develop a customized branch-and-bound framework to solve the SPCA-MIO problem. We build an enumeration tree that branches on the components of the support vector $\mathbf{y}$ that underlies $\mathbf{x}$, and we develop bounds from closely studying the algebraic properties of the tree's subproblems.

To aid in the development of this algorithm, we will consider a family of problems derived from the original SPCA-MIO problem that we call SPCA with Partially Determined Support (SPCA-PDS). Each of these problems represents a subproblem of SPCA-MIO where we fix some components of $\mathbf{y}$.

To form these problems, we will use lower and upper bound vectors $\mathbf{l}, \mathbf{u} \in \{0,1\}^n$ to bound $\mathbf{y}$. This captures decisions we have made about a particular y_i. To enforce $y_i = 1$ we set $l_i = 1$, and to enforce $y_i = 0$ we set $u_i = 0$.

$$\begin{aligned}
\max_{\mathbf{x},\mathbf{y}} \quad & \mathbf{x}'\mathbf{Q}\mathbf{x} \\
\text{s.t.} \quad & \sum_{i=1}^{n} x_i^2 = 1 \\
& -y_i \le x_i \le y_i \quad i = 1, \dots, n \\
& \sum_{i=1}^{n} y_i = k \\
& \mathbf{x} \in [-1,1]^n \\
& \mathbf{y} \in \{0,1\}^n \\
& \mathbf{l} \le \mathbf{y} \le \mathbf{u}, \quad \mathbf{l}, \mathbf{u} \in \{0,1\}^n.
\end{aligned} \qquad \text{SPCA-PDS} \ (6)$$

For ease of reference, we will use the notation $\mathbb{X}(\mathbf{l}, \mathbf{u}, k)$ to mean the set of feasible $\mathbf{x}$ vectors in Problem (6), so we can write the SPCA problem with partially determined support (SPCA-PDS), as simply:

$$\max_{\mathbf{x} \in \mathbb{X}(\mathbf{l},\mathbf{u},k)} \mathbf{x}'\mathbf{Q}\mathbf{x}. \tag{7}$$

We can form an SPCA-PDS problem by choosing any $\mathbf{l}, \mathbf{u} \in \{0,1\}^n$ and $0 \le k \le n$. The problem will have a non-empty feasible set as long as $\mathbf{l} \le \mathbf{u}$, $\sum l_i \le k$, $\sum u_i \ge k$. In particular, if we choose $\mathbf{l} = \{0\}^n, \mathbf{u} = \{1\}^n$ then we have exactly Problem (5), SPCA-MIO. If instead we make $\mathbf{l} = \mathbf{u}$, we have SPCA-MIO for a fixed support. In this case, the variable $\mathbf{y}$ becomes a constant, and by eliminating the rows and columns of $\mathbf{Q}$ that correspond with $y_i = 0$, the problem reduces to Problem (1), PCA. For choices of $\mathbf{l}, \mathbf{u}$ between these two extremes, we get an SPCA problem with partially determined support.

This problem formulation gives us a vocabulary with which to discuss the SPCA branch-and-bound tree. Each node of the tree is identified with a pair $(\mathbf{l}, \mathbf{u})$. At the root node we have the original SPCA-MIO problem ($\mathbf{l} = \{0\}^n, \mathbf{u} = \{1\}^n$), and at each branch, the bounds are tightened in one dimension.

If we were to enumerate the entire tree, every leaf would have $\mathbf{l} = \mathbf{u}$. However, some leaves would represent empty feasible sets, because they would require $\sum y_i \ne k$. For example, consider a problem with $n = 4, k = 2$. If $(\mathbf{l}, \mathbf{u}) = (\{1\}^4, \{1\}^4)$ for example, there could be no $\mathbf{x}$ satisfying $\|\mathbf{x}\|_0 \le k$ in the feasible set. Instead of exploring parts of the tree with empty feasible sets, we will stop when we reach a node that represents at most one feasible $\mathbf{y}$ vector. For example, the bottom left node in Figure 1 is terminal for this problem, even though $\mathbf{l} \ne \mathbf{u}$, because $\sum l_i = k$, and so it corresponds to exactly one support vector, $\mathbf{y} = (1, 1, 0, 0)$.

To determine when we have arrived at a terminal node, we can use the following terminal function for upper and lower bound vectors $\mathbf{l}, \mathbf{u}$ and $k > 0$:

$$isTerminal(\mathbf{l}, \mathbf{u}, k) = \begin{cases} true, & \text{if } \sum l_i \ge k, \\ true, & \text{if } \sum u_i \le k, \\ false, & otherwise. \end{cases} \tag{8}$$

At one of these terminal nodes, SPCA-PDS reduces to PCA restricted to that support, and therefore the optimal solution at one of these terminal nodes is the first principal component of $\mathbf{Q}$ restricted to the rows and columns where the support is non-zero. Since we will not branch

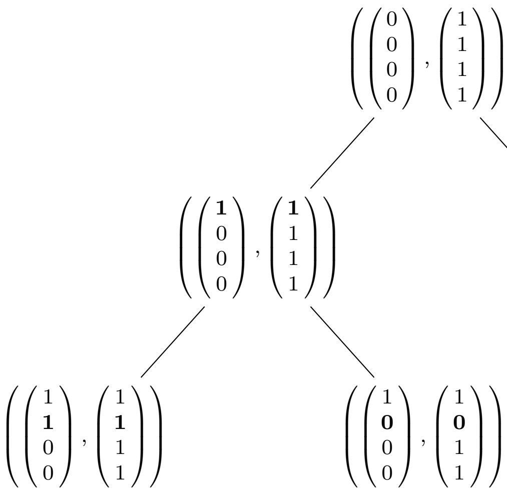

Fig. 1 An example of two branching steps in an SPCA enumeration tree.

further on these nodes, we can discard them after checking to see if they improve upon the best feasible solution.

The real benefit of branch-and-bound, however, is that we do not need to enumerate the entire tree. We only have to explore subtrees further where superior solutions may potentially exist—that is, we need only explore subtrees with upper bounds above the best feasible solution found so far. To take advantage of this, we will build methods to compute upper and lower bounds at each node for the subtree rooted at that node.

This algorithm relies on two key methods: *lower*$(\cdot)$ and *upper*$(\cdot)$, that calculate lower and upper bounds for SPCA-PDS:

$$\mathit{lower}(\mathbf{l},\mathbf{u},k) = \left(\mathbf{x}'\mathbf{Q}\mathbf{x} \text{ for some } \mathbf{x} \in \mathbb{X}(\mathbf{l},\mathbf{u},k)\right), \tag{9}$$

$$\mathit{upper}(\mathbf{l},\mathbf{u},k) \geq \left(\mathbf{x}'\mathbf{Q}\mathbf{x} \text{ for all } \mathbf{x} \in \mathbb{X}(\mathbf{l},\mathbf{u},k)\right). \tag{10}$$

For now, we will take for granted that we have ways of computing these bounds. In Section 3, we will make explicit how these bounds are computed.

2.1 Introduction to Algorithm 1: Optimal-SPCA

We now present Algorithm 1 and the accompanying Theorem 1 to make these ideas explicit, and prove the validity of the approach. The goal of Algorithm 1 is to find $\epsilon-$optimal solutions to SPCA-MIO, for any $\epsilon \geq 0$. This means Optimal-SPCA will find a solution $\hat{\mathbf{x}}$ with objective value within ϵ of the optimal objective value. Since SPCA-MIO is a special case of SPCA-PDS with $\mathbf{l} = \mathbf{0}, \mathbf{u} = \mathbf{1}$ we can write the feasibility set of SPCA-MIO as $\mathbb{X}(\mathbf{0},\mathbf{1},k)$, so that we can write the definition of an $\epsilon-$optimal solution as some $\hat{\mathbf{x}}$ that satisfies:

$$\hat{\mathbf{x}}'\mathbf{Q}\hat{\mathbf{x}} \geq \mathbf{x}'\mathbf{Q}\mathbf{x} - \epsilon \quad \forall \mathbf{x} \in \mathbb{X}(\mathbf{0},\mathbf{1},k).$$

We do allow the choice of $\epsilon = 0$ here. Since there are finitely many nodes in the enumeration tree, the algorithm can find the provably optimal solution with $\epsilon = 0$. In practice, choosing a small $\epsilon > 0$ reduces run-time without significantly sacrificing solution quality.

Algorithm 1 considers a tree consisting of pairs $(\mathbf{l}, \mathbf{u})$ of lower and upper bounds on the support $\mathbf{y}$. A list of unexplored nodes is maintained, and at each iteration of the algorithm, an unexplored node is selected and split into two complementary nodes by choosing an index that is not fixed and setting it to $l_i = 1$ and $u_i = 0$ in the two child nodes, respectively. If the new nodes are terminal, the SPCA-PDS problem is equivalent to PCA, and so the exact solution is returned. If the node is not terminal, the algorithm continues by computing additional bounds. Segments of the tree are removed as their upper bounds fall below the best known solution, until the highest upper bound is within ϵ of the best feasible solution, guaranteeing ϵ-tolerance.

Algorithm 1: Optimal-SPCA

```
   Input : Covariance matrix $\mathbf{Q}$, target sparsity $k$, optimality tolerance $\epsilon \geq 0$
   Output: Sparse vector $\hat{\mathbf{x}}$ with $\|\hat{\mathbf{x}}\|_0 = k$ and $\|\hat{\mathbf{x}}\|_2 \leq 1$ that maximizes $\hat{\mathbf{x}}'\mathbf{Q}\hat{\mathbf{x}}$ within the optimality
           tolerance $\epsilon$
1  initialize $n_0 = (\mathbf{l}, \mathbf{u}) = (\{0\}^n, \{1\}^n)$ as the root node
2  initialize node set $\mathcal{N} = \{n_0\}$
3  initialize $\hat{\mathbf{x}} = \{0\}^n$
4  initialize lower bound $lb = 0$, upper bound $ub =$ the largest eigenvalue of $\mathbf{Q}$, $\lambda_{\max}(\mathbf{Q})$
5  while $ub - lb > \epsilon$; do
6      select $(\mathbf{l}, \mathbf{u}) \in \mathcal{N}$
7      select some index $i$ where $l_i = 0, u_i = 1$
8      for $val = 0, 1$ do
9          newnode $= ((\ell_1, \dots, \ell_{i-1}, val, \ell_{i+1}, \dots, \ell_n), (u_1, \dots, u_{i-1}, val, u_{i+1}, \dots, u_n)$
10         if isTerminal(newnode) then
11             compute the optimal solution $\mathbf{x}$ to the problem at newnode,
12             set $upper = lower = \mathbf{x}'\mathbf{Q}\mathbf{x}$
13         else
14             compute $lower = lower(newnode)$, with corresponding feasible point $\mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)$
15             compute $upper = upper(newnode)$
16         if $lower > lb$ then
17             update $lb = lower$
18             update $\hat{\mathbf{x}} = \mathbf{x}$
19             remove any node $\in \mathcal{N}$ with $upper \leq lb$
20         if $upper > lb$ then
21             add newnode to $\mathcal{N}$
22     remove $(\mathbf{l}, \mathbf{u})$ from $\mathcal{N}$
23     update $ub$ to be the greatest value of $upper$ over $\mathcal{N}$
24 return $\hat{\mathbf{x}}$
```

Theorem 1 (Correctness of Optimal-SPCA) *Optimal-SPCA terminates in finitely many iterations at an $\epsilon-$optimal solution for SPCA-MIO.*

Proof The validity of Optimal-SPCA relies primarily on the validity of the branch-and-bound approach for discrete optimization problems [43]. We outline here a few key points involved in the application of this method to our problem.

