Fast Low-Rank Semidefinite Programming for Embedding and Clustering

Brian Kulis[*]
Department of Computer Sciences
University of Texas at Austin
Austin, TX 78759 USA
kulis@cs.utexas.edu

Arun C. Surendran
Microsoft adCenter Labs
1 Microsoft Way
Redmond, WA 98052 USA
acsuren@microsoft.com

John C. Platt
Microsoft Research
1 Microsoft Way
Redmond, WA 98052 USA
john.platt@microsoft.com

Abstract

Many non-convex problems in machine learning such as embedding and clustering have been solved using convex semidefinite relaxations. These semidefinite programs (SDPs) are expensive to solve and are hence limited to run on very small data sets. In this paper we show how we can improve the quality and speed of solving a number of these problems by casting them as low-rank SDPs and then directly solving them using a non-convex optimization algorithm. In particular, we show that problems such as the k-means clustering and maximum variance unfolding (MVU) may be expressed exactly as low-rank SDPs and solved using our approach. We demonstrate that in the above problems our approach is significantly faster, far more scalable and often produces better results compared to traditional SDP relaxation techniques.

1 Introduction

A recent trend in machine learning research has been to use *semidefinite programming* for a number of learning tasks. Semidefinite programs (SDPs) seek to optimize a linear objective with linear constraints as well as a semidefinite constraint. The standard primal form for an SDP is:

$$\begin{aligned} \min_{Z \in R^{n \times n}} \quad & \mathrm{tr}(CZ) \\ \text{subject to} \quad & \mathrm{tr}(ZA_i) = b_i, \quad 0 \leq i \leq m \\ & Z \succeq 0, \end{aligned} \tag{1}$$

where the C and A_i matrices are given, as well as the b_i values. The constraint $Z \succeq 0$ requires that Z be positive semi-definite.

[*] Work done while at Microsoft Research

Examples of applications that use SDPs include sparse PCA [6], maximum variance unfolding [17], robust Euclidean embedding [4], minimizing normalized or ratio cuts in a graph [14, 18], kernel matrix learning [9], and many others. Typically, off-the-shelf techniques are used for solving these problems, which severely limits scalability. Many of these methods scale as $O(n^3)$ or worse, where n is the number of rows (or columns) of the semidefinite matrix. Furthermore, they require $O(n^2)$ memory to store the semidefinite matrix. Thus, they are rarely used for problems when n is more than a few thousand. Despite the computational and memory restrictions inherent in SDPs, semidefinite programming is a standard technique for solving many problems.

In many machine learning problems, e.g. in embedding via maximum variance unfolding (MVU), the original non-convex problem is transformed into a convex one by semidefinite relaxation in the original dimension of the data. Once a solution to this large SDP is found, the final solution to the original problem is obtained using low-rank embedding of the semidefinite solution. A similar process exists for SDP relaxations to clustering problems; after solving a large SDP, the final clusters are obtained using rounding, usually by finding the low rank projection of the data onto the top few eigenvectors of the SDP solution.

Since the final goal in many of these problems is the low-rank solution, it is often wasteful to solve the original full-rank problem. In this paper, we take a different approach to optimizing these objective functions. We show that, for several embedding and clustering problems, the original (non-convex) objective function can be recast directly without relaxation as a *low-rank SDP*: an SDP with an additional rank constraint. Because of the rank constraint, we can directly solve for low-rank factorizations of the semidefinite matrix, treating the problem as a non-linear optimization problem. Rather than being seduced by convexity, we appeal to nonlinear programming techniques

to solve exact reformulations of the original problems. This notion of forgoing convexity to gain speed and scalability is beginning to gain popularity in machine learning, notably in the recent work of [5].

To solve the low-rank SDP problems, we generalize a recent algorithm by Burer and Monteiro [3] that uses low-rank factorizations to optimize full-rank SDPs. We perform all computations in the low-rank space, leading to dramatic improvements in speed and memory requirements. Moreover, the results we obtain are often significantly better than the results of the standard relaxation technique. We apply this algorithm to embedding and graph clustering problems, demonstrating improved results as compared to existing methods. For MVU, we compare our algorithm to the semidefinite relaxation method, and for our clustering experiments we compare it to other state-of-the-art graph clustering algorithms.

2 Low-Rank SDPs for Clustering and Embedding

As mentioned above, the general idea is to convert a variety of problems into low-rank semidefinite programs. In Sections 2.1 and 2.2, we demonstrate how to achieve this for a clustering and an embedding problem. In Sections 2.1 and 4 we will briefly show how to extend this approach to other clustering and embedding problems.

