Computational Enhancements in Low-Rank Semidefinite Programming

Samuel Burer* Changhui Choi†

July 12, 2004

Abstract

We discuss computational enhancements for the low-rank semidefinite programming algorithm, including the extension to block semidefinite programs, an exact linesearch procedure, and a dynamic rank reduction scheme. A truncated Newton method is also introduced, and several preconditioning strategies are proposed. Numerical experiments illustrating these enhancements are provided.

Keywords: Semidefinite programming, low-rank matrices, vector programming, nonlinear programming, numerical experiments.

1 Introduction

In this paper, we present ideas and techniques for efficiently solving large-scale semidefinite programs (SDPs). To state the standard-form SDP problem, let $\mathcal{S}^n$ denote the set of $n \times n$ symmetric matrices, and for a given sequence $n_1, \ldots, n_\ell$ of positive integers summing to n, let $\mathcal{S}^{n_1 \cdots n_\ell}$ denote the subset of $\mathcal{S}^n$ consisting of block-diagonal matrices, where the blocks are of sizes $n_1, \ldots, n_\ell$, respectively. In addition, let $\mathcal{S}^n_+ \subset \mathcal{S}^n$ denote the subset of positive semidefinite matrices, and similarly let $\mathcal{S}^{n_1 \cdots n_\ell}_+ \subset \mathcal{S}^{n_1 \cdots n_\ell}$ denote those matrices with all blocks positive semidefinite; note that $\mathcal{S}^{n_1 \cdots n_\ell}_+ = \mathcal{S}^n_+ \cap \mathcal{S}^{n_1 \cdots n_\ell}$. Then, given a cost matrix $C \in \mathcal{S}^{n_1 \cdots n_\ell}$ as well as constraint matrices $A_i \in \mathcal{S}^{n_1 \cdots n_\ell}$ and right-hand sides b_i, for $i = 1, \ldots, m$, the primal SDP is stated as

$$\min\{C \bullet X : \mathcal{A}(X) = b,\ X \in \mathcal{S}^{n_1 \cdots n_\ell}_+\}, \tag{1}$$

and its dual is

$$\max\{b^T y : \mathcal{A}^*(y) + S = C,\ S \in \mathcal{S}^{n_1 \cdots n_\ell}_+\}, \tag{2}$$

where the symbol $\bullet$ denotes the inner product for matrices, i.e, $P \bullet Q := \text{trace}(P^T Q)$, $[\mathcal{A}(X)]_i := A_i \bullet X$ for all $i = 1, \ldots, m$, and $\mathcal{A}^*(y) := \sum_{i=1}^m y_i A_i$. We assume that both (1)

*Department of Management Sciences, University of Iowa, Iowa City, IA 52242-1000, USA. (Email: samuel-burer@uiowa.edu). This research was supported in part by NSF Grant CCR-0203426.

†Department of Management Sciences, University of Iowa, Iowa City, IA 52242-1000, USA. (Email: changhui-choi@uiowa.edu). This research was supported in part by NSF Grant CCR-0203426.

and (2) are feasible and that strong duality holds, i.e., there exist optimal solutions X^* and (y^*, S^*) such that $X^* \bullet S^* = 0$.

Interior-point methods for solving (1) and (2) to a desired accuracy in polynomial time have been studied extensively, and there are currently several high quality implementations of IPMs publicly available [4, 3, 18, 30, 34, 33]. These implementations are notable for their accuracy and robustness on many problem classes, but they suffer from some inherent bottlenecks that limit their performance on large-scale SDPs, namely computation with and storage of the generically dense matrices X and M, where X is as in (1) and M represents the $m \times m$ Schur complement matrix arising in the calculation of the Newton direction in each iteration.

To overcome some of these bottlenecks, researchers have also considered alternative approaches to the classical IPMs, including: (i) modified IPMs in which the Newton direction is calculated using the method of preconditioned conjugate gradients [21, 25, 12, 32, 31, 6]; and (ii) first-order nonlinear programming algorithms [8, 9, 10, 13, 15, 16]. As a result, a number of SDPs that were unreachable by IPMs have been solved. It is important to note that, in general, the current implementations of (i) and (ii) have a difficult time solving (1) and (2) to high accuracy since they are not able to deal with the inevitable ill-conditioning encountered near optimality. Particularly in the case of (i), constructing good preconditioners is a formidable challenge since M is dense and unstructured. Moreover, computational experience with (i) and (ii) has mostly involved SDP relaxations of combinatorial optimization problems. It is unclear if the benefits of (i) and (ii) extend to more general classes of problems.

In this paper, we attempt to address issues of robustness and accuracy in first-order algorithms for (1) and (2). More specifically, we consider how to extend and improve the so-called "low-rank" semidefinite programming algorithm of Burer and Monteiro [8, 7]. The main idea behind this algorithm is the following result of Pataki [27] and Barvinok [2]:

Theorem 1.1 *Suppose $\ell = 1$, i.e., there is a single block in (1), and let $\bar{X}$ be an extreme point of (1). Then $\bar{r} = rank(\bar{X})$ satisfies $\bar{r}(\bar{r}+1)/2 \leq m$.*

Since at least one optimal solution of (1) is an extreme point, Theorem 1.1 implies that, when $\ell = 1$, we may restrict our search for an optimal solution to those X having rank $(X) \leq r$, where $r := \min\{\bar{r} : \bar{r}(\bar{r}+1)/2 \geq m\}$; note that $r \approx \sqrt{2m}$. Thus, (1) is equivalent to the nonlinear program

$$\min\{C \bullet RR^T : \mathcal{A}(RR^T) = b,\ R \in \Re^{n \times r}\}. \tag{3}$$

Because r is typically much smaller than n (especially when m is small), this change of variables allows one to avoid storing the dense $n \times n$ matrix X. In addition, a first-order approach for solving (3) is attractive when the data matrices C, A_i are sparse (or well structured) because the gradients of the objective and constraints can be computed taking advantage of this structure. In [8], Burer and Monteiro propose an augmented Lagrangian algorithm (see [26]) for solving (3) which uses the limited memory BFGS (L-BFGS) approach with strong Wolfe-Powell (WP) linesearch, and in [7], they prove the convergence of the algorithm to a global optimal solution even in the face of the nonconvexity of (3). In both papers, they present strong computational results on some large-scale SDP relaxations of combinatorial optimization problems.

Using the algorithm of [8, 7] as our starting point, we discuss several simple but powerful enhancements to the algorithm. In particular, it is unclear at first glance how to extend the low-rank algorithm to SDPs having multiple blocks, i.e., those with $\ell > 1$. In Section 2, we propose a simple extension that maintains the practical benefits and theoretical properties of the $\ell = 1$ case. In addition, we illustrate how the strong WP linesearch may require excessive function and gradient evaluations by detailing an exact line search procedure that costs just three function and zero gradient evaluations. Moreover, it has been observed in practice that the actual rank of an optimal solution is often much smaller than the theoretically maximum rank satisfied by extreme points. We outline a practical procedure for identifying and exploiting this smaller rank algorithmically.

In Section 3, we propose a variant of the truncated Newton method as an alternative to the L-BFGS approach. In contrast to the L-BFGS approach, which attempts to incorporate a reasonable amount of implicit second-order information, the truncated Newton method uses this information explicitly. We develop formulas for the Hessian of the augmented Lagrangian function, highlight its special structure, and show how Hessian-vector products can be computed efficiently.

In Sections 2 and 3, we also provide computational results on a collection of large-scale SDPs representing a number of different problem classes. Although the computational results do not suggest that one particular version of the low-rank algorithm is best for all problem instances, they illustrate the ability of the low-rank algorithm to solve a variety of large-scale block SDPs.