The maximum number of iterations of the algorithm is capped at the number of possible nodes (which are distinct elements of $\{0, 1\}^{2n}$), ensuring the algorithm will terminate in finitely many iterations. Nodes are only removed in Steps 19 and 22, if they are dominated by the best feasible solution so far, or because they have been partitioned into pairs of complementary nodes. Therefore no optimal solution will be removed before termination. The algorithm will not terminate until $ub - lb \leq \epsilon$ so that some feasible solution has been identified that is ϵ-optimal.

We conclude that Optimal-SPCA is an exact algorithm for computing an $\epsilon-$optimal solution to the SPCA problem, and is guaranteed to terminate in finite time, given valid methods *upper* and *lower* computable at every node $(\mathbf{l}, \mathbf{u})$.

2.2 A Note on Subsequent Components

The intended purpose of Optimal-SPCA is to find the optimal first sparse principal component, but it can also be used to find subsequent sparse components. Given a first component $\mathbf{x}_1$, we can project the matrix $\mathbf{Q}$ into the space perpendicular to $\mathbf{x}_1$ as in Equation 2:

$$\mathbf{Q}_2 = (I - \mathbf{x}_1\mathbf{x}_1')\mathbf{Q}(I - \mathbf{x}_1\mathbf{x}_1').$$

A second sparse principal component can then be found by applying Optimal-SPCA to the new matrix $\mathbf{Q}_2$. This second component $\mathbf{x}_2$ will explain the optimal amount of remaining variance once $\mathbf{x}_1$ is accounted for.

In traditional PCA, this process of repeated projections of the matrix $\mathbf{Q}$ and the choice of the principal eigenvector of $\mathbf{Q}$ results in a sequence of orthogonal components. Moreover, for all p, the first p components jointly explain the greatest variance of all possible sets of p vectors.

Our SPCA algorithm returns subsequent components that differ in two respects. Subsequent components are not guaranteed to be orthogonal (though they may be), and since the components are computed consecutively, they may not be jointly optimal for the multivariate problem. However, the computational performance of this approach, demonstrated in Section 5.3, is strong compared to other methods that do not take an iterative approach to multiple components.

It is conceivable to adapt Optimal-SPCA to maximize the total variance explained over multiple sparse principal components at once. The branch-and-bound formulation extends easily to multiple components by allowing nodes to represent lower and upper bounds on the supports of multiple components at once. The computational difficulty arises in calculating bounds and in exploring the exponentially larger tree of possible supports. While this would be possible to formulate, it would only be able to solve the smallest of SPCA problems.

3 Linear Algebra Bounds for SPCA with Partially-Determined Support

We turn now to constructing

$$\begin{aligned} \mathit{lower}(\mathbf{l}, \mathbf{u}, k) &= \left(\mathbf{x}'\mathbf{Q}\mathbf{x} \text{ for some } \mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)\right), \\ \mathit{upper}(\mathbf{l}, \mathbf{u}, k) &\geq \left(\mathbf{x}'\mathbf{Q}\mathbf{x} \text{ for all } \mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)\right), \end{aligned}$$

which are upper and lower bounds to the optimal value of the problem SPCA-PDS:

$$\max_{\mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)} \mathbf{x}'\mathbf{Q}\mathbf{x}.$$

We start by considering trivial constructions:

$$lb_0 = 0, \tag{11}$$

$$ub_0 = \lambda_{\max}(\mathbf{Q}). \tag{12}$$

These are trivially valid lower and upper bounds. Optimal-SPCA will eventually arrive at the optimal solution of SPCA-MIO by using lb_0 and ub_0. However, by providing tighter bounds, we can improve the performance of Optimal-SPCA and reduce the number of nodes that must be explored in order to prove optimality.

In this section, we propose several methods for computing tighter bounds that will enable us to solve the SPCA problem while exploring a much smaller subset of the branch-and-bound tree. First, we introduce a subroutine that will be useful throughout the development of these bounds.

3.1 A Useful Truncation Routine

Many of the bounds we develop involve a step that takes an arbitrary length n vector $\mathbf{x}$ as input and outputs a vector $\hat{\mathbf{x}}$ in the feasible set $\mathbb{X}(\mathbf{l},\mathbf{u},k)$. We will want to make this selection to minimize $\|\mathbf{x}-\hat{\mathbf{x}}\|_2$ over $\hat{\mathbf{x}} \in \mathbb{X}(\mathbf{l},\mathbf{u},k)$.

It can be shown that this problem is solved by preserving the largest loadings subject to feasibility constraints. That is, $\hat{\mathbf{x}}$ contains the largest k elements of $\mathbf{x}$ subject to two restrictions: that the elements must include those with $l_i = 1$ and must not include those with $u_i = 0$. A routine that makes this procedure explicit will be useful for the work ahead. We provide this routine explicitly in Algorithm 2.

Algorithm 2: $Trunc(\cdot,\mathbf{l},\mathbf{u},k)$

```
  Input  : Vector $\mathbf{x}$, target sparsity $k$, lower support bound $\mathbf{l}$, upper support bound $\mathbf{u}$
  Output: Sparse vector $\hat{\mathbf{x}} \in \mathbb{X}(\mathbf{l},\mathbf{u},k)$ that minimizes $\|\mathbf{x}-\hat{\mathbf{x}}\|_2$
1 Form the set $\mathcal{I}_1 = \{i|l_i = 1\}$
2 Form the set $\mathcal{I}_2 = \arg\max_{I\subset\{1,\dots,n\}} \sum |x_i|$ s.t. $l_i = 0, u_i = 1 \forall i \in I, |I| = k - \sum l_i$. Ties are broken by
  choosing the smallest index.
3 $\hat{x}_i = \begin{cases} x_i & i \in \mathcal{I}_1 \cup \mathcal{I}_2, \\ 0, & \text{otherwise.} \end{cases}$
4 return $\hat{\mathbf{x}}$
```

The function $Trunc$ will be used in two ways. It will be used in Sections 3.5 and 3.6 for projecting an arbitrary vector onto a vector that is feasible for the SPCA-PDS problem (after normalization). $Trunc$ can also be thought of as solving the maximization Problem (13), which makes $Trunc$ useful for computing the worst case eigenvalue bounds over all feasible supports in Sections 3.3 and 3.4.

$$\begin{aligned} \min_{\mathbf{y}} \quad & \sum_i x_i^2(1-y_i) \\ \text{s.t.} \quad & \mathbf{l} \le \mathbf{y} \le \mathbf{u}, \sum y_i = k. \end{aligned} \tag{13}$$

3.2 An Upper Bound from Eigenvalues

First, we consider how simple eigenvalues can provide a tighter upper bound than ub_0 for the problem SPCA-PDS. Note that a feasible $\mathbf{x} \in \mathbb{X}(\mathbf{l},\mathbf{u},k)$ is also feasible for the problem with the sparsity constraint lifted. That is, $\mathbf{x}$ is feasible for the problem:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{x}'\mathbf{Q}\mathbf{x} \\ \text{s.t.} \quad & \sum_{i=1}^n x_i^2 = 1 \\ & x_i = 0 \quad \forall i \in \{i|u_i = 0\}. \end{aligned} \tag{14}$$

But since $x_i = 0$ for all i with $u_i = 0$, we can rewrite the objective using a different covariance matrix,

$$\mathbf{x}'\mathbf{Q}\mathbf{x} = \sum_{\substack{i,j \text{ s.t.} \\ u_i, u_j = 1}} q_{ij} x_i x_j = \mathbf{x}'\mathbf{Q}_{\mathbf{u}}\mathbf{x}, \tag{15}$$

where we define $\mathbf{Q}_{\mathbf{u}}$ as:

$$(\mathbf{Q}_{\mathbf{u}})_{ij} = \begin{cases} q_{ij}, & u_i = u_j = 1, \\ 0, & \text{otherwise.} \end{cases} \tag{16}$$

Problem (14) thus simplifies to:

$$\begin{aligned} \max_{\mathbf{x}} \quad & \mathbf{x}'\mathbf{Q}_{\mathbf{u}}\mathbf{x} \\ \text{s.t.} \quad & \|\mathbf{x}\|_2 = 1. \end{aligned} \tag{17}$$

This is now a non-sparse PCA problem with optimal solution value $\lambda_{\max}(\mathbf{Q}_{\mathbf{u}})$. Since Problem (14) is a relaxation of Problem (5), $\lambda_{\max}(\mathbf{Q}_{\mathbf{u}})$ provides an upper bound to Problem (5). This gives us an upper bound for SPCA-PDS as well, which we will write as:

$$ub_1(\mathbf{l}, \mathbf{u}, k) = \lambda_{\max}(\mathbf{Q}_{\mathbf{u}}). \tag{18}$$

3.3 An Upper Bound from the Matrix Trace

Linear algebra gives the fact that the sum of the eigenvalues of a matrix is the trace of that matrix. Since our matrix $\mathbf{Q}$ is positive semi-definite, its eigenvalues are non-negative, and therefore the trace of $\mathbf{Q}$ provides an upper bound on each of the eigenvalues of $\mathbf{Q}$. From this fact, we can derive another upper bound for the SPCA problem with partially determined support.

Consider any $\mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)$, and again let $\mathbf{y}$ be the corresponding support vector of $\mathbf{x}$ ($y_i = 0$ if and only if $x_i = 0$). Then $\mathbf{x}$ is feasible for the problem where the support is restricted to $\mathbf{y}$. That is, we know $\mathbf{x} \in \mathbb{X}(\mathbf{y}, \mathbf{y}, k)$.