2.1 Low-Rank SDPs for Clustering

We first consider low-rank SDPs for clustering. Specifically, we establish that the k-means objective function can be expressed equivalently as a low-rank SDP problem. Later, in section 3, we develop a general algorithm for directly optimizing low-rank SDPs. This algorithm minimizes the k-means objective function and has running time of $O(zk)$ per iteration, where z is the number of non-zero elements of the Gram matrix (or the number of edges in a graph, for graph clustering). Unlike spectral methods for optimizing this objective, there is no relaxation involved in expressing the k-means objective function as a low-rank SDP, so post-processing or rounding of the resulting solution is not required to obtain a discrete clustering of the data.

2.1.1 The k-means objective function

Given a set of data points $\{\mathbf{x}_i\}_{i=1}^n$, and the number of desired clusters, k, the k-means objective function attempts to minimize the sum of squared Euclidean distances between each point and its corresponding cluster centroid:

$$\begin{aligned} \mathcal{D}(\{\pi_c\}_{c=1}^k) &= \sum_{c=1}^{k} \sum_{\mathbf{x}_i \in \pi_c} \|\mathbf{x}_i - \mathbf{m}_c\|^2, \\ \text{where } \mathbf{m}_c &= \frac{1}{|\pi_c|} \sum_{\mathbf{x}_i \in \pi_c} \mathbf{x}_i. \end{aligned}$$

It has been established (for example, in [19]) that the above problem can be recast as a trace maximization problem, which leads to a spectral relaxation for k-means. Peng [14] introduced a different but equivalent reformulation for this objective, yielding the following optimization problem:

$$\begin{aligned} \max_{Z \in R^{n \times n}} \quad & \operatorname{tr}(KZ) \\ \text{subject to} \quad & \operatorname{tr}(Z) = k \\ & Z \geq 0 \text{ elementwise} \\ & Z\mathbf{e} = \mathbf{e} \\ & Z^2 = Z, Z = Z^T. \end{aligned} \tag{2}$$

where K is the Gram matrix (or kernel matrix) of the data points (i.e. if X is the matrix whose rows are the $\mathbf{x}_i$ vectors, then $K = XX^T$).

We can make two observations about the problem as formulated in (2). First, it is known that solving this problem is equivalent to finding a global solution to the integer programming problem for k-means [14]. Second, (2) is related to the generic SDP formulation in (1), except that the semidefinite constraint $Z \succeq 0$ is replaced by $Z^2 = Z$ and $Z = Z^T$. These constraints force all the eigenvalues of Z to be 0 or 1 (i.e., $\lambda(Z) \in \{0, 1\}$), hence this is a special form called a 0-1 SDP [14]. This integer constraint on the eigenvalues makes it intractable to directly optimize the problem in (2). As a result, relaxation methods are often employed to obtain approximations of the original problem: one can relax the constraint $Z^2 = Z$ to a weaker constraint such as $\lambda(Z) \in [0, 1]$ or $Z \succeq 0$. Depending on the relaxation, spectral methods or semi-definite programming are then employed to find a globally optimal solution to the relaxed problem. For example, Xing and Jordan [18] considered a weighted form of this objective (i.e., normalized cut), and solved it approximately via semidefinite programming using such a relaxation.

2.1.2 Equivalence to Low-Rank SDPs

We now show how we can convert (2) explicitly into a low-rank SDP problem. The first step is to replace the constraints $Z^2 = Z$ and $Z = Z^T$ with a rank constraint on Z and a positive semi-definite constraint. The following theorem explains why this rank constraint is meaningful, and will also show us how to do this recasting:

Theorem 1. *Given*

$$\begin{aligned} &a) \quad tr(Z) = k \\ &b) \quad Z \geq 0 \ \textit{elementwise} \\ &c) \quad Z\mathbf{e} = \mathbf{e} \\ &d) \quad Z = Z^T, \end{aligned}$$

then rank$(Z) = k$ *if and only if* $Z^2 = Z$.

Proof. First we show $\text{rank}(Z) = k \Rightarrow Z^2 = Z$. Since $\text{rank}(Z) = k$, Z has $n - k$ eigenvalues equal to 0. Recall that a matrix is reducible if and only if it can be put into block upper-triangular form with simultaneous permutations of rows and columns, and a square matrix that is not reducible is irreducible. If Z were irreducible, we could conclude that $Z\mathbf{e} = \mathbf{e}$ and $Z \geq 0$ imply (by the Perron-Frobenius Theorem [15]) that all but the top eigenvalue are strictly less than 1. But since there are only k non-zero eigenvalues, this would make it impossible to satisfy $\text{tr}(Z) = k$. Thus, Z is reducible, and because it is symmetric, it is permutation-similar to a block-diagonal matrix. Each block Z_i satisfies $Z_i\mathbf{e} = \mathbf{e}$, and hence each block has an eigenvalue of 1. The Perron-Frobenius theorem applies to these block-diagonal, irreducible matrices, and thus each block has the property that it has a leading eigenvalue of 1, and all other eigenvalues are strictly less than one. We must therefore have k blocks, each with a single positive eigenvalue of 1; otherwise it would be impossible to satisfy $\text{tr}(Z) = k$. Therefore, each of the k non-zero eigenvalues equals 1, so $\lambda(Z) \in \{0, 1\}$, and hence $Z^2 = Z$. In the other direction, $Z^2 = Z$ is equivalent to $\lambda(Z) \in \{0, 1\}$. Thus, $\text{tr}(Z) = k$ implies that $\text{rank}(Z) = k$. □