In Section 4, we discuss some progress towards solving (1) and (2) to high accuracy using the truncated Newton method. We propose several preconditioning strategies and explore their relative advantages and disadvantages on some medium-scale problems. It is important to note that the preconditioning strategies are made possible by the special structure of the Hessian in the low-rank algorithm, which is in contrast to preconditioning the Schur complement matrix M in IPMs.

Finally, in Section 5, we conclude with a few final remarks.

2 Some Simple Enhancements

In this section, we discuss several simple, but powerful, enhancements to the low-rank algorithm of Burer and Monteiro [8, 7]. Computational results illustrating the improvements are also provided.

2.1 Extension to Block SDPs

As mentioned in the introduction, the low-rank algorithm has been introduced for the case $\ell = 1$ based on Theorem 1.1, but many classes of SDPs have multiple blocks. We would like to apply the generic change of variables "$X = RR^T$" to each block of the variable X in (1), but it is unclear what rank to choose for each block (or equivalently, what number of columns to choose for "R"). More specifically, we will denote the k-th block of X by X_k, for $k = 1, \ldots, \ell$ and will substitute $X_k = R_k R_k^T$, where $R_k \in \Re^{n_k \times r_k}$. Our concern is how to choose r_k.

Pataki [27] provides the following generalization of Theorem 1.1.

Theorem 2.1 *Let $\bar{X}$ be an extreme point of (1) such that the rank of $\bar{X}_k$ is $\bar{r}_k$, for $k = 1, \ldots, \ell$. Then $\sum_{k=1}^{\ell} \bar{r}_k(\bar{r}_k + 1)/2 \leq m$.*

It is instructive to observe that this theorem generalizes the well-known theorem of linear programming (LP), which bounds the number of nonzeros of an extreme point by m. This can be seen by simply noting that (1) is an LP when $n_1 = \cdots = n_\ell = 1$. Theorem 2.1, however, does not immediately suggest how to generalize the low-rank algorithm to multiple blocks since it only bounds the ranks $r_1, \ldots, r_\ell$ in aggregate, instead of individually.

The following proposition, which can be seen as applying Theorem 1.1 separately to each block, provides the individual bounds that allow us to choose r_k. The proposition uses the number

$$m_k := |\{i : [A_i]_k \neq 0\}|, \tag{4}$$

which counts the number of constraints that explicitly involve the k-th block of X.

Proposition 2.2 *Let $\bar{X}$ be an extreme point of (1) such that the rank of $\bar{X}_k$ is $\bar{r}_k$, for $k = 1, \ldots, \ell$. Then $\bar{r}_k(\bar{r}_k + 1)/2 \leq m_k$.*

Proof. The proposition follows easily by Theorem 1.1 and the fact that $\bar{X}_k$ is an extreme point of the single-block SDP gotten from (1) by considering the blocks X_j, $j \neq k$, to be fixed at the values $\bar{X}_j$. ∎

In accordance with Proposition 2.2, we choose

$$r_k := \min\{\bar{r}_k : \bar{r}_k(\bar{r}_k + 1)/2 \geq m_k\}. \tag{5}$$

To express the reformulation of (1) under the low-rank change of variables, we define $r := r_1 + \cdots + r_\ell$ and also introduce a certain subset $\Re^{(n_1 \times r_1) \cdots (n_\ell \times r_\ell)}$ of the space $\Re^{n \times r}$ of rectangular $n \times r$ matrices. $\Re^{(n_1 \times r_1) \cdots (n_\ell \times r_\ell)}$ consists of all "block-diagonal" matrices, where the blocks are of sizes $n_1 \times r_1$, $\ldots$, $n_\ell \times r_\ell$, respectively, and are positioned in a diagonal pattern. Then (1) is equivalent to

$$\min\{C \bullet RR^T : \mathcal{A}(RR^T) = b,\ R \in \Re^{(n_1 \times r_1) \cdots (n_\ell \times r_\ell)}\}, \tag{6}$$

and (6) will serve as the basis of our low-rank algorithm.

We mention briefly that one can certainly apply different strategies other than factoring each and every block as $X_k = R_k R_k^T$. One idea would be to apply the low-rank factorization to each block where r_k is significantly smaller than n_k, i.e., where one could expect a substantial savings from the change of variables. The remaining constraints $X_k \in \mathcal{S}_+^{n_k}$ could be handled using techniques from IPMs, for example. (In fact, we mention some preliminary numerical experiments with this approach in Section 5.) For uniformity in our approach and for clarity in the exposition, however, we have decided to apply the low-rank idea to each block, even if r_k is close to n_k.

2.2 Exact Linesearch

When applying the augmented Lagrangian approach to (6), the main computational effort is spent in solving unconstrained problems of the form $\min\{\mathcal{L}(R) : R \in \Re^{(n_1 \times r_1)\cdots(n_\ell \times r_\ell)}\}$, where

$$\mathcal{L}(R) := C \bullet RR^T + y^T(b - \mathcal{A}(RR^T)) + \frac{\sigma}{2}\|b - \mathcal{A}(RR^T)\|^2, \tag{7}$$

$y \in \Re^m$ is the current dual multiplier, and $\sigma > 0$ is the current penalty parameter. Both y and σ can be considered fixed in the minimization of $\mathcal{L}(R)$. As discussed in the introduction, Burer and Monteiro [8] employ a L-BFGS approach with strong WP linesearch for the minimization.

Let $D \in \Re^{(n_1 \times r_1)\cdots(n_\ell \times r_\ell)}$ be a descent direction for (7) produced by the L-BFGS approach. A quick summary of the WP linesearch in this context is that it performs an inexact minimization of the one-dimensional function $f(\alpha) := \mathcal{L}(R + \alpha D)$ over $\alpha \geq 0$, obtaining $\bar{\alpha}$, so that the next point $\bar{R} := R + \bar{\alpha}D$ satisfies a certain curvature condition with R, which in turn allows subsequent descent directions to be defined by the L-BFGS approach. The linesearch is performed by calculating $f(\alpha)$ and $f'(\alpha)$ for various test values of α, which require the calculation of $\mathcal{L}(R + \alpha D)$ and $\nabla\mathcal{L}(R + \alpha D)$. As a result, the efficiency of the linesearch corresponds to the number of required function and gradient evaluations.

Suppose that a maximum stepsize $\alpha_{\max}$ has been specified for the linesearch. Since $f(\alpha)$ is a one-dimensional function, it can theoretically be minimized over $[0, \alpha_{\max}]$ by calculating its critical points in $[0, \alpha_{\max}]$ and comparing the associated function values as well as $f(0)$ and $f(\alpha_{\max})$. Calculating the critical points is too costly for a general minimization, but in the case of $f(\alpha)$, which is a quartic polynomial since $\mathcal{L}$ is quartic, there are at most three critical points, which can be quickly and easily calculated.