We can use the same trick as in Equation (15) to equate,

$$\mathbf{x}'\mathbf{Q}\mathbf{x} = \mathbf{x}'\mathbf{Q}_{\mathbf{y}}\mathbf{x} \tag{19}$$

where $\mathbf{Q}_{\mathbf{y}}$ is formed as $\mathbf{Q}_{\mathbf{u}}$ was formed, by zeroing out rows and columns of $\mathbf{Q}$ where $y_i = 0$. Then as before, $\mathbf{x}'\mathbf{Q}\mathbf{x} \leq \lambda_{\max}(\mathbf{Q}_{\mathbf{y}})$, and this time we apply the new bound of the trace:

$$\mathbf{x}'\mathbf{Q}\mathbf{x} \leq \lambda_{\max}(\mathbf{Q}_{\mathbf{y}}) \leq \text{trace}(\mathbf{Q}_{\mathbf{y}}). \tag{20}$$

We can compute an upper bound like this for all points $\mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)$ by considering the maximum possible value of trace$(\mathbf{Q}_{\mathbf{y}})$ over all feasible $\mathbf{y}$ satisfying $\mathbf{l} \leq \mathbf{y} \leq \mathbf{u}$, $\sum y_i = k$. In this case, we take $Trunc(diag(\mathbf{Q}), \mathbf{l}, \mathbf{u}, k)$ and sum the components. This gives an upper bound on trace$(\mathbf{Q}_{\mathbf{y}})$ over all feasible $\mathbf{y}$, and therefore an upper bound ub_2 for the entire problem.

$$ub_2(\mathbf{l}, \mathbf{u}, k) = \sum_i Trunc(diag(\mathbf{Q}), \mathbf{l}, \mathbf{u}, k)_i. \tag{21}$$

3.4 An Upper Bound from the Gershgorin Circle Theorem

The Gershgorin Circle Theorem [61] states that the eigenvalues of a positive semi-definite matrix $\mathbf{Q}$ are bounded by the largest absolute column sum of $\mathbf{Q}$. We define the Gershgorin operator as:

$$Gersh(\mathbf{Q}) = \max_j \sum_i |Q_{ij}|. \tag{22}$$

Then we can state the Gershgorin Circle Theorem as:

$$Gersh(\mathbf{Q}) \geq \lambda_{\max}(\mathbf{Q}). \tag{23}$$

This theorem inspires an additional upper bound for SPCA-PDS.

Again, we start by considering a point $\mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)$. Let $\mathbf{y}$ be the corresponding support vector of $\mathbf{x}$. Then $\mathbf{x}$ is feasible for the problem where the support is restricted to $\mathbf{y}$. That is, we know $\mathbf{x} \in \mathbb{X}(\mathbf{y}, \mathbf{y}, k)$, and we have as before

$$\mathbf{x}'\mathbf{Q}\mathbf{x} = \mathbf{x}'\mathbf{Q_y}\mathbf{x} \leq \lambda_{\max}(\mathbf{Q_y}). \tag{24}$$

Now we can bound $\lambda_{\max}(\mathbf{Q_y})$ from above using the Gershgorin Circle Theorem:

$$\mathbf{x}'\mathbf{Q}\mathbf{x} \leq \lambda_{\max}(\mathbf{Q_y}) \leq Gersh(\mathbf{Q_y}). \tag{25}$$

Since this is true for any feasible $\mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)$ the problem is overall bounded by the maximum value of $Gersh(\mathbf{Q_y})$ over all feasible supports $\mathbf{y}$. Explicitly,

$$\begin{aligned} \max_{\mathbf{x}\in\mathbb{X}(\mathbf{l},\mathbf{u},k)} \mathbf{x}'\mathbf{Q}\mathbf{x} &\leq \max_{\substack{\mathbf{l}\leq\mathbf{y}\leq\mathbf{u}\\ \sum y_i = k}} Gersh(\mathbf{Q_y}) \\ &= \max_{\substack{\mathbf{l}\leq\mathbf{y}\leq\mathbf{u}\\ \sum y_i = k}} \left(\max_j \sum_i |(\mathbf{Q_y})_{ij}| \right) \\ &= \max_j \left(\max_{\substack{\mathbf{l}\leq\mathbf{y}\leq\mathbf{u}\\ \sum y_i = k}} \sum_i |(\mathbf{Q_y})_{ij}| \right) \\ &= \max_{j,\ u_j=1} \left(\sum_i Trunc(|\mathbf{Q}_j|, \mathbf{l}, \mathbf{u}, k)_i \right). \end{aligned} \tag{26}$$

Note that we only have to take the maximum over columns j with $u_j = 1$, since any column with $u_j = 0$ will be zeroed out in the matrix $\mathbf{Q_y}$ in the previous step. This maximum is then a new upper bound ub_3 for the SPCA problem with partially-determined support $(\mathbf{l}, \mathbf{u})$:

$$ub_3(\mathbf{l}, \mathbf{u}, k) = \max_{j, u_j=1} \left(\sum_i Trunc(|\mathbf{Q}_j|, \mathbf{l}, \mathbf{u}, k)_i \right). \tag{27}$$

3.5 A Lower Bound from Eigenvalues

Any element $\mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)$ will suffice to establish a lower bound. A principled way of choosing $\mathbf{x}$ is to select some feasible support vector $\mathbf{y}$, and take $\mathbf{x}$ as the principal eigenvector of $\mathbf{Q_y}$.

One fast method of finding a promising $\mathbf{y}$ is to take the principal eigenvector of $\mathbf{Q_u}$, that is, $\mathbf{v}_{\max}(\mathbf{Q_u})$, feed it to the truncation function $Trunc(\cdot, \mathbf{l}, \mathbf{u}, k)$ and let $\mathbf{y}$ be the support of this vector. Then we simply take $\mathbf{x}$ to be the principal eigenvector of $\mathbf{Q_y}$. The lower bound is then,

$$lb_1(\mathbf{l}, \mathbf{u}, k) = \lambda_{\max}(\mathbf{Q_y}) \text{ s.t. } \mathbf{y} = \text{supp}(Trunc(\mathbf{v}_{\max}(\mathbf{Q_u}), \mathbf{l}, \mathbf{u}, k)). \tag{28}$$

3.6 A Lower Bound from Yuan and Zhang (2013)

Yuan and Zhang [70] created an iterative truncation algorithm that inspires an additional lower bound on our SPCA-PDS problem. The algorithm in [70] begins with a k-sparse vector $\mathbf{x}$ and alternates between a power method step ($\mathbf{x} = \mathbf{Qx}$) and a truncation step, limiting the support to the dimensions with the k largest absolute loadings.

We can adapt this algorithm to accommodate the partially determined support $(\mathbf{l}, \mathbf{u})$ by making a change to the truncation step. Instead of simply preserving the k components with the largest absolute value, we must include components where $l_i = 1$, and we are forbidden from including components where $u_i = 0$. That is, we involve our truncation routine $Trunc(\cdot, \mathbf{l}, \mathbf{u}, k)$ The complete algorithm is then:

Algorithm 3: A lower bound for SPCA with partially-determined support from thresholding

```
Input : Covariance matrix $\mathbf{Q}$, target sparsity $k$, lower support bound $\mathbf{l}$, upper support bound $\mathbf{u}$,
        optimality tolerance $\epsilon \geq 0$
Output: Sparse vector $\mathbf{x} \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k)$
1 initialize $\mathbf{x}^0 = \{0\}^n,\ \mathbf{x}^1 \in \mathbb{X}(\mathbf{l}, \mathbf{u}, k), \mathbf{x}^1 \neq \mathbf{x}^0$ (for instance, let $\mathbf{x}^1 = Trunc(\mathbf{v}_{\max}(\mathbf{Q}), \mathbf{l}, \mathbf{u}, k)$ )
2 while $\|\mathbf{x}_{m+1} - \mathbf{x}_m\|_2 \geq \epsilon$; do
3     $\mathbf{x}^m = \mathbf{Q}\mathbf{x}^{m-1}$
4     $\mathbf{x}^m = Trunc(\mathbf{x}^m, \mathbf{l}, \mathbf{u}, k)$
5     $\mathbf{x}^m = \mathbf{x}^m / \|\mathbf{x}^m\|_2$
6 return $\mathbf{x}^m$
```

The bound lb_2 takes longer to compute than lb_1, but tends to dominate lb_1 in value. This is especially important if the algorithm is terminated early, to provide the best chance at finding an optimal, or a near-optimal solution.

$$lb_2(\mathbf{l}, \mathbf{u}, k) = Alg3(\mathbf{l}, \mathbf{u}, k, \mathbf{Q}, \epsilon). \tag{29}$$

3.7 Collecting Upper and Lower Bounds

In Section 3.2-3.6, we constructed three upper bounds and two lower bounds. In practice, we can derive overall bounds by combining all these methods, which we will run at every node:

$$ub = \min(ub_1, ub_2, ub_3) \tag{30}$$
$$lb = \max(lb_1, lb_2). \tag{31}$$

Note that we do not need to include lb_0, ub_0 in the comparisons because in all cases $lb_1 \geq lb_0 = 0$ and $ub_1 \leq ub_0 = \lambda_{\max}(\mathbf{Q})$, making the trivial bounds redundant.

The remaining bounds, however, are critically not redundant. In particular, ub_2 and ub_3 dominate one another depending on other characteristics of the covariance matrix $\mathbf{Q}$. Consider the following matrices with $k = 2$.

$$\mathbf{M}_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{M}_2 = \begin{pmatrix} 13 & 8 & 0 \\ 8 & 5 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{32}$$

For the matrix $\mathbf{M}_1$, we have $ub_2 = 2$, the sum of the largest two diagonal elements, while $ub_3 = 1$, the largest absolute column sum of two elements, which corresponds to the true largest eigenvalue of the matrix. In the matrix $\mathbf{M}_2$, we have $ub_2 = 18$ while $ub_3 = 21$, this time ub_2 dominating and coming close to the true largest 2-sparse eigenvalue of 17.9. In both cases, ub_1 dominates ub_2 and ub_3, but this is due to the third dimension in the constructed examples being uncorrelated with the first two; ub_1 is often the weakest bound. Since it is difficult to know a priori but not difficult to calculate which bound will be dominant, it benefits the algorithm to calculate all three and choose the smallest value for an overall upper bound.

4 Computational Tactics

So far, we have presented Optimal-SPCA at a high level. We have discussed how lower and upper bounds for the SCPA problem with partially determined support allow us to explore the space of support vectors in order to find a provably optimal solution more efficiently than by brute force enumeration. We have suggested a number of lower and upper bounds which, taken in combination, provide overall bounds on the problems at each node.

These discussions have not fully determined the details of Optimal-SPCA, however. In implementation, we must consider a number of parameters that affect how we make branching, bounding, and other decisions. First we will look at the choice between exploring the breadth or the depth of the tree, controlling the maximum number of unexplored nodes at any point in time. We consider several approaches to choosing which dimension of $\mathbf{x}$ to branch on, controlling the strategy and the number of steps to take in searching for the branching dimension. Finally we consider emphasizing lower or upper bounds, controlling the number of local search steps to use when establishing a lower bound at each node.

In this section, we discuss in detail the choice of these parameters, providing numerical experiments on the effects of the parameters on run-times, and recommended values that we will use in the discussion in Section 5 of Optimal-SPCAs accuracy and scalability. For each parameter set, we performed 500 iterations of sampling 80 out of the 101 variables in the Communities data set from the UCI database [48]. We ran Optimal-SPCA on each reduced problem, with a target sparsity $k = 10$. We report the average values of time in seconds for the algorithm to prove optimality ("time to upper bound"), and with hindsight, the time at which the algorithm identified as a feasible solution what would prove to be the optimal solution ("time to lower bound"). We also report the total number of nodes explored in the enumeration tree. We use these results to provide recommended parameter values. While testing each parameter, we held the others constant at default values: a maximum number of nodes of $10,000$, two local search steps at each node, and ten steps for selecting a branching dimension at each node.