It follows from the above result that we can rewrite the k-means objective function in (2) as an SDP with an additional rank constraint:

$$\begin{aligned} \max_{Z \in R^{n \times n}} \quad & \text{tr}(KZ) \\ \text{subject to} \quad & \text{tr}(Z) = k \\ & Z \geq 0 \text{ elementwise} \\ & Z\mathbf{e} = \mathbf{e} \\ & \text{rank}(Z) = k \\ & Z \succeq 0. \end{aligned} \tag{3}$$

We have shown that Z permutes to a block-diagonal matrix that captures the disjoint clustering of the data and has rank k. We also know that Z is an orthogonal projection matrix, so it follows that Z can be factored as $Z = YY^T$, where Y is an $n \times k$ matrix. The entries of Y are discrete-valued and give the cluster assignments; in particular, $Y_{ij} = 0$ if point i is not in cluster j, and is equal to $1/\sqrt{k_j}$ otherwise, where k_j is the number of points in the cluster to which j is assigned. Thus, the optimal Y gives the optimal clustering for the k-means objective.

The final piece in the solution is to explicitly factor Z as a low-rank matrix $Z = YY^T$, and solve for Y instead. We later discuss a method for solving this low-rank SDP directly in Section 3.

2.1.3 Implications

Surprisingly, the solution to the continuous optimization problem (3) produces a feasible point for the 0-1 optimization problem, without requiring the 0-1 constraint. That is, the problem (3) returns a solution that encodes a discrete cluster indicator matrix. Thus, unlike standard relaxation techniques for k-means, we require no post-processing of the solution.

Furthermore, problem (3) can easily be extended to solve a number of graph partitioning problems. As discussed in Dhillon et al. [7], if we introduce a weighted form of the k-means objective function, there exists an equivalence to graph clustering objectives such as normalized cut, ratio cut, and ratio association for an appropriate choice of weights. For example, we perform graph clustering using the ratio association objective:

$$\max \sum_{i=1}^{k} \frac{\text{links}(\mathcal{V}_i, \mathcal{V}_i)}{|\mathcal{V}_i|},$$

where $\mathcal{V}_i$ refers to the set of nodes in cluster i, and $\text{links}(\mathcal{V}_i, \mathcal{V}_i)$ gives the sum of all the edges from nodes in cluster i to other nodes in cluster i. The ratio association objective is a natural generalization of k-means to graphs, since optimizing the k-means objective function over a set of vectors is equivalent to optimizing the ratio association objective over the Gram matrix of those vectors.

Additional graph clustering objectives are possible in this framework as well. For example, by introducing a cluster size constraint as discussed in Peng [14] (in particular, the constraint $\text{diag}(Y^TY) \geq \frac{1}{s}\mathbf{e}$, where s is the minimum size of a cluster), we can easily encode balancing constraints for the clusters. Thus, various clustering objective functions can be encoded as low-rank SDPs.

2.2 Low-Rank SDPs for Embedding

We now turn our attention to the problem of embedding a set of high-dimensional vectors into a lower-dimensional space, while retaining characteristics of the original data. Embedding is a natural application of low-rank semidefinite programming since the rank

constraint corresponds to the desired dimensionality of the embedding in various embedding problems.

In maximum variance unfolding, we seek to find low-dimensional representations of our input data that have maximum possible variance, while preserving local distances of the input data. The underlying optimization problem can be expressed as [17]:

$$\begin{aligned} \max_{Y \in R^{n \times r}} \quad & \sum_i \|\mathbf{y}_i\|^2 = \frac{1}{2n} \sum_{i,j} \|\mathbf{y}_i - \mathbf{y}_j\|^2 \quad (4) \\ \text{subject to} \quad & \sum_i \mathbf{y}_i = 0 \\ & \|\mathbf{y}_i - \mathbf{y}_j\|^2 = D_{ij} \ \forall (i,j) \in \mathcal{E}. \end{aligned}$$