In our implementation, we calculate the critical points using the publicly available GNU Scientific Library [14], which simply requires the coefficients of the cubic polynomial $f'(\alpha)$. It is straightforward to verify that

$$\begin{aligned} f(\alpha) &= a\,\alpha^4 + b\,\alpha^3 + c\,\alpha^2 + d\,\alpha + e, \\ f'(\alpha) &= 4\,a\,\alpha^3 + 3\,b\,\alpha^2 + 2\,c\,\alpha + d, \end{aligned}$$

where

$$\begin{aligned} a &:= \frac{\sigma}{2}\|q_2\|^2, \\ b &:= \sigma\, q_1^T q_2, \\ c &:= p_2 - (y + \sigma q_0)^T q_2 + \frac{\sigma}{2}\|q_1\|^2, \\ d &:= p_1 - (y + \sigma q_0)^T q_1, \\ e &:= p_0 + y^T q_0 + \frac{\sigma}{2}\|q_0\|^2 \end{aligned}$$

and

$$\begin{aligned}
p_0 &:= C \bullet RR^T, \\
p_1 &:= C \bullet (RD^T + DR^T), \\
p_2 &:= C \bullet DD^T, \\
q_0 &:= b - \mathcal{A}(RR^T), \\
q_1 &:= \mathcal{A}(RD^T + DR^T), \\
q_2 &:= \mathcal{A}(DD^T).
\end{aligned}$$

Note that $e = \mathcal{L}(R)$. It is reasonable to assume that p_0 and q_0 have been computed as a byproduct of computing $\mathcal{L}(R)$, and so the main work in computing the coefficients of $f'(\alpha)$ are in computing p_1, p_2, q_1, and q_2. It is not difficult to see that, in turn, these can be calculated in time less than three evaluations of $\mathcal{L}$.

As a result, we can expect the exact linesearch approach, which also guarantees the curvature condition necessary for L-BFGS, to be more efficient than the WP linesearch if the WP linesearch performs work totaling more than the equivalent of three function evaluations. Although we do not provide specific measurements and counts comparing the two linesearch methods, we do present in Section 2.4 computational results that clearly demonstrate the superiority of the exact linesearch when viewed over the entire course of the algorithm.

The fact that the exact linesearch yields a truly precise stepsize also has the benefit that the minimization of $\mathcal{L}(R)$ is numerically more stable using the exact linesearch, which improves the accuracy of the overall algorithm. In fact, we feel that the exact linesearch is an indispensable enhancement to the low-rank algorithm if one wishes to achieve robustness and accuracy when solving (1).

2.3 Dynamic Rank Updating

The rank r_k of the factorization $X_k = R_k R_k^T$ clearly plays an important role not only in the theoretical equivalence of (1) and (6) but also in the computational complexity of the various operations in the low-rank algorithm, such as function and gradient evaluations of the augmented Lagrangian. The choice of r_k according to Proposition 2.2 and (5) is a conservative choice in comparison what is often observed in practice, namely that the true rank of an optimal solution is much less than the theoretically predicted rank. However, it is of course not possible to guess the optimal rank beforehand.

Is there a practical way to determine the optimal rank as the algorithm progresses? For the purposes of discussion, at any point during the execution of the algorithm, let $s_k \leq r_k$ be the currently enforced rank for R_k. We may perform an LQ factorization of the matrix R_k with row pivoting (see Businger and Golub [11]), i.e.,

$$R_k = P\, L\, Q, \tag{8}$$

where $P \in \Re^{n_k \times n_k}$ is a permutation matrix, $L \in \Re^{n_k \times s_k}$ is lower triangular, and $Q \in \Re^{s_k \times s_k}$ is orthogonal. (The LQ factorization is performed according to the BLAS-3 algorithm by Quintana-Ortí et. al. [28], which is implemented in the LAPACK library [1].) We then

determine the smallest $s \leq s_k$ such that the partition

$$L = \begin{pmatrix} L_{11} & 0 \\ L_{21} & L_{22} \end{pmatrix}, \quad L_{11} \in \Re^{s \times s}$$

yields L_{22} sufficiently small compared to R_k; note that s may equal s_k. The precise condition that we use for L_{22} to be considered small is

$$\|L_{22}\|_F \leq \varepsilon \, \|R_k\|_F,$$

where ε is the machine precision and $\|\cdot\|_F$ indicates the Frobenius norm. We then consider the (numerical) rank of R_k to be s. Once s has been determined, we update s_k to s'_k as follows:

$$s'_k := \begin{cases} s+1 & \text{if } s < s_k, \\ r_k & \text{if } s = s_k. \end{cases} \tag{9}$$

If $s'_k \leq s_k$, we update R_k as

$$R'_k := P \, \tilde{L}, \quad \tilde{L} := \begin{pmatrix} L_{11} \\ L_{21} \end{pmatrix},$$

the idea being that $L_{22} \approx 0$ implies $LL^T \approx \tilde{L}\tilde{L}^T$, which in turn implies

$$R_k R_k^T = P \, L \, Q \, Q^T L^T P^T = P \, L \, L^T P^T \approx P \, \tilde{L} \, \tilde{L}^T P^T = R'_k (R'_k)^T.$$

On the other hand, if $s'_k > s_k$, then we update R_k to $R'_k = \begin{pmatrix} R_k & E \end{pmatrix}$, where $E \in \Re^{n_k \times (s'_k - s_k)}$ is a random matrix of small norm, the idea being to augment R_k by $s'_k - s_k$ independent columns while keeping $R'_k (R'_k)^T$ relatively close to $R_k R_k^T$.

It is important to point out that, in theory, determining the numerical rank s by the procedure outlined above is not always reliable, i.e., on some inputs, it can identify the numerical rank incorrectly. Practically speaking, however, it is almost always accurate, and in fact is widely used in practice because of its robustness and simplicity.

The update formula (9) is a conservative strategy that tries to avoid the possibility of a "false positive" regarding the optimal rank of R_k, that is, a situation in which we accidentally enforce a rank s_k such that all optimal solutions of (1) have $\text{rank}(X_k) > s_k$. Accordingly, we consider it dangerous to keep $s_k < r_k$ if $s = s_k$ since this equality would seem to indicate that the numerical rank may increase if s_k is relaxed. This explains the second half of the update rule, and the reasoning for the first half is similar. The inequality $s < s_k$ indicates that the optimal rank may be less than s_k, and we reduce the rank accordingly, but only to $s + 1$. If the optimal rank is indeed s, then this slightly relaxed rank will strengthen our confidence that we have not accidentally limited the rank too severely.

In the next subsection, we give computational results indicating that the dynamic adjustment of rank can benefit the performance of the low-rank algorithm.

2.4 Computational Results

In this subsection, we give computational results illustrating the performance of the low-rank algorithm on block SDPs coming from several different problem classes. We also provide

evidence that the exact linesearch performs better than the WP linesearch and that the rank reduction procedure speeds up the low-rank algorithm in many cases.

We first discuss some basic aspects and parameters of the low-rank algorithm that will apply not only in this subsection but also in Sections 3.1 and 4.1. Although it is likely that different parameters will cause different algorithmic behavior, we have chosen the ones mentioned below because we believe they are good choices for a variety of problems.

In every problem instance, we set the initial primal variable R to a random matrix of Frobenius norm 1 centered about 0, the initial dual variable y to 0, and the initial penalty parameter σ to $1/n$. In addition, during the course of the algorithm, our strategy for updating y and σ is as follows: after the solution of every tenth augmented Lagrangian subproblem, i.e., minimizing (7) for a fixed pair (y, σ), we multiply σ by 2.0; after all other subproblems, we update y by the standard augmented Lagrangian update rule. Finally, our stopping criterion for any given subproblem is

$$\frac{\|\nabla \mathcal{L}(R)\|_F}{1 + |C_{\max}|} \leq \frac{\rho_c}{\sigma},$$

where ρ_c is a positive parameter and $C_{\max}$ is defined to be the largest entry of C in absolute value. In this subsection and Section 3.1, we take $\rho_c = 10^{-1}$, which achieves medium accuracy. In Section 4.1, we set $\rho_c = 10^{-3}$ since our goal then will be to achieve higher accuracy. This stopping criterion has two aspects: first, since $\nabla \mathcal{L}(R) = 2\,S\,R$ (see equation (11) in Section 3), one can interpret the inequality as enforcing complementary slackness between $X := RR^T$ and S; second, as σ increases during the course of the algorithm, the tolerance becomes tighter and tighter. The overall algorithm is terminated once the primal feasibility becomes small, i.e.,

$$\frac{\|b - \mathcal{A}(RR^T)\|}{1 + |b_{\max}|} \leq \rho_f, \tag{10}$$

where ρ_f is a parameter and $b_{\max}$ is the largest entry of b in absolute value. In this subsection and Section 3.1, $\rho_f = 10^{-5}$ and in Section 4.1, we take $\rho_f = 10^{-8}$. Finally, every instance is given an upper bound of 5 hours (or 18,000 seconds) running time, and all problems are run on a Pentium 4 2.4 GHz with 1 GB of RAM under the Linux operating system.