4.1 Choosing Nodes: Best-First vs. Depth-First

The two node selection heuristics we focus on are best-first and depth-first search. In best-first search, at each iteration we choose the node with the highest remaining upper bound. This ensures that the overall upper bound decreases at most iterations (as long as there are not ties). In depth-first search, we continue to develop the most recently added node until it either reaches termination or its upper bound falls below the best feasible solution and the subtree at that node can be eliminated. Depth-first search keeps the number of remaining nodes small, but spends less time tightening the overall upper bound.

An algorithm running best-first search alone would quickly grow the number of remaining nodes, continually replacing a single node with the largest upper bound by two child nodes, only reducing the number of nodes in the tree as the upper bound begins to converge to the lower bound and the algorithm nears completion. The size of the tree could scale like $\binom{n}{k}$. On the other hand, a depth-first search algorithm would keep the number of nodes exceedingly small, keeping the size of the tree on the order of nk. While the methods differ in the order of nodes explored and how many nodes are kept in memory, they typically result in the same number of total nodes explored by the end of the algorithm.

To trade-off between these approaches, we consider the maximum number of nodes that we keep in a queue in memory at a given time. When the remaining nodes are fewer than this cutoff, the algorithm branches on nodes with the highest upper bound (best-first search). When the number of nodes crosses over the cutoff, the algorithm turns to depth-first search, starting with the most recently added nodes, until the number of active nodes is reduced.

Table 1 Effects of best- vs depth-first search (controlled by the maximum size of the branch-and-bound tree) on run-times and tree size in Optimal-SPCA. A combination of the two approaches leads to the lowest running times.

Maximum Number of Nodes	Time (seconds) to Lower Bound	Time (seconds) to Upper Bound	Number of Nodes Explored
10	0.1734	6.7330	22,804
100	0.1011	6.5560	22,801
1,000	0.1028	6.5482	22,803
10,000	0.1034	6.9058	22,805
100,000	0.1033	7.8298	22,801

In Table 1 we see minor effects of the maximum queue size on the computational speed of the algorithm. Generally, higher values of the limit result in longer times to prove optimality, with no significant difference in the number of nodes explored. This indicates that the algorithm is spending more time at each branching step when the limit is higher, since the algorithm needs to search over all the remaining nodes to choose one to branch on and to update the overall upper bound. (In turn, this was faster than maintaining a list of nodes sorted by upper bound value.)

We do not recommend choosing the smallest possible limit, however. While low limits do not significantly hinder time to convergence, they mask the progress of the algorithm by making slow progress on the reduction of the upper bound until the algorithm has nearly terminated. In Figure 2 we ran Optimal-SPCA 100 times with a range of queue limits on the original Communities problem. We plotted the average upper bound at each time point, with the lines terminating when optimality is proven. In Table 2 we report in detail the time until optimality is proven in this set of experiments.

The results from Figure 2 and Table 2 lead us to two conclusions. First, emphasizing best-first search (allowing the maximum number of nodes in the queue to grow large) results in the best upper bound achievable at each time point. Second, emphasizing depth-first search (keeping the number of nodes in the queue small) results in the fastest termination of the algorithm. The relative importance of these priorities depends on the application, scale, and goals of the modeler, and so the final parameter value is largely a matter of taste.

4.2 Choosing Dimensions: Random, Fixed, and Adaptive

Once a leaf node has been selected for branching, the algorithm must decide which as-yet-undetermined index to fix in the child nodes. This corresponds to Step 7 of Optimal-SPCA. The

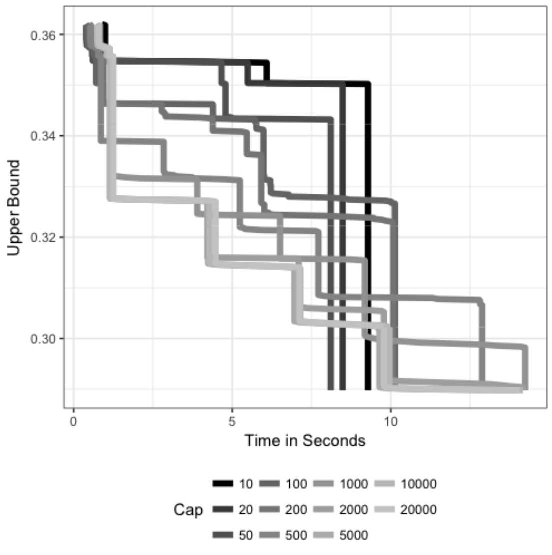

Fig. 2 Optimal-SPCA's rate of convergence of upper bound on Communities data set depends on the maximum number of nodes allowed in the queue. The larger the queue is allowed to grow, the tighter the upper bound at each point in time, but at the expense of taking longer overall to converge.

Table 2 Optimal-SPCA's time to prove optimality on the Communities dataset depends on the maximum number of nodes allowed in the queue. Generally, larger queues result in longer times to converge, but overly restricting the size of the queue also impedes run time.

Maximum Number of Nodes	Time (seconds) to prove optimality
10	9.28
20	8.48
50	8.11
100	10.13
200	10.08
500	12.88
1,000	14.23
2,000	14.17
5,000	13.99
10,000	14.17
20,000	14.06

careful selection of a branching dimension has a significant impact on the bounds we can achieve down the tree. By encouraging the algorithm to select dimensions that explain large portions of the as-yet-unexplained variance, we can better partition the solution space.

A naive approach is to use a completely random selection of dimensions. However, this wastes significant time partitioning the solution space into very similar looking subspaces with similar upper and lower bounds, making little progress.

One simple prioritization is to branch on the dimension of those available (i with $l_i = 0, u_i = 1$) that has the largest possible absolute component of the principal eigenvector of $\mathbf{Q}$, that is, $i = \arg\max_{l_i=0,u_i=1} |\lambda_{\max}(\mathbf{Q})_i|$. At the root node, this is in fact the dimension that explains the greatest possible variance. Lower in the tree, this dimension may not necessarily explain the most variance, but the heuristic continues to function better than random selection.

Another possible approach is to prioritize the dimensions not by loadings of the first eigenvector but by the eigenvalues of the standard basis vectors ($i = \arg\max_i \mathbf{e}_i'\mathbf{Q}\mathbf{e}_i = \arg\max_i Q_{ii}$). This approach performs well when there is a significant spread in the values Q_{ii}, but the approach degenerates when $\mathbf{Q}$ is a correlation matrix (so that all $Q_{ii} = 1$).

A more sophisticated approach is to prioritize by the loadings of the principal eigenvector of $\mathbf{Q_u}$, which captures the changes in variance-explaining power that take place as we descend the tree.

In Table 3 we compare the four methods of prioritizing branching dimensions: random, by $|\mathbf{v}_{\max}(\mathbf{Q})_i|$, by Q_{ii}, and by $|\mathbf{v}_{\max}(\mathbf{Q_u})_i|$. In the case of ties, the branching dimension is chosen randomly from among the top-ranking indices. We considered both the covariance matrix and the correlation matrix of the communities data set. To keep run-time reasonable for the covariance matrix test, we reduced the problem to select 50 out of the 101 variables, and we used $k = 5$. For the correlation matrix, we selected 80 of the 101 variables and used $k = 5$.

The results show that choosing branching dimensions according to Q_{ii} and $|\mathbf{v}_{\max}(\mathbf{Q_u})_i|$ dominate overall, with Q_{ii} offering some advantage when eigenvalues are widely distributed, and $|\mathbf{v}_{\max}(\mathbf{Q_u})_i|$ outperforming Q_{ii} when data is normalized. Our general recommendation is to prioritize branching dimensions according to $|\mathbf{v}_{\max}(\mathbf{Q_u})_i|$, since this method performs well for finding principal components of both covariance and correlation matrices. We included prioritization by Q_{ii} as a secondary option in the algorithm.

Table 3 Effects of dimension selection methodology on run-times and tree size in Optimal-SPCA. Prioritizing the dimensions with large loadings on the eigenvector of the reduced matrix $\mathbf{Q_u}$ performs consistently across problem structures, with other methods performing well in more limited contexts.

Data set	Method for Dimension Selection	Time (seconds) to Lower Bound	Time (seconds) to Upper Bound	Number of Nodes Explored
communities	random	0.0679	8.3950	20,092
communities	$\lvert\mathbf{v}_{\max}(\mathbf{Q})_i\rvert$	0.0694	1.6149	3,365
communities	Q_{ii}	0.0649	0.0771	9
communities	$\lvert\mathbf{v}_{\max}(\mathbf{Q_u})_i\rvert$	0.0659	0.1624	109
normCommunities	random	0.1016	0.3590	184
normCommunities	$\lvert\mathbf{v}_{\max}(\mathbf{Q})_i\rvert$	0.0975	0.2808	85
normCommunities	Q_{ii}	0.0994	0.3399	148
normCommunities	$\lvert\mathbf{v}_{\max}(\mathbf{Q_u})_i\rvert$	0.1024	0.2663	69

Computing the first eigenvector of $\mathbf{Q_u}$ for every node, however, is computationally expensive. As a middle ground, we consider taking several power-method steps from the principal eigenvalue of $\mathbf{Q}$ to that of $\mathbf{Q_u}$. The more steps we take, the more confident we are in our choice of branching dimension, but the more we will pay in computational time per node.

In Table 4 we see the trade-off between run-time per node and number of nodes explored as we compute more iterations of the power method at each node to help us select a branching dimension. The trade-off in this case is beneficial up to about 20 iterations. In larger problems, the time cost of computing these iterations will be higher, and in order to provide a default that will work for a range of problem sizes, we recommend a smaller value of 10 iterations. In some cases, this value can be increased to improve performance.

Table 4 Effects of dimension selection steps on run-times and tree size in Optimal-SPCA. More time spent selecting the dimension to branch on results in fewer nodes explored in the tree, and overall smaller run-times.

Iterations for Dimension Selection	Time (seconds) to Lower Bound	Time (seconds) to Upper Bound	Number of Nodes Explored
0	0.1020	183.5768	694,490
1	0.0987	32.8741	118,874
2	0.0993	20.9729	73,563
3	0.0982	15.4463	52,784
4	0.0965	12.9729	44,201
5	0.0969	11.1544	37,810
6	0.0961	10.6936	36,225
7	0.0951	10.0357	34,059
8	0.0954	8.1165	27,381
9	0.0943	7.2210	23,848
10	0.0953	6.8468	22,805
15	0.0956	4.5605	14,958
20	0.0947	4.2638	13,809
30	0.0962	3.1453	9,955
40	0.0962	4.1013	13,654
50	0.0974	3.0518	9,727

4.3 Feasibility vs. Optimality: De-emphasizing Lower Bounds

We must also consider how much effort should be expended to work to find the best lower bounds at each node. The parameter that controls the effort put into the calculation of lower bounds is the number of local search steps we take at each node using Algorithm 3. If we take zero steps, we trivially use as a lower bound the support formed by prioritizing the dimensions with largest absolute loadings in the first eigenvector of $\mathbf{Q}$. As we take more steps, we come closer to running the entirety of Yuan and Zhang's algorithm at each node, improving the quality of the bounds, but drastically increasing the work per node.