The set $\mathcal{E}$ contains the nearest neighbors of the input data, and D_{ij} gives the distance between neighbors i and j in the input space. The above problem is non-convex, but by relaxing the problem to be convex, it can be expressed as the following SDP [17]:

$$\begin{aligned} \max_{Z \in R^{n \times n}} \quad & \operatorname{tr}(Z) \quad (5) \\ \text{subject to} \quad & \mathbf{e}^T Z \mathbf{e} = 0 \\ & Z_{ii} + Z_{jj} - 2Z_{ij} = D_{ij} \ \forall (i,j) \in \mathcal{E} \\ & Z \succeq 0. \end{aligned}$$

The standard method for computing the embedding is to solve the above SDP for Z, and then to project the data onto the first few eigenvectors of Z. Note that when projecting the data onto the top eigenvectors of Z, we are longer guaranteed that the resulting embedding satisfies the given constraints.

However, by adding an explicit rank constraint to (5) for the desired dimensionality of the embedding, it is easy to show that we are directly optimizing the original problem (4). This follows by factorizing $Z = YY^T$ and rewriting the problem in terms of Y: $\operatorname{tr}(Z) = \operatorname{tr}(YY^T) = \sum_i \|\mathbf{y}_i\|^2$. The constraint $\mathbf{e}^T Z \mathbf{e} = 0$ maps to $\sum_i \mathbf{y}_i = 0$ and the distance constraints $Z_{ii} + Z_{jj} - 2Z_{ij} = D_{ij}$ map to $\mathbf{y}_i^T \mathbf{y}_i + \mathbf{y}_j^T \mathbf{y}_j - 2\mathbf{y}_i^T \mathbf{y}_j = \|\mathbf{y}_i - \mathbf{y}_j\|^2$. Thus, adding the constraint $\operatorname{rank}(Z) = r$ to (5) leads to a low-rank SDP that is equivalent to the original MVU problem. Furthermore, because we work directly in the embedded space, convergence of the low-rank SDP guarantees that all constraints are satisfied, whereas projecting a solution to (5) to a lower dimension can cause the constraints to be violated.

3 Solving the Low-Rank Semidefinite Programs

In the previous sections we showed how we can formulate two problems as low-rank SDPs. In this section, we discuss a general technique for optimizing the low-rank formulations.

Recently, Burer and Monteiro [3] considered a method for solving the standard (full-rank) SDP (1) using a change of variables. Since there exists an optimal solution to (1) whose rank satisfies $r(r+1)/2 \leq m$ (proven independently by Barvinok [1] and Pataki [13]), Burer and Monteiro introduce a change of variables $Z = YY^T$, where $Y \in R^{n \times r}$ and r satisfies the inequality above. Rewriting the optimization over Y, the problem can be expressed as a quadratic program with (generally non-convex) quadratic constraints:

$$\begin{aligned} \max_{Y \in R^{n \times r}} \quad & \operatorname{tr}(Y^T C Y) \quad (6) \\ \text{subject to} \quad & \operatorname{tr}(Y^T A_i Y) = b_i, \quad 0 \leq i \leq m. \end{aligned}$$

A key aspect of this formulation is that it avoids the positive semidefinite constraint, since $Z = YY^T$ automatically enforces the constraint. A disadvantage to this formulation is that the problem has gone from convex to non-convex, so we have given up the guarantee of converge to the globally optimal solution.

Burer and Monteiro proposed to solve this quadratic programming problem using the augmented Lagrangian technique. The augmented Lagrangian for (6) is given by:

$$\begin{aligned} L(Y, \lambda, \sigma) \quad = \quad & \operatorname{tr}(Y^T C Y) - \sum_{i=1}^m \lambda_i (\operatorname{tr}(Y^T A_i Y) - b_i) \\ + \quad & \frac{\sigma}{2} \sum_{i=1}^m (\operatorname{tr}(Y^T A_i Y) - b_i)^2. \end{aligned}$$

The third term is the penalty term for the augmented Lagrangian, introducing a variable σ. To minimize the augmented Lagrangian, we alternate minimizing this function with respect to Y and with respect to λ and σ. To optimize with respect to Y, Burer and Monteiro use a limited-memory BFGS algorithm, which has the advantage of maintaining $O(nr)$ memory overhead, but also has the speed of a quasi-Newton method. This requires computation of the gradient of L with respect to Y, $\nabla_Y L(Y, \lambda, \sigma)$, given by:

$$2CY - 2 \sum_{i=1}^m (\lambda_i - \sigma(\operatorname{tr}(Y^T A_i Y) - b_i)) A_i Y.$$

To update λ and σ (assuming equality constraints), the standard augmented Lagrangian technique updates apply: set $\lambda_i = \lambda_i - \sigma(\operatorname{tr}(Y^T A_i Y) - b_i)$, and update σ by a multiplicative factor. On several test problems, Burer and Monteiro demonstrated that this technique always empirically converges to the globally optimal solution, despite the fact that the quadratic programming problem is non-convex. Furthermore,

they showed that their algorithm is significantly faster than existing SDP solvers.