In our computational results, we report four different pieces of information for each algorithmic variant on each problem instance. We give the final objective value $C \bullet RR^T$, the final infeasibility $\|b - \mathcal{A}(RR^T)\|/(1 + |b_{\max}|)$ in accordance with (10), and the time (in seconds). We also report the minimum eigenvalue of the final $S := C - \mathcal{A}^*(y)$ produced by the algorithm, which indicates how close the algorithm came to attaining dual feasibility and hence is a measure of optimality. The eigenvalue is calculated using the package ARPACK [20]. Note that the computation times do not include the calculation of $\lambda_{\min}[S]$.

Our test problems are a collection of large-scale SDPs from a variety of sources. Problem statistics and original references are given in Table 1. (For the interested reader, most of the problems are archived at H. Mittelmann's website [23].)

Tables 2 and 3 give results for three variants of the L-BFGS low-rank algorithm: with strong WP linesearch (labelled WP), exact linesearch (labelled EXACT), and exact linesearch with rank reduction (labelled REDUCE). Comparing WP and EXACT, we see a general trend that EXACT takes much less time to meet the feasibility tolerance while achieving slightly

Table 1: Large-Scale Test Problems

name	name (short)	ℓ	n	r	m	nonzeros	source
cancer_100	cancer	1	569	145	10,469	172,634	[23]
G59mc	G59	1	5,000	101	5,000	38,858	[12]
ice_2.0	ice	1	8,113	128	8,113	40,380	[17]
brock400_4.co_th	brock	1	400	201	20,078	100,677	[8]
p_hat300-1.co_th	p_hat	1	300	261	33,918	79,367	[8]
cphil12	cphil12	1	364	158	12,376	66,430	[23]
neu3	neu3	2	420	123	7,364	87,575	[23]
neu3g	neu3g	1	462	127	8,007	106,953	[23]
rose15	rose15	2	137	89	3,860	9,184	[23]
BeH_2Sigma+_STO-6GN5r12g1T2	BeH	22	1,406	559	948	66,572	[35]
BH2_2A1_STO-6GN7r14g1T2	BH2	22	2,166	755	1,743	143,510	[35]
H3O+_1-A1_STO-6GN10r16g1T2_5	H3O	22	3,162	980	2,964	279,898	[35]
AlH_1-Sigma+_STO-6GN14r20g1T2_5	AlH	22	5,990	1,524	7,230	857,380	[35]
shmup5	shmup5	3	11,041	123	1,800	119,223	[36]

better accuracy in the optimality measurement. Even on those problems which did not converge in the time allotted, EXACT has better feasibility and optimality measurements.

Next comparing EXACT and REDUCE, it is clear that, on some problems, the rank reduction technique can reduce the time substantially. On other problems, there is not much difference between the two, and on still others, REDUCE is worse than EXACT. In terms of the feasibility and optimality measures, there does not seem to be a clear winner. Overall, we feel that these results indicate that the rank reduction is a promising method (perhaps on particular types of problems) that should be explored further. In fact, we adopt the rank reduction procedure for the computational results in Sections 3.1 and 4.1.

3 Incorporating Second-Order Information

When calculating descent steps for the minimization of (7), it is a good idea to include as much second-derivative information as possible. Roughly speaking, two extremes are possible: the steepest descent direction $-\nabla\mathcal{L}(R)$ and the Newton direction D, which solves $\mathcal{L}''(R)[D] = -\nabla\mathcal{L}(R)$. Here, $\mathcal{L}''(R)$ indicates the Hessian in the form of a linear operator acting on the space $\Re^{(n_1\times r_1)\cdots(n_\ell\times r_\ell)}$. Of course, the difficulty is that working with $\mathcal{L}''(R)$ either directly or indirectly can require large amounts of computation and/or storage. One approach to incorporate some second-order information is the L-BFGS approach, which attempts to approximate $\mathcal{L}''(R)$ using a small collection of points $\{R_j\}$ and the corresponding gradients $\{\nabla\mathcal{L}(R_j)\}$.

Another common approach is the truncated Newton method. The basic idea is to attempt the calculation of the Newton direction using the method of conjugate gradients (CG) for positive definite systems, which generates better and better descent steps approximating the Newton direction during its execution. However, since the Hessian may not be positive definite, the calculation is terminated, or truncated, as soon as negative eigenvalues for the Hessian are detected, and the method then returns the most recently available descent direction. The two main computational issues with this method are how to compute

Table 2: Objective values and times (in seconds) for the L-BFGS low-rank algorithm with strong WP linesearch ("WP"), exact linesearch ("exact"), and exact linesearch with rank reduction ("reduce"). A prefixed asterisk (∗) indicates that the algorithm did not reach the feasibility tolerance of 10^{-5} within the 18,000 seconds allotted.

PROBLEM	OBJECTIVE VALUE			TIME (s)		
	WP	exact	reduce	WP	exact	reduce
cancer	-27631.6751	-27629.7874	-27629.1628	*18,000	*18,000	*18,000
G59	-14624.6488	-14624.6531	-14624.6530	2,415	2,383	2,231
ice	6810.1062	6809.1454	6809.2753	2,677	2,173	2,076
brock	-43.5620	-42.2123	-42.0171	*18,000	*18,000	*18,000
p_hat	-10.0680	-10.0680	-10.0680	15,350	10,137	13,087
cphil12	0.0000	0.0000	0.0000	195	117	119
neu3	0.0000	0.0000	0.0000	760	446	824
neu3g	0.0001	0.0000	0.0000	1,267	1,265	910
rose15	-0.0047	-0.0048	-0.0048	1,811	1,025	1,009
BeH	16.6944	16.6943	16.6944	963	493	546
BH2	30.4356	30.4327	30.4329	8,846	3,526	2,778
H3O	90.1136	90.1117	90.1213	6,717	7,986	3,934
AlH	246.0347	245.7722	245.1691	*18,021	*18,001	*18,008
shmup5	-26152.5137	-24118.4723	-24920.6478	*18,000	*18,000	*18,000

Table 3: Infeasibility and optimality ($\lambda_{\min}[S]$) measurements for the L-BFGS low-rank algorithm with strong WP linesearch ("WP"), exact linesearch ("exact"), and exact linesearch with rank reduction ("reduce"). For both measurements, values closer to zero are better. A prefixed asterisk (∗) indicates that the algorithm did not reach the feasibility tolerance of 10^{-5} within the 18,000 seconds allotted.