In Table 5 we see little benefit from more than a couple iterations of Algorithm 3. In this experiment, dedicating more time to improving lower bounds saves at most 0.005 seconds off the convergence of the lower bound, but adds significantly to the time it takes for the algorithm to converge, since it requires more computation at each node. In most cases, only one or two iterations of Algorithm 3 are necessary to achieve strong lower bounds.

4.4 Computing Eigenvalues and Eigenvectors

Many steps of the algorithm involve computing the principal eigenvalue of a matrix $\mathbf{Q}_\mathbf{y}$. In each case we used a power method approach with some adaptations.

Two of the upper bounds (ub_2, ub_3) do not require an eigenvalue computation. Since our final upper bound will be the minimum of all three upper bounds, and because the power method's process of determining the largest eigenvalue is monotonically increasing, we can run the eigenvalue upper bound ub_1 last, and terminate the power method early if the norm of $\mathbf{x}$ crosses above the lesser of ub_2 and ub_3.

If the problem is very high dimensional, with the matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ having many more dimensions than samples $(n \gg m)$, then we can save some additional time in computing eigenvalues. In this case, the data matrix $\mathbf{A}$ is actually much smaller than the covariance matrix $\mathbf{Q} = \mathbf{A}'\mathbf{A}/(m-1) \in \mathbb{R}^{n\times n}$. We can avoid repeated multiplication by the large matrix $\mathbf{Q}$ by

Table 5 Effects of local search effort for lower bounds on run-times and tree size in Optimal-SPCA. Additional maximum allowed local search steps slightly improve the time to identify the optimal solution, but significantly increase the time to prove optimality.

Maximum Steps	Time (seconds) to Lower Bound	Time (seconds) to Upper Bound	Number of Nodes Explored
0	0.0999	5.3822	22,804
1	0.0989	6.1295	22,802
2	0.0989	6.8352	22,803
3	0.0975	7.5748	22,802
4	0.0973	8.3241	22,801
5	0.0969	9.0840	22,802
6	0.0958	9.7792	22,800
7	0.0963	10.5060	22,801
8	0.0947	11.2302	22,804
9	0.0956	11.9454	22,806
10	0.0956	12.6792	22,805
15	0.0962	16.2266	22,793
20	0.0969	19.8704	22,805
30	0.0970	26.9684	22,802
40	0.0988	34.0215	22,802
50	0.1000	41.3359	22,802

noting the equivalence::

$$\begin{aligned}\mathbf{Q}^p &= \frac{(\mathbf{A}'\mathbf{A})^p}{(m-1)^p}\\ &= \frac{(\mathbf{A}')(\mathbf{A}\mathbf{A}')^{p-1}(\mathbf{A})}{(m-1)^p}.\end{aligned}$$

Thus, instead of repeatedly multiplying a vector by $\mathbf{Q} \in \mathbb{R}^{n\times n}$, we can first multiply it by $\mathbf{A}$, then multiply the result repeatedly by the much smaller matrix $\mathbf{A}\mathbf{A}' \in \mathbb{R}^{m\times m}$, finishing by multiplying the result of that step by $\mathbf{A}'$. For data sets from applications like genomics, where n far outstrips m, this will save significant computational time.

Another trick that helps in high dimensions is running computations not in the original space, but in the space of the components involved. For example, when computing the eigenvalue of some $\mathbf{Q}_\mathbf{y}$, instead of repeatedly multiplying the zero values in $\mathbf{Q}_\mathbf{y}$ by a vector, it is preferable to form a new $\tilde{\mathbf{Q}}_\mathbf{y}$ that only has height and width equal to $\sum y_i$, find its principal eigenvalue, and map those loadings back to the support vector $\mathbf{y}$. This dimension reduction is also important for the local search for lower bounds.

5 Computational Results

In this section, we implement Optimal-SPCA on a number of real data sets and compare the performance (solution quality and run-time) of our method against the state of the art. We compare Optimal-SPCA to those by d'Aspremont et al. [19], Hein and Bühler [29], Richtárik et al. [60], Zou et al. [73], Jolliffe et al. [38], Journée et al. [39], and Yuan and Zhang [70]. Since Yuan and Zhang's method is adapted for use as a primal heuristic within Optimal-SPCA, Yuan and Zhang and Optimal-SPCA often arrive at the same solution in these experiments, with Optimal-SPCA taking additional time and providing a certificate of optimality.

Experiments for Tables 6-11 were performed on a four-core 2.4 GHz processor with 16 GB of RAM. Experiments for Tables 12 and 13 were limited to a single core on MIT's Engaging cluster, with a 2.0 GHz processor and 32 GB of RAM. The methods by d'Aspremont et al., Hein and Bühler, and Richtárik et al. were run with the authors' published packages in Matlab. The methods by Zou et al. and Jolliffe et al. were run with their published packages in R. The methods by Journée et al., Yuan and Zhang, and Optimal-SPCA were written for use in this paper in Julia 0.6 without the use of mathematical programming solvers. The experiments in Couenne for Table 10 used Couenne version 0.5 with AmplNLWriter in Julia. For all experiments, the default parameters were used for Optimal-SPCA: a maximum number of nodes of $10,000$, two local search steps at each node, and ten steps for selecting a branching dimension at each node. For experiments where computational time is included, the results reported are the average times across all the experimental runs.

5.1 Description of Data Sources

We performed experiments on five data sets: the frequently used Pitprops set, three sets from the UCI database, and a gene expression data set used to study breast cancer. The Pitprops data set was used in four of the seven papers we compare to (Zou et al. [73], Jolliffe et al. [38], Journée et al. [39], and Yuan and Zhang [70]). Five of the methods used gene expression data from various sources (d'Aspremont et al. [19], Hein and Bühler [29], Zou et al. [73], Journée et al. [39], and Yuan and Zhang [70]). A few additional data sets have been studied in these papers, including sets from the UCI data base, and a number of synthetically generated data sets.

Pitprops The Pitprops data set consists of a 13×13 covariance matrix formed from 180 observations of "pit props made of Corsican pine grown in East Anglia" [27]. The variables included measurements such as the diameter and length of the prop, the number of rings and number of knots. Jeffers [35] attempted to interpret the first six principal components of this covariance matrix, highlighting the difficulty in such interpretation, since each component had significant loadings in each dimension. Since then, a number of papers have performed PCA and variations like SPCA on the data to create more interpretable results, primarily by generating sparser solutions. Below, we will compare our results to the results from [35] and other SPCA papers to see how the principal components compare in sparsity and variance explained.

Wine The Wine data set from the UCI database [48] consists of 178 measurements of 13 chemical attributes of wines, with the objective of classifying the wines by origin. For our work, we ignore the origin categorical variable and just consider the others, which include variables like alcohol level, color intensity, and acidity. The data set has been studied primarily for classification problems with redundancy analysis (RDA).

MiniBooNE The MiniBooNE data set from UCI [48] has 130,065 observations of 50 variables, measuring attributes of neutrinos in an experiment meant to distinguish electron neutrinos from muon neutrinos. As with the Wine data set, we have ignored the classification variable, and created both raw and normalized versions of the problem.

Communities The Communities and Crime data set from UCI [48] originally contained 1994 observations of 128 variables, containing socio-economic data, law enforcement data, and crime data from 1990 and 1995 across the United States. Categorical variables and variables not available for the entire data set were removed to produce a set with 101 numeric variables.

Cancer The breast cancer gene expression dataset, originated in papers by Wang et al. in 2005 and Minn et al. in 2007. The set contains 344 measurements of 22,283 variables.

Since our method assumes centered (mean zero) data, we pre-processed these datasets by subtracting the empirical mean of each variable from each measurement. Moreover, we created additional data sets by normalizing the data (so that each variable has unit variance) to see how the algorithms perform on correlation matrices. The normalized problems are referred to in the reports that follow by prefixing "norm" to the problem name. Since the Pitprops data is presented as a correlation matrix and the original data is lost, this distinction is only relevant for the remaining data sets.

5.2 Comparison of Solution Quality Across Methods

In Tables 6 and 7, we report the variance explained by the components discovered by each method, for each problem, with a fixed target sparsity of $k = 5$ and $k = 10$, respectively. The values marked by an asterisk represent values that were not dominated by another method. Where the method by Journée et al. [39] is marked n/a, there was no parameter choice that resulted in the target sparsity.

Table 6 Variance of real-world data sets explained by solutions with sparsity $k = 5$. Asterisks indicate optimal values. Optimal-SPCA finds the optimal solution in all cases, while other methods only sometimes identify this solution.

Data set	Dimensions	Optimal-SPCA	d'Aspremont et al. [19]	Hein and Bühler [29]	Jolliffe et al. [38]	Journée et al. [39]	Richtárik et al. [60]	Yuan and Zhang [70]	Zou et al. [73]
Pitprops	13	*3.40615	*3.40615	*3.40615	*3.40615	*3.40615	2.72258	*3.40615	*3.40615
Wine	13	*99201.31	*99201.31	*99201.31	*99201.31	*99201.31	*99201.31	99199.39	*99201.28
normWine	13	*3.43978	*3.43978	2.40834	3.43663	n/a	3.43663	3.43663	3.31752
miniBooNE	50	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9
normMiniBooNE	50	*5.00000	4.99971	4.99998	*5.00000	n/a	*5.00000	*5.00000	4.28611
Communities	101	*0.27689	0.27315	0.27292	0.41344	n/a	*0.27689	*0.27689	0.20588
normCommunities	101	*4.86051	4.62919	4.62919	4.51048	n/a	4.45186	*4.86051	4.13879

The key insight from this analysis is that each of the existing methods fails to find the optimal solution on at least one data set. Some data sets, like the miniBooNE set, are easy for all of the methods to solve, while others, like the normalized Wine problem, and both the original and normalized Communities problem, are not solved to optimality by most of the methods.

Of the methods considered in this analysis, those by d'Aspremont et al. and Yuan and Zhang are most successful in finding solutions to difficult problems, and one of these two methods was able to solve every problem.

Additionally, we ran each method on the Pitprops data set for every sparsity level $k = 1, \ldots, 13$. The variances explained by the results are reported in Table 8 and Figure 3. Optimal-SPCA, d'Aspremont et al., and Yuan and Zhang found the optimal solution at every sparsity level, and Hein and Bühler succeeded in all but one. The other methods were far less consistent over the range of sparsity levels.

Table 7 Variance explained by SPCA methods on real-world data sets with $k = 10$. Asterisks indicate optimal values. Optimal-SPCA finds the optimal solution in all cases, while other methods only sometimes identify this solution.