We propose to apply the same technique to our low-rank SDPs.[1] We differ from Burer and Monteiro in that we solve SDPs with explicit rank constraints, so the size of the matrix Y is precisely determined by the rank constraint. Note that the rank of our solution does not in general satisfy the inequality $r(r+1)/2 \leq m$; however, empirically this does not in general lead to poor solutions.

As an example, the low-rank SDP for k-means discussed earlier (3) can be recast as an optimization problem over the factorization $Z = YY^T$, where Y is $R^{n \times k}$:

$$\begin{aligned} \max_{Y \in R^{n \times k}} \quad & \operatorname{tr}(Y^T K Y) \\ \text{subject to} \quad & \|Y\|_F^2 = k \\ & YY^T \mathbf{e} = \mathbf{e} \\ & Y \geq 0 \text{ elementwise,} \end{aligned} \tag{7}$$

where we have replaced the constraint $YY^T \geq 0$ elementwise with the stronger constraint $Y \geq 0$ elementwise, which is easier to enforce in the low-rank setting. Any feasible solution to this problem yields a discrete cluster indicator matrix Y.

The main computational cost involved in running the low-rank SDP algorithm is in computing the Lagrangian and the gradient of the Lagrangian. In the case of k-means, the most expensive step in computing the Lagrangian is the computation of $\operatorname{tr}(Y^T K Y)$, which can be done in $O(zk)$ time, where z is the number of non-zero entries of the matrix K. For computing the gradient of the Lagrangian, the most expensive step is in forming the matrix KY, which can also be performed in $O(zk)$ time. Thus, the function and gradient evaluations for k-means can both be performed in $O(zk)$ time.

3.1 Inequality Constraints for Low-Rank SDPs

Another departure from Burer and Monteiro's algorithm is that our low-rank SDPs contain inequality constraints, which are not readily handled by [3]; thus, we now generalize the method to handle inequality constraints. There are several options for handling linear inequality constraints in the low-rank SDP formulation; for example, one possibility is to use a log barrier function. However, it turns out that for the augmented Lagrangian approach, it is easier to solve directly for updates of the Lagrange multipliers for both equality and inequality constraints.

By treating a constraint $\operatorname{tr}(Y^T A_i Y) \geq b_i$ as the constraints $\operatorname{tr}(Y^T A_i Y) = b_i + s_i$ and $s_i \geq 0$, we can explicitly solve for the updates of the λ_i terms. For inequality constraints, this yields

$$\lambda_i = \max(\lambda_i - \sigma(\operatorname{tr}(Y^T A_i Y) - b_i), 0).$$

This agrees with the requirement that $\lambda_i \geq 0$ for Lagrange multipliers for inequality constraints. Computing the Lagrangian and the gradient of the Lagrangian are also straightforward for inequality constraints. The second and third terms of the Lagrangian change: for each constraint i, the Lagrangian term $-\lambda_i(\operatorname{tr}(Y^T A_i Y) - b_i) + \frac{\sigma}{2}(\operatorname{tr}(Y^T A_i Y) - b_i)^2$ is unchanged if $\sigma(\operatorname{tr}(Y^T A_i Y) - b_i) \leq \lambda_i$. Otherwise, this term becomes $-\lambda_i^2/2\sigma$. Similarly, in the computation of the gradient of the Lagrangian, we only contribute the term $2(\lambda_i - \sigma(\operatorname{tr}(Y^T A_i Y) - b_i))A_i Y$ if $\sigma(\operatorname{tr}(Y^T A_i Y) - b_i) \leq \lambda_i$. Otherwise, nothing is added to the gradient. It is easy to extend this idea to the actual elementwise constraint $Y \geq 0$ that we use in our k-means formulation in (7).

See [12] for further information on the augmented Lagrangian technique.

4 Related Problems

Min balanced cut embedding (discussed by Lang [10] and others) is another important embedding technique. It avoids the so-called "octopus" structure that exists in many embedding methods, especially when dealing with power-law graphs. This problem is demonstrated in [10]: typically, embedding algorithms embed the points densely close to the origin, with many points projected outward in various directions like tentacles. By adding constraints that both embeds the points onto a sphere and promotes balanced cuts, this problem is avoided.