PROBLEM	INFEASIBILITY			OPTIMALITY ($\lambda_{\min}[S]$)		
	WP	exact	reduce	WP	exact	reduce
cancer	*9e−03	*6e−03	*5e−03	-5e−03	-7e−03	-3e−03
G59	9e−06	9e−06	1e−05	-7e−05	-3e−06	-3e−06
ice	6e−06	7e−06	8e−06	-2e−07	-1e−07	-2e−07
brock	*2e−02	*1e−02	*1e−02	-4e−02	-4e−02	-6e−02
p_hat	1e−05	1e−05	1e−05	-3e−04	-2e−04	-4e−04
cphil12	9e−06	1e−05	1e−05	-9e−04	-8e−04	-8e−04
neu3	1e−05	1e−05	1e−05	-2e−03	-6e−03	-1e−03
neu3g	1e−05	1e−05	1e−05	-2e−02	-1e−03	-8e−03
rose15	1e−05	1e−05	1e−05	-2e−02	-5e−03	-1e−03
BeH	1e−05	1e−05	1e−05	-1e−03	-3e−03	-2e−03
BH2	8e−06	8e−06	1e−05	-3e−03	-2e−04	-1e−03
H3O	1e−05	1e−05	1e−05	-4e−03	-2e−04	-5e−04
AlH	*4e−03	*3e−03	*7e−03	-4e−02	-3e−03	-1e−02
shmup5	*1e+01	*3e+00	*1e+01	-2e−01	-5e−02	-4e−02

Hessian-vector products for the CG method and how many Hessian-vector products must be performed by the method.

In the case of the low-rank algorithm, we propose the truncated Newton method as an intriguing alternative to the L-BFGS approach. We first discuss how to compute Hessian-vector products and then discuss a practical method for reducing the number of Hessian-vector products required. Preconditioning strategies for further reducing the number of Hessian-vector products are discussed in Section 4.

The gradient of (7) is given by

$$\nabla \mathcal{L}(R) = 2\, S\, R, \qquad S := C - \mathcal{A}^*(y + \sigma(b - \mathcal{A}(RR^T))), \tag{11}$$

and it is not difficult to see that the Hessian operator is given by

$$\mathcal{L}''(R)[D] = 2\, S\, D + 2\, \sigma\, \mathcal{A}^*(\mathcal{A}(RD^T + DR^T))R. \tag{12}$$

Hence, a single Hessian-vector product, i.e., one evaluation of $\mathcal{L}''(R)$ at an arbitrary D, requires at most the equivalent of two function evaluations and two gradient evaluations.

Looking into the structure (12) a bit more closely, we gain further insight.

Proposition 3.1 *The constituent operator*

$$\mathcal{M}(R)[D] := \mathcal{A}^*(\mathcal{A}(RD^T + DR^T))R$$

of (12) can be expressed as

$$\mathcal{M}(R)[D] = 2 \sum_{i=1}^{m} (M_i \bullet D) M_i, \qquad M_i := A_i R \tag{13}$$

and as such is a rank-m positive semidefinite operator.

Proof. Equation (13) is established as

$$\begin{aligned} \mathcal{A}^*(\mathcal{A}(RD^T + DR^T))R &= \left(\sum_{i=1}^{m} \left(A_i \bullet (RD^T + DR^T) \right) A_i \right) R \\ &= \sum_{i=1}^{m} \left(2\, A_i \bullet RD^T \right) A_i R = 2 \sum_{i=1}^{m} \left(A_i R \bullet D \right) A_i R, \end{aligned}$$

and this clearly implies that the dimension of the range space of $\mathcal{M}(R)$ is at most m. Finally, $\mathcal{M}(R)$ is positive semidefinite because

$$\mathcal{M}(R)[D] \bullet D = \left(2 \sum_{i=1}^{m} (M_i \bullet D) M_i \right) \bullet D = 2 \sum_{i=1}^{m} (M_i \bullet D)^2 \geq 0.$$

■

A simple corollary is the following:

Corollary 3.2 *If S is positive (semi)definite, then $\mathcal{L}''(R)$ is positive (semi)definite.*

Said differently, if the Hessian $\mathcal{L}''(R)$ is not positive definite, then it is due completely to negative eigenvalues in S. This will be important in Section 4 when developing preconditioners for the truncated Newton method. It is also interesting to note that the convergence proof for the low-rank algorithm [7] implies that the sequence of matrices S produced by the algorithm converges to a positive semidefinite matrix.

Proposition 3.1 indicates that preprocessing the matrices M_i in (13) can speed up the truncated Newton method by facilitating the calculation of $\mathcal{M}(R)[D]$. Specifically, M_i is nonzero only in those rows where A_i is nonzero, which implies that M_i can be computed and stored taking advantage of sparsity. Then calculating $\mathcal{M}(R)[D]$ requires m sparse dot products and vector additions. Besides these computational advantages, we have found that preprocessing M_i and computing $\mathcal{M}(R)[D]$ as described yields improved accuracy over a direct calculation of $\mathcal{M}(R)[D]$ based on a literal implementation of (12).

As discussed above, the truncated Newton method is designed to terminate once the CG method detects negative eigenvalues in the Hessian. However, we propose a different stopping criterion that we have found produces much better descent directions in practice. The idea uses the fact that we can compute an exact linesearch in about three function evaluations of $\mathcal{L}$ as described in Section 2.2. Suppose that $\{D_j\}$ is the sequence of descent directions produced by the CG method, and let $\{\bar{R}_j\}$ be defined as $\bar{R}_j := R + \bar{\alpha}_j D_j$, where the stepsize $\bar{\alpha}_j$ minimizes the one-dimensional function $\mathcal{L}(R + \alpha D_j)$ over $\alpha \in [0, 1]$. Our approach is to compute $\bar{\alpha}_j$ and $\mathcal{L}(\bar{R}_j)$ every five CG iterations and to terminate the CG method once we see a deterioration in the value $\mathcal{L}(\bar{R}_j)$, that is, we return the descent direction D_{j-5} if $\mathcal{L}(\bar{R}_{j-5})$ is less than $\mathcal{L}(\bar{R}_j)$. Overall, this strategy produces better directions and reduces the number of Hessian-vector products — enough so that the extra calculations are well worth the expense.

One additional detail regarding the truncated Newton method is that, on top of the stopping criterion mentioned above, we also implement a stopping tolerance on the residual of the Newton equation. More specifically, we terminate the CG iteration once a descent direction D is obtained satisfying

$$\|\mathcal{L}''(R)[D] + \nabla\mathcal{L}(R)\|_F \leq \min\left(0.1, \|\nabla\mathcal{L}(R)\|_F\right).$$

3.1 Computational Results

In Table 4, we give computational results on the test problems from Table 1 comparing the L-BFGS low-rank algorithm with exact linesearch and rank reduction (this is method REDUCE from Tables 2 and 3) with the truncated Newton algorithm just introduced, also with exact linesearch and rank reduction. All features of the table are as discussed in Section 2.4.

The most striking observation concerning Table 4 is that, on most problems for which the L-BFGS method finished within the time allotted, the truncated Newton took considerably more time. Clearly, the truncated Newton method can underperform the L-BFGS method on certain problems. On several problems, however, namely *G59*, *ice*, and *shmup5*, the truncated Newton method clearly outperformed the L-BFGS method. In particular, on *G59*

Table 4: Performance of the L-BFGS and truncated Newton variants of the low-rank algorithm. Both methods use the exact linesearch and rank reduction technique. A prefixed asterisk ($*$) indicates that the algorithm did not reach the feasibility tolerance of 10^{-5} within the 18,000 seconds allotted. A prefixed dagger ($\dagger$) indicates that the ARPACK calculation of $\lambda_{\min}[S]$ did not converge; instead the best estimate of $\lambda_{\min}[S]$ was returned.