Data set	Dimensions	Optimal-SPCA	d'Aspremont et al. [19]	Hein and Bühler [29]	Jolliffe et al. [38]	Journée et al. [39]	Richtárik et al. [60]	Yuan and Zhang [70]	Zou et al. [73]
Pitprops	13	*4.17264	*4.17264	*4.17264	4.15996	4.14411	4.15996	3.40615	4.11101
Wine	13	*99201.78	*99201.78	*99201.78	*99201.78	*99201.78	*99201.78	99199.58	*99201.78
normWine	13	*4.59429	*4.59429	*4.59429	*4.59429	n/a	*4.59429	3.92632	4.58251
miniBooNE	50	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9	*1.96827e9
normMiniBooNE	50	*9.99999	9.99965	9.03575	*9.99999	n/a	*9.99999	*9.99999	7.55189
Communities	101	*0.44491	*0.44491	0.26666	*0.44491	n/a	*0.44491	0.44312	0.35353
normCommunities	101	*8.82313	7.18684	*8.82313	8.39797	n/a	6.18039	*8.82313	7.66345

Table 8 Variance of Pitprops explained by SPCA methods for various k. Asterisks indicate optimal values. Several methods, including Optimal-SPCA, obtain the optimal solution at each sparsity level.

k	Optimal-SPCA	d'Aspremont et al. [19]	Hein and Bühler [29]	Jolliffe et al. [38]	Journée et al. [39]	Richtárik et al. [60]	Yuan and Zhang [70]	Zou et al. [73]
1	*1.000	*1.000	*1.000	NA	*1.000	*1.000	*1.000	*1.000
2	*1.954	*1.954	*1.954	1.679	1.813	1.081	*1.954	*1.954
3	*2.475	*2.475	*2.475	1.884	2.204	2.077	*2.475	2.329
4	*2.937	*2.937	*2.937	*2.937	*2.937	2.549	*2.937	2.883
5	*3.406	*3.406	*3.406	*3.406	*3.406	2.895	*3.406	*3.406
6	*3.771	*3.771	*3.771	*3.771	*3.771	*3.771	*3.771	*3.771
7	*3.996	*3.996	*3.996	3.840	*3.996	*3.996	*3.996	3.822
8	*4.069	*4.069	*4.069	4.054	*4.069	4.054	*4.069	*4.069
9	*4.139	*4.139	4.116	4.060	*4.139	4.116	*4.139	4.116
10	*4.173	*4.173	*4.173	4.111	4.160	4.160	*4.173	4.160
11	*4.208	*4.208	*4.208	4.113	*4.208	*4.208	*4.208	*4.208
12	*4.218	*4.218	*4.218	4.130	*4.218	*4.218	*4.218	*4.218
13	*4.219	*4.219	*4.219	*4.219	*4.219	*4.219	*4.219	*4.219

5.3 Method performance on the computation of multiple sparse principal components

In the series of experiments reported in Table 9, we observed the performance on SPCA methods that computed multiple components at once and compared these to the outcome of applying Optimal-SPCA sequentially. We did not include the methods by d'Aspremont et al., Journée et al., or Yuan and Zhang since they did not provide methods for computing multiple principal components at once, and the implementation of Hein and Bühler's method in Mathlab failed to run on the large normMiniBoo problem. While Optimal-SPCA is guaranteed to maximize the variance explained by PC1, it does not have a statistical guarantee over multiple components. Nonetheless, we found that the approach of maximizing total variance explained for one compo-

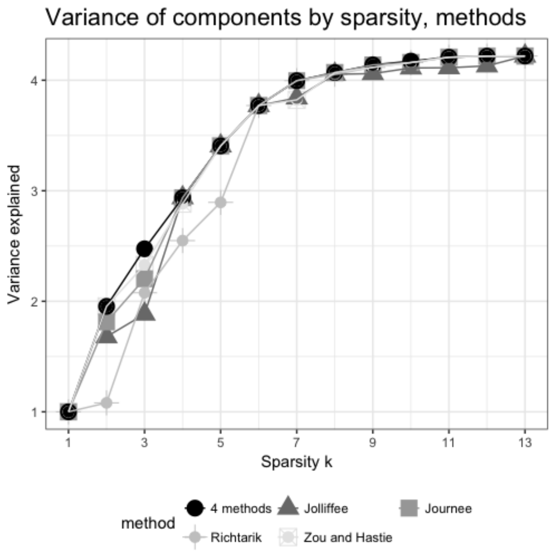

Fig. 3 Variance of Pitprops explained by SPCA methods for various k. Optimal-SPCA, d'Aspremont et al., Hein and Bühler, and Yuan and Zhang are combined into "4 methods" in this plot.

nent at a time (see Section 2.2) was also effective in maximizing total variance explained across multiple components.

Since the principal components in SPCA are not necessarily orthogonal, the computation of marginal variance explained by subsequent components, as well as total variance explained, must be computed carefully. Following the approach of Zou et al. [73], we compute the variance explained by subsequent components by projecting the components into the subspace of the data perpendicular to preceding components.

5.4 Comparison of run times across methods

On each of the datasets used for testing in Section 5.2, we also ran timed tests of each algorithm, five times each. In Table 10 we report the average run time in seconds of each method on each dataset. It should be noted that the run-times are impacted both by the algorithmic complexity of the methods and by the underlying performance of the coding languages. We wrote Optimal-SPCA in the Julia language, which has a run-time advantage over R (Jolliffe et al., Zou et al.) and Matlab (d'Aspremont et al., Hein and Bühler, Richtárik et al.) [18].

Two findings are notable. First, Optimal-SPCA identifies the optimal solution to the SPCA problem within the time frame of the existing algorithms. Second, Optimal-SPCA proves optimality for small k in seconds, and slightly larger k in minutes. For problems with small k, Optimal-SPCA scales similarly to Yuan and Zhang, which lies within it. For more complex problems, Optimal-SPCA spends significantly more time refining upper bounds on the large space of supports in order to prove optimality. While the time to provable optimality does not scale as well with k, the time until the optimal solution is found as a feasible solution remains competitive with other algorithms. This suggests Optimal-SPCA can be beneficial for higher k problems with early termination.

Table 9 Variance of Pitprops explained by SPCA methods across the first three components for $k = 5$. In each case, Optimal-SPCA obtains the highest total variance explained.

Data set	Adj Variance of:	Optimal-SPCA	Hein and Bühler [29]	Jolliffe et al. [38]	Richtárik et al. [60]	Zou et al. [73]
Pitprops	PC1	3.40615	3.40615	2.66602	3.40615	3.27020
Pitprops	PC2	2.15779	1.94730	2.02014	1.23414	1.97907
Pitprops	PC3	1.90637	2.07497	1.94153	0.00176	2.11999
Pitprops	Total	7.47032	7.42842	6.62769	4.64205	7.36925
normWine	PC1	3.43978	3.43663	3.01445	3.43978	3.01516
normWine	PC2	2.38627	2.37488	2.23001	0.00000	2.23558
normWine	PC3	2.09970	1.80185	1.84457	0.01496	1.82003
normWine	Total	7.92575	7.61336	7.08904	3.45473	7.07077
normMiniBoo	PC1	5.00000	N/A*	1.48050	5.00000	3.97389
normMiniBoo	PC2	5.00000	N/A*	1.37906	0.00000	2.27701
normMiniBoo	PC3	4.99999	N/A*	1.28584	1.75260	1.29458
normMiniBoo	Total	15.00000	N/A*	4.14540	6.75260	7.54549

Table 10 Run-times (in seconds) of SPCA methods on real-world datasets. Optimal-SPCA finds the best feasible solution in a time competitive with other methods. Although proving optimality takes significantly longer, the times are still tractable for the largest problems.

Dataset	Dimensions	k	Optimal-SPCA - optimal	Optimal-SPCA - feasible	d'Aspremont et al. [19]	Hein and Bühler [29]	Jolliffe et al. [38]	Journée et al. [39]	Richtárik et al. [60]	Yuan and Zhang [70]	Zou et al. [73]
pitprops	13	5	0.158	0.121	0.250	0.042	0.222	5.507	0.786	0.051	0.103
pitprops	13	10	0.136	0.051	0.233	0.046	0.303	5.510	0.791	0.047	0.104
wine	13	5	0.056	0.053	0.255	0.051	0.097	5.508	0.853	0.052	0.102
wine	13	10	0.043	0.042	0.248	0.051	0.085	5.510	0.818	0.044	0.094
normWine	13	5	0.067	0.056	0.240	0.062	0.188	5.509	0.804	0.048	0.105
normWine	13	10	0.075	0.050	0.236	0.049	0.114	5.510	0.802	0.047	0.098
miniBoo	50	5	0.062	0.061	0.332	0.800	10.204	5.808	105.672	0.078	0.112
miniBoo	50	10	0.064	0.064	0.320	0.906	53.000	5.762	106.200	0.077	0.114
normMiniBoo	50	5	0.076	0.073	0.317	1.993	32.899	6.225	107.263	0.082	0.120
normMiniBoo	50	10	0.071	0.069	0.313	2.000	13.200	6.090	105.800	0.081	0.127
communities	101	5	8.179	0.768	0.378	0.062	3.510	5.656	5.927	0.164	0.125
communities	101	10	167.000	0.101	0.374	0.098	1.244	5.510	6.156	0.114	0.137
normCommunities	101	5	0.393	0.105	0.365	0.075	1.146	5.521	5.893	0.096	0.126
normCommunities	101	10	3.244	0.103	0.379	0.080	3.994	5.520	5.890	0.095	0.127

5.5 Tractability of Optimal-SPCA Compared to Exact Solutions From Non-Convex Solvers

Optimal-SPCA is not the only way to compute exact solutions to SPCA-MIO. We could also send the problem to a solver designed to handle non-convex mixed integer nonlinear programs. We chose to work with Couenne[49] (which stands for Convex Over and Under ENvelopes for Nonlinear Estimation), an open-source solver published by the COIN-OR community. Couenne reformulates problems in order to identify upper bounds, and uses a combination of branch-and-bound, heuristic, and branching techniques that can be customized and augmented. For our experiment, we formulated SPCA-MIO using Julia and JuMP [24] and called Couenne with the AmplNLWriter package in Julia. We used the default options for heuristics and branching procedures in the software.

In Table 11, we report on the run-times of Couenne compared to Optimal-SPCA on our sample of real-world problems, running each algorithm until the optimality gap (upper bound divided by lower bound) was less than 0.01, or until an hour had passed. For larger problems, Couenne did not prove optimality in under an hour, and in these cases we reported the remaining optimality gap. Optimal-SPCA, on the other hand, proved optimality in under an hour in every case, while Couenne took considerably longer, and often did not prove optimality in an hour. When Couenne did prove optimality, it needed to explore more nodes in the problem tree, and did so with more time spent exploring each node. Because of these differences, we do not consider solving SPCA-MIO with a general purpose solver to be tractable, while Optimal-SPCA is tractable even for large problems.