The SDP for min balanced cut is expressed as follows:

$$\begin{aligned} \min_{Z \in R^{n \times n}} \quad & \operatorname{tr}(LZ) \\ \text{subject to} \quad & \operatorname{diag}(Z) = \mathbf{e} \\ & \mathbf{e}^T Z \mathbf{e} = 0. \end{aligned} \tag{8}$$

The $\operatorname{diag}(Z) = \mathbf{e}$ constraint places all the data points onto the unit sphere, while the $\mathbf{e}^T X \mathbf{e} = 0$ constraint forces the data to be mean-centered at 0. The matrix L is the Laplacian of the input graph corresponding to the points. Unlike the earlier cases of k-means and maximum variance unfolding, adding a rank constraint to the SDP does not correspond to solving

[1] In some cases such as sparse PCA, the limited-memory BFGS algorithm cannot be employed; a subgradient technique would be more appropriate for optimizing the augmented Lagrangian. ADMM ?

an exact form of the underlying objective function, but we can still apply the low rank transformation $Z = YY^T$ to write problem (8) as a quadratic program with quadratic constraints (QCQP) in Y, leading to a significantly more scalable algorithm for min balanced cut embedding.

Finally, we can apply low-rank SDPs to the problem of finding sparse principal components. A recent paper by d'Aspremont et al. [6] expressed the sparse PCA problem for a single component as:

$$\begin{aligned} \max_{\mathbf{y}} \quad & \mathbf{y}^T A \mathbf{y} \\ \text{subject to} \quad & \|\mathbf{y}\|^2 = 1 \\ & \text{card}(\mathbf{y}) \leq k. \end{aligned}$$

The function card() refers to the cardinality of $\mathbf{y}$, i.e. the number of non-zero elements. Since $\mathbf{y}^T A \mathbf{y} = \text{tr}(A\mathbf{y}\mathbf{y}^T)$, this problem can be expressed as a low-rank SDP:

$$\begin{aligned} \max_{Z \in R^{n \times n}} \quad & \text{tr}(AZ) \\ \text{subject to} \quad & \text{tr}(Z) = 1 \\ & \text{card}(Z) \leq k^2 \\ & \text{rank}(Z) = 1 \\ & X \succeq 0. \end{aligned}$$

Solving this problem is difficult because of the cardinality and rank constraints. The authors of [6] relax the sparsity constraint, instead imposing $\mathbf{e}^T|Z|\mathbf{e} \leq k$, where $|Z|$ is a matrix whose entries correspond to the absolute values of the entries of Z. They also drop the rank constraint, instead projecting the solution of the full-rank SDP to the principal eigenvector of Z.

By treating this problem as a low-rank SDP, we can directly optimize $\mathbf{y}$ through the low-rank substitution $Z = \mathbf{y}\mathbf{y}^T$; this avoids relaxing the rank constraint. Furthermore, we can enforce a simple constraint directly on $\mathbf{y}$, such as $\|\mathbf{y}\|_1 \leq k$, to promote a sparse solution.

5 Experimental Results

5.1 Methodology

We first compare results of the low-rank SDP solver against a state-of-the-art interior point SDP solver, DSDP [2]. We chose to compare against DSDP because, as compared to other freely-available solvers, it performed the best on sparse SDP benchmark experiments [11]. By comparing the low-rank solver with DSDP, we can determine 1) how the computational times compare between a traditional SDP solver and a low-rank SDP solver, and 2) how the accuracy compares between these solvers. Because DSDP is limited to small data sets, we run large-scale experiments using only the low-rank solver.

Data set	Number of data points
Iris	150
Pima	768
Vehicle	846
Sonar	208
Segment	2310
Satimage	2000
Pendigits	1092

Table 1: Data sets used for small-scale maximum variance experiments

Data set	Number of nodes	Number of Edges
add20	2395	7462
data	2851	15093
uk	4824	6837
add32	4960	9462
rajat06	10922	18061
crack	10240	30380
whitaker3	9880	28989

Table 2: Graphs used for graph clustering experiments

We tested the two solvers on SDP problems discussed earlier: maximum variance unfolding and graph clustering. Due to space constraints, we cannot present results on sparse PCA or min balanced cut embedding. For maximum variance unfolding, we fixed the number of neighbors to be 5, and generated the constraints from these 5 nearest neighbors. In a recent paper [16], Weinberger et al. discuss a procedure for consolidating constraints. By removing redundant constraints, the authors demonstrate how to substantially improve the running times of the SDP. We did not perform such a step in our algorithm, though doing so would almost certainly result in further improvements of the running times of our algorithm.

For graph clustering, we optimize the ratio association objective and compare results of our low-rank solver against various existing state-of-the-art methods: kernel k-means, the spectral relaxation, and the multilevel method developed in [8]. For postprocessing the solutions of the relaxation algorithms, we used standard k-means. Note that since SDP relaxation methods for k-means do not scale to graphs larger than a few hundred nodes, comparisons to SDP relaxations were not possible for these experiments.

Default settings were applied to DSDP, and for the low-rank SDP, we used a multiplicative factor of 1.5 for σ and uniform initialization for Y.