PROBLEM	OBJECTIVE VALUE		TIME (s)		INFEASIBILITY		OPTIMALITY ($\lambda_{\min}[S]$)	
	L-BFGS	T-NEWT	L-BFGS	T-NEWT	L-BFGS	T-NEWT	L-BFGS	T-NEWT
cancer	-27629.16276	-27627.65200	*18,000	*18,033	5e−03	3e−03	-3e−03	-2e−04
G59	-14624.65301	-14624.65313	2,231	1,609	1e−05	9e−06	-3e−06	-1e−06
ice	6809.27527	6809.06282	2,076	957	8e−06	3e−06	-2e−07	-2e−06
brock	-42.01712	-40.32657	*18,000	*18,000	1e−02	5e−03	-6e−02	-2e−02
p_hat	-10.06805	-10.18626	13,087	*18,015	1e−05	2e−03	-4e−04	-4e−02
cphil12	0.00000	0.00000	119	981	1e−05	7e−06	-8e−04	-2e−04
neu3	0.00000	0.00002	824	*18,744	1e−05	2e−05	-1e−03	-3e−03
neu3g	0.00001	0.00003	910	*18,161	1e−05	2e−05	-8e−03	-2e−03
rose15	-0.00483	-0.00484	1,009	4,351	1e−05	1e−05	-1e−03	-1e−03
BeH	16.69436	16.69389	546	909	1e−05	6e−06	-2e−03	-1e−04
BH2	30.43289	30.43113	2,778	*13,037	1e−05	9e−06	-1e−03	-3e−04
H3O	90.12125	90.11516	3,934	*19,867	1e−05	1e−04	-5e−04	-2e−02
AlH	245.16909	243.84060	*18,008	*18,116	7e−03	2e−02	-1e−02	-4e−04
shmup5	-24920.64776	-23821.89118	*18,000	*18,071	1.E+01	5e−05	-4e−02	†-1e+00

and *ice*, the truncated method finished earlier, and on *shmup5*, it was on track to finish earlier as evidenced by the low amount of infeasibility (5e−05 compared to 1e+01). Unfortunately, the ARPACK calculation of $\lambda_{\min}[S]$ failed for the truncated Newton method on *shmup5*, and so we are unable to judge optimality with much confidence.

Although these results clearly do not suggest that one should use the truncated Newton method on all problems, we believe that they do demonstrate the truncated Newton method is an important alternative to the L-BFGS method, which works very well on some problems.

4 Preconditioning the Newton System

In this section, we discuss preconditioning strategies to further reduce the number of Hessian-vector products required by the truncated Newton method in each iteration of the low-rank algorithm. Although the preconditioning can be effective in reducing the number of Hessian-vector products, on most problems the gains from this reduction are unfortunately offset by the cost of constructing the preconditioner and applying it in each CG iteration. Some computational results illustrating these two points are given in Section 4.1. Nonetheless, we include our ideas here in hopes that they will provide insight on preconditioning issues in the low-rank algorithm.

We propose two types of preconditioning strategies. The first can be considered an "automatic" strategy, i.e., one that is constructed without specifically exploiting any properties of the Hessian, while the second makes use of the structures of the Hessian described in Section 3.

The first is the limited memory BFGS preconditioner of Morales and Nocedal [24]. Al-

though this preconditioner is based on the same basic concepts as the L-BFGS method, the truncated Newton method with this preconditioner is not equivalent to the L-BFGS method. The gist of this preconditioner is that, during the execution of the CG method, information is produced which can be used to approximate the Hessian. It is then only required to organize and store this information appropriately for use as a preconditioner. Since this idea is simple to implement and fits with our overall goal of exploiting as much second-order information as possible without too much expense, we compare this preconditioner with the ones introduced next.

The second type of preconditioner that we propose is based on the structure of the Hessian as described in (12) and Proposition 3.1. In particular, we construct a preconditioner $\mathcal{P}[D]$ of the form

$$\mathcal{P}[D] := 2\,\hat{S}\,D + 2\,\sigma\,\hat{\mathcal{M}}[D], \qquad \hat{\mathcal{M}}[D] := 2\sum_{i=1}^{\hat{m}}(\hat{M}_i \bullet D)\hat{M}_i, \tag{14}$$

where $\hat{S}$ is a positive definite approximation of S and $\hat{\mathcal{M}}$ is a low-rank ($\hat{m} \ll m$) positive semidefinite approximation of $\mathcal{M}(R)$. Once $\hat{S}$ and $\hat{\mathcal{M}}$ are constructed, $\mathcal{P}$ can be inverted efficiently using the Sherman-Morrison-Woodbury formula, which requires the inversion of $\hat{S}$ and a related $\hat{m} \times \hat{m}$ positive definite matrix involving both $\hat{S}$ and $\hat{\mathcal{M}}$.

Although there can be many choices for $\hat{S}$, we experimented with two specific choices, namely a diagonal approximation $\hat{S}_{(d)}$ and an approximation $\hat{S}_{(c)}$ gotten by performing an incomplete Cholesky factorization of S. More specifically,

$$[\hat{S}_{(d)}]_{jj} = \begin{cases} |S_{jj}| & \text{if } S_{jj} \neq 0 \\ 1 & \text{otherwise} \end{cases} \quad \forall \;\; j = 1, \ldots, n,$$

and

$$\hat{S}_{(c)} = LL^T,$$

where L is a no-fill incomplete Cholesky factor of $S + \delta I$, for some $\delta \geq 0$. The final value of δ is determined by first attempting the incomplete Cholesky of $S + \delta I$ with $\delta = 0$ and then increasing δ if necessary until the factorization is successful. A no-fill factorization is chosen in order to exploit the sparsity of S.

For $\hat{\mathcal{M}}$, we experimented with three choices. The first, which we call $\hat{\mathcal{M}}_{(z)}$, takes $\hat{m} = 0$ so that $\hat{\mathcal{M}}_{(z)} = 0$. The second, which we call $\mathcal{M}_{(n)}$, takes $\hat{m} \approx \sqrt{m}$, and the collection $\{\hat{M}_i\}$ is the $\hat{m}$ largest members of $\{M_i\}$ in the Frobenius norm. The following generic proposition, whose proof we sketch, demonstrates the quality of $\hat{\mathcal{M}}_{(n)}$ among all approximations $\hat{\mathcal{M}}$ of rank $\hat{m}$ such that each $\hat{M}_i$ is taken as some convex combination of $M_1, \ldots, M_m$ but no single M_i is assigned more total weight than 1 among all $\hat{M}_i$, which helps to ensure diversity in $\hat{\mathcal{M}}$. (Note: The proposition uses the notation $A \succeq B$, which denotes that $A - B$ is positive semidefinite.)

Proposition 4.1 *For $p \geq q$, suppose $W \in \Re^{p\times q}$ has full column rank, and for a given $\hat{q} \leq q$, define $W^* \in \Re^{p\times\hat{q}}$ to be any matrix whose columns are the $\hat{q}$ largest-norm columns of W. Also define*

$$\mathcal{V} := \{V \in \Re^{q\times\hat{q}} : V^T e = e,\; Ve \leq e,\; V \geq 0\},$$

where e signifies the all-ones vector of appropriate dimension, and

$$\mathcal{W} := \{WV : V \in \mathcal{V}\}.$$

Then $W^* \in \mathcal{W}$, and $WW^T \succeq \hat{W}\hat{W}^T$ for all $\hat{W} \in \mathcal{W}$. Moreover, $W^*(W^*)^T$ yields the best approximation of WW^T in the sense that W^* minimizes $\text{trace}(WW^T - \hat{W}\hat{W}^T)$ *over $\hat{W} \in \mathcal{W}$.*

Proof. It is clear that there exists $V^* \in \mathcal{V}$ having a single 1 in each column such that $W^* = WV^*$, i.e., $W^* \in \mathcal{W}$.