We believe the significant differences here are due to the inability for Couenne to identify reasonable feasible solutions and tight upper bounds. The advantages of SPCA-MIO are that we use a powerful heuristic to generate lower bounds (inspired by Yuan and Zhang [70]), and take advantage of algebraic structure for upper bounds that are not considered as part of Couenne's process of setting bounds.

Table 11 Performance of Optimal-SPCA and Couenne on SPCA-MIO. Despite solving the same formulation of the problem, the solver Couenne takes hundreds of times longer than Optimal-SPCA to converge and explores much more of the enumeration tree to identify the optimal solution.

		Optimal-SPCA			Couenne		
Dataset	k	Explored	Time	Gap	Explored	Time	Gap
pitprops	5	6	0.079	-	993,683	1,528.020	-
pitprops	10	17	0.141	-	1,562,100	>1 hour	323%
wine	5	2	0.054	-	204	1.420	-
wine	10	2	0.047	-	18,738	44.220	-
normWine	5	4	0.049	-	612,615	951.300	-
normWine	10	6	0.049	-	512,639	>1 hour	294%
miniBoo	5	2	0.060	-	128	9.390	-
miniBoo	10	2	0.060	-	1,600	32.120	-
normMiniBoo	5	2	0.063	-	89,000	>1 hour	31,771%
normMiniBoo	10	2	0.079	-	75,900	>1 hour	18,559%
communities	5	23,779	12.870	-	13,800	>1 hour	485%
communities	10	498,309	338.950	-	12,500	>1 hour	44,522%
normCommunities	5	39	0.180	-	23,300	>1 hour	211,018%
normCommunities	10	593	0.726	-	23,100	>1 hour	∞

5.6 Scaling of Optimal-SPCA for large scale datasets

In this set of experiments, we generated problems of various sizes by selecting random sets of various dimensions from the Cancer dataset and the Micromass dataset, and running Optimal-SPCA with $k = 5$ on each problem. We terminated the algorithm after five hours if it had not completed. In Table 12, we report average results over 100 experiments performed for each problem size. The columns "Portion Proved in 1h" and "Portion Proved in 5h" document the percent of experiments in which Optimal-SPCA was able to prove optimality within 1 hour and 5 hours respectively.

Table 12 Optimal-SPCA is able to find optimal solutions and prove optimality in several hours for the largest real-world problems, and on some data sets converges in even less time.

Dataset	Dimensions	Time (sec) to Best Solution	Time (sec) to Prove Optimality	Portion Proved in 1h	Portion Proved in 5h
cancer	50	1.52	1.57	1.00	1.00
cancer	100	4.25	4.75	1.00	1.00
cancer	250	5.92	41.80	1.00	1.00
cancer	500	7.97	61.27	0.97	0.97
cancer	750	10.29	126.91	0.96	0.96
cancer	1000	15.21	429.44	0.96	0.99
cancer	2500	105.17	751.24	0.91	0.96
cancer	5000	353.47	424.83	0.94	0.94
cancer	10000	1448.80	1595.40	1.00	1.00
micromass	50	0.30	0.32	1.00	1.00
micromass	100	0.09	0.09	1.00	1.00
micromass	250	0.21	0.22	1.00	1.00
micromass	500	0.49	0.58	1.00	1.00
micromass	750	0.85	0.97	1.00	1.00
micromass	1000	1.94	2.11	1.00	1.00

These experiments show that Optimal-SPCA can be used to solve problems and prove optimality in most cases for up to $10,000$ dimensions. We also conclude that Optimal-SPCA can return optimal solutions reliably, even when the algorithm does not have enough time to prove optimality. This suggests that Optimal-SPCA can be used to great effect even on the largest problems, generating optimal solutions in short periods of time, even if it cannot quickly provide a certificate of optimality. In the next section we further explore the quality of solutions achievable by Optimal-SPCA under short time limits.

5.7 Quality of Feasible Solutions with Early Termination in High Dimensions

As we have seen, Optimal-SPCA typically establishes the correct lower bound long before the upper bound is tightened. That is, it discovers the optimal solution in much less time than it takes to prove optimality. Because of this, we have confidence that this algorithm can be used on even larger data sets, or in more time-constrained applications, simply by setting a time limit and taking the best solution at the end of that period, instead of running to optimality. Alternatively, the acceptable optimality gap (the difference between upper and lower bounds) can be widened to save run-time without sacrificing the quality of the final solution.

As problem sizes scale, many of the methods we have discussed will fail to finish in 1 or even 10 minutes, even with the fewest number of iterations. We will focus our attention on comparing our method to Yuan and Zhang [70], which is the fastest of the existing methods and most reliable at high dimensions. In these experiments, we selected a subset of the variables in the Cancer data set and ran both Yuan and Zhang and Optimal-SPCA with a time limits of one minute, ten minutes, and one hour. We reported the algorithms as having tied if they resulted in variances explained within 0.001% of one another, otherwise we reported which algorithm dominated.

Table 13 Yuan and Zhang and Optimal-SPCA are competitive for finding feasible solutions on high-dimensional subsets of the Cancer dataset.

		Experiments dominated by		
dim	time (s)	Optimal-SPCA	Yuan and Zhang	Tie
50	60	0.5%	1.0%	98.5%
100	60	2.5%	2.0%	95.5%
250	60	2.3%	1.0%	96.7%
500	60	1.0%	1.0%	98.0%
750	60	0.8%	0.3%	99.0%
1000	60	0.5%	1.0%	98.5%
50	600	2.0%	1.3%	96.7%
100	600	0.5%	1.0%	98.5%
250	600	1.3%	0.8%	98.0%
500	600	1.0%	0.8%	98.2%
750	600	1.3%	0.5%	98.2%
1000	600	1.3%	0.8%	98.0%
50	3600	1.3%	1.0%	97.7%
100	3600	2.5%	3.0%	94.5%
250	3600	1.5%	0.8%	97.7%
500	3600	1.5%	0.5%	98.0%
750	3600	1.8%	0.8%	97.5%
1000	3600	1.5%	0.8%	97.8%

We see in Table 13 that both methods returned the same result in most cases. The portion of the time that each algorithm outperformed the another was comparable. This demonstrates that Optimal-SPCA, in addition to providing provable optimality in the long-run, provides results through the early stopping heuristic that are competitive with even the fastest algorithms at large scale.

5.8 Conclusions for Computation

In these experiments, we have shown that Optimal-SPCA, uniquely among SPCA algorithms, provides the provably optimal solution to the problem in finite time. By tuning the algorithm's parameters to suit the problem and experimental priorities, Optimal-SPCA is competitive with existing algorithms in run-time. For large-scale problems where many methods fail to find a solution in tractable time, an early-terminated Optimal-SPCA offers solutions typically of equal or higher quality than the most scalable methods.

We believe these experiments demonstrate that Optimal-SPCA is suitable for general use on both small and large scale problems, produces similar or superior solutions to other methods, and is the only method guaranteed to return optimal solutions in finite time.

6 Conclusions

In this paper, we have derived an algorithm, or rather a class of algorithms, for solving the sparse principal component analysis problem to optimality. The algorithm proves optimality and allows direct control of sparsity, while remaining computationally tractable for large problems. There are many possibilities for computational improvements to these algorithms. It is possible that more careful storage and accounting of existing nodes, reuse of eigenvalues found in the tree as warm starts for lower nodes, and other ideas can continue to reduce run-times and make a branch-and-bound approach even more appealing for solving SPCA. Generally, this work demonstrates the applicability of discrete optimization techniques to problems in statistics and machine learning that have historically and primarily been tackled using continuous optimization techniques. This paper adds to a growing number of works that provide tractable discrete optimization approaches to machine learning problems and improve solution quality over existing methods.

References

1. Amini, A.A., Wainwright, M.J.: High-dimensional analysis of semidefinite relaxations for sparse principal components. In: IEEE International Symposium on Information Theory, pp. 2454–2458. IEEE (2008)
2. Asteris, M., Papailiopoulos, D., Kyrillidis, A., Dimakis, A.G.: Sparse PCA via bipartite matchings. In: Advances in Neural Information Processing Systems, pp. 766–774 (2015)
3. Bair, E., Hastie, T., Paul, D., Tibshirani, R.: Prediction by supervised principal components. Journal of the American Statistical Association **101**(473), 119–137 (2006)
4. Beck, A., Vaisbourd, Y.: The sparse principal component analysis problem: Optimality conditions and algorithms. Journal of Optimization Theory and Applications **170**(1), 119–143 (2016)
5. Bennett, K.P., Parrado-Hernández, E.: The interplay of optimization and machine learning research. Journal of Machine Learning Research **7**(Jul), 1265–1281 (2006)
6. Bertsimas, D., Copenhaver, M.S.: Characterization of the equivalence of robustification and regularization in linear and matrix regression. European Journal of Operational Research (2017)
7. Bertsimas, D., Copenhaver, M.S., Mazumder, R.: Certifiably optimal low rank factor analysis. Journal of Machine Learning Research **18**(29), 1–53 (2017)
8. Bertsimas, D., Dunn, J.: Optimal classification trees. Machine Learning pp. 1–44 (2017)
9. Bertsimas, D., King, A.: An algorithmic approach to linear regression. Operations Research **64**(1), 2–16 (2016)
10. Bertsimas, D., King, A., Mazumder, R., et al.: Best subset selection via a modern optimization lens. The Annals of Statistics **44**(2), 813–852 (2016)
11. Bertsimas, D., Shioda, R.: Classification and regression via integer optimization. Operations Research **55**(2), 252–271 (2007)
12. Bixby, R.E.: A brief history of linear and mixed-integer programming computation. Documenta Mathematica pp. 107–121 (2012)
13. Candès, E.J., Li, X., Ma, Y., Wright, J.: Robust principal component analysis? Journal of the ACM (JACM) **58**(3), 11 (2011)
14. Carrizosa, E., Guerrero, V.: rs-Sparse principal component analysis: A mixed integer nonlinear programming approach with VNS. Computers & Operations Research **52**, 349–354 (2014)
15. Chamberlain, G., Rothschild, M.: Arbitrage, factor structure, and mean-variance analysis on large asset markets (1982)
16. Chan, S.O., Papailiopoulos, D., Rubinstein, A.: On the worst-case approximability of sparse PCA. arXiv preprint arXiv:1507.05950 (2015)
17. Chen, Y., Jalali, A., Sanghavi, S., Xu, H.: Clustering partially observed graphs via convex optimization. Journal of Machine Learning Research **15**(1), 2213–2238 (2014)
18. Computing, J.: Julia micro-benchmarks (2018). URL `https://julialang.org/benchmarks/`
19. d'Aspremont, A., Bach, F., Ghaoui, L.E.: Optimal solutions for sparse principal component analysis. Journal of Machine Learning Research **9**(Jul), 1269–1294 (2008)
20. d'Aspremont, A., El Ghaoui, L., Jordan, M.I., Lanckriet, G.R.: A direct formulation for sparse PCA using semidefinite programming. SIAM Review **49**(3), 434–448 (2007)
21. Deluzio, K., Astephen, J.: Biomechanical features of gait waveform data associated with knee osteoarthritis: an application of principal component analysis. Gait & posture **25**(1), 86–93 (2007)
22. Ding, C., He, X.: K-means clustering via principal component analysis. In: Proceedings of the twenty-first international conference on Machine learning, p. 29. ACM (2004)
23. Du, Q., Fowler, J.E.: Hyperspectral image compression using jpeg2000 and principal component analysis. IEEE Geoscience and Remote sensing letters **4**(2), 201–205 (2007)
24. Dunning, I., Huchette, J., Lubin, M.: JuMP: A modeling language for mathematical optimization. SIAM Review **59**(2), 295–320 (2017). DOI 10.1137/15M1020575
25. Gurobi Optimization Inc.: Gurobi 7.0 performance benchmarks. `http://www.gurobi.com/pdfs/benchmarks.pdf` (2015). Accessed 17 December 2016
26. Gurobi Optimization Inc.: Gurobi optimizer reference manual (2017). URL `http://www.gurobi.com`
27. Hand, D.J., Daly, F., McConway, K., Lunn, D., Ostrowski, E.: A handbook of small data sets, vol. 1. CRC Press (1993)
28. Hastie, T., Tibshirani, R., Wainwright, M.: Statistical learning with sparsity: the lasso and generalizations. CRC press (2015)
29. Hein, M., Bühler, T.: An inverse power method for nonlinear eigenproblems with applications in 1-spectral clustering and sparse PCA. In: Advances in Neural Information Processing Systems, pp. 847–855 (2010)
30. Hotelling, H.: Relations between two sets of variates. Biometrika **28**(3/4), 321–377 (1936)
31. Hsu, Y.L., Huang, P.Y., Chen, D.T.: Sparse principal component analysis in cancer research. Translational cancer research **3**(3), 182 (2014)
32. IBM: IBM ILOG CPLEX User's manual (2017). URL `https://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/`