5.2 Data sets

Tables 1 and 2 lists the small data sets used in our experiments. Table 1 consists of vector-based data sets used for maximum variance unfolding, and Table 2 consists of graph-based data sets used for graph clustering experiments.

5.3 Results

In Table 3, we present results of maximum variance unfolding. For these experiments, we ran DSDP on each of the data sets, recording the time taken to converge, the final variance (trace) value, and the maximum infeasibility (i.e., the amount of feasibility for the constraint that is *most* violated) first after running just DSDP, then after projecting to the top 10 eigenvectors (DSDP10). We similarly ran our low-rank solver on the same data sets, and recorded these values. However, for the low-rank solver, we also ran over varying ranks: 5, 10, and 50.

Several interesting observations can be made about these results. First, the dashed lines represent DSDP failing to converge (due to memory problems, generally). This happened on two of the data sets (and many larger data sets not presented here), even though none of these data sets is particularly large. Second, even when DSDP converged, it did not always converge to the optimal solution. In fact, it converged to a feasible solution that was better than all low-rank solutions on only one of the data sets (sonar). On other data sets, at least one of the low-rank solutions produced higher variance. Third, the running times for DSDP are generally much slower than the low-rank SDP program. DSDP was hundreds of times slower than the low-rank SDP for $r = 5$ or $r = 10$ on several data sets. Fourth, the solutions to DSDP generally have better feasibility properties in terms of maximum infeasibility and average infeasibility (not presented in the table), but the low-rank SDP solutions generally have maximum infeasibility lower than 1×10^{-3}, which is reasonable for machine learning applications. Fifth, after projecting down to 10 dimensions, generally the variance given by DSDP is close to that of the unprojected variance, except in the case of vehicle, where the variance went from 1.4 to 1.3 and the maximum infeasibility jumped from 10^{-11} to 10^{-3}. Finally, overall it seems that for $r = 10$, we get a good quality/speed tradeoff for the low-rank algorithm. In fact, sometimes the $r = 10$ solution is the best (for example, in pendigits), and generally the solution for $r = 10$ is nearly as good as the higher rank solutions. On the other hand, for $r = 5$, the solution was substantially suboptimal in some cases (for example, on pima).

Now we focus on graph clustering. In this experiment, we clustered the graphs given in Table 2 by optimizing the ratio association objective function. We compared four methods: baseline kernel k-means with random initialization, the spectral relaxation technique (with k-means postprocessing), Graclus [8] (a kernel k-means approach that is enhanced using multilevel techniques), and the low-rank SDP. We could not run any stronger relaxations than the spectral relaxation due to the size of the graphs (semidefinite relaxations for k-means are generally limited to only a few hundred points).

Results are given in Table 4. On four of the seven graphs, the low-rank SDP yields the highest ratio association score. Both the low-rank SDP method and Graclus consistently outperformed the spectral method

Data set	Run	Trace Val	Max. Inf.	Time
Iris	DSDP	8.834e-1	5.007e-9	184
	DSDP10	8.834e-1	7.648e-8	184
	LR5	1.206e+0	3.257e-5	15
	LR10	1.220e+0	3.381e-5	17
	LR50	1.227e+0	3.285e-5	80
Pima	DSDP	1.098e+1	5.292e-12	25109
	DSDP10	1.098e-4	1.813e-6	25109
	LR5	8.512e+0	4.966e-4	294
	LR10	1.103e+1	2.261e-4	153
	LR50	1.113e+1	2.733e-4	744
Vehicle	DSDP	1.441e+1	1.410e-11	40851
	DSDP10	1.300e+1	5.387e-3	40851
	LR5	1.636e+1	2.098e-4	120
	LR10	1.681e+1	1.543e-4	167
	LR50	1.665e+1	1.493e-4	978
Sonar	DSDP	6.476e+1	2.951e-12	71
	DSDP10	6.476e+1	7.648e-8	71
	LR5	5.298e+1	1.666e-2	24
	LR10	6.473e+1	8.369e-6	7
	LR50	6.471e+1	3.700e-6	40
Segment	DSDP	—	—	—
	DSDP10	—	—	—
	LR5	1.300e+2	7.530e-3	450
	LR10	1.212e+2	1.167e-3	1040
	LR50	1.259e+2	2.044e-3	4791
Satimage	DSDP	—	—	—
	DSDP10	—	—	—
	LR5	5.378e+0	6.141e-5	233
	LR10	5.443e+0	3.162e-5	281
	LR50	5.437e+0	3.851e-5	1366
Pendigits	DSDP	1.005e+4	3.051e+1	1761
	DSDP10	8.995e+3	2.881e+1	1761
	LR5	3.003e+0	5.615e-5	137
	LR10	3.159e+0	6.796e-5	159
	LR50	3.142e+0	6.531e-5	1092