Let $\hat{W} \in \mathcal{W}$ and let V satisfy $\hat{W} = WV$. Using that W has full column rank, the condition $WW^T \succeq \hat{W}\hat{W}^T$ is equivalent to the condition $I \succeq VV^T$, which holds if and only if $\lambda_{\max}[VV^T] \leq 1$, which can in turn be shown by the Gerschgorin circle theorem and the definition of $\mathcal{V}$.

Minimizing the function $\text{trace}(WW^T - \hat{W}\hat{W}^T)$ over $\hat{W} \in \mathcal{W}$ is equivalent to maximizing $\|WV\|_F^2$ over $V \in \mathcal{V}$, i.e., maximizing the sum of the squared norms of the columns of WV. Since we are maximizing a convex function over a convex set, an optimal solution occurs at an extreme point of $\mathcal{V}$. It is not difficult to see that the extreme points of $\mathcal{V}$ have exactly one 1 per column, from which it easily follows that W^* minimizes $\text{trace}(WW^T - \hat{W}\hat{W}^T)$ over $\hat{W} \in \mathcal{W}$. ∎

The third choice for $\hat{\mathcal{M}}$, which we call $\hat{\mathcal{M}}_{(e)}$, also takes $\hat{m} \approx \sqrt{m}$, but in this case, the collection $\{\hat{M}_i\}$ is determined as the $\hat{m}$ largest eigenvectors of $\mathcal{M}(R)$, each scaled by the square root of its corresponding eigenvalue. The quality of $\hat{\mathcal{M}}_{(e)}$ is given by the following theorem (Schmidt [29], Mirsky [22]).

Theorem 4.2 *Let $U \in \mathcal{S}^q$ have spectral decomposition QDQ^T. For any $\hat{q} \leq q$, let $\hat{D}$ be constructed from D by zeroing out the $q - \hat{q}$ smallest eigenvalues. Then $\hat{U} := Q\hat{D}Q^T$ is the closest rank-$\hat{q}$ matrix to U in both the Frobenuius norm and the 2-norm.*

For all but the smallest problem instances, it will only be possible to calculate $\hat{\mathcal{M}}_{(e)}$ using an iterative method for eigenvalues and eigenvectors, such as the Lanczos method. For this, we employ the library ARPACK [20].

It is not difficult to see that $\hat{S}_{(d)}$ is quicker to compute with than $\hat{S}_{(c)}$, but one would expect $\hat{S}_{(c)}$ to be a more effective preconditioner. Likewise, $\hat{\mathcal{M}}_{(z)}$ is quicker than $\hat{\mathcal{M}}_{(n)}$, which is quicker than $\hat{\mathcal{M}}_{(e)}$, but it is likely that $\hat{\mathcal{M}}_{(e)}$ makes the best preconditioner. We give computational results in the next subsection which support these ideas.

4.1 Computational Results

As mentioned at the beginning of this section, we unfortunately cannot claim that preconditioning the truncated Newton method works very well in practice, at least with the preconditioning strategies outlined above. However, in this subsection, we would at least like to demonstrate that preconditioning does reduce the number of Hessian-vector multiplications (or equivalently, the number of CG iterations). As a result, our conclusion is that constructing and applying our preconditioning strategies is currently too expensive. This in

Table 5: Medium-Scale Test Problems

name	ℓ	n	r	m	nonzeros	source
vibra2	3	337	35	144	1996	[19]
maxG11	1	800	41	800	2918	[5]

Table 6: Results of the truncated Newton variant of the low-rank algorithm when solving *vibra2* to high accuracy using different preconditioning methods. The columns give the preconditioning method, the final objective value, the time (in seconds), the total number of conjugate gradient iterations, the final infeasibility measure, and the final optimality measure ($\lambda_{\min}[S]$).

PC	OBJ	TIME	CGS	INFEAS	OPT
(none)	-166.0153616	58	52,656	8e−10	-9e−09
L-BFGS	-166.0153616	63	41,178	9e−09	-6e−07
$\hat{S}_{(d)}$ $\hat{\mathcal{M}}_{(z)}$	-166.0153616	90	46,533	4e−10	-6e−10
$\hat{S}_{(d)}$ $\hat{\mathcal{M}}_{(n)}$	-166.0153616	152	89,903	7e−09	-2e−07
$\hat{S}_{(d)}$ $\hat{\mathcal{M}}_{(e)}$	-166.0153616	206	49,562	6e−10	-7e−10
$\hat{S}_{(c)}$ $\hat{\mathcal{M}}_{(z)}$	-166.0153616	37	20,663	5e−10	-1e−09
$\hat{S}_{(c)}$ $\hat{\mathcal{M}}_{(n)}$	-166.0153616	38	19,812	9e−10	-7e−10
$\hat{S}_{(c)}$ $\hat{\mathcal{M}}_{(e)}$	-166.0153616	164	16,989	4e−09	-3e−09

turn leaves open the question of whether cheap, effective preconditioners can be constructed for the low-rank algorithm.

In Tables 6 and 7, we give results for eight preconditioning strategies on two medium-scale test problems; the problems are described in Table 5. The information displayed in the tables is the same as in Tables 2, 3, and 4 with the addition of the columns "CGS," which gives the total number of CG iterations required by each variant of the algorithm. Our goal is to illustrate the effect of preconditioning when solving for high accuracy, and so we set $\rho_c = 10^{-3}$ and $\rho_f = 10^{-8}$; explanations of these parameters and all other algorithmic decisions are given in Section 2.4. The eight strategies are: no preconditioning at all, the limited memory BFGS preconditioner, and all six possible combinations of $\{\hat{S}_{(d)}, \hat{S}_{(c)}\}$ and $\{\hat{\mathcal{M}}_{(z)}, \hat{\mathcal{M}}_{(n)}, \hat{\mathcal{M}}_{(e)}\}$ discussed above.

The results, which are indicative of many tests that we performed, illustrate that the number of CG iterations can be substantially reduced using preconditioning, but unfortunately a smaller running time is not guaranteed due to the overhead associated with preconditioning. Moreover, there is no clear pattern as to which is the "best" preconditioner. The performance of each particular preconditioner seems to be problem dependent.

5 Final Remarks

In this paper, we have explored how to improve the low-rank semidefinite programming algorithm. As a result, the algorithm is now applicable to any block SDP, and the speed,

Table 7: Results of the truncated Newton variant of the low-rank algorithm when solving *maxG11* to high accuracy using different preconditioning methods. The columns give the preconditioning method, the final objective value, the time (in seconds), the total number of conjugate gradient iterations, the final infeasibility measure, and the final optimality measure ($\lambda_{\min}[S]$).

PC	OBJ	TIME	CGS	INFEAS	OPT
(none)	-629.1647830	69	17,045	4e−09	-1e−08
L-BFGS	-629.1647830	95	6,574	7e−09	-5e−10
$\hat{S}_{(d)}$ $\hat{\mathcal{M}}_{(z)}$	-629.1647830	209	49,392	8e−09	-5e−09
$\hat{S}_{(d)}$ $\hat{\mathcal{M}}_{(n)}$	-629.1647830	329	16,653	9e−09	-1e−07
$\hat{S}_{(d)}$ $\hat{\mathcal{M}}_{(e)}$	-629.1647830	2,962	16,176	9e−09	-2e−08
$\hat{S}_{(c)}$ $\hat{\mathcal{M}}_{(z)}$	-629.1647830	87	13,357	9e−09	-1e−08
$\hat{S}_{(c)}$ $\hat{\mathcal{M}}_{(n)}$	-629.1647830	380	23,952	6e−09	-9e−08
$\hat{S}_{(c)}$ $\hat{\mathcal{M}}_{(e)}$	-629.1647830	3,049	23,137	5e−09	-5e−08

robustness, and accuracy of the method have been improved. We have introduced an alternative to the L-BFGS approach, namely the truncated Newton method, which works very well on some problems and hence extends the flexibility of the method. Preconditioning strategies for the truncated Newton method — with advantages and disadvantages — have also been discussed.