33. Iezzoni, A.F., Pritts, M.P.: Applications of principal component analysis to horticultural research. HortScience **26**(4), 334–338 (1991)
34. Iguchi, T., Mixon, D.G., Peterson, J., Villar, S.: Probably certifiably correct k-means clustering. Mathematical Programming **165**(2), 605–642 (2017)
35. Jeffers, J.N.: Two case studies in the application of principal component analysis. Applied Statistics pp. 225–236 (1967)
36. Jolliffe, I.T.: Rotation of principal components: choice of normalization constraints. Journal of Applied Statistics **22**(1), 29–35 (1995)
37. Jolliffe, I.T.: Principal component analysis. Wiley Online Library (2002)
38. Jolliffe, I.T., Trendafilov, N.T., Uddin, M.: A modified principal component technique based on the LASSO. Journal of Computational and Graphical Statistics **12**(3), 531–547 (2003)
39. Journée, M., Nesterov, Y., Richtárik, P., Sepulchre, R.: Generalized power method for sparse principal component analysis. The Journal of Machine Learning Research **11**, 517–553 (2010)
40. Kaiser, H.F.: The varimax criterion for analytic rotation in factor analysis. Psychometrika **23**(3), 187–200 (1958)
41. Kumar, V., Kanal, L.N.: Parallel branch-and-bound formulations for and/or tree search. IEEE transactions on pattern analysis and machine intelligence **42**(6), 768–778 (1984)
42. Labib, K., Vemuri, V.R.: An application of principal component analysis to the detection and visualization of computer network attacks. Annales des Telecommunications/Annals of Telecommunications **61**(1-2), 218–234 (2006)
43. Land, A.H., Doig, A.G.: An automatic method of solving discrete programming problems. Econometrica: Journal of the Econometric Society pp. 497–520 (1960)
44. Lee, S., Epstein, M.P., Duncan, R., Lin, X.: Sparse principal component analysis for identifying ancestry-informative markers in genome-wide association studies. Genetic epidemiology **36**(4), 293–302 (2012)
45. Lee, Y.K., Lee, E.R., Park, B.U.: Principal component analysis in very high-dimensional spaces. Statistica Sinica pp. 933–956 (2012)
46. Leng, C., Wang, H.: On general adaptive sparse principal component analysis. Journal of Computational and Graphical Statistics **18**(1), 201–215 (2009)
47. Li, G.J., Wah, B.W.: Coping with anomalies in parallel branch-and-bound algorithms. IEEE Transactions on Computers **100**(6), 568–573 (1986)
48. Lichman, M.: UCI machine learning repository (2013). URL `http://archive.ics.uci.edu/ml`
49. Lougee-Heimer, R.: The common optimization interface for operations research. IBM Journal of Research and Development **47**(1), 57–66 (2003)
50. Luss, R., Teboulle, M.: Conditional gradient algorithms for rank-one matrix approximations with a sparsity constraint. SIAM Review **55**(1), 65–98 (2013)
51. Ma, Z., et al.: Sparse principal component analysis and iterative thresholding. The Annals of Statistics **41**(2), 772–801 (2013)
52. Mangasarian, O.L.: Exact 1-norm support vector machines via unconstrained convex differentiable minimization. Journal of Machine Learning Research **7**(Jul), 1517–1530 (2006)
53. Mazumder, R., Radchenko, P., Dedieu, A.: Subset selection with shrinkage: Sparse linear modeling when the snr is low. arXiv preprint arXiv:1708.03288 (2017)
54. Moghaddam, B., Weiss, Y., Avidan, S.: Spectral bounds for sparse PCA: Exact and greedy algorithms. In: Advances in neural information processing systems, pp. 915–922 (2005)
55. Nemhauser, G.L.: Integer Programming: the Global Impact. Presented at EURO, INFORMS, Rome, Italy, 2013. `http://euro-informs2013.org/data/http_/euro2013.org/wp-content/uploads/nemhauser.pdf` (2013). Accessed 9 September 2015
56. Papailiopoulos, D.S., Dimakis, A.G., Korokythakis, S.: Sparse PCA through low-rank approximations. In: ICML (3), pp. 747–755 (2013)
57. Platt, J.C.: 12 fast training of support vector machines using sequential minimal optimization. Advances in kernel methods pp. 185–208 (1999)
58. Price, A.L., Patterson, N.J., Plenge, R.M., Weinblatt, M.E., Shadick, N.A., Reich, D.: Principal components analysis corrects for stratification in genome-wide association studies. Nature genetics **38**(8), 904–909 (2006)
59. Richman, M.B.: Rotation of principal components. Journal of climatology **6**(3), 293–335 (1986)
60. Richtárik, P., Takáč, M., Ahipaşaoğlu, S.D.: Alternating maximization: unifying framework for 8 sparse PCA formulations and efficient parallel codes. arXiv preprint arXiv:1212.4137 (2012)
61. Scott, D.S.: On the accuracy of the Gerschgorin circle theorem for bounding the spread of a real symmetric matrix. Linear algebra and its applications **65**, 147–155 (1985)
62. Snoek, J., Larochelle, H., Adams, R.P.: Practical bayesian optimization of machine learning algorithms. In: Advances in neural information processing systems, pp. 2951–2959 (2012)
63. Sra, S., Nowozin, S., Wright, S.J.: Optimization for machine learning. Mit Press (2012)
64. Tibshirani, R.: Regression shrinkage and selection via the lasso. Journal of the Royal Statistical Society. Series B (Methodological) pp. 267–288 (1996)

65. Top500 Supercomputer Sites: Performance development. `http://www.top500.org/statistics/perfdevel/` (2016). Accessed 17 December 2016
66. Wilkinson, J.H.: The algebraic eigenvalue problem, vol. 87. Clarendon Press Oxford (1965)
67. Witten, D., Tibshirani, R., Hastie, T.: A penalized matrix decomposition, with applications to sparse principal components and canonical correlation analysis. Biostatistics **10**(3), 515–534 (2009)
68. Witten, D.M., Tibshirani, R.J.: Extensions of sparse canonical correlation analysis with applications to genomic data. Statistical applications in genetics and molecular biology **8**(1), 1–27 (2009)
69. Yanover, C., Meltzer, T., Weiss, Y.: Linear programming relaxations and belief propagation–an empirical study. Journal of Machine Learning Research **7**(Sep), 1887–1907 (2006)
70. Yuan, X.T., Zhang, T.: Truncated power method for sparse eigenvalue problems. Journal of Machine Learning Research **14**(Apr), 899–925 (2013)
71. Zeng, Z.Q., Yu, H.B., Xu, H.R., Xie, Y.Q., Gao, J.: Fast training support vector machines using parallel sequential minimal optimization. In: Intelligent System and Knowledge Engineering, 2008. ISKE 2008. 3rd International Conference on, vol. 1, pp. 997–1001. IEEE (2008)
72. Zhang, Y., Ghaoui, L.E.: Large-scale sparse principal component analysis with application to text data. In: Advances in Neural Information Processing Systems, pp. 532–539 (2011)
73. Zou, H., Hastie, T., Tibshirani, R.: Sparse principal component analysis. Journal of Computational and Graphical Statistics **15**(2), 265–286 (2006)

A Overview of the Optimal-SPCA implementation in Julia

The linked repository contains an implementation of Optimal-SPCA written in Julia 0.6.0. The latest version of this software is available on GitHub at `https://github.com/lauren897/Optimal-SPCA`. The Algorithm directory contains the Julia files that comprise the algorithm, and the Data directory contains an example dataset.

In order to run this software, you must install a recent version of Julia from `http://julialang.org/downloads/`. The most recent version of Julia at the time this code was last tested before publication was Julia 0.6.0.

Two packages must be installed in Julia before the code can be run. These packages are `DataFrames`, and `StatsBase`. They can be added by running `Pkg.add("DataFrames")` and `Pkg.add("StatsBase")` respectively.

At this point, the file `test.jl` should run successfully. To run the script, navigate to the Algorithm directory, and run `include("test.jl")`. The script will run Optimal-SPCA on the Pitprops dataset, and then generate an additional random problem and run the algorithm on that problem. It will then identify the first few sparse principal components using Optimal-SPCA sequentially and reporting the cumulative variance explained.

The key method used in the algorithm is is `branchAndBound`. It takes two required arguments: `prob`, and `k`. The variable `prob` uses a custom type that holds the original data as well as the covariance matrix associated with the problem. (If data is not available, the Cholesky factorization of the covariance matrix will suffice.) The data is presented in an $m \times n$ array, with m data points in n dimensions. The corresponding covariance matrix is $n \times n$. The parameter `k` is a positive integer less than n and represents the desired sparsity.

By default, `branchAndBound` solves the problem and returns the objective function value, solution vector, and a few performance metrics, including time elapsed and the number of nodes explored. There are many optional parameters, some of which are discussed in detail in our paper. Other parameters have to do with technical aspects of the algorithm, like convergence criteria and resizing arrays. These are commented on in detail in the `branchAndBound.jl` file where the function is defined.