Table 3: Results for maximum variance unfolding

Graph	Kernel k-means	Spectral	Graclus	Low-Rank SDP
add20	54.72	72.44	20.71	**75.27**
add32	11.79	17.94	15.19	**18.94**
uk	6.29	11.29	11.22	**11.43**
rajat06	9.69	12.04	**13.08**	12.64
data	33.39	44.13	41.06	**44.71**
crack	17.70	21.92	**23.37**	22.72
whitaker3	17.51	23.04	**23.29**	23.13

Table 4: Ratio association results on graph clustering experiments ($k = 4$)

(except on add20, where spectral outperformed Graclus). Though not given here, we also ran experiments for minimizing the normalized cut; however, we found that Graclus outperformed the low-rank SDP in this case. As future work, we would like to explore multilevel methods using the low-rank SDP algorithm, which may make it more competitive for minimizing the normalized cut.

6 Conclusions

In this paper, we have studied low-rank semidefinite programming. We demonstrated how we can convert various clustering and embedding problems such as k-means and maximum variance unfolding into low-rank SDPs without any convex relaxations. We then directly optimized these low-rank SDP formulations using non-convex algorithms, performing computations directly in the low-rank space. We showed that our approach is significantly faster, better and more scalable than standard SDP approaches for embedding. We also showed that for graph clustering, our approach matches or outperforms state-of-the-art techniques.

References

[1] A. Barvinok. Problems of distance geometry and convex properties of quadratic maps. *Discrete Computational Geometry*, 13:189–202, 1995.

[2] S. J. Benson, Y. Ye, and X. Zhang. Solving large-scale sparse semidefinite programs for combinatorial optimization. *SIAM Journal on Optimization*, 10(2):443–461, 2000.

[3] S. Burer and R. D. C. Monteiro. A nonlinear programming algorithm for solving semidefinite programs via low-rank factorization. *Mathematical Programming (Series B)*, 95:329–357, 2003.

[4] L. Cayton and S. Dasgupta. Robust euclidean embedding. In *Proc. of the 23rd Intl. Conf. on Machine Learning*, 2006.

[5] R. Collobert, F. Sinz, J. Weston, and L. Bottou. Trading convexity for scalability. In *Proc. 23 ICML Conference*, 2006.

[6] A. d'Aspremont, L. El Ghaoui, M. I. Jordan, and G. R. G. Lanckriet. A direct formulation for sparse pca using semidefinite programming. *SIAM Review*, 2006.

[7] I. Dhillon, Y. Guan, and B. Kulis. A unified view of kernel k-means, spectral clustering and graph cuts. Technical Report TR-04-25, University of Texas at Austin, 2004.

[8] I. Dhillon, Y. Guan, and B. Kulis. A fast kernel-based multilevel algorithm for graph clustering. In *11th ACM SIGKDD Conference*, 2005.

[9] G. R. G. Lanckriet, N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan. Learning the kernel matrix with semidefinite programming. *Journal of Machine Learning Research*, 5:27–72, 2004.

[10] K. Lang. Fixing two weaknesses of the spectral method. In *Neural Information Processing Systems 18*, 2005.

[11] H. Mittelmann. Several SDP-codes on sparse and other SDP problems. `http://plato.asu.edu/ftp/sparse_sdp.html`, Jul 2006.

[12] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 1999.

[13] G. Pataki. On the rank of extreme matrices in semidefinite programs and the multiplicity of optimal eigenvalues. *Mathematics of Operations Research*, 23:339–358, 1998.

[14] J. Peng. 0-1 semidefinite programming for spectral clustering: Modeling and approximation. Technical Report 2005/12, Advanced Optimization Laboratory, McMaster University, 2005.

[15] S. U. Pillai, T. Suel, and S. Cha. The Perron-Frobenius theorem. *IEEE Signal Processing Magazine*, 22(2):62–75, March 2005.

[16] K. Q. Weinberger, B. D. Packer, and L. K. Saul. Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In *Proc. of the 10th Intl. Workshop and Artificial Intelligence and Statistics*, 2005.

[17] K. Q. Weinberger, F. Sha, and L. K. Saul. Learning a kernel matrix for nonlinear dimensionality reduction. In *Proc. of the 21st Intl. Conf. on Machine Learning*, 2004.

[18] E. P. Xing and M. I. Jordan. On semidefinite relaxations for normalized k-cut and connections to spectral clustering. Technical Report CSD-03-1265, University of California at Berkeley, 2003.

[19] H. Zha, C. Ding, M. Gu, X. He, and H. Simon. Spectral relaxation for k-means clustering. In *Neural Information Processing Systems 14*, 2001.