During the course of this study, we have considered other possible enhancements to the algorithm that unfortunately did not seem very promising in practice. We mention them here for completeness. First, we tested a variant of the algorithm in which SDP blocks X_k of small size were not factored as $X_k = R_k R_k^T$ but instead were handled in the augmented Lagrangian via a log-barrier term. This method was successful but converged much more slowly than the method described in this paper. Second, under the hypothesis that poor scaling in the data matrices A_i could cause problems for the low-rank algorithm, we premultiplied the constraints $\mathcal{A}(RR^T) = b$ by the the inverse of the Cholesky factorization of the operator $\mathcal{A} \circ \mathcal{A}^*$, so that the resultant system $\tilde{\mathcal{A}}(RR^T) = \tilde{b}$ would satisfy $\text{cond}[\tilde{\mathcal{A}} \circ \tilde{\mathcal{A}}^*] = 1$. The linear algebra involving the Cholesky factor was handled in a sparse fashion, but in this case as well, no real improvement was observed, except on one or two problems that we tested.

Further work is needed to enable high accuracy on general large-scale SDPs, and preconditioning the low-rank algorithm seems to be a promising area for future research. Another possible idea for improvement is parallelization; nearly all of the basic operations in the low-rank algorithm (e.g., function and gradient evaluations) are capable of being parallelized.

References

[1] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999.

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

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

[4] B. Borchers. CSDP, a C library for semidefinite programming. *Optimization Methods and Software*, 11(1):613–623, 1999.

[5] B. Borchers. SDPLIB 1.2, a library of semidefinite programming test problems. *Optimization Methods and Software*, 11(1):683–690, 1999.

[6] S. Burer. Semidefinite programming in the space of partial positive semidefinite matrices. *SIAM Journal on Optimization*, 14(1):139–172, 2003.

[7] S. Burer and R.D.C. Monteiro. Local minima and convergence in low-rank semidefinite programming. Manuscript, Department of Management Sciences, University of Iowa, Iowa City, IA, September 2003. Submitted to *Mathematical Programming*.

[8] 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.

[9] S. Burer, R.D.C. Monteiro, and Y. Zhang. Solving a class of semidefinite programs via nonlinear programming. *Mathematical Programming*, 93:97–122, 2002.

[10] S. Burer, R.D.C. Monteiro, and Y. Zhang. A computational study of a gradient-based log-barrier algorithm for a class of large-scale SDPs. *Mathematical Programming (Series B)*, 95:359–379, 2003.

[11] Peter Businger and Gene H. Golub. Handbook series linear algebra. Linear least squares solutions by Householder transformations. *Numerische Mathematik*, 7:269–276, 1965.

[12] C. Choi and Y. Ye. Solving sparse semidefinite programs using the dual scaling algorithm with an iterative solver. Manuscript, Department of Management Sciences, University of Iowa, Iowa City, IA 52242, USA, 2000.

[13] M. Fukuda and M. Kojima. Interior-point methods for lagrangian duals of semidefinite programs. Manuscript, Department of Mathematical and Computing Sciences, Tokyo Institute of Technology, 2-12-1 Oh-Okayama, Meguru-ku, Tokyo 152, Japan, December 2000.

[14] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, M. Booth, and F. Rossi. *GNU Scientific Library Reference Manual*, second edition. http://www.gnu.org/software/gsl/.

[15] C. Helmberg and K.C. Kiwiel. A spectral bundle method with bounds. ZIB Preprint SC-99-37, Konrad-Zuse-Zentrum, Berlin, Germany, December 1999. To appear in *Mathematical Programming*.

[16] C. Helmberg and F. Rendl. A spectral bundle method for semidefinite programming. *SIAM Journal on Optimization*, 10:673–696, 2000.

[17] J. Keuchel, C. Schnörr, C. Schellewald, and D. Cremers. Binary partitioning, perceptual grouping, and restoration with semidefinite programming. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(11):1364–1379, November 2003.

[18] Michal Kočvara and Michael Stingl. Pennon: a code for convex nonlinear and semidefinite programming. *Optim. Methods Softw.*, 18(3):317–333, 2003.

[19] M. Kočvara. On the modelling and solving of the truss design problem with global stability constraints. *Structural and Multidisciplinary Optimization*, 23(3):189–203, April 2002.

[20] R. B. Lehoucq, D. C. Sorensen, and C. Yang. *ARPACK Users' Guide: Solution of Large-Scale Eigenvalue Problems with Implicitly Restarted Arnoldi Methods*. Society for Industrial and Applied Mathematics, 1998.

[21] C-J. Lin and R. Saigal. On solving large scale semidefinite programming problems: a case study of quadratic assigment problem. Technical Report, Dept. of Industrial and Operations Engineering, The University of Michigan, Ann Arbor, MI 48109-2177, 1997.

[22] L. Mirsky. Symmetric gauge functions and unitarily invariant norms. *Quart. J. Math. Oxford Ser. (2)*, 11:50–59, 1960.

[23] H. D. Mittelmann. http://plato.la.asu.edu.

[24] José Luis Morales and Jorge Nocedal. Automatic preconditioning by limited memory quasi-Newton updating. *SIAM J. Optim.*, 10(4):1079–1096 (electronic), 2000.

[25] K. Nakata, K. Fujisawa, and M. Kojima. Using the conjugate gradient method in interior-points for semidefinite programs. *Proceedings of the Institute of Statistical Mathematics*, 46:297–316, 1998. In Japanese.

[26] J. Nocedal and S. Wright. *Numerical Optimization*. Springer-Verlag, 1999.

[27] 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.

[28] Gregorio Quintana-Ortí, Xiaobai Sun, and Christian H. Bischof. A BLAS-3 version of the QR factorization with column pivoting. *SIAM J. Sci. Comput.*, 19(5):1486–1494 (electronic), 1998.

[29] E. Schmidt. *Mathematische Annalen*, 63:433–476, 1907.

[30] Jos F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optim. Methods Softw.*, 11/12(1-4):625–653, 1999. Interior point methods.

[31] K.C. Toh. Solving large scale semidefinite programs via an iterative solver on the agumented systems. Manuscript, Department of Mathematics, National University of Singapore, 2 Science Drive, Singapore 117543, Singapore, January 2003.

[32] K.C. Toh and M. Kojima. Solving some large scale semidefinite programs via the conjugate residual method. *SIAM Journal on Optimization*, 12:669–691, 2002.

[33] R. H. Tütüncü, K. C. Toh, and M. J. Todd. *SDPT3: A Matlab software package for semidefinite-quadratic-linear programming, version 3.0*, August 2001. Available from `http://www.math.nus.edu.sg/~mattohkc/sdpt3.html`.

[34] Makoto Yamashita, Katsuki Fujisawa, and Masakazu Kojima. Implementation and evaluation of SDPA 6.0 (semidefinite programming algorithm 6.0). *Optim. Methods Softw.*, 18(4):491–505, 2003. The Second Japanese-Sino Optimization Meeting, Part II (Kyoto, 2002).

[35] Z. Zhao, B. J. Braams, M. Fukuda, M. L. Overton, and J. K. Percus. The reduced density matrix method for electronic structure calculations and the role of three-index representability conditions. *The Journal of Chemical Physics*, 120(5):2095–2104, 2004.

[36] J. Zowe, M. Kočvara, and M. P. Bendsøe. Free material optimization via mathematical programming. *Math. Programming*, 79(1-3, Ser. B):445–466, 1